IMPROVING THE TWO-ELECTRON INTEGRAL COMPUTATION IN MUNGAUSS

KUSHAN SAPUTANTRI

Improving the Two-Electron Integral Computation in MUNgauss

by

©Kushan Saputantri
Memorial University of Newfoundland, St.John's,
Newfoundland and Labrador, Canada

M.Sc., People's Friendship University Moscow, USSR(1995)

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Science

Computational Science
Memorial University of Newfoundland

November 2007

Improving the Two-Electron Integral Computation in MUNgauss

by

Kushan Saputantri

Memorial University of Newfoundland, St.John's, Newfoundland and Labrador, Canada

Abstract

A major step in the self consistent field method is assembling and diagonalizing the Fock matrix. In order to form the Fock matrix, one-electron integrals, overlap integrals, and two-electron integrals must be computed. Computation of the two-electron integrals is the most time consuming and computationally difficult among them. The focus of this research is to improve the two-electron integral computation code in MUNgauss.

Improved the two-electron integral computation code by introducing new Fortran 90/95 features, where applicable. Specialized subroutines were introduced to replace general two-electron computation subroutine `I2ER_SPDF`, in order to compute less complicated two-electron integrals. Parallel implementation of subroutine `I2ER_SPDF` was also investigated.

The two-electron integral code was improved by using Fortran 90 modules to improve readability, efficiency, and organization. Derived types, *"select case"*, and *"do"* con-

structs were introduced to improve readability. The introduction of dynamic memory allocation reduced the usage of memory and improved the code efficiency.

Subroutine I2ER_SPDF can be specialized by identifying the shell(s,p,d,f) quadruplets before calling the subroutine to compute the two-electron integrals. Specialized subroutines were introduced to compute two-electron integrals belonging to the ssss quadruplet to dddd quadruplet. Introduction of specialized subroutines to compute simpler types of integrals reduces the memory usage and reduces the time spent on two-electron integral computations.

Two-electron integrals are labeled using indices I, J, K, and L. For a given set of indices I, J, K, and L, there are three unique blocks of two-electron integrals. They are: IJKL, ILJK and IKJL. Three mutually independent blocks, IJKL, ILJK and IKJL were computed in parallel using three processors on the same node. Shared memory parallel computing with OpenMP was used for this purpose.

Results of the improved code and old code were compared for accuracy as well as time spent on computation using SGI Altix(Verdandi) housed at Memorial University.

Acknowledgement

First and foremost I would like to express my sincere gratitude to supervisor, Dr. Raymond Poirier, for continuous guidance, encouragements, and funding. I would like to thank ACEnet for computing facilities and all the students of theoretical chemistry research group for their effort in helping to make this project a success. Finally, I would like to thank my wife for all the encouragement, guidance, and understanding.

Contents

Abstract i

Acknowledgement iii

List of Figures vi

List of Tables vii

List of Appendices viii

1 The Electronic Schrödinger Equation 1

1.1 The wavefunction 3

1.2 The Hartree-Fock equation 4

1.2.1 The Introduction of Basis Set 6

1.3 Roothaan's Equation 6

1.3.1 The Orthogonalization of the Basis Set 8

1.4 Self-consistent-field (SCF) Procedure 10

1.5 Two-Electron Integrals 11

1.5.1 Product of Two Gaussian Functions 11

1.5.2 The Laplace Transform 14

1.5.3 The Gamma Function Integral 15

1.5.4 Two-Electron Integrals with Gamma Function Method 16

1.5.5 Two-Electron Integrals with Higher Angular Momentum 17

1.5.6 Two-Electron Integrals with Rys Quadrature Method 20

2 Why use Fortran 90/95 ? 22

2.1 New Fortran 90 Features 23

2.1.1 Free Source Form 23

2.1.2 *"if-then-else if-end if"* Construct 23

2.1.3 *"select case"* Statement 24

2.1.4 *"do"* Construct 24

2.1.5 Modules 25

2.1.6 Dynamic Memory Allocation 26

2.2 New features in Fortran 95 27

2.3 OpenMP 27

2.3.1 Shared Memory Architecture 28

2.3.2 Why use OpenMP 29

3 Computation of Two-Electron Integrals 33

3.1 Implementation of the Rys polynomial Method 34

3.1.1 Module mod_idfclc 34

3.1.2 Subroutine IDFCLC 34

3.1.3 Subroutine I2ER_SPDF 36

3.1.4 Subroutine I2ER_GSPDF 38

3.1.5 Specialized Subroutines 40

3.1.6 Parallel Computation of Integrals 42

4 Conclusion 49

A Two-Electron Integral for s Type Functions 51

B Module *mod_idfclc* 55

Bibliography 70

List of Figures

1-1 A molecular coordinate system: i, j − electrons; A, B− nuclei 2

2-1 A canonical shared memory architecture 31

2-2 A canonical message passing architecture 32

List of Tables

2.1 Comparing shared memory and distributed memory models 30

3.1 Three blocks of integrals in the case of $(ss \mid pp)$ for $I \neq J \neq K \neq L$. . 40

3.2 Integral computation timing with specialized subroutines using basis set STO-3G. 45

3.3 Integral computation timing with specialized subroutines using basis set 6-31G(d). 46

3.4 Integral computation timing with specialized subroutines using basis set 6-311G(d,p). 47

3.5 Average integral computation timing in seconds using sequential and parallel $I2ER\_SPDF$ with STO-3G basis set. 48

3.6 Average integral computation timing in seconds using sequential and parallel $I2ER\_SPDF$ with the 6-31G(d) basis set. 48

List of Appendices

Two-Electron Integral for s Type Functions **51**

Module *mod_idfele* **55**

Glossary of terms and symbols

$\hat{H}$: hamiltonian operator

Φ : wavefunction

E : energy of the system

r : position vector

α : alpha spin function

β : beta spin function

ω : spin variable

Ψ : slater determinant

χ : spin orbital

$\hat{\mathscr{J}}$: coulomb operator

$\hat{\mathscr{K}}$: exchange operator

ε : orbital energy

$\hat{f}$: fock operator

ϕ : basis function

ψ : molecular orbital

S : overlap matrix

F : Fock matrix

N : Number of electrons

Z : Atomic number of nucleus

∇ : Laplacian operator

X : Transformation matrix

P : Density matrix

G : Two-electron part of the Fock matrix

ϵ_1, ϵ_2 : Exponents of new Gaussians

G_A : Gaussian function centered at A

Γ : Gamma function

Chapter 1

The Electronic Schrödinger Equation

According to the Born-Oppenheimer approximation, nuclei are much heavier than electrons and move more slowly, therefore to a good approximation one can consider the electrons in a molecule to be moving in a field of fixed nuclei. The Schrödinger equation for a system of electrons in a field of fixed nuclei is called the electronic Schrödinger equation,

$$\hat{H}_{elec} \mid \Phi_{elec}\rangle = E_{elec} \mid \Phi_{elec}\rangle \tag{1.1}$$

where, $\hat{H}_{elec}$ is the Hamiltonian operator for a system like that shown in Figure 1-1, $\mid \Phi_{elec}\rangle$ is the wavefunction, and E_{elec} is the total electronic energy of the system. Nuclei are described by the position vectors R_A and R_B. Electrons are described by position vectors r_i and r_j. Hamiltonian operator is given in eq.(1.2),

$$\hat{H}_{elec} = -\frac{1}{2}\sum_{i=1}^{N} \nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{1}{r_{ij}} \tag{1.2}$$

where, first term represents the kinetic energy of electrons, second term represents attraction between electrons and nuclei, and third term represents repulsion between electrons. The solution to the electronic Schrödinger equation is the electronic wavefunction. The electronic energy (E_{elec}), and electronic wavefunction (Φ_{elec}) depend

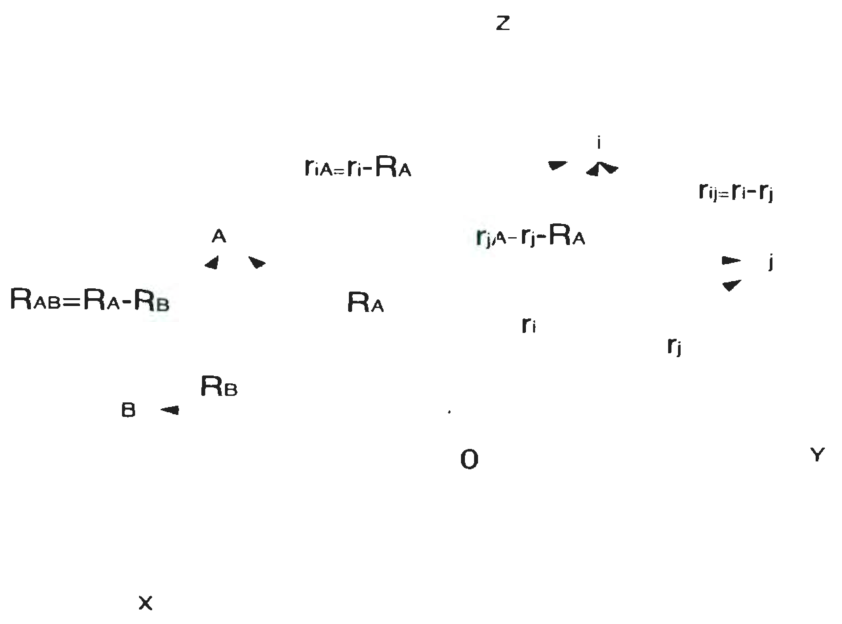

Figure 1-1: A molecular coordinate system: i, j – electrons; A, B– nuclei

explicitly on electronic coordinates, but depend parametrically on the nuclear coordinates [1].

$$\Phi_{elec} = \Phi(\{r_i\};\{R_A\}) \tag{1.3}$$

$$E_{elec} = E(\{R_A\}) \tag{1.4}$$

The total energy is given as:

$$E_{tot} = E_{elec} + \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}} \tag{1.5}$$

1.1 The wavefunction

According to eq.(1.2), the electronic Hamiltonian depends only on the spatial coordinates of the electrons, but to describe an electron completely, spin must also be specified. Therefore, two spin functions, $\alpha(\omega)$ and $\beta(\omega)$, which represent spin up and spin down are introduced in relation to nonrelativistic theory. These are functions of the spin variable ω, and they are taken to be complete and orthonormal. Accordingly, an electron can be described with three spatial coordinates and a spin coordinate. If these four coordinates are denoted collectively as x, then the wavefunction for an N-electron system can be described as $\Phi(x_1, x_2, ...x_N)$. According to the antisymmetry principle, electrons must be described by wavefunctions which are antisymmetric with respect to the interchange of the coordinates of a pair of electrons. A wavefunction that satisfies Schrödinger's equation and the antisymmetry property can be obtained by using Slater determinants. The Slater determinant for a system with N-electrons can be written as follows,

$$\Psi(x_1, x_2, ..., x_N) = |\chi_i\chi_j...\chi_k\rangle = (N!)^{-1/2}\begin{vmatrix} \chi_i(x_1) & \chi_j(x_1) & \dots & \chi_k(x_1) \\ \chi_i(x_2) & \chi_j(x_2) & \dots & \chi_k(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \chi_i(x_N) & \chi_j(x_N) & \dots & \chi_k(x_N) \end{vmatrix}$$

where $(N!)^{-1/2}$ is the normalization factor, and χ represents spin orbitals. This Slater determinant has N electrons occupying N spin orbitals, and the electrons are indistinguishable.

1.2 The Hartree-Fock equation

According to the Hartree-Fock approximation, the simplest antisymmetric wavefunction which can be used to describe the ground state of an N-electron system is a single Slater determinant,

$$|\Psi_0\rangle = |\chi_1\chi_2\cdots\cdots\chi_a\chi_b\cdots\chi_N\rangle \tag{1.6}$$

According to the variational principle, the best spin orbitals are those spin orbitals which minimize the electronic energy E_0,

$$\begin{aligned} E_0 &= \langle\Psi_0|\hat{H}|\Psi_0\rangle = \sum_a \langle a|h|a\rangle + \frac{1}{2}\sum_{ab}\langle ab||ab\rangle \\ &= \sum_a \langle a|\hat{h}|a\rangle + \frac{1}{2}\sum_{ab}\langle ab|ab\rangle - \langle ab|ba\rangle \end{aligned} \tag{1.7}$$

Spin orbitals can be systematically changed while preserving the orthonormality until the energy is minimized and the equation that represents the best spin orbitals can be obtained, eq.(1.8).

$$\begin{aligned} &\hat{h}(x_1)\chi_a(x_1) + \sum_{b\neq a}\left[\int_{-\infty}^{\infty} |\chi_b(x_2)|^2 r_{12}^{-1}\right] dx_2\chi_a(x_1) \\ &- \sum_{b\neq a}\left[\int_{-\infty}^{\infty} \chi_b^*(x_2)\chi_a(x_2)r_{12}^{-1}\right] dx_2\chi_b(x_1) = \varepsilon_a\chi_a(x_1) \end{aligned} \tag{1.8}$$

where,

$$\hat{h}(x_1) = -\frac{1}{2}\nabla^2 - \sum_A \frac{Z_A}{r_{1A}} \tag{1.9}$$

is the sum of the kinetic energy and potential energy of an electron. In eq.(1.8), the second and third terms represent the electron-electron interaction. The first of the two-electron terms is called the coulomb term, and the second is called the exchange term. The coulomb term is the total average potential acting on the electron χ_a arising from $N-1$ electrons in other spin orbitals. It is convenient to define the coulomb operator as:

$$\hat{\mathscr{J}}_b(x_1) = \int_{-\infty}^{+\infty} |\chi_b(x_2)|^2 r_{12}^{-1} dx_2 \tag{1.10}$$

The exchange term which arises from the antisymmetry property of the single determinant wavefunction does not have a classical explanation as for the coulomb term. The exchange operator is defined as,

$$\hat{\mathscr{K}}_b(x_1)\chi_a(x_1) = \left[\int_{-\infty}^{+\infty} \chi_b^*(x_2) r_{12}^{-1} \chi_a(x_2) dx_2\right] \chi_b(x_1) \tag{1.11}$$

Using the above two terms, eq.(1.8) can be written as follows:

$$\left[\hat{h}(x_1) + \sum_{b\neq a} \hat{\mathscr{J}}_b(x_1) - \sum_{b\neq a} \hat{\mathscr{K}}_b(x_1)\right] \chi_a(x_1) = \varepsilon_a \chi_a(x_1) \tag{1.12}$$

By removing the restriction $b \neq a$, eq.(1.12) can be converted to an eigenvalue type equation,

$$\hat{f}(x_1)|\chi_a\rangle = \varepsilon_a|\chi_a\rangle \tag{1.13}$$

where, $\hat{f}(x_1)$ is the Fock operator

$$\hat{f}(x_1) = \hat{h}(x_1) + \sum_b [\hat{\mathscr{J}}_b(x_1) - \hat{\mathscr{K}}_b(x_1)] \tag{1.14}$$

The Hartree-Fock equation over spin orbitals can be converted to an equation over spatial orbitals by integrating out the spin functions. A spatial orbital $\psi(r)$ is a function of the position vector r and describes the spatial distribution of an electron such that $|\psi_i(r)|^2 dr$ is the probability of finding the electron in the small volume element dr surrounding r. The resulting equation is given as follows[1]:

$$\hat{f}(r_1)\psi_i(r_1) = \varepsilon_i \psi_i(r_1) \tag{1.15}$$

1.2.1 The Introduction of Basis Set

The Hartree-Fock equation can be converted to an algebraic equation by introducing the basis set. Using K known basis functions $\{\varphi_\mu(r)|\mu = 1,2,\dots K\}$, the unknown molecular orbitals can be written as linear expansions,

$$\psi_i = \sum_{\mu=1}^{K} C_{\mu i}\phi_\mu \qquad i = 1,2,\dots K \tag{1.16}$$

where $C_{\mu i}$ is expansion coefficient and ϕ_μ is basis function. Molecular calculations are performed using basis sets, which are composed of a finite number of basis functions centered at each atomic nucleus. The most widely used types of basis functions are Slater type functions and Gaussian type functions. Even though Slater type functions can more accurately represent atomic orbitals, Gaussian type functions are used widely because of the ease of computation. The unnormalized Cartesian Gaussian function with center $\mathbf{R}$ can be written as:

$$\phi(\mathbf{r};\alpha,\overline{n},\mathbf{R}) = (x-R_x)^l(y-R_y)^m(z-R_z)^n\exp\left[-\alpha(\mathbf{r}-\mathbf{R})^2\right] \tag{1.17}$$

where $\mathbf{r} = (x,y,z)$ represents the coordinates of the electron, α is the Gaussian exponent, and $\overline{n}$ denotes a set of integers l, m and n.

