GPUs in GAMESS: The story of libcchem

Dave Tomlinson
Iowa State University

Outline

- Introduction to GAMESS and Background of methods
- Electron Repulsion Integrals (ERI) and Hartree-Fock
- Coupled Cluster

GAMESS

- General Atomic and Molecular Electronic Structure System
- One of the most widely used electronic structure codes
- Maintained by the Gordon Group at Iowa State University
- In development for over 35 years with hundreds of developers all over the world
- Over 1 million lines of Fortran

"Advances in electronic structure theory: GAMESS a decade later" M.S. Gordon, M.W. Schmidt pp. 1167-1189, in "Theory and Applications of Computational Chemistry: the first forty years" C. E. Dykstra, G. Frenking, K. S. Kim, G. E. Scuseria (editors), Elsevier, Amsterdam, 2005.

Introduction to *ab initio* methods

- *ab initio* – from first principles
- Solving the Schrödinger equation
  - $H\Psi=E\Psi$
- Very accurate energies structures of molecular systems
- Hartree-Fock
- Coupled Cluster

Overview of selected *ab initio* methods in GAMESS

- Hartree-Fock
  - Most basic *ab initio* method
  - Formally Scales $O(N^4)$, can be optimized down to $\sim O(N^3)$ or better
  - Most computationally expensive step is electron repulsion integrals (ERI) over atomic orbitals (AOs)

$$\int\int c_m(1)\,c_n(1)\,[1/r_{12}]\,c_l(2)\,c_s(2)\,dV_1\,dV_2$$

Overview of Selected *ab initio* Methods in GAMESS (cont.)

- Coupled Cluster
  - Cluster Expansion
    - $\Psi=\Psi_0 e^T$
    - where $T=T_1+T_2+T_3+\ldots+T_N$
      - T_i=i-particle operator
    - CCSD scales $O(N^6)$; CCSDT scales $O(N^8)$, …
    - Compromise = CCSD(T): triples perturbatively $O(N^7)$
      - If the problem size is doubled, 128x more expensive