1.3 Roothaan's Equation

By replacing the molecular orbitals in eq.(1.15) with a linear combination of basis functions, the following is obtained,

$$\hat{f}(r_1)\sum_\nu C_{\nu i}\phi_\nu(r_1) = \varepsilon_i\sum_\nu C_{\nu i}\phi_\nu(r_1) \tag{1.18}$$

Multiplying on the left by ϕ_μ^* and integrating, gives

$$\sum_\nu C_{\nu i}\int_{-\infty}^{+\infty}\phi_\mu^*(r_1)\hat{f}(r_1)\phi_\nu(r_1)dr_1 = \varepsilon_i\sum_\nu C_{\nu i}\int_{-\infty}^{+\infty}\phi_\mu^*(r_1)\phi_\nu(r_1)dr_1 \tag{1.19}$$

The overlap matrix and Fock matrix are defined as follows.

The overlap matrix:

$$S_{\mu\nu} = \int_{-\infty}^{+\infty} \phi_{\mu}^{*}(r_1)\phi_{\nu}(r_1)dr_1 \tag{1.20}$$

The Fock matrix:

$$F_{\mu\nu} = \int_{-\infty}^{+\infty} \phi_{\mu}^{*}(r_1)\hat{f}(r_1)\phi_{\nu}(r_1)dr_1 \tag{1.21}$$

In eq.(1.20), S is a $K \times K$ Hermitian matrix where K is the number of basis functions. Although basis functions are normalized and linearly independent, they are not orthogonal to each other. Therefore, the basis functions overlap with a magnitude of $0 \leq |S_{\mu\nu}| \leq 1$. In eq.(1.21), the Fock matrix is a $K \times K$ Hermitian matrix. The one-electron Fock operator $\hat{f}(r_1)$ and a set of one-electron basis functions ϕ_{μ} define the Fock matrix. Using the above definitions of the Fock matrix and the overlap matrix, the Hartree-Fock equation can be written as follows:

$$\sum_{\nu} F_{\mu\nu}C_{\nu i} = \varepsilon_i \sum_{\nu} S_{\mu\nu}C_{\nu i} \qquad i = 1, 2, ..., K \tag{1.22}$$

The single matrix equation, or the compact form of the above, is called Roothaan's equation, and is given by,

$$FC = SC\varepsilon \tag{1.23}$$

where C is the $K \times K$ square matrix of expansion coefficients,

$$\mathbf{C} = \begin{pmatrix} C_{11} & C_{12} & \cdots & C_{1k} \\ C_{21} & C_{22} & \cdots & C_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ C_{k1} & C_{k2} & \cdots & C_{kk} \end{pmatrix}$$

and ε is a diagonal matrix of orbital energies.

$$\varepsilon = \begin{pmatrix} \varepsilon_{11} & 0 & \dots & 0 \\ 0 & \varepsilon_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \varepsilon_{kk} \end{pmatrix}$$

Using the linear expansion of molecular orbitals, the $F_{\mu\nu}$ element of the Fock matrix can be written as

$$\begin{aligned} F_{\mu\nu} &= H_{\mu\nu}^{core} + \sum_{a}^{N/2} \sum_{\lambda\sigma} C_{\lambda a} C_{\sigma a}^{*} \left[2\langle \mu\nu|\lambda\sigma\rangle - \langle \mu\nu|\sigma\lambda\rangle\right] && (1.24) \\ &= H_{\mu\nu}^{core} + \sum_{\lambda\sigma} P_{\lambda\sigma} \left[\langle \mu\nu|\lambda\sigma\rangle - \frac{1}{2}\langle \mu\nu|\sigma\lambda\rangle\right] && (1.25) \\ &= H_{\mu\nu}^{core} + G_{\mu\nu} && (1.26) \end{aligned}$$

where, $H_{\mu\nu}^{core}$ is the one-electron part and $G_{\mu\nu}$ is the two-electron part that depends on the density matrix P and contribution from the two-electron integrals. The density matrix can be written using expansion coefficients as follows:

$$P_{\mu\nu} = 2 \sum_{a}^{N/2} C_{\mu a} C_{\nu a}^{*} \qquad (1.27)$$

1.3.1 The Orthogonalization of the Basis Set

For an orthonormal basis set, Roothaan's equation is an eigenvalue equation with S being a unit matrix. In this case, the eigenvalues and eigenvectors are found simply by diagonalizing the Fock matrix. For this purpose, an orthonormal basis set (ϕ') can be found using a transformation matrix X,

$$\varphi_{\mu}' = \sum_{\nu} X_{\nu\mu} \phi_{\nu}, \qquad \mu = 1, 2, ..., K \qquad (1.28)$$

where

$$\int_{-\infty}^{+\infty} \phi_\mu^{'*}(r)\phi_\nu^{'}(r)dr = \delta_{\mu\nu}, \tag{1.29}$$

and Kronecker delta δ is defined as,

$$\delta_{ij} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases} \tag{1.30}$$

Substituting eq.(1.28) into eq.(1.29), the following equation is obtained.

$$\begin{aligned} \int_{-\infty}^{+\infty} \phi_\mu^{'*}(r)\phi_\nu^{'}(r)dr &= \int_{-\infty}^{+\infty} \left[\sum_\lambda X_{\lambda\mu}^{*}\phi_\lambda^{*}(r)\right]\left[\sum_\sigma X_{\sigma\nu}\phi_\sigma(r)\right] dr \\ &= \sum_\lambda\sum_\sigma X_{\lambda\mu}^{*}\int_{-\infty}^{\infty} \phi_\lambda^{*}(r)\phi_\sigma(r)dr X_{\sigma\nu} \\ &= \sum_\lambda\sum_\sigma X_{\lambda\mu}^{*}S_{\lambda\sigma}X_{\sigma\nu} = \delta_{\mu\nu} \end{aligned} \tag{1.31}$$

Therefore X must be chosen such that,

$$X^{+}SX = I \tag{1.32}$$

where X^{+} is the complex conjugate of the matrix X. There are two common ways to obtain a transformation matrix. The first method is called symmetric orthogonalization, and uses the inverse square root of S for X. The second method is called canonical orthogonalization, in which the transformation matrix is obtained by dividing columns of unitary matrix by the square root of the corresponding eigenvalue. Once the transformation matrix is known, the relationship between the current coefficient matrix and the previous coefficient matrix can be obtained.

$$C^{'} = X^{-1}C, \qquad C = XC^{'} \tag{1.33}$$

Substitution of eq.(1.33) into eq.(1.23) gives

$$FXC^{'} = SXC^{'}\varepsilon \tag{1.34}$$

Multiplying eq.(1.34) on the left by X^+ gives

$$(X^+FX)C' = (X^+SX)C'\varepsilon \tag{1.35}$$

and replacing, X^+FX by F' gives

$$F'C' = C'\varepsilon \tag{1.36}$$

Roothaan's equation, eq.(1.36), is now a classic eigenvalue equation, and can be solved for C' by diagonalizing F'. Given C', C can be found using $C = XC'$.

1.4 Self-consistent-field (SCF) Procedure

The self-consistent-field procedure is the computational procedure for obtaining the restricted closed shell wavefunction for a molecule. The self-consistent-field solution is obtained with a finite basis set. The general SCF procedure is as follows[1]:

1. Determine the nuclear coordinates R_A, atomic number Z_A, number of electrons N, and a basis set for a given molecule.
2. Compute overlap integrals $S_{\mu\nu}$, one-electron integrals $H^{core}_{\mu\nu}$, and two-electron integrals $(\mu\nu|\lambda\sigma)$.
3. Obtain the transformation matrix X.
4. Compute a reasonable guess for the density matrix P.
5. Compute the matrix G, using eq.(1.25).
6. Add G and the core Hamiltonian H^{core} to obtain the matrix F using eq.(1.26).
7. Compute the transformed Fock matrix $F' = X^+FX$.
8. Diagonalize F' to obtain C' and ε.

9. Compute $C = XC'$.

10. Form the new density matrix $P = 2CC^{+}$ using eq.(1.27).

11. Determine whether the procedure has converged, comparing the previous density matrix and the current density matrix. If not go to step 5.

12. If the procedure has converged, stop the process.

1.5 Two-Electron Integrals

Two-electron integrals over basis functions have the following form:

$$(\mu_A \nu_B | \lambda_C \sigma_D) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \phi_\mu^{A*}(r_1) \phi_\nu^B(r_1) r_{12}^{-1} \phi_\lambda^{C*}(r_2) \phi_\sigma^D(r_2) dr_1 dr_2 \qquad (1.37)$$

where ϕ_μ^A is a basis function on nucleus A, centered at R_A, and the integral involves four centers: R_A,R_B, R_C and R_D. Linear combinations of Gaussian type functions are used as an approximation for basis fuctions eq.(1.17).

1.5.1 Product of Two Gaussian Functions

Two-electron integrals can be calculated relatively easily with Gaussian functions, because the product of two Gaussian functions, each on different centers is another Gaussian function on a third center, which lies on the line joining the first two Gaussian functions[2]. Accordingly, the product of two Gaussian functions centered on A and B is a new Gaussian with center P eq.(1.38) and the constant K_P is given in eq.(1.40). The exponent of the new Gaussian P is $\epsilon_1 = \alpha + \beta$. Similarly the product of two Gaussian functions centered on C and D produces a new Gaussian centered on Q eq (1.39),and the constant K_Q is given in the eq.(1.41). The exponent of the Gaussian Q is $\epsilon_2 = \gamma + \delta$.

$$\exp(-\alpha r_A^2)\exp(-\beta r_B^2) = K_p \exp(-\epsilon_1 r_P^2) \qquad (1.38)$$

$$\exp(-\gamma r_C^2)\exp(-\delta r_D^2) = K_Q \exp(-\epsilon_2 r_Q^2) \tag{1.39}$$

and,

$$K_P = \exp\left(-\frac{\alpha\beta}{\alpha+\beta}\overline{AB}^2\right) \tag{1.10}$$

$$K_Q = \exp\left(-\frac{\gamma\delta}{\gamma+\delta}\overline{CD}^2\right) \tag{1.11}$$

where $(\overline{AB})^2 = (\overline{A} - \overline{B})^2$ and $(\overline{CD})^2 = (\overline{C} - \overline{D})^2$. The coordinates of the new Gaussian centers P and Q are given by the following formulas:

$$\overline{P} = \frac{\alpha\overline{A} + \beta\overline{B}}{(\alpha+\beta)}, \tag{1.12}$$

$$\overline{Q} = \frac{\gamma\overline{C} + \delta\overline{D}}{(\gamma+\delta)} \tag{1.13}$$

The general expression for the product of two Gaussian type functions is as follows:

$$G_A G_B = x_A^l y_A^m z_A^n \exp(-\alpha r_A^2) x_B^{l'} y_B^{m'} z_B^{n'} \exp(-\beta r_B^2) \tag{1.14}$$

where G_A and G_B are Gaussian functions centered on nucleus A and nucleus B respectively. The Gaussian product has been generalized by including the angular part as cubic harmonic functions: $(x_A^l y_A^m z_A^n)$ $(x_B^{l'} y_B^{m'} z_B^{n'})$. The normalization factors for the Gaussian functions are not included in this equation. In order to derive the product of the angular part, x_A can be expressed as follows:

$$\begin{aligned} x_A &= x - A_x \\ &= (x - P_x) + (P_x - A_x) \\ &= x_p + \overline{PA}_x \end{aligned} \tag{1.15}$$

Then x_A^l becomes:

$$\begin{aligned} x_A^l &= (x_p + \overline{PA}_x)^l \\ &= \sum_{i=0}^{l} \overline{PA}_x^{l-i} \binom{l}{i} x_p^i \end{aligned} \qquad (1.46)$$

The product $x_A^l x_B^{l'}$ can be written as:

$$\begin{aligned} x_A^l x_B^{l'} &= \sum_{i=0}^{l} \sum_{j=0}^{l'} \overline{PA}_x^{l-i} \overline{PB}_x^{l'-j} \binom{l}{i} \binom{l'}{j} x_p^{i+j} \\ &= \sum_{k=0}^{l+l'} \left[\sum_{\substack{i=0 \\ i+j=k}}^{l} \sum_{j=0}^{l'} \overline{PA}_x^{l-i} \overline{PB}_x^{l'-j} \binom{l}{i} \binom{l'}{j} \right] x_p^k \\ &= \sum_{k=0}^{l+l'} f_k(l, l', \overline{PA}_x, \overline{PB}_x) x_p^k \end{aligned} \qquad (1.47)$$

The product of two Gaussians, eq.(1.44), becomes,

$$G_A G_B = K_P \sum_{k_x=0}^{l+l'} \sum_{k_y=0}^{m+m'} \sum_{k_z=0}^{n+n'} f_{k_x} f_{k_y} f_{k_z} x_p^{k_x} y_p^{k_y} z_p^{k_z} \exp(-\gamma_1 r_p^2) \qquad (1.48)$$

The above equation can be separated into Cartesian components as

$$G_A G_B = K_P G_p^x G_p^y G_p^z \qquad (1.49)$$

where

$$G_p^h = \sum_{k_h} f_{k_h} h_p^{k_h} \exp(-\gamma_1 h_p^2) \qquad (1.50)$$

h represents x, y or z, and

$$f_{k_h} = \sum_{\substack{i=0 \\ i+j=k}}^{l} \sum_{j=0}^{l'} \overline{PA}_x^{l-i} \overline{PB}_x^{l'-j} \binom{l}{i} \binom{l'}{j} \qquad (1.51)$$

1.5.2 The Laplace Transform

The Laplace transform is used to replace the $\frac{1}{r}$ term in the two-electron integral with a Gaussian type function eq.(1.37). A new integration variable ρ is introduced, and a general expression for $(\frac{1}{r})^\lambda$ is derived in place of $\frac{1}{r}$. The explicit form of the Laplace transform is:

$$L(t) = \int_0^{+\infty} \exp(-t\rho)g(\rho)d\rho \tag{1.52}$$

Replacing t by r^2 and $g(\rho)$ by $\rho^{\frac{\lambda}{2}-1}$ in eq.(1.52) gives the following

$$L(r^2) = \int_0^{+\infty} \exp(-r^2\rho)\rho^{\frac{\lambda}{2}-1}d\rho \tag{1.53}$$

Letting $u = \rho r^2$ gives,

$$L(r^2) = \left(\frac{1}{r}\right)^\lambda \int_0^{+\infty} \exp(-u)u^{\frac{\lambda}{2}-1}du \tag{1.54}$$

Using the Gamma function method we obtain

$$L(r^2) = \left(\frac{1}{r}\right)^\lambda \Gamma\left(\frac{\lambda}{2}\right) \tag{1.55}$$

$$\Leftrightarrow \left(\frac{1}{r}\right)^\lambda = \frac{1}{\Gamma(\lambda/2)} \int_0^{\infty} \exp(-r^2\rho)\rho^{\frac{\lambda}{2}-1}d\rho \tag{1.56}$$

Eq.(1.56) can also be written as

$$\left(\frac{1}{r}\right)^\lambda = \frac{1}{\Gamma(\lambda/2)} \int_0^{+\infty} \exp(-r^2u^2)u^{\frac{\lambda}{2}-1}du \tag{1.57}$$

by letting $u^2 = \rho$.

1.5.3 The Gamma Function Integral

The Gamma function is described by the integral

$$F_k(\gamma) = \int_{-\infty}^{+\infty} \exp(-\gamma t^2) t^k dt \tag{1.58}$$

This integral is nonzero as long as $\bar{k}$ is an even integer. Therefore eq.(1.58) can be written as:

$$\begin{aligned}
F_{\bar{k}}(\gamma) &= 2\int_0^{+\infty} \exp(-\gamma t^2) t^{\bar{k}} dt \\
&= \gamma^{-(\bar{k}+1)/2} \int_0^{+\infty} \exp(-u) u^{(\bar{k}-1)/2} du \quad (\text{if } u = \gamma t^2) \\
&= \gamma^{-(\bar{k}+1)/2} \Gamma\left(\frac{\bar{k}+1}{2}\right)
\end{aligned} \tag{1.59}$$

The Gamma function can be written recursively as,

$$\Gamma(v) = (v-1)\Gamma(v-1) \tag{1.60}$$

where v is an integer greater than one. This recursive formula can be derived by following the steps given below:

$$\Gamma(v) = \int_0^{+\infty} u^{v-1} \exp(-u) du \tag{1.61}$$

Letting $r = u^{v-1}$ and $dv = \exp(-u)du$, the following expression is obtained:

$$\Gamma(v) = \left[u^{(v-1)} \exp(-u)\right]_0^{\infty} + (v-1)\int_0^{+\infty} \exp(-u) u^{v-2} du = (v-1)\Gamma(v-1) \tag{1.62}$$

Thus:

$$F_{\bar{k}}(\gamma) = \gamma^{-(\bar{k}+1)/2} \left(\frac{\bar{k}-1}{2}\right)!!\,\Gamma\left(\frac{1}{2}\right) \tag{1.63}$$

where,

$$\left(\frac{\bar{k}-1}{2}\right)!! = \left(\frac{\bar{k}-1}{2}\right)\left(\frac{\bar{k}-3}{2}\right)\left(\frac{\bar{k}-5}{2}\right)\cdots\left(\frac{3}{2}\right)\left(\frac{1}{2}\right)$$

In the particular case where $\bar{k} = 0$,

$$F_0(\gamma) = \int_{-\infty}^{+\infty} \exp(-\gamma t^2) dt = \sqrt{\frac{\pi}{\gamma}}$$
$$= \gamma^{-1/2} \Gamma\left(\frac{1}{2}\right) \tag{1.64}$$

Therefore,

$$\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi} \tag{1.65}$$

In the case where $\bar{k}$ is an even integer

$$F_{\bar{k}}(\gamma) = \gamma^{-(\bar{k}+1)/2} \frac{(\bar{k}-1)!!}{2^{\bar{k}/2}} \sqrt{\pi}, \tag{1.66}$$

in the case where $\bar{k}$ is an odd integer:

$$F_{\bar{k}}(\gamma) = 0 \tag{1.67}$$

1.5.4 Two-Electron Integrals with Gamma Function Method

The general form of the two-electron integrals between Gaussian-type functions $(ab|cd)$ is given by:

$$(ab|cd) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} G_A^*(r_1) G_B(r_1) \frac{1}{r_{12}} G_C^*(r_2) G_D(r_2) dr_1 dr_2, \tag{1.68}$$

where, G_A, G_B, G_C,and G_D are four Gaussian functions centered on A, B, C and D respectively. Variables r_1 and r_2 represent the distance from the center of the coordinate system to the first and the second electron respectively. In the above equation $\frac{1}{r_{12}}$ can be replaced by the following term using Laplace transform eq.(1.57):

$$\frac{1}{r_{12}} = \frac{1}{\sqrt{\pi}} \int_0^{+\infty} \exp(-r_{12}^2)\rho^{-1/2} d\rho \tag{1.69}$$

If the final product of Gaussians is separated according to the contribution from the x, y, and z coordinates, the following form can be obtained:

$$(ab|cd) = \frac{K_P K_Q}{\sqrt{\pi}} \int_0^{+\infty} \rho^{-1/2} U_x^{(R)} U_y^{(R)} U_z^{(R)} d\rho \tag{1.70}$$

where K_P and K_Q are constants given in eq.(1.40) and eq.(1.41), $U_x^{(R)}$,$U_y^{(R)}$ and $U_z^{(R)}$ are of the form:

$$\begin{aligned} U_h^{(R)} = & \int_0^{+\infty} \int_0^{+\infty} h_A^{k_h} h_B^{l_h} h_C^{m_h} h_D^{n_h} \exp(-\rho(h_2 - h_1)^2) \\ & \times \exp(-\epsilon_1(h_1 - P_h)^2) \exp(-\epsilon_2(h_2 - Q_h)^2) dh_1 dh_2 \end{aligned} \tag{1.71}$$

$h \in \{x, y, z\}$. The final form of the simplest two-electron integral $(ss|ss)$, given in eq.(1.72), can be obtained by following the steps given in Appendix A,

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{\epsilon_1 \epsilon_2 (\epsilon_1 + \epsilon_2)^{\frac{1}{2}}} F_0 \left(\frac{\epsilon_1 \epsilon_2 \overline{PQ}^2 t^2}{\epsilon_1 + \epsilon_2} \right) \tag{1.72}$$

1.5.5 Two-Electron Integrals with Higher Angular Momentum

Given the formula for two-electron integrals over s functions, two-electron integrals for higher angular momentum have to be derived. S. F. Boys[3] introduced a method that uses differentiation of two-electron integrals of s type functions over nuclear coordinates to obtain those over higher angular momentum.

Lambda represents the sum of l,m and n given in eq.(1.17), and it is closely related to the total angular momentum number. The Angular momentum index is represented by $\overline{n} = (l, m, n)$. The functions with Lambda equal to 0, 1, 2 .. are referred to as s, p, d .. respectively. The single component of the s shell with angular momentum index $\overline{0} = (0, 0, 0)$ is designated as s. The components of the p shell have angular momentum indices $\overline{1}_i (i = x, y, z)$, where $\overline{1}_i$ is defined by $\overline{1}_i = (\delta_i x, \delta_i y, \delta_i z)$ using Kronecker delta. In this particular case these components will be designated p_x, p_y

and p_z. The Cartesian Gaussian function eq.(1.17) satisfies the following differential relationship[1],

$$\frac{\partial}{\partial R_i}\phi(\mathbf{r},\alpha,\overline{n},\mathbf{R}) = 2\alpha\phi(\mathbf{r},\alpha,\overline{n}+\overline{1}_i,\mathbf{R}) - N_i(\overline{n})\phi(\mathbf{r},\alpha,\overline{n}-\overline{1}_i,\mathbf{R}) \qquad (1.73)$$

where, $i = x, y, z$, and $N_i(\overline{n})$ is the value of the i^{th} component of the angular momentum index $\overline{n}$. The differential relation in eq.(1.73) allows the derivatives of two-electron integrals to be written as linear combinations of two-electron integrals of lower angular momentum. The first derivative can be written as follows:

$$\frac{\partial}{\partial A_i}(ab,cd)^{(\bar{k})} = 2\alpha\left[(a+\overline{1}_i)b,cd\right]^{(\bar{k})} - N_i(a)\left[(a-\overline{1}_i)b,cd\right]^{(k)} \qquad (1.74)$$

Recursive equations can be derived after some mathematical manipulation of eq.(1.74). Those equations can be used to obtain the recurrence expressions for two-electron integrals over Cartesian Gaussian functions. The two-electron integral $[(a+\overline{1}_i)b,cd]^{(k)}$ can be decomposed into lower angular momentum functions as follows:

$$\begin{aligned}
[(a+1_i)b,cd]^{(\bar{k})} \quad = \quad & (P_i - A_i)(ab,cd)^{(\bar{k})} + (W_i - P_i)(ab,cd)^{(k+1)} \\
& + \frac{1}{2\epsilon_1}N_i(a)\left[[(a-1_i)b,cd]^{(\bar{k})} - \frac{\rho}{\epsilon_2}[(a-1_i)b,cd]^{(k+1)}\right] \\
& + \frac{1}{2\epsilon_1}N_i(b)\left[[a(b-1_i),cd]^{(\bar{k})} - \frac{\rho}{\epsilon_2}[a(b-1_i),cd]^{(k+1)}\right] \\
& + \frac{1}{2(\epsilon_1+\epsilon_2)}N_i(c)[ab,(c-1_i)d]^{(k+1)} \\
& + \frac{1}{2(\epsilon_1+\epsilon_2)}N_i(d)[ab,c(d-1_i)]^{(\bar{k}+1)} \qquad (1.75)
\end{aligned}$$

For example, using eq.(1.75), the following recurrence expression for $(p_i s_j, s_k s_l)^{(0)}$ can be obtained:

$$(p_i s_j, s_k s_l)^{(0)} = (P_i - A_i)(s_i s_j, s_k s_l)^{(0)} + (W_i - P_i)(s_i s_j, s_k s_l)^{(1)}$$

The two-electron integral $[a(b+1_j),cd]^{(\bar{k})}$ can be decomposed into lower angular momentum functions as follows:

$$\begin{aligned}
[a(b+1_j),cd]^{(\bar{k})} &= (P_j - B_j)(ab,cd)^{(k)} + (W_j - P_j)(ab,cd)^{(k+1)} \\
&+\frac{1}{2\epsilon_1}N_j(a)\left[[(a-1_j)b,cd]^{(k)} - \frac{\rho}{\epsilon_2}[(a-1_j)b,cd]^{(k+1)}\right] \\
&+\frac{1}{2\epsilon_1}N_j(b)\left[[a(b-1_j),cd]^{(k)} - \frac{\rho}{\epsilon_2}[a(b-1_j),cd]^{(k+1)}\right] \\
&+\frac{1}{2(\epsilon_1+\epsilon_2)}N_j(b)[ab,(c-1_j)d]^{(\bar{k}+1)} \\
&+\frac{1}{2(\epsilon_1+\epsilon_2)}N_j(a)[ab,c(d-1_j)]^{(\bar{k}+1)} \qquad (1.76)
\end{aligned}$$

The following expression for $(p_i p_j, s_k s_l)^{(0)}$ can be obtained from the above,

$$\begin{aligned}
(p_i p_j, s_k s_l)^{(0)} = (P_j - B_j)(p_i s_j, s_k s_l)^{(0)} + (W_j - P_j)(p_i s_j, s_k s_l)^{(1)} + \\
\frac{\delta_{ij}}{2\alpha}\left((s_i s_j, s_k s_l)^{(0)} - \frac{\rho}{\alpha}(s_i s_j, s_k s_l)^{(1)}\right)
\end{aligned}$$

The two-electron integral $[ab,(c+1_l)d]^{(k)}$ can be expressed using lower angular momentum functions as follows:

$$\begin{aligned}
[ab,(c+1_k)d]^{(k)} &= (Q_k - C_k)(ab,cd)^{(k)} + (W_k - Q_k)(ab,cd)^{(k+1)} \\
&+\frac{1}{2\epsilon_2}N_k(d)\left[[ab,c(d-1_k)]^{(\bar{k})} - \frac{\rho}{\epsilon_2}[ab,c(d-1_k)]^{(k+1)}\right] \\
&+\frac{1}{2\epsilon_2}N_k(c)\left[[ab,(c-1_k)d]^{(\bar{k})} - \frac{\rho}{\epsilon_2}[ab,(c-1_k)d]^{(\bar{k}+1)}\right] \\
&+\frac{1}{2(\epsilon_1+\epsilon_2)}N_k(b)[a(b-1_k),cd]^{(\bar{k}+1)} \\
&+\frac{1}{2(\epsilon_1+\epsilon_2)}N_k(a)[(a-1_k)b,cd]^{(\bar{k}+1)} \qquad (1.77)
\end{aligned}$$

An expression for the $(p_i s, p_k s)^{(0)}$ can be obtained from the above,

$$(p_i s_j, p_k s_l)^{(0)} = (Q_k - C_k)(p_i s_j, s_k s_l)^{(0)} + (W_k - Q_k)(p_i s_j, s_k s_l)^{(1)} + \frac{\delta_{ik}}{2(\epsilon_1)}(s_i s_j, s_k s_l)^{(1)}$$