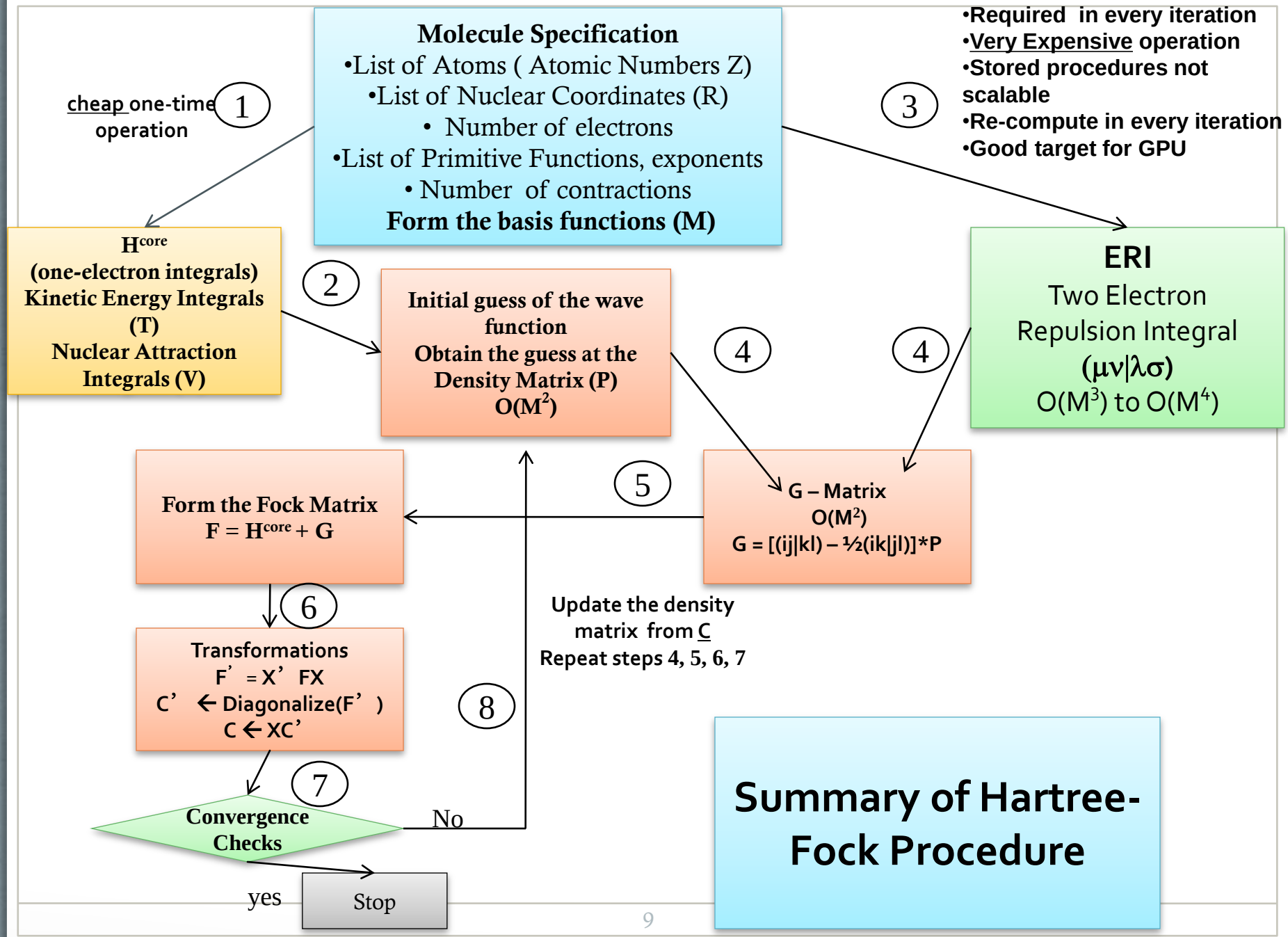
Libcchem Background

- External C++ library for performance critical code
- Originally developed to allow GAMESS to be run on GPUs
- Very Efficient CPU code as well

A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)

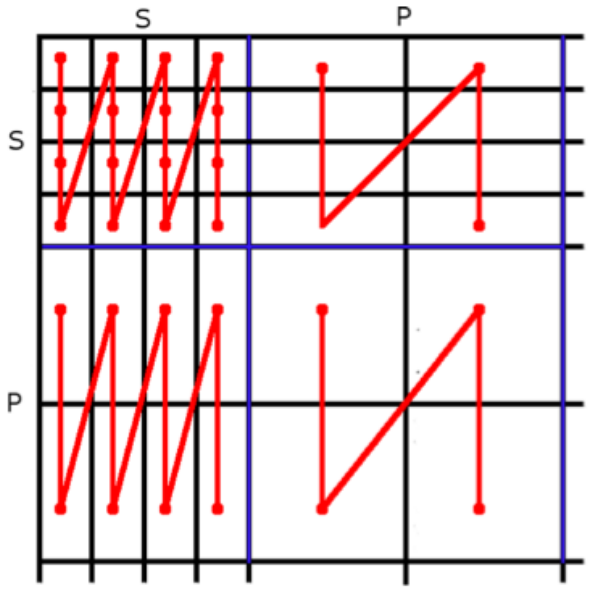
Electron Repulsion Integrals

- Major computational step in both *ab initio* and DFT methods
- Complexity is $O(M^3)$-$O(M^4)$, M = number of Gaussian basis functions
- Rys Quadrature – proposed by Dupuis, Rys, King (DRK)


Molecule Specification
•List of Atoms (Atomic Numbers Z)
•List of Nuclear Coordinates (R)
• Number of electrons
•List of Primitive Functions, exponents
• Number of contractions
Form the basis functions (M)
cheap one-time operation
1
3
•Required in every iteration
•Very Expensive operation
•Stored procedures not scalable
•Re-compute in every iteration
•Good target for GPU
H^core
(one-electron integrals)
Kinetic Energy Integrals (T)
Nuclear Attraction Integrals (V)
2
Initial guess of the wave function
Obtain the guess at the Density Matrix (P)
O(M^2)
4
4
ERI
Two Electron Repulsion Integral
(μν|λσ)
O(M^3) to O(M^4)
Form the Fock Matrix
F = H^core + G
5
G – Matrix
O(M^2)
G = [(ij|kl) – ½(ik|jl)]*P
6
Transformations
F' = X' FX
C' ← Diagonalize(F')
C ← XC'
8
Update the density matrix from C
Repeat steps 4, 5, 6, 7
7
Convergence Checks
No
yes
Stop
Summary of Hartree-Fock Procedure

Libcchem RHF

- Restricted Hartree-Fock
- S and P refer to S and P orbitals
- Basis set sorted to improve data locality



A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)

Libcchem RHF (Cont.)