The general expression for the two-electron integral $[ab,c(d+1_l)]^{(k)}$ can be written as follows:

$$\begin{aligned}
[ab, c(d+1_l)]^{(k)} &= (Q_l - D_l)(ab, cd)^{(k)} + (W_l - Q_l)(ab, cd)^{(k+1)} \\
&+ \frac{1}{2\epsilon_2} N_l(d) \left[[ab, c(d-1_l)]^{(k)} - \frac{\rho}{\epsilon_2}[ab, c(d-1_l)]^{(k+1)} \right] \\
&+ \frac{1}{2\epsilon_2} N_l(c) \left[[ab, (c-1_l)d]^{(k)} - \frac{\rho}{\epsilon_2}[ab, (c-1_l)d]^{(k+1)} \right] \\
&+ \frac{1}{2(\epsilon_1 + \epsilon_2)} N_l(b)[a(b-1_l), cd]^{(k+1)} \\
&+ \frac{1}{2(\epsilon_1 + \epsilon_2)} N_l(a)[(a-1_l)b, cd]^{(k+1)} \qquad (1.78)
\end{aligned}$$

An expression for the $(p_i s_j, p_k s_l)^{(0)}$ can be obtained from the above.

$$\begin{aligned}
(p_i p_j, p_k p_l)^{(0)} &= (Q_l - D_l)(p_i p_j, p_k s_l)^{(0)} + (W_l - Q_l)(p_i p_j, p_k s_l)^{(1)} \\
&+ \frac{1}{2(\alpha + \beta)} \left(\delta_{il}(s_i p_j, p_k s_l)^{(1)} + \delta_{jl}(p_i s_j, p_k s_l)^{(1)} \right) \\
&+ \frac{\delta_{kl}}{2\beta} \left((p_i p_j, s_k s_l)^{(0)} - \frac{\rho}{\beta}(p_i p_j, s_k s_l)^{(1)} \right)
\end{aligned}$$

1.5.6 Two-Electron Integrals with Rys Quadrature Method

The Rys quadrature method gives an alternative method to the Gamma function method in computing two-electron integrals over Gaussian functions. In this method the Gaussian product theorem is applied twice to eq.(1.70), after introducing the following variables,

$$\eta_2 = \epsilon_2 + \rho$$

$$\bar{h}_2 = \frac{\epsilon_2 Q_h + \rho h_1}{\epsilon_2 + \rho}$$

$$\eta_1 = \epsilon_1 + \frac{\epsilon_2 \rho}{\epsilon_2 + \rho}$$

$$\bar{h}_1 = \frac{\epsilon_1 P_h + (\epsilon_2 \rho/(\epsilon_2 + \rho))Q_h}{\epsilon_1 + \epsilon_2 \rho/(\epsilon_2 + \rho)}$$

The following terms are introduced to simplify the equations.

$$\theta = \frac{\epsilon_1\epsilon_2}{\epsilon_1+\epsilon_2}$$

$$\nu_h = \eta_1(h_1-\overline{h}_1)^2+\eta_2(h_2-\overline{h}_2)^2$$

The two-electron repulsion integral can be written as follows,

$$(ab|cd) = \frac{K_pK_q}{\sqrt{\pi}}\int_0^\infty \rho^{-1/2}\exp\left(-\theta\overline{PQ}^2\frac{\rho}{\theta+\rho}\right)$$
$$\times\prod_{h=x,y,z}\int_{-\infty}^{\infty}h_A^{k_h}h_B^{l_h}\int_{-\infty}^{\infty}h_C^{m_h}h_D^{n_h}\exp(-\nu_h)dh_1dh_2d\rho \tag{1.79}$$

Introducing the new variable t, which satisfies the relationship $t^2=\frac{\rho}{\rho+\theta}$, eq.(1.79) becomes,

$$(ab|cd) = \frac{2K_pK_q}{\sqrt{\pi\theta(\epsilon_1+\epsilon_2)^{3/2}}}\int_0^1\exp(-\theta\overline{PQ}^2t^2)I_x^{(R)}I_y^{(R)}I_z^{(R)}dt \tag{1.80}$$

where,

$$I_h^{(R)} = (\eta_1\eta_2)^{1/2}\int_{-\infty}^{\infty}h_A^{k_h}h_B^{k_h}\int_{-\infty}^{\infty}h_C^{m_h}h_D^{n_h}\exp[-\nu_h(t^2)]dh_1dh_2 \tag{1.81}$$

$h\in\{x,y,z\}$. In the above expression $\eta_1\eta_2=\epsilon_1\epsilon_2/(1-t^2)$. Therefore, $I_h^{(R)}$ is an even polynomial of t. This integral can be computed by Rys quadrature method with[5, 6, 7]:

$$P_L(t^2) = I_x^{(R)}I_y^{(R)}I_z^{(R)}$$

$$(ab|cd) = \frac{2K_pK_q}{\theta\sqrt{\pi}}\sum_{i=1}^{k}I_x^{(R)}(t_i)I_y^{(R)}(t_i)I_z^{(R)}(t_i)W_i \tag{1.82}$$

where,$k>L/2$, and t_i and W_i are the roots and the weights of the k^{th} Rys polynomial.

Chapter 2

Why use Fortran 90/95 ?

Fortran 90/95 has been used in programming the two-electron package in MUNgauss. A team lead by John Backus developed the Fortran or the Formula translation system in 1954. It is one of the earliest high level programming languages, and its first standard was created in 1966. A new standard was created in 1976 and was named Fortran 77. The need to modernize the Fortran 77 became apparent with the emergence of new languages like C and C+ +.

The new standard of Fortran 90 has all the good features of Fortran 77 as well as many new features of modern languages. Fortran 90 has almost all the features that are important to scientific programming and most of the features of object oriented languages. Unlike most languages, Fortran is designed to generate executable codes that are highly optimized and run extremely fast. Fortran has been widely used by scientists and engineers for many years, and algorithms and code already exist for many problems. Fortran 77 remains a subset of Fortran 90/95 and code written in Fortran 77 can be used along with the Fortran 90/95 code.

With the increase in size and complexity, modern computing is moving toward the use of parallel computers. However most of the procedural programming languages use a linear memory model with the exception of Fortran 90/95. A linear memory model assumes that consecutive elements of an array are consecutive in memory.

This is a reasonable assumption for traditional computers but completely incorrect when it comes to parallel computing. Fortran 90 has addressed this problem, and has provided standardized language support for parallel computing. This includes array syntax and many intrinsic functions for doing array operations[8].

2.1 New Fortran 90 Features

The two-electron integral package uses a number of new features available in Fortran 90. Some of these features are described below.

2.1.1 Free Source Form

One important feature in Fortran 90 is the free source form, which makes it possible to have:

- names as long as 31 characters
- lines up to 132 characters in length
- semi-colon as the statement separator for multiple statements per line
- option to include source text from files.

2.1.2 *"if-then-else if-end if"* Construct

The *"if-then-else if-end if"* construct makes the code more readable.

```
if (logical-expression) then
  statement-1
else if (logical-expression) then
  statement-2
end if
```

In order to enhance the readability, commands like Commands like *"continue"* and *"goto"* are used only rarely.

2.1.3 *"select case"* Statement

Another selective execution statement Fortran 90 has is the *"select case"* statement.

```
selectcase(I)
  case(1)
    statement-1
  case(2)
    statement-2
     :
  case default
    statement-default
end select
```

The *"select case"* expression is evaluated, and the resulting value is the case index. The case index is compared to the case selector of each case statement. If a match occurs, the statement block associated with that case statement is executed. If no match occurs no statement block is executed. After the execution of the construct is complete or no match occurred the control is transferred to the statement after the *"end select"* statement.

2.1.4 *"do"* Construct

The *"do"* construct specifies the repeated execution of a statement block.

```
outer: do i=1,n
inner:  do j=1,m
          statement block
          if(condition) cycle
          if(condition) exit outer
        end if
      end do inner
```

```
        end do  outer
```

The iteration count of a loop can be determined at the beginning of execution of the *"do"* construct, unless it is indefinite. You can curtail a specific iteration with the *"cycle"* statement, and the *"exit"* statement terminates the loop.

2.1.5 Modules

Modules are collections of data, type definitions and procedure definitions, which gives a more secure and general replacement for the common block concept. Variables, data or subprograms that are declared in Fortran 90 modules can be made available by the compiler to all subprograms which use the module using *"include"* statement. Following is an example of a module:

```
    module mod_idfclc
!*****************************************************************
!     Date last modified:                          Version 2.0  *
!     Author:                                                   *
!     Description:                                              *
!                                                               *
!*****************************************************************
!Modules:
      USE program_manager
      USE program_defaults
      USE constants
!
      implicit none
!
!Public variables
      integer, public :: LAMAX,LBMAX,LCMAX,LDMAX !SHARED
      integer, public :: MTYPE,LENTQ
      integer, public :: IGEND,JGEND,KGEND,LGEND
      double precision, public :: XA,YA,ZA,XB
      integer,public :: Irange,Jrange,Krange,Lrange

!Private variables
      integer, private :: IX,IY,IZ,JX,JY,JZ,KX,KY,KZ,LX,LY,LZ
      double precision, private :: G2DFX(13),G2DFY(13),G2DFZ(13)
!Module contains a subroutine
CONTAINS!>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
!
      subroutine I2ER_SPDF
        !code belong to the subroutine goes here
      end subroutine I2ER_SPDF
!
   end module mod_idfclc
```

Module `mod_idfclc` uses three modules: `program_manager`, `program_defaults`, and constants. Variables are declared next and module contains the subroutine `I2ER_SPDF`.

2.1.6 Dynamic Memory Allocation

Another new feature of Fortran 90 that is beneficial for scientific codes is the dynamic memory management. It is the allocation of memory during the runtime. Fortran 77 is capable of only static allocation, as a result code has to be recompiled for different problem sizes or different number of processors. The only alternative is to declare a single large array at compile time and then, at runtime, assign indices to point to different locations within the array for use by different quantities. This results in code that is very difficult to maintain.

With Fortran 90 we can declare the arrays as allocatable, and the problem size and the number of processors can be read in at runtime, with arrays allocated to the size required. Depending on the problem, arrays may be different sizes on different processors. This improves the performance through improved cache utilization as well as saving memory.

2.2 New features in Fortran 95

Another new standard of Fortran, Fortran 95 was introduced in 1996. It is a relatively small change compared to the change between Fortran 77 and Fortran 90. There are some major features, some minor corrections, and few new intrinsic functions in the new standard. Some of these are to keep Fortran in step with the work in the High Performance Fortran (HPF) area. The major features include[9]:

- FORALL statement and construct
- pure and elemental user defined subprograms
- initial association status for pointers
- implicit initialization of derived type objects.

Some of the minor features introduced in Fortran 95 are as follows:

- new intrinsic function NULL
- new intrinsic function `CPU_TIME`
- automatic deallocation of allocatable arrays

2.3 OpenMP

OpenMP is an application program interface which supports multi-platform shared memory parallel programming in C,C++, and Fortran. OpenMP is compatible with most architectures including Unix and Windows NT. Jointly developed by a group of software and hardware vendors, OpenMP has now become the standard of the shared memory parallel computing, and is recognized by most of the major computer manufacturers.

OpenMP is not a programming language. It is consist of a set of compiler directives that describe the parallelism of the source code, along with a supporting library of subroutines available to applications. These directives are instructions to a compiler supporting OpenMP. They are ignored as comments by compilers that do not support OpenMP.

2.3.1 Shared Memory Architecture

OpenMP has been designed primarily for shared memory multiprocessors. Figure 2-1 shows the architecture of shared memory computers. The most important characteristic of shared memory computers is the ability of all processors to access memory directly through a logically directed connection. Distributed shared memory(DSM) computers belong to the same family.

The alternative to the shared memory configuration is distributed memory, as shown in Figure 2-2. In distributed memory, each processor in the system is only capable of directly addressing memory physically associated with it. In order to access information in memory connected to other processors, the user must explicitly pass messages through some network connecting the processors. Usually distributed memory systems are programmed using message passing libraries such as Message Passing Interface(MPI). Distributed shared memory systems can be programmed using OpenMP and message passing interface together for efficiency. Most larger systems are distributed memory computers because there is a practical limit to the number of processors that can be supported in traditional shared memory architecture. A comparison of two programming models are given in Table 2.1.

2.3.2 Why use OpenMP

Recently there has been an increase of shared memory parallel systems since they are affordable and contain increasing number of processors. Meanwhile, most of the programming models available are designed mainly for distributed memory systems. Therefore, OpenMP was created as a standard and portable application programming interface for writing shared memory parallel programs. OpenMP is a parallel programming model for shared memory and distributed shared memory multiprocessors. There are other implementation models that could be used instead of OpenMP, including Pthreads and MPI.

MPI is portable, widely used, and it is the accepted standard for message passing programs, but in a shared memory machine, message passing is not required to share data. Message passing is considered a difficult way to program because it requires a great deal of time. In addition to that the program data structure has to be explicitly partitioned, as well as the entire application must be parallelized.

Pthreads is an accepted shared memory model but there is little Fortran support for Pthreads. Even for C and C++ based applications, it is considered low level and awkward. It is also more suitable for task parallelism rather than data parallelism.

OpenMP became the industry standard recently, and it is a step in a long history of shared memory programming models. Most of the shared memory hardware vendors support some subset of the OpenMP functionality, but complete application portability had not been achieved yet. OpenMP uses compiler directives to support parallelism through calls to runtime library routines. These directives can be embedded within a program written in Fortran, C or C++.

One big advantage of OpenMP is that it can be used in writing new parallel code as well as in parallelizing existing code. In addition, it is easier to write portable

Table 2.1: Comparing shared memory and distributed memory models

Feature	Shared memory	Distributed Memory
Ability to Parallelize small parts of an application	Relatively easy to do. Reward versus effort varies widely	Relatively difficult. overheads are high
Complexity of code	Simple algorithms are easy to implement	Significant additional overhead, even for simple algorithms
Amount of additional code required	Small increase of code 2-25 percent	Significant amount of message handling code
Readability	easy to read	difficult to read

code using directives, because they are automatically ignored by compilers that do not support OpenMP. All of the functionality of OpenMP can not be achieved by only using directives. In addition to directives, OpenMP also includes a small set of runtime library routines and environment variables. OpenMP is independent of the operating system or the machine, and compilers exist for almost all the versions of UNIX as well as Windows NT[10].

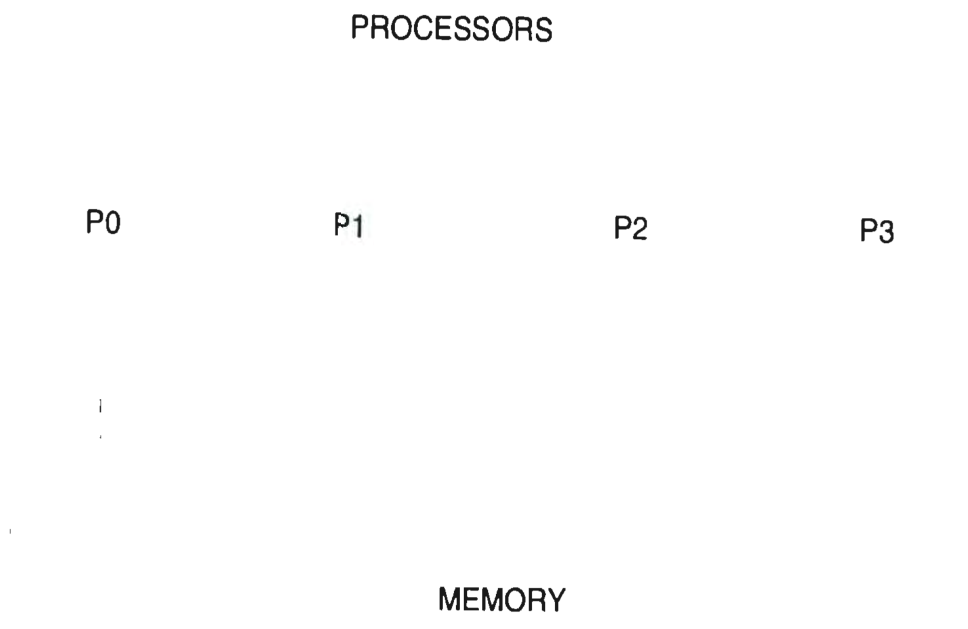

Figure 2-1: A canonical shared memory architecture, where P0-P3 are processors.

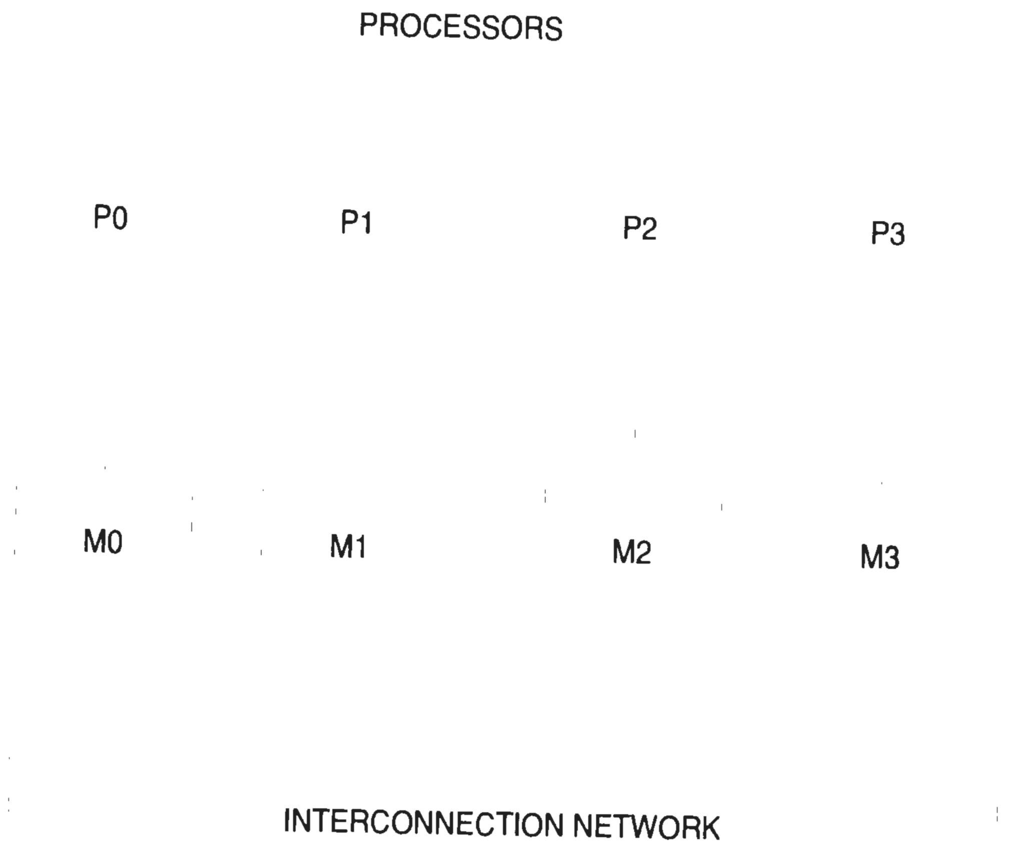

Figure 2-2: A canonical message passing architecture, where P0-P3 are processors, and M0-M3 are memory associated with processors.

Chapter 3

Computation of Two-Electron Integrals

Two methods are available in MUNgauss to evaluate two-electron integrals. The Rys polynomial method eq.(1.82) is used to compute two-electron integrals between s-type, p-type, d-type and f-type shells. The current implementation of the Gamma function method eq.(1.74) is limited to computation of two-electron integrals involving s-type and p-type shells. Subroutine `I2ECLC` by default calls both subroutine `IDFCLC` and subroutine `ISPCLC`, but the user has the option of choosing one of them. Subroutine `IDFCLC` is called first to compute the integrals with d-type and f-type shells, and subroutine `ISPCLC` is called to compute the integrals between s-type and p-type shells as given below.

```
if(LI2EDF)then
  call IDFCLC
  if(MUN_PRTLEV.GT.0)call PRT_I2E_details ('IDFCLC', IDFCNT)
end if ! LI2EDF
if(LI2ESP)then
  call ISPCLC
  if(MUN_PRTLEV.GT.0)call PRT_I2E_details ('ISPCLC', ISPCNT)
end if ! LI2ESP
```

3.1 Implementation of the Rys polynomial Method

3.1.1 Module mod_idfclc

Module `mod_idfclc` contains all the variables as well as all the subroutines used in the Rys polynomial part of the code. The variables accessible inside and outside the module are declared as public, while the variables accessible only inside the module are declared as private. The variables initialized inside the parallel part of the code are declared as thread private with the *"threadprivate"* directive. The *"threadprivate"* directive is used to identify a list of variables as being private to each thread, and a private copy of that list of variables is created for each thread. Therefore, every reference to a thread private variable within the parallel section of the code refers to a variable instance of the private copy of the executing thread. Threads cannot refer to thread private variables belonging to another thread. Other than the declarations, module `mod_idfclc` contains general subroutines used in two-electron integral computation, specialized subroutines for two-electron integral computation, subroutines used in sorting integrals and storing, as well as all the subroutines called inside integral computation subroutines.

3.1.2 Subroutine IDFCLC

The subroutine `IDFCLC` loops over the shell types (s,p,d,f), identifying the shell quadruplets (the type of integral), and calls the appropriate specialized subroutine. The generalized subroutine `I2ER_SPDF` is called as the default case. For example, a minimal basis set of each carbon atom in molecule C_{60} consists of 1s, 2s, and 2p shell types, therefore carbon has three shell types. Altogether, molecule C_{60} has a total of $60 \times 3 = 180$ shells which are described as atom shells.

The identification of shell quadruplets before the atom shell loop reduces checking, and improves the efficiency of the code. The function which determines the type of the integrals, DEFCASE, is also included in subroutine IDFCLC using the Fortran command *"contains"*. Part of function IDFCLC is illustrated below.

```
Loop over Ishell
 Loop over Jshell
  Loop over Kshell
   Loop over Lshell
   !allocate arrays to store the integrals
   !determine the ICASE(identification of shell quadruplets)
   !using DEFCASE which returns ICASE
   select case (ICASE)
    case (1)
    call I2ER_SSSS
    case (2)
    call I2ER_SSSP
    case (3)
    call I2ER_SSPP
    case (4)
    call I2ER_SPPP
    case (5)
    call I2ER_PPPP
    case (6)
    call I2ER_SSSD
    case (10)
    call I2ER_SSDD
    case (13)
    call I2ER_SDDD
    case (15)
    call I2ER_DDDD
    case (7,8,9,11,12,14,16:)
```

```
        call I2ER_SPDF
       end select
      ! deallocate the arrays
      end do ! Lshell
     end do ! Kshell
    end do ! Jshell
   end do ! Ishell
```

3.1.3 Subroutine I2ER_SPDF

Module `mod_idfclc` contains a generalized subroutine called `I2ER_SPDF` which computes integrals belonging to all the shell quadruplet types using the Rys polynomial method.

```
subroutine I2ER_SPDF
! Modules used in subroutine
  implicit none
! Variable declarations
! Loop over the atomic shells

  do Iatmshl=IFRST,ILAST
   do Jatmshl=JFRST,JLAST
    do Katmshl=KFRST,KLAST
     do Latmshl=LFRST,LLAST

! Determine the coincidences (Mtype)

      call DEF_shells (Jatmshl, Katmshl, Latmshl, Jshell, Kshell, Lshell)
      if((ABEXP+CDEXP).le.I2E_expcut)then
       Int_pointer=>IJKLs
```

```
          call I2ER_GSPDF
         end if
         if(LTWOINT)then
          call DEF_shells (Latmshl, Jatmshl, Katmshl, Lshell, Jshell, Kshell)
          if((ABEXP+CDEXP).le.I2E_expcut)then
           Int_pointer=>ILJKs
           call I2ER_GSPDF
          end if
         end if
         if(Mtype.eq.1)then
          call DEF_shells (Katmshl, Jatmshl, Latmshl, Kshell, Jshell, Lshell)
          if((ABEXP+CDEXP).le.I2E_expcut)then
           Int_pointer=>IKJLs
           call I2ER_GSPDF
          end if
         end if
        end do
       end do
      end do
     end do
```

First, subroutine I2ER_SPDF loops over atom shells belonging to specific atoms, and determines all the shell information. Subroutine DEF_shells takes the four atom shells as parameters in the order of the integral, and initializes the variables accordingly. Subroutine I2ER_GSPDF computes all the integrals for the given atom shell quadruplet.

For a given set of $\{ijkl\}$ atom shell indices, there are three unique integrals. They are: (ij|kl), (il|jk), and (ik|jl). In the sequential version, these three blocks of integrals are computed by calling the subroutine DEF_shells and the subroutine I2ER_GSPDF three times sequentially.

Determination of coincidences (Mtype) in the shell quadruplet helps to reduce the number of times the I2ER_GSPDF is called. If there are coincidences in the shell quadruplet, symmetry can be used and all the integrals can be computed by calling GSPDF once or twice. Part of the subroutine I2ER_SPDF is given below,

3.1.4 Subroutine I2ER_GSPDF

Subroutine I2ER_GSPDF computes all the two-electron integrals for a given atom shell quadruplet.

```
  subroutine I2ER_GSPDF
 !Modules:
 implicit none
 !Local variables and data goes here
 !Initialization of arrays
 ! Loop over the Gaussian expansions

 do Igauss=IGBGN,IGEND
  do Jgauss=JGBGN,JGEND
   do Kgauss=KGBGN,KGEND
    do Lgauss=LGBGN,LGEND
     !Obtain the gaussian exponents
     !Loop over roots of the Rys Polynomial
      do IZERO=1,NZERO
       !loop over atomic orbitals.
      do IAO=ISTART,IEND
       do JAO=JSTART,JENDM
        do KAO=KSTART,KENDM
         do LAO=LSTART,LENDM
         !Two-electron integral over primitives are computed using eq.(1.82)
         end do ! LAO
```

```
          end do ! KAO
         end do ! JAO
        end do ! IAO
       end do ! IZERO

      do IAO=1,Irange
       do JAO=1,JENDM
        do KAO=1,KENDM
         do LAO=1,LENDM
         ! Apply contraction coefficients.
        end do ! LAO
       end do ! KAO
      end do ! JAO
     end do ! IAO
!
     end do Dloop ! Lgauss
    end do Cloop ! Kgauss
   end do Bloop ! Jgauss
  end do Aloop ! Igauss

! End of loop over gaussians!
! End of routine I2ER_SPDF
  return
  end subroutine I2ER_GSPDF
  %
```

First, subroutine `I2ER_GSPDF` loops over the Gaussian expansions, followed by looping over the roots of the Rys polynomial and computing the components $I_x^{(R)}(t_i)$, $I_y^{(R)}(t_i)$, and $I_z^{(R)}(t_i)$ (see eq.(1.82)). In the loop over the AOs (for example, p_x, p_y and p_z are the AOs belonging to the p-type atom shell), the two-electron integrals are computed using eq.(1.82). Finally, contraction coefficients are applied and integrals are stored in arrays IJKLs, ILJKs, IKJLs. The three blocks of integrals IJKLs, ILJKs, and IKJLs resulting from three calls to subroutine I2ER_GSPDF are as given in Table 3.1 for the case of (ss|pp).

Table 3.1: Three blocks of integrals in the case of $(ss \mid pp)$ for $I \neq J \neq K \neq L$.

IJKL	ILJK	IKJL
SSP_XP_X	SP_XSP_X	SP_XSP_X
SSP_XP_Y	SP_XSP_Y	SP_XSP_Y
SSP_XP_Z	SP_XSP_Z	SP_XSP_Z
SSP_YP_X	SP_YSP_X	SP_YSP_X
SSP_YP_Y	SP_YSP_Y	SP_YSP_Y
SSP_YP_Z	SP_YSP_Z	SP_YSP_Z
SSP_ZP_X	SP_ZSP_X	SP_ZSP_X
SSP_ZP_Y	SP_ZSP_Y	SP_ZSP_Y
SSP_ZP_Z	SP_ZSP_Z	SP_ZSP_Z

3.1.5 Specialized Subroutines

Other than I2ER_SPDF, there are currently nine specialized subroutines used in MUNgauss to compute two-electron integrals. Two-electron integrals between the s-type shells (ss|ss) to d-type shells (dd|dd) are done using specialized subroutines. For example, subroutine I2ER_SSSS contains the atom shell loop similar to the I2ER_SPDF, and instead of calling the generalized subroutine I2ER_GSPDF it calls the corresponding specialized subroutine I2ER_GSSSS.

Use of specialized subroutines for the nine simplest types of integrals increases the efficiency of the code. It reduces the amount of memory used, because the general case allocates fixed size arrays that do not change with the problem size. The specialized case allocates arrays specific to the problem size. The number of subroutine calls inside the specialized functions are less, compared to the general case, because most of the code is written inline. The specialized subroutine for I2ER_GSSSS does not loop over the Rys polynomial roots, because the number of roots is one. Similarly, it does not loop over the AOs, because all four orbitals are s-type.

```
   subroutine I2ER_GSSSS
!  !Modules:
      implicit none
!  !Local Scalars
!  !Commence loops over Gaussian expansion.
    do Igauss=IGBGN,IGEND
     do Jgauss=JGBGN,JGEND
      do Kgauss=KGBGN,KGEND
       do Lgauss=LGBGN,LGEND

        Computes the integral using eq.(1.82)
        apply contraction coefficients.

      end do Dloop ! Lgauss
     end do Cloop ! Kgauss
    end do Bloop ! Jgauss
   end do Aloop ! Igauss
! End of loop over gaussians
  return
  end subroutine I2ER_GSSSS
```

Tables 3.2-3.4 show the improvement achieved by the addition of specialized subroutines for STO-3G, 6-31G(d), and 6-311G(d,p) basis sets respectively. Specialized subroutines have been added individually to the code to observe the improvement. Percentage of improvements were computed relative to the general case subroutine I2ER_GSPDF.