- Only the needed integrals are computed for each block
  - All integrals are not computed at once
- Integrals are sorted for increased efficiency
- Can be run on GPUs
- Number of integrals ~ $\frac{n^4}{8}$
  - For 1000 basis functions, number of integrals is ~125,000,000,000

Rys Quadrature Implementation

- Two low-level implementations
  - Fully unrolled and simplified kernels for low angular momentum (L)
  - Partially unrolled for more complex integrals (higher L)
  - Make use of C++ templates & automatically generated code
- Human hands-on code small: ~ 2,000 lines of code
- Code kept small due to objects & generic templates
- GPU implementation driven by complexity of integrals
- Explicit unrolling can be controlled at different levels such as shells, roots to test for performance improvements

Integrals Conclusion

- Very easy to generate the possible ERI shell combinations using templates
  - Automatic code generation (both python & C++)
- Explicit unrolling can be controlled at different levels such as shells, roots to test for performance improvements

Input	Basis	Basis Functions	CPU only time	K80 +CPU	K80 Speedup
Ginkgo	ccd	555	844.1	155.9	5.41x

Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz

Coupled Cluster

- Highly accurate family of methods
- Most popular method is coupled-cluster with iterative singles and doubles and non-iterative triples (CCSD(T))
- Easy to use “black box” method

Coupled Cluster (cont.)

- The CC wavefunction can be written as

$$\Psi_{CC} = e^{T}\Phi_0 = \left(1 + T + \frac{T^2}{2!} + \frac{T^3}{3!} + \cdots\right)\Phi_0$$

- T is the cluster operator defined as

$$T = T_1 + T_2 + T_3 + \cdots + T_N$$

- The "CCSD" in CCSD(T) means the cluster operator is truncated after T_2 giving

$$T_{CCSD} = T_1 + T_2$$

(T) Algorithm

```
for c in V {
  for b in c {
    for a in b {

      load t(o,o,a,b)
      load t(o,o,a,c)
      load t(o,o,b,c)

      load v(o,o,o,a)
      load v(o,o,o,b)
      load v(o,o,o,c)

      load v(o,o,v,a)
      load v(o,o,v,b)
      load v(o,o,v,c)

      load v(o,v,b,c)
      load v(o,v,c,b)
      load v(o,v,a,c)
      load v(o,v,c,a)
      load v(o,v,a,b)
      load v(o,v,b,a)

      // t(i,j,e,a)*V(e,k,b,c) corresponds to
      // dgemm(t(ij,e), V(e,k)), etc
      t(i,j,k) = t(i,j,e,a)*V(e,k,b,c) - t(i,m,a,b)*V(j,k,m,c)
      t(i,k,j) = t(i,k,e,a)*V(e,j,c,b) - t(i,m,a,c)*V(k,j,m,b)
      t(k,i,j) = t(k,i,e,c)*V(e,j,a,b) - t(k,m,c,a)*V(i,j,m,b)
      t(k,j,i) = t(k,j,e,c)*V(e,i,b,a) - t(k,m,c,b)*V(j,i,m,a)
      t(j,k,i) = t(j,k,e,b)*V(e,i,a,c) - t(j,m,b,c)*V(k,i,m,a)
      t(j,i,k) = t(j,i,e,b)*V(e,k,c,a) - t(j,m,b,a)*V(i,k,m,c)
      ...
    }
  }
}
```

A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)

Single Node GPU For CC performance (minutes)

Input	$C_8H_{10}N_4O_2$/ccPVTZ	$SiH_4B_2H_6$/aug-ccPVQZ	$C_4N_3H_5$/aug-ccPVTZ
Direct	124	131	36
Direct+GPU[1]	53	65	26
CCSD	163	142	42
CCSD+GPU[1]	95	75	33
CCSD Speed-up[2]	1.4x	1.9X	1.3X

[1] GPU enabled

[2] Overall CCSD speed-up

A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)

Future Work

- Gradients
- Open Shell Methods
- New Coupled Cluster
- Further Optimizations

Thanks for listening.