For the STO-3G basis set (Table 3.2), test cases HF and H_2O do not show any improvement with the addition of specialized subroutines. Sensitivity of the timing is not enough to show differences in small test cases. The other test cases show about 60% improvement with the addition of SSSS-PPPP subroutines, but show very little improvement with the addition of SSSD-DDDD subroutines because there are no d type functions involved in most of the test cases. For the $6-31G(d)$ basis set (Table 3.3), smaller test cases showed about 15% improvement with the addition of SSSS-DDDD subroutines, which is reduced to about 10% for the bigger molecules. The improvement achieved with the addition of SSSS-PPPP subroutines is far greater than the improvement achieved with the addition of SSSD-DDDD subroutines. For the $6-311G(d,p)$ basis set (Table 3.4), an improvement of about 40% is observed for all the test cases. In the case of larger basis sets with d functions, subroutine `I2ER_GSPDF` is used often, therefore the improvement achieved by the use of specialized subroutines is reduced.

3.1.6 Parallel Computation of Integrals

Coincidences occur in shell quadruplets only about 20% of the time. Therefore, all three integral blocks `(ij|kl)`, `(il|jk)` and `(ik|jl)` have to be calculated for most of the shell quadruplets. If the three blocks can be done in parallel using three processors, computation time can be improved. This can be done using the OpenMP directive *"sections"*. The *"sections"* directive is useful where three tasks do not depend on each others results. With directive *"sections"*, it is possible to perform an entire section of code in parallel, assigning each task to a different thread. The parallel section starts with the directive *"parallel"* and ends with *"end parallel"*. The code for the entire sequence of tasks, or sections, begins with a *"sections"* directive and ends with a *"end sections"* directive. The part of the code executed on each processor has to be separated by a *"section"* directive.

By default al the variables are declared shared except the variables declared thread private in the respective modules. Therefore every variable that is initialized inside the parallel section has to be declared thread private. The *"barrier"* directive of the end of the parallel section is used to synchronize the process. If there are no coincidences in the shell quadruplet, an equal amount of work is done on all three processors. The parallel implementation of I2ER_SPDF is as follows.

```
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP SECTIONS
!$OMP SECTION
   call DEF_shells (Jatmshl, Katmshl, Latmshl, Jshell, Kshell, Lshell)
   if((ABEXP+CDEXP).le.I2E_expcut)then
     Int_pointer=>IJKLs
     call I2ER_GSPDF
   end if
!$OMP SECTION
   if(LTWOINT)then
    call DEF_shells (Latmshl, Jatmshl, Katmshl, Lshell, Jshell, Kshell)
    if((ABEXP+CDEXP).le.I2E_expcut)then
     Int_pointer=>ILJKs
     call I2ER_GSPDF
    end if
   end if
!$OMP SECTION
   if(Mtype.eq.1)then
    call DEF_shells (Katmshl, Jatmshl, Latmshl, Kshell, Jshell, Lshell)
    if((ABEXP+CDEXP).le.I2E_expcut)then
     Int_pointer=>IKJLs
     call I2ER_GSPDF
    end if
   end if
!$OMP END SECTIONS
!$OMP BARRIER
```

```
!$OMP END PARALLEL
```

Parallel and sequential versions of subroutine I2ER_SPDF were compared using a set of selected test cases on a SGI Altix (Verdandi) housed at Memorial University. The STO-3G and 6-31G(d) basis sets were used for the computation. The complete code for the parallel mod_idfclc is given in the Appendix B.

An average timing improvement of 25% is observed for molecules containing atoms from the third and fourth row of the periodic table. For the test cases containing a large number of atoms from the first and second row elements, an average improvement of 7% is observed. Every OpenMP directive involves overhead [11]. If the work performed in the parallel sections is smaller, and the number of OpenMP directives is higher, overhead can reduce the gain achieved by the parallel processing. Therefore, big test cases with large numbers of atoms belonging to first and second row elements tend to give less improvement with this type of implementation. Work performed inside the parallel sections has to be considerably large to achieve good results with parallel processing.

Table 3.2: Integral computation timing in seconds with specialized subroutines. The basis set used was STO-3G. Specialized subroutines SSSS, SSSP, SSPP, SPPP, PPPP,SSSD,SSDD,SDDD,and DDDD have been added one by one to the code. The machine used was a SGI Altix (Verdandi).

Molecule	SPDF	SSSS	SSSP	SSPP	SPPP	PPPP	SSSD	SSDD	SDDD	DDDD	Improvement
HF	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	00%
H_2O	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	00%
CCl_4	3.67	3.56	3.55	2.32	1.74	1.57	1.57	1.57	1.57	1.57	57%
C_2H_6	0.33	0.26	0.14	0.13	0.13	0.13	0.13	0.13	0.13	0.13	60%
C_3H_8	1.21	1.01	0.98	0.52	0.48	0.47	0.47	0.47	0.47	0.47	61%
C_5H_{12}	6.38	5.24	2.69	2.46	2.42	2.43	2.43	2.43	2.42	2.42	62%
C_6H_{14}	11.42	9.46	9.45	4.79	4.35	4.28	4.28	4.28	4.28	4.28	62%
$1G\_pep$	2.35	2.10	2.11	1.13	0.96	0.93	0.93	0.93	0.93	0.93	60%
$2G\_pep$	16.52	14.91	14.91	7.83	6.50	6.23	6.22	6.23	6.23	6.22	62%
$3G\_pep$	47.32	42.79	42.77	22.25	18.10	17.16	17.13	17.14	17.13	17.13	63%
$4G\_pep$	94.11	85.06	85.04	44.14	35.56	33.66	33.60	33.60	33.60	33.54	64%

Table 3.3: Integral computation timing in seconds with specialized subroutines. The basis set used was 6-31G(d). Specialized subroutines SSSS, SSSP, SSPP, SPPP, PPPP SSSD, SSDD, SDDD, and DDDD have been added one by one to the code. The machine used was a SGI Altix (Verdandi).

Molecule	SPDF	SSSS	SSSP	SSPP	SPPP	PPPP	SSSD	SSDD	SDDD	DDDD	Improvement
HF	0.05	0.04	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.03	40%
H_2O	0.08	0.07	0.06	0.05	0.05	0.05	0.05	0.03	0.05	0.05	37%
CCl_4	38.08	37.26	34.29	29.07	25.38	23.80	23.09	22.92	22.92	22.88	39%
C_2H_6	1.70	1.45	1.18	1.04	1.01	1.01	0.89	0.88	0.88	0.88	48%
C_3H_8	6.48	5.60	4.57	3.99	3.93	3.88	3.44	3.36	3.37	3.37	47%
C_5H_{12}	33.71	29.63	24.33	21.10	20.42	20.40	18.32	17.86	17.88	17.92	46%
C_6H_{14}	58.99	52.10	42.86	37.15	35.89	36.29	32.60	31.62	31.65	31.98	45%
$1G\_pep$	15.66	14.60	12.63	10.96	10.46	10.40	9.52	9.28	9.29	9.30	40%
$2G\_pep$	103.65	97.54	87.69	73.52	69.82	69.09	64.19	63.04	63.31	62.85	39%
$3G\_pep$	280.17	274.27	234.83	201.03	190.21	188.31	176.05	172.49	172.93	172.69	38%
$4G\_pep$	569.69	532.28	462.56	402.89	394.88	378.04	356.08	351.01	351.01	350.09	38%

Table 3.4: Integral computation timing in seconds with specialized subroutines. The basis set used was 6-311G(d,p). Specialized subroutines SSSS, SSSP, SSPP, SPPP, PPPP SSSD, SSDD, SDDD, and DDDD have been added one by one to the code. The machine used was a SGI Altix (Verdandi).

Molecule	SPDF	SSSS	SSSP	SSPP	SPPP	PPPP	SSSD	SSDD	SDDD	DDDD	Improvement
HF	0.12	0.11	0.10	0.09	0.08	0.08	0.08	0.08	0.08	0.08	33%
H_2O	0.24	0.22	0.19	0.16	0.15	0.15	0.14	0.14	0.14	0.14	41%
CCl_4	30.66	30.21	28.45	25.10	22.79	21.72	21.31	21.15	21.19	21.18	31%
C_2H_6	6.29	5.78	4.77	3.94	3.72	3.68	3.49	3.46	3.45	3.46	45%
C_3H_8	23.42	21.65	18.00	14.97	14.12	13.98	13.22	13.09	13.08	13.11	44%
C_5H_{12}	117.14	108.29	91.22	76.70	72.58	71.74	68.15	67.46	67.47	67.53	42%
C_6H_{14}	202.45	187.53	158.56	133.80	127.70	125.19	118.89	117.89	117.76	118.29	41%
$1G\_pep$	38.64	36.65	31.88	27.13	25.45	25.05	23.74	23.47	23.45	23.56	39%
$2G\_pep$	245.08	231.47	203.38	174.20	164.18	164.02	154.61	152.15	151.72	152.5	37%
$3G\_pep$	673.59	643.22	567.11	492.35	469.78	461.59	441.48	437.32	438.90	435.44	35%
$4G\_pep$	1380.73	1317.84	1175.55	1030.12	977.00	962.22	927.88	921.63	922.65	920.88	34%

Table 3.5: Average integral computation timing in seconds using sequential and parallel *I2ER SPDF* with STO-3G basis set. The machine used was a Altix (Verdandi).

Molecule	sequential	Parallel	Improvement
HF	0.01	0.01	00%
SnH_4	1.83	1.16	36%
Sn_2H_6	14.50	9.53	34%
Sn_4H_{10}	103.42	74.60	27%
CCl_4	3.68	2.91	21%
Ge_3H_8	16.69	12.90	23%
Ge_4H_{10}	39.08	30.04	23%
$1G\_pep$	2.35	1.93	18%
$2G\_pep$	17.52	13.5	23%
$3G\_pep$	49.31	39.31	20%
$4G\_pep$	97.1	73.6	24%

Table 3.6: Average integral computation timing in seconds using sequential and parallel *I2ER SPDF* with 6-31G(d) basis set. The machine used was a SGI Altix (Verdandi).

Molecule	sequential	Parallel	Improvement
HF	0.05	0.05	00%
SnH_4	3.57	2.43	32%
Sn_2H_6	29.15	19.06	35%
CCl_4	38.13	26.22	31%
Ge_3H_8	44.22	30.13	32%
Ge_4H_{10}	116.39	79.19	32%
$1G\_pep$	16.67	12.34	26%
$2G\_pep$	103.72	90.12	13%
$3G\_pep$	283.42	265.49	6.3%
$4G\_pep$	567.10	542.18	4.3%

Chapter 4

Conclusion

In ab initio SCF computations, two-electron integral computation is one of the major time consuming steps. Improving the two-electron integral part of the code is very important for the improvement of SCF computation timing. This research focuses on the improvement of the MUNgauss code in timing, organization, and readability.

The Organization and readability of two-electron integral code in MUNgauss has been improved using new Fortran 90/95 features. Introduction of specialized subroutines reduces the memory usage, and improves the computation timing. Specialized subroutines do not follow all the steps of the generalized subroutine, there by saving time when computing less complicated types of integrals. Average improvement of the timing with the introduction of nine subroutines to compute ssss to dddd type integrals is about 10%. With the increasing complexity of the specialized subroutines, improvement gained with the addition of specialized subroutine is reduced.

Three unique blocks of integrals, IJKL, IKJL, and ILJK, for a given set of indices were computed in parallel, using OpenMP. About 25% average improvement in SCF timing achieved for test cases containing atoms from the third and fourth row of the periodic table, while little improvement is observed with large test cases containing more atoms from the first and second row of the periodic table. If the workload performed in the parallel sections is small, the overhead of the parallel directives can exceed the gain achieved by parallel processing. Large overhead compared to the work done on the processors explains the smaller improvement for test cases with comparatively more first and second row elements.

As future research, parallelization can be moved up to the atom shell level using OpenMP loop scheduling. This increases the amount of work done on the processors, which should improve the timing for every test case.

Appendix A

Two-Electron Integral for s Type Functions

Starting from eq.(1.72), two-electron integral equation for s type functions can be derived as follows. Given $h_1 = h_Q + Q_h$ and $h_2 = h_p + P_h$, we obtain

$$U_h^{(R)} = \int_0^\infty \int_0^\infty h_A^{k_h} h_B^{l_h} h_C^{m_h} h_D^{n_h} \exp(-\rho(h_Q - h_p - \overline{PQ}_h)^2) \times \exp(-\epsilon_1(h_p)^2) \times \exp(-\epsilon_2(h_Q)^2) dh_1 dh_2 \tag{A.1}$$

Since $\exp(-\rho\overline{PQ}_h^2)$ does not depend on h_1 and h_2, it can be taken out of the integral.

$$U_h^{(R)} = \exp(-\rho\overline{PQ}_h^2) \int_0^\infty h_A^{k_h} h_B^{l_h} \exp[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p] \int_0^\infty h_C^{m_h} h_D^{n_h} \exp[-(\epsilon_2 + \rho)h_Q^2 + 2\rho\overline{PQ}_h h_Q + 2\rho h_p h_Q] dh_1 dh_2 \tag{A.2}$$

In the case of s type functions, $k_h = l_h = m_h = n_h = 0$ where, $h = x, y$ or z. Therefore, the integral can be solved as follows,

$$U_h^{(R)}(ssss) = \exp(-\rho\overline{PQ}_h^2) \int_0^\infty \exp\left[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p + \frac{\rho^2(h_p + \overline{PQ}_h)^2}{\epsilon_2 + \rho}\right] dh_1 \int_0^\infty \exp[-(\epsilon_2 + \rho)Z^2] dZ \tag{A.3}$$

where $Z = h_Q - \rho(h_p + \overline{PQ}_h)/(\epsilon_2 + \rho)$. The integral over Z has the solution $Z = \sqrt{\pi/(\epsilon_2 + \rho)}$ which gives,

$$U_h^{(R)}(ssss) = \exp(-\rho\overline{PQ}_h^2)\sqrt{\pi/(\epsilon_2 + \rho)} \int_0^\infty \exp\Big[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p + \frac{\rho^2(h_p + \overline{PQ}_h)^2}{\epsilon_2 + \rho}\Big] dh_1 \tag{A.1}$$