Acknowledgments

- Prof. Mark Gordon
- Dr. Mike Schmidt
- Dr. Andrey Asadchev
- NVIDIA
- AFOSR-BRI

– Recall electron repulsion integrals over AOs

$$\iint c_m(1)\,c_n(1)[1/r_{12}]\,c_l(2)\,c_s(2)dV_1dV_2$$

– E(PT2) requires transformation of these ERI from AOs to molecular orbitals (MOs) ϕ_i

- Most time-consuming step in PT2
- Large number of these integrals, cannot store in memory on single CPU
- Highly coupled transformation, tough to make parallel

- Cluster expansion: Coupled cluster method

– $\Psi=\Psi_0\, e^T$: $T=T_1+T_2+T_3+\ldots+T_N$

- T_i=i-particle operator

– CCSD scales~N^6; CCSDT scales~N^8, …

– Compromise = CCSD(T): triples perturbatively ~N^7

Heterogeneous Computing

- Using multiple architectures on the same system
  - CPU with a GPU
- Faster overall computations
- Power savings

Outline

- Introduction
- Libcchem Background
- ROHF Background
- Results and Conclusions
- Future Work

Outline

- Introduction
- Libcchem Background
- ROHF Background
- Results and Conclusions
- Future Work

Outline

- Introduction
- Libcchem Background
- ROHF Background
- Results and Conclusions
- Future Work

Rys Quadrature algorithm

Rys Quadrature Algorithm

for all *l* do
  for all *k* do
    for all *j* do
      for all *i* do

$$I(m,n,l,s) = \sum_{w} I_x(w,m_x,n_x,l_x,s_x) I_y(w,m_y,n_y,l_y,s_y) I_z(w,m_z,n_z,l_z,s_z)$$

      end for
    end for
  end for
end for

- Summation over the roots over all the intermediate 2-D integrals
- floating point operations = $3 * N * \left(\frac{L_a + 1}{2} \right) \left(\frac{L_b + 1}{2} \right) \left(\frac{L_c + 1}{2} \right) \left(\frac{L_d + 1}{2} \right)$
- Recurrence, transfer and roots have predictable memory access patterns, fewer flops. Quadrature step is the main focus here.

Automatic Code Generation

- Number of registers per thread, shared memory per thread block limits the thread blocks that can be assigned per SM
- Loops implemented directly result in high register usage
- Explicitly unroll the loops. How? *Manually it's tedious and error-prone*
- Use a common template and generate all the cases
- Python based Cheetah template engine is used- reuse existing Python utilities and program support modules easily.

CCSD Algorithm

```
for b in v { // loop over virtual b index
  Dt(i,j,a) = 0

  load t(o,o,v,b)
  load V(o,o,v,b)
  load V(o,v,o,b)
  load V(o,o,o,b)

  Dt += Vt

  // terms with t
  for u in v {
    load t'(o,o,v,u)
    // evaluate terms with t'
    Dt += Vt'
  }
  // terms with v
  for u in v {
    load v'(o,o,v,u)
    // evaluate terms with v'
    Dt += V't
  }
  store Dt(o,o,v,b)
}
```