The remaining part of the equation can be solved using a similar substitution as follows,

$$\begin{aligned}
&\int_0^\infty \exp\left[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p + \frac{\rho^2(h_p + \overline{PQ}_h)^2}{(\epsilon_2 + \rho)}\right] dh_1 \\
= &\int_0^\infty \exp\left[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p + \frac{\rho^2 h_p^2 + \rho^2\overline{PQ}_h^2 + 2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2 + \rho)}\right] dh_1 \\
= &\int_0^\infty \exp\left[-(\epsilon_1 + \rho)h_p^2 - 2\rho\overline{PQ}_h h_p + \frac{\rho^2 h_p^2}{(\epsilon_2 + \rho)} + \frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2 + \rho)} + \frac{2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2 + \rho)}\right] dh_1 \\
= &\int_0^\infty \exp\left[-\frac{((\epsilon_1 + \rho)(\epsilon_2 + \rho) - \rho^2)h_p^2}{\epsilon_2 + \rho} - 2\rho\overline{PQ}_h h_p + \frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2 + \rho)} + \frac{2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2 + \rho)}\right] dh_1
\end{aligned}$$

$$\begin{aligned}
= \exp\left[\frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2 + \rho)}\right] \int_0^\infty \exp\Bigg[&-\frac{((\epsilon_1 + \rho)(\epsilon_2 + \rho) - \rho^2)h_p^2}{\epsilon_2 + \rho} - 2\rho\overline{PQ}_h h_p \\
&+ \frac{2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2 + \rho)}\Bigg] dh_1
\end{aligned}$$

$$\begin{aligned}
= \exp\left[\frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2 + \rho)}\right] \int_0^\infty \exp\Bigg[&-\frac{((\epsilon_1 + \rho)(\epsilon_2 + \rho) - \rho^2)h_p^2}{\epsilon_2 + \rho} - \frac{2\rho\overline{PQ}_h h_p(\epsilon_2 + \rho)}{(\epsilon_2 + \rho)} \\
&+ \frac{2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2 + \rho)}\Bigg] dh_1
\end{aligned}$$

$$= \exp\left[\frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2+\rho)}\right]\int_0^\infty \exp\left[-\frac{((\epsilon_1+\rho)(\epsilon_2+\rho)-\rho^2)h_p^2}{\epsilon_2+\rho} - \frac{2\rho^2 h_p\overline{PQ}_h + 2\rho h_p\overline{PQ}_h\epsilon_2}{(\epsilon_2+\rho)} + \frac{2\rho^2 h_p\overline{PQ}_h}{(\epsilon_2+\rho)}\right] dh_1$$

$$= \exp\left[\frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2+\rho)}\right]\int_0^\infty \exp\left[-\frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)h_p^2}{\epsilon_2+\rho} - \frac{2\rho h_p\overline{PQ}_h\epsilon_2}{(\epsilon_2+\rho)}\right] dh_1 \quad (A.5)$$

Eq.(A.5) is simplified by introducing M, where M is defined as:

$$M = h_p - \frac{\rho\overline{PQ}_h\epsilon_2}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)}$$

$$\Rightarrow M^2 = h_p^2 + \frac{\rho^2\overline{PQ}_h^2\epsilon_2^2}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)^2} - \frac{2\rho\overline{PQ}_h\epsilon_2 h_p}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)}$$

$$\Rightarrow \frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)}{\epsilon_2+\rho}M^2 = \frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho}{\epsilon_2+\rho}h_p^2 + \frac{\rho^2\overline{PQ}_p^2\epsilon_2^2}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)(\epsilon_2+\rho)} - \frac{2\rho\overline{PQ}_h\epsilon_2 h_p}{(\epsilon_2+\rho)}$$

By introducing the above term into eq.(A.5), the following expression is be obtained:

$$U_h^{(R)}(ssss) = \exp\left[\frac{\rho^2\overline{PQ}_h^2}{(\epsilon_2+\rho)}\right]\exp\left[\frac{\rho^2\overline{PQ}_h^2\epsilon_2^2}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)(\epsilon_2+\rho)}\right]\int_0^\infty \exp\left[-\frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)Z^2}{(\epsilon_2+\rho)}\right] dZ$$

$$U_h^{(R)}(ssss) = \exp\left[\frac{\rho^2\overline{PQ}_h^2}{\epsilon_2+\rho} + \frac{\rho^2\overline{PQ}_h^2\epsilon_2^2}{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)(\epsilon_2+\rho)}\right]\int_0^\infty \exp\left[-\frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)Z^2}{(\epsilon_2+\rho)}\right] dZ$$

$$U_h^{(R)}(ssss) = \exp\left[\frac{\rho^2\overline{PQ}_h^2(\epsilon_1+\epsilon_2)}{\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho}\right]\int_0^\infty \exp\left[-\frac{(\epsilon_1\epsilon_2+(\epsilon_1+\epsilon_2)\rho)Z^2}{(\epsilon_2+\rho)}\right] dZ \quad (A.6)$$

By integrating the above expression, the following is obtained.

$$U_h^{(R)}(ssss) = \frac{\pi}{[\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)]^{\frac{1}{2}}} \exp\left(-\frac{\epsilon_1\epsilon_2\overline{PQ}_h^2\rho}{\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)\rho}\right) \tag{A.7}$$

By replacing the expression for U_h into the eq.(1.70), the following form of the two-electron integral $(ss|ss)$ can be obtained:

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{[\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)]^{\frac{3}{2}}} \int_0^\infty \rho^{-1/2} \exp\left(-\frac{\epsilon_1\epsilon_2(\overline{PQ}_X^2 + \overline{PQ}_Y^2 + \overline{PQ}_Z^2)\rho}{\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)\rho}\right) d\rho \tag{A.8}$$

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{[\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)]^{\frac{3}{2}}} \int_0^\infty \rho^{-1/2} \exp\left(-\frac{\epsilon_1\epsilon_2(\overline{PQ}^2)\rho}{\epsilon_1\epsilon_2 + (\epsilon_1 + \epsilon_2)\rho}\right) d\rho \tag{A.9}$$

In the above equation $1 + [\rho(\epsilon_1 + \epsilon_2)]/\epsilon_1\epsilon_2$ term is replaced by $(1 - t^2)^{-1}$. Therefore the upper limit of integration changes from infinity to 1, and lower limit does not change.

$$\frac{1}{(1-t^2)} = 1 + \frac{\rho(\epsilon_1 + \epsilon_2)}{\epsilon_1\epsilon_2} \quad \text{and} \quad \frac{\epsilon_1\epsilon_2}{1-t^2} = \epsilon_1\epsilon_2 + \rho(\epsilon_1 + \epsilon_2)$$

$$\frac{\epsilon_1\epsilon_2 t^2}{(\epsilon_1 + \epsilon_2)(1-t^2)} = \rho \quad \text{and} \quad \frac{2\epsilon_1\epsilon_2 t(1-t^2)^{-2}}{\epsilon_1 + \epsilon_2} dt = d\rho$$

Using these equations in eq.(A.9) gives,

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{\left[\frac{\epsilon_1\epsilon_2}{1-t^2}\right]^{\frac{3}{2}}} \int_0^1 \left[\frac{\epsilon_1\epsilon_2 t^2}{(\epsilon_1 + \epsilon_2)(1-t^2)}\right]^{-1/2}$$

$$\exp\left(-\frac{\epsilon_1\epsilon_2\overline{PQ}_h^2 \frac{\epsilon_1\epsilon_2 t^2}{(\epsilon_1+\epsilon_2)(1-t^2)}}{\frac{\epsilon_1\epsilon_2}{1-t^2}}\right) \frac{2\epsilon_1\epsilon_2 t(1-t^2)^{-2}}{\epsilon_1 + \epsilon_2} dt \tag{A.10}$$

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{\epsilon_1\epsilon_2(\epsilon_1 + \epsilon_2)^{\frac{1}{2}}} \int_0^1 \exp\left(-\frac{\epsilon_1\epsilon_2\overline{PQ}^2 t^2}{\epsilon_1 + \epsilon_2}\right) dt \tag{A.11}$$

The final form of the simplest two electron integral $(ss|ss)$ is:

$$(ss|ss) = \frac{K_P K_Q \pi^{\frac{5}{2}}}{\epsilon_1\epsilon_2(\epsilon_1 + \epsilon_2)^{\frac{1}{2}}} F_0\left(\frac{\epsilon_1\epsilon_2\overline{PQ}^2 t^2}{\epsilon_1 + \epsilon_2}\right) \tag{A.12}$$

Appendix B

Module *mod_idfclc*

Code for the parallel implementation of `mod_idfclc` is given here.

```
      MODULE mod_idfclc
!*********************************************************************
!     Date last modified: July 2006                        Version 2.0  *
!                                                                     *
!     Description: module used by the DF two-electron integral          *
!     parallel implementation                                         *
!*********************************************************************
!Modules:
      USE program_manager
      USE program_defaults
      USE constants
      USE type_molecule
      USE type_basis_set
      USE i2e_module

      implicit none
```

```
!Global variables from module_df_integrals
!Variables that gets initialized in separate threads
!are declared as thread private.
      integer:: LAMAX
      integer,private ::LBMAX,LCMAX,LDMAX !SHARED
!$OMP THREADPRIVATE(LBMAX,LCMAX,LDMAX)
      integer:: MTYPE,ICASE !Iand ICASE do not belong
      integer :: LENTQ
      integer:: IGEND
      integer,private::JGEND,KGEND,LGEND
!$OMP THREADPRIVATE(JGEND,KGEND,LGEND)
      integer:: IGBGN
      integer,private::JGBGN,KGBGN,LGBGN
!$OMP THREADPRIVATE(JGBGN,KGBGN,LGBGN)
      integer,private:: CCAsave,,CCBsave,CCCsave,CCDsave
!$OMP THREADPRIVATE(CCAsave,CCBsave,CCCsave,CCDsave)
      integer,private:: AOI,AOJ,AOK,AOL !Used in IDFCLC1 and 2
      logical:: LTWOINT
      logical:: Lsort
      double precision,dimension(:),allocatable :: IJKLS
      double precision,dimension(:),allocatable :: IKJLS
      double precision,dimension(:),allocatable :: ILJKS
      double precision,dimension(:),allocatable :: TQ

!public variables used in I2E_SPDF,def and specialized subroutines
      double precision:: XA,YA,ZA
      double precision,private::XB,YB,ZB,XC,YC,ZC,XD,YD,ZD !shared
!$OMP THREADPRIVATE(XB,YB,ZB,XC,YC,ZC,XD,YD,ZD)
      integer:: Irange
      integer,private::Jrange,Krange,Lrange !shared
!$OMP THREADPRIVATE(Jrange,Krange,Lrange)
      integer:: IATOM
      integer,private::JATOM,KATOM,LATOM
!$OMP THREADPRIVATE(JATOM,KATOM,LATOM)
      integer:: Iatmshl,Jatmshl,Katmshl,Latmshl
      double precision:: PIconst !Global
      double precision,private :: EXPARG
!$OMP THREADPRIVATE(EXPARG)
      double precision,private:: ABEXP,CDEXP
!$OMP THREADPRIVATE(ABEXP,CDEXP)
!public variables used in def and specialized subroutine
      integer,private:: LABMAX,LCDMAX,LPQMAX
!$OMP THREADPRIVATE(LABMAX,LCDMAX,LPQMAX)
      integer,private:: NZERO
!$OMP THREADPRIVATE(NZERO)
      integer:: IEND
      integer,private::KEND,LEND,JEND
!$OMP THREADPRIVATE(JEND,KEND,LEND)
      integer:: ISTART
      integer,private ::JSTART,KSTART,LSTART
!$OMP THREADPRIVATE(JSTART,KSTART,LSTART)
      logical,private:: LRABCD,LRAB,LRCD
!$OMP THREADPRIVATE(LRABCD,LRAB,LRCD)
      logical,private:: IIKK,IATMSHL_EQ_KATMSHL,JATMSHL_EQ_LATMSHL
      logical,private:: IATMSHL_EQ_JATMSHL,IJIJ,KATMSHL_EQ_LATMSHL
!$OMP THREADPRIVATE(IIKK,IATMSHL_EQ_KATMSHL,JATMSHL_EQ_LATMSHL)
!$OMP THREADPRIVATE(IATMSHL_EQ_JATMSHL,IJIJ,KATMSHL_EQ_LATMSHL)
      double precision,private:: RABSQ,RCDSQ
!$OMP THREADPRIVATE(RABSQ,RCDSQ)
```

```
CONTAINS!>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

      subroutine I2ER_SPDF
!***********************************************************************
!     Date last modified: September 23, 1996                Version 2.0 *
!     Author: R.A. Poirier                                              *
!     Description:                                                      *
!***********************************************************************
! Modules:

      implicit none
!
! Local Scalars
      logical Ldebug
      integer i
      call PRG_manager ('enter', 'I2ER_SPDF', 'UTILITY')
      Ldebug=Local_Debug

      include "start_atmshl_loop"

! parallel section, variables are shared by default, and
! number of threads used is declared as three.
! Only if the ICASE is greater than 2 computation is done in parallel.
! Number of threads used is three.
!$OMP PARALLEL DEFAULT(SHARED) IF(ICASE>2) NUM_THREADS(3)
!$OMP SECTIONS
!$OMP SECTION
      call DEF_shells (Jatmshl, Katmshl, Latmshl, Jshell, Kshell, Lshell)
      if((ABEXP+CDEXP).le.I2E_expcut)then
```

```
          call I2ER_GSPDF(IJKLs)
        end if
!$OMP SECTION
        if(LTWOINT)then
        call DEF_shells (Latmshl, Jatmshl, Katmshl, Lshell, Jshell, Kshell)
        if((ABEXP+CDEXP).le.I2E_expcut)then
          call I2ER_GSPDF(ILJKs)
        end if
        end if
!$OMP SECTION
        if(Mtype.eq.1)then
        call DEF_shells (Katmshl, Jatmshl, Latmshl, Kshell, Jshell, Lshell)
        if((ABEXP+CDEXP).le.I2E_expcut)then
          call I2ER_GSPDF(IKJLs)
        end if
        end if
!$OMP END SECTIONS
!$OMP END PARALLEL

        ISTART=Basis%shell(Ishell)%XSTART
        IEND=Basis%shell(Ishell)%XEND
        Irange=IEND-ISTART+1

        JSTART=Basis%shell(Jshell)%XSTART
        JEND=Basis%shell(Jshell)%XEND
        Jrange=JEND-JSTART+1

        KSTART=Basis%shell(Kshell)%XSTART
        KEND=Basis%shell(Kshell)%XEND
        Krange=KEND-KSTART+1
```

```
      LSTART=Basis%shell(Lshell)%XSTART
      LEND=Basis%shell(Lshell)%XEND
      Lrange=LEND-LSTART+1
      LENTQ=Irange*Jrange*Krange*Lrange
      if(Mtype.eq.1)then
        Lsort=.false.
      end if
      if(ICASE.eq.2)Lsort=.false.
      if(ICASE.eq.6)Lsort=.false.

       !if(Mtype.eq.3)then
        !write(6,'(4i6)')Iatmshl
        !do i= 1,lentq
        ! write(6,'(3F20.6)')IJKLs(i),ILJKs(i),IKJLs(i)
       ! end do
       !end if
      if(Lsort)then
       call  SORTALL(IJKLs,ILJKs,IKJLs,TQ,Irange,Jrange,Krange, &
       Lrange,LENTQ,MTYPE)
      end if

      call I2E_SHLN
!
! Note that IKJLS and ILJKS are switched
      end do Lloop ! Latmshl
      end do Kloop ! Katmshl
      end do Jloop ! Jatmshl
      end do Iloop ! Iatmshl
! End of loop over shells.
```

```
!

! End of routine I2ER_SPDF
      call PRG_manager ('exit', 'I2ER_SPDF', 'UTILITY')
      return
    end subroutine I2ER_SPDF

    subroutine I2ER_GSPDF(IJKLs)
!***********************************************************************
!     Date last modified: September 23, 1996               Version 2.0 *
!     Author: R.A. Poirier                                             *
!     Description:                                                     *
!***********************************************************************
! Modules:

      implicit none
! Input array

      double precision, dimension(MAXINT),INTENT(INOUT) :: IJKLS
! Local Scalars
      integer :: COR_INDEX
      logical :: Ldebug
      integer :: IV,INTC,IENDM,KENDM,LENDM,NJKL,NKL,NJK
      integer :: IX,IY,IZ,JX,JY,JZ,KX,KY,KZ,LX,LY,LZ
      integer :: IGAUSS,JGAUSS,KGAUSS,LGAUSS
      integer :: IAO,JAO,KAO,LAO
      integer :: IZERO, IERROR
      integer :: CCA,CCB,CCC,CCD
      integer :: INDIX(20),INDIY(20),INDIZ(20),INDJX(20),INDJY(20)
      integer :: INDJZ(20),INDKX(20),INDKY(20),INDKZ(20)
      INDLX(20),INDLY(20),INDLZ(20)
      double precision :: XINT(11)
      double precision :: XAP,YAP,ZAP,XBP,YBP,ZBP,XCQ,YCQ,ZCQ,XDQ,YDQ, &
                          ZDQ,PQX,PQY,PQZ,RPQSQ
      double precision :: A2DF(174),CCPX(48),CCPY(48),CCPZ(48),CCQX(48), &
                          CCQY(48),CCQZ(48)
      double precision :: G2DFX(13),G2DFY(13),G2DFZ(13),XIP(256),YIP(256),&
                          ZIP(256)
      double precision :: TP(7),WP(7)!private
      double precision :: CC1,CC2,CC3 !moved from the module_DF_int
      double precision :: AS,BS,CS,DS !moved from module_DF_int
      double precision :: RHO,TWORHO,DXYZ,ZTEMP,RHOT2,ZCONST
      double precision :: EPAB,EPABI,EPABA, EPABB,EQCD,EQCDI,EQCDC, &
                          EQCDD,EABCD,EP2I,EABCDI
      double precision, dimension(:), allocatable :: TQprim

!

      DATA INDIX/0,64,0,0,128,0,0,64,64,0,192,0,0,64,128,128,64,0,0,64/, &
           INDIY/0,0,64,0,0,128,0,64,0,64,0,192,0,128,64,0,0,64,128,64/, &
           INDIZ/0,0,0,64,0,0,128,0,64,64,0,0,192,0,0,64,128,128,64,64/, &
           INDJX/0,16,0,0,32,0,0,16,16,0,48,0,0,16,32,32,16,0,0,16/, &
           INDJY/0,0,16,0,0,32,0,16,0,16,0,48,0,32,16,0,0,16,32,16/, &
           INDJZ/0,0,0,16,0,0,32,0,16,16,0,0,48,0,0,16,32,32,16,16/, &
           INDKX/0,4,0,0,8,0,0,4,4,0,12,0,0,4,8,8,4,0,0,4/, &
           INDKY/0,0,4,0,0,8,0,4,0,4,0,12,0,8,4,0,0,4,8,4/, &
           INDKZ/0,0,0,4,0,0,8,0,4,4,0,0,12,0,0,4,8,8,4,4/, &
           INDLX/1,2,1,1,3,1,1,2,2,1,4,1,1,2,3,3,2,1,1,2/, &
           INDLY/1,1,2,1,1,3,1,2,1,2,1,4,1,3,2,1,1,2,3,2/, &
           INDLZ/1,1,1,2,1,1,3,1,2,2,1,1,4,1,1,2,3,3,2,2/, &
           XINT/1.0D0,2.0D0,3.0D0,4.0D0,5.0D0,6.0D0,7.0D0,8.0D0, &
```

```
                9.0D0,10.0D0,11.0D0/
!
! Begin:
      A2DF(1:174)=ZERO
      CCPX(1:48)=ZERO
      CCPY(1:48)=ZERO
      CCPZ(1:48)=ZERO
      CCQX(1:48)=ZERO
      CCQY(1:48)=ZERO
      CCQZ(1:48)=ZERO
      allocate(TQprim(10000),STAT=IERROR)
!
      include 'mungauss_gaussian_AB'
      include 'mungauss_gaussian_CD'

      TQprim(1:LENTQ)=ZERO

      IF(EXPARG.LE.I2E_expcut)THEN
! NOTE: Must zero CCP's if this step is skipped!!!
      if(LRAB)then
        EPABA=AS*EPABI
        EPABB=BS*EPABI
        XAP= EPABB*(XB-XA)
        YAP= EPABB*(YB-YA)
        ZAP= EPABB*(ZB-ZA)
        XBP=-EPABA*(XB-XA)
        YBP=-EPABA*(YB-YA)
        ZBP=-EPABA*(ZB-ZA)
        call GETCC_XYZ (CCPX, CCPY, CCPZ, XAP, YAP, ZAP, XBP, YBP, &
        ZBP, LAMAX, LBMAX)
        end if

        if(LRCD)then
          EQCDC=CS*EQCDI
          EQCDD=DS*EQCDI
          XCQ= EQCDD*(XD-XC)
          YCQ= EQCDD*(YD-YC)
          ZCQ= EQCDD*(ZD-ZC)
          XDQ=-EQCDC*(XD-XC)
          YDQ=-EQCDC*(YD-YC)
          ZDQ=-EQCDC*(ZD-ZC)
          call GETCC_XYZ (CCQX, CCQY, CCQZ, XCQ, YCQ, ZCQ, XDQ, YDQ, &
          ZDQ, LCMAX, LDMAX)
        end if
!
        PQX=(CS*XC+DS*XD)*EQCDI-(AS*XA+BS*XB)*EPABI
        PQY=(CS*YC+DS*YD)*EQCDI-(AS*YA+BS*YB)*EPABI
        PQZ=(CS*ZC+DS*ZD)*EQCDI-(AS*ZA+BS*ZB)*EPABI

        RPQSQ=PQX*PQX+PQY*PQY+PQZ*PQZ
!
        EABCD=EPAB*EQCD
        EABCDI=ONE/(EPAB+EQCD)
        RHO=EABCD*EABCDI

        ZTEMP=PIconst*DEXP(-EXPARG)*DSQRT(EABCDI)/EABCD

        EP2I=ONE/(EPAB+EPAB)
        call IJKLA2 (A2DF, LABMAX, LCDMAX, EQCD, EP2I)
!
```

```
      DXYZ=RHO*RPQSQ
      call RPOLX (NZERO, DXYZ, TP, WP)
      TWORHO=RHO+RHO

      do IZERO=1,NZERO
      RHOT2=TWORHO*TP(IZERO)
      ZCONST=ZTEMP*WP(IZERO)
      if(ZCONST.le.I2E_PQCUT2)cycle
      G2DFX(1)=ONE
      G2DFY(1)=ONE
      G2DFZ(1)=ZCONST
      XIP(1)=ONE
      YIP(1)=ONE
      ZIP(1)=G2DFZ(1)
!
      IF(LPQMAX.GE.2)THEN
      G2DFX(2)=RHOT2*PQX
      G2DFY(2)=RHOT2*PQY
      G2DFZ(2)=RHOT2*PQZ*G2DFZ(1)
!
      IF(LPQMAX.GE.3)THEN
        do IV=3,LPQMAX
          G2DFX(IV)=RHOT2*(PQX*G2DFX(IV-1)-XINT(IV-2)*G2DFX(IV-2))
          G2DFY(IV)=RHOT2*(PQY*G2DFY(IV-1)-XINT(IV-2)*G2DFY(IV-2))
          G2DFZ(IV)=RHOT2*(PQZ*G2DFZ(IV-1)-XINT(IV-2)*G2DFZ(IV-2))
        end do
      end if ! LPQMAX.GE.3
!
      IF(LRABCD)THEN
        call ABCD4C (A2DF, G2DFX, CCQX, CCPX, XIP, LAMAX, LBMAX, &
          LCMAX, LDMAX)
          call ABCD4C (A2DF, G2DFY, CCQY, CCPY, YIP, LAMAX, LBMAX, &
          LCMAX, LDMAX)
          call ABCD4C (A2DF, G2DFZ, CCQZ, CCPZ, ZIP, LAMAX, LBMAX, &
          LCMAX, LDMAX)
        else IF(LRCD)THEN
          call AACD3C (A2DF, G2DFX, CCQX, XIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
          call AACD3C (A2DF, G2DFY, CCQY, YIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
          call AACD3C (A2DF, G2DFZ, CCQZ, ZIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
        else IF(LRAB)THEN
          call ABCC3C (A2DF, G2DFX, CCPX, XIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
          call ABCC3C (A2DF, G2DFY, CCPY, YIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
          call ABCC3C (A2DF, G2DFZ, CCPZ, ZIP, LAMAX, LBMAX, LCMAX, &
          LDMAX)
        else
          call AABB2C (A2DF, G2DFX, XIP, LAMAX, LBMAX, LCMAX, LDMAX)
          call AABB2C (A2DF, G2DFY, YIP, LAMAX, LBMAX, LCMAX, LDMAX)
          call AABB2C (A2DF, G2DFZ, ZIP, LAMAX, LBMAX, LCMAX, LDMAX)
        end if ! LRABCD
        end if ! LPQMAX.GE.2
!
! Commence loop over atomic orbitals.
        INTC=0
        JENDM=JEND
        KENDM=KEND
```

```
 do IAO=ISTART,IEND
   IF(Iatmshl_EQ_Jatmshl)JENDM=IAO
   IF(IJIJ)KENDM=IAO
   IX=INDIX(IAO)
   IY=INDIY(IAO)
   IZ=INDIZ(IAO)
   do JAO=JSTART,JENDM
     JX=INDJX(JAO)+IX
     JY=INDJY(JAO)+IY
     JZ=INDJZ(JAO)+IZ
     do KAO=KSTART,KENDM
       LENDM=LEND
       IF(Katmshl_EQ_Latmshl)LENDM=KAO
       IF(IJIJ.AND.IAO.EQ.KAO)LENDM=JAO
       KX=INDKX(KAO)+JX
       KY=INDKY(KAO)+JY
       KZ=INDKZ(KAO)+JZ
       do LAO=LSTART,LENDM
         LX=INDLX(LAO)+KX
         LY=INDLY(LAO)+KY
         LZ=INDLZ(LAO)+KZ
         INTC=INTC+1
         TQprim(INTC)=TQprim(INTC) &
         +(XIP(LX)*YIP(LY)*ZIP(LZ))
       end do ! LAO
     end do ! KAO
  end do ! JAO
 end do ! IAO
end do ! IZERO
end if ! EXPARG.LE.I2E_expcut
```

```
! Apply contraction coefficients.
!
     NJKL = Jrange*Krange*Lrange
     NKL = Krange*Lrange
     NJK = Jrange*Krange

     INTC = 0
     COR_INDEX = 0
     JENDM = Jrange
     KENDM = Krange
     do IAO=1,Irange
       CC1=BASIS%ccbyao(CCA+IAO-1)
       IF(Iatmshl_EQ_Jatmshl)JENDM = IAO
       IF(IJIJ)KENDM = IAO
         do JAO=1,JENDM
           CC2=CC1*BASIS%ccbyao(CCB+JAO-1)
           do KAO=1,KENDM
             CC3=CC2*BASIS%ccbyao(CCC+KAO-1)
             LENDM=Lrange
             IF(Katmshl_EQ_Latmshl)LENDM=KAO
             IF(IJIJ.AND.IAO.EQ.KAO)LENDM=JAO
               do LAO=1,LENDM
                 INTC=INTC+1
                 COR_INDEX=(IAO-1)*NJKL+(JAO-1)*NKL+(KAO-1)*Lrange+LAO
                 IJKLS(COR_INDEX)=IJKLS(COR_INDEX) &
                 +TQprim(INTC)*CC3*BASIS%ccbyao(CCD+LAO-1)
               end do ! LAO
           end do ! KAO
         end do ! JAO
```

```
        end do ! IAO
  !
        CCD=CCD+Lrange
        end do Dloop ! Lgauss
        CCC=CCC+Krange
        end do Cloop ! Kgauss
        CCB=CCB+Jrange
        end do Bloop ! Jgauss
        CCA=CCA+Irange
        end do Aloop ! Igauss
        deallocate(TQprim,STAT=IERROR)
 ! End of loop over gaussians
 !
 ! End of routine I2ER_GSPDF
      return
      end subroutine I2ER_GSPDF
end module mod_idfclc
```

Bibliography

[1] Szabo, A. and Ostlund, N. S. (1996). *In Modern Quantum Chemistry.* 2nd Edition, Dover publications Inc., pg 39–41, pg 43–46, pg 111–114, pg 136–137, pg 145–146.

[2] Dudel, R., LeRoy, G., Peeters. D.and Sena, M. (1983). *In Quantum Chemistry.* John Wiley and sons, New York, pg 176–199.

[3] Boys, S.F. (1950). *Electronic wavefunctions, A General method of calculation for stationary states of any molecular system,* Proc. R. Soc. London, pg 542–554.

[4] Obara, S. and Saika, A. (1986). *Efficient recursive computation of molecular integrals over Cartesian Gaussian functions,* J.Chem. Phys. vol.84, pg 3963–3973.

[5] Rys, J. and Dupuis, M. (1976) *Numerical Integration using Rys Polynomials,* Journal of Comput.Chem.,vol 21, pg 144–165.

[6] Rys, J.,Dupuis, M. and King, H.,F. (1976) *Numerical Integration using Rys Polynomials,* Journal of Comput.Chem.,vol 21, pg 144–165.

[7] Rys, J.,Dupuis, M. and King, H.F. (1983) *Computation of Electron Repulsion Integrals using Rys Quadrature method,* Journal of comput.Chem.,vol 4,No. 2, pg 154–157.

[8] Redwine, C. (1995) *Upgrading to Fortran 90,* Springer, pg 1–100.

[9] Sleightholme, J., *Information Services and Systems.* The new features of Fortran 95, 22nd May 2006.

[10] Chandra,R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. and Menon, R.(2001) *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, pg 15–200.

[11] Bull, J.M. (1999) *Mesuring Synchronisation and Scheduling Overheads in OpenMP*, Proceedings of the First European Workshop on OpenMP, Lund, Sweden, pg 99–105.