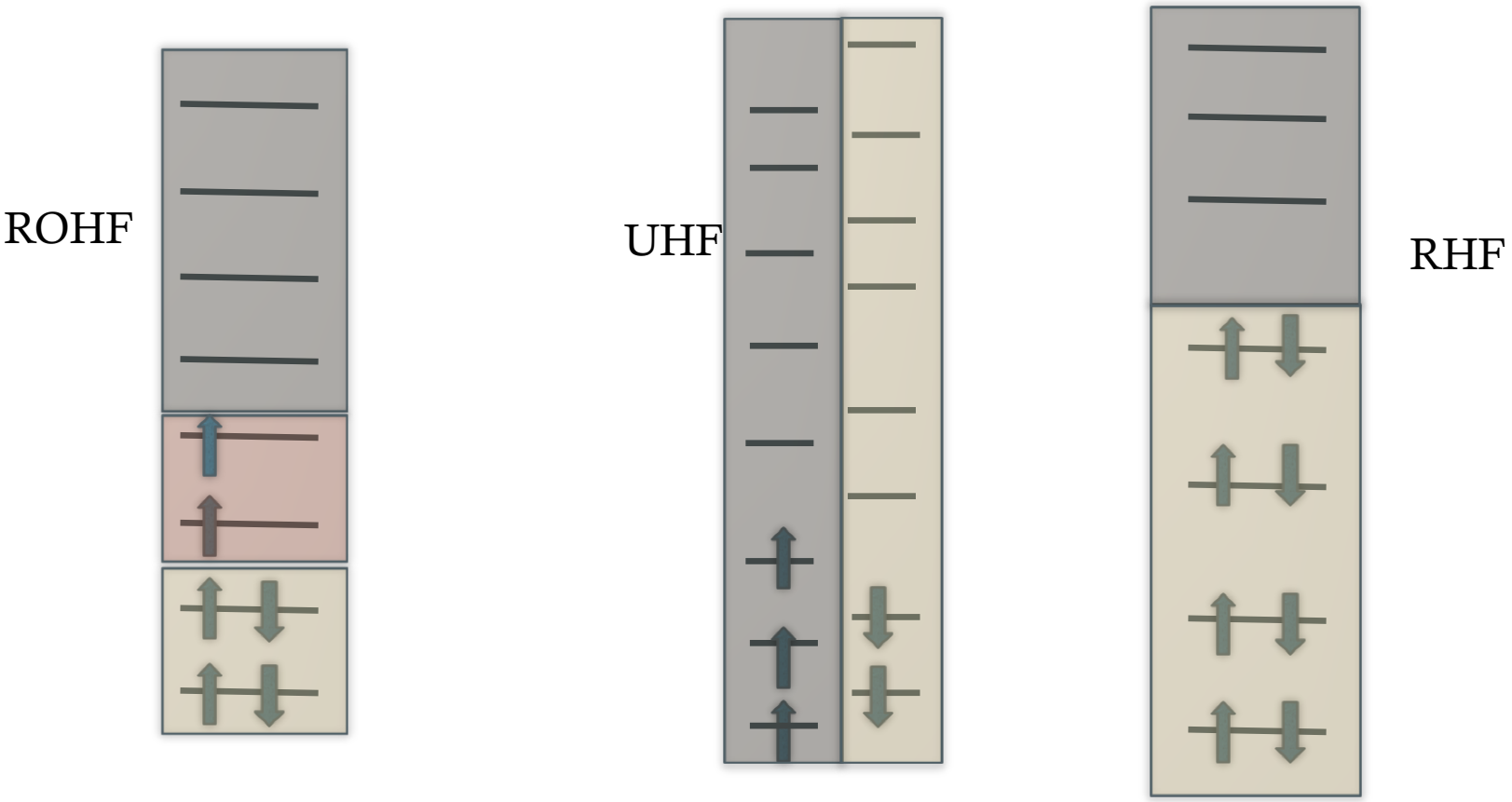
A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)

ROHF Background

- Restricted open-shell Hartree-Fock
- Restricted in the sense that pairs of alpha and beta electrons occupy the same orbitals
- Used for open-shell calculations
- Originally formulated by Roothaan in 1960[1]

1. C. C. J. Roothaan, **Rev. Mod. Phys.**, 32, 179(1960)

ROHF vs. UHF vs. RHF Orbital diagram



- Comparison of ROHF, UHF, and RHF

Rys Quadrature algorithm

Rys Quadrature Algorithm

for all *l* do
  for all *k* do
    for all *j* do
      for all *i* do

$$I(i,j,k,l) = \sum_{w} I_x(w,i_x,j_x,k_x,l_x) I_y(w,i_y,j_y,k_y,l_y) I_z(w,i_z,j_z,k_z,l_z)$$

      end for
    end for
  end for
end for

Overview of Selected *ab initio* Methods in GAMESS (cont.)

- MP2
  - Møller-Plesset 2nd order perturbation theory
  - Scales as N^5,
    - if the problem size is doubled, 32x more expensive
  - Requires the integral transformation from AOs to molecular orbitals (MOs)

RHF Results

Input	Basis	Basis Functions	CPU only time	K80 +CPU	K80 Speedup
Ginkgo	ccd	555	844.1	155.9	5.41x

CCSD Intermediates Algorithm

```
for S in Shells {
  for Q ≤ S {
    for R in Shells {
      for P in Shells {
        // skip insignificant ints
        if (!screen(P,Q,R,S)) continue;
        // evaluate 2-e integrals(PQ|RS)
        V(P,R,Q,S) = eri(P,Q,R,S);
      }
      // i and j are unrestricted
      // loops over all P functions are implied
      // loops over shells Q,S are implied
      for r in R {
        U1(i,j,q,s) = ...
        U12(i,j,q,s) = ...
        load t(o,o,n,r)
        U2(i,j,q,s) += t(i,j,p,r)*V(p,r,q,s)
      }
    }
    store U1(i,j,Q,S), U1(j,i,S,Q)
    store U12(i,j,Q,S), U12(j,i,S,Q)
    store U2(i,j,Q,S), U2(j,i,S,Q)
  }
}
```

A. Asadchev, M. S. Gordon, **J. Chem. Theory Comput.**, 8, 4166(2012)