Electron correlation: theory and practice

Ryan Michael Olson
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd
Part of the Physical Chemistry Commons

Recommended Citation
http://lib.dr.iastate.edu/rtd/15490

This Dissertation is brought to you for free and open access by Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Electron correlation: theory and practice

by

Ryan Michael Olson

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Physical Chemistry

Program of Study Committee:
Mark S. Gordon, Major Professor
  William S. Jenks
  Ricky A. Kendall
  Gordon J. Miller
  Jacob W. Petrich

Iowa State University
Ames, Iowa
2007
# TABLE OF CONTENTS

**LIST OF TABLES** ................................................................. vi

**LIST OF FIGURES** ............................................................ x

**GENERAL INTRODUCTION** ....................................................... 1

**CHAPTER 1. Introduction** ..................................................... 2

1.1 Forward ................................................................. 2

1.2 Methods and Theories .................................................... 4

1.3 Thesis Organization ..................................................... 13

1.4 Bibliography ............................................................. 17

**PART I  GOLD CLUSTERS** ......................................................... 19

**CHAPTER 2. Where does the planar-to-nonplanar turnover occur in small gold clusters?** .............................................. 20

2.1 Introduction ............................................................. 20

2.2 Computational Approach ............................................... 22

2.3 Results and Discussion .................................................. 25

2.4 Conclusions ............................................................... 28

2.5 Bibliography ............................................................. 30

**CHAPTER 3. Theory and Basis Set Effects on Au$_8$** ............... 34

3.1 Introduction ............................................................. 34

3.2 Computational Details .................................................. 39
3.3 Discussion .................................................. 41
3.4 Conclusions ............................................. 46
3.5 Bibliography .............................................. 48

CHAPTER 4. The binding of the noble metal cations Au$^+$ and Ag$^+$ to propene ........................................... 53
4.1 Introduction ............................................. 53
4.2 Computational Details .................................. 54
4.3 Discussion ............................................... 55
4.4 Bibliography .............................................. 59

PART II PARALLEL COMPUTATIONAL CHEMISTRY 62

CHAPTER 5. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model ................................................. 63
5.1 Introduction ............................................. 64
5.2 The Distributed Data Interface .......................... 66
  5.2.1 DDI Programming Model .......................... 66
  5.2.2 DDI Implementation ............................... 68
  5.2.3 Native SHMEM Implementation .................. 69
  5.2.4 Non-Native Implementations ..................... 69
5.3 Improved DDI Data Server Model ..................... 72
5.4 Results / Timings ....................................... 77
5.5 Conclusions ............................................. 84
5.6 Bibliography .............................................. 86

CHAPTER 6. A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO) .................................. 88
6.1 Introduction ............................................. 89
6.2 Methodology and Implementation ..................... 90
CHAPTER 7. A Novel Approach to Parallel Coupled Cluster Calculations:
Combining Distributed and Shared Memory Techniques for Modern Cluster Based Systems

7.1 Introduction ......................................................... 115
7.2 CCSD/CCSD(T) Theory ........................................... 121
7.3 Parallel Design ...................................................... 125
  7.3.1 Parallel Programming Model ................................. 126
  7.3.2 Memory ......................................................... 128
  7.3.3 Molecular Integral Transformation ......................... 129
  7.3.4 Memory Requirements and Bottlenecks .................... 130
7.4 Parallel Implementation ........................................... 135
  7.4.1 CCSD ......................................................... 135
  7.4.2 “Direct” AO-driven Four-Virtual Term ..................... 138
  7.4.3 Triples Correction, MP-(T) ............................... 145
7.5 Computational Details ............................................. 146
7.6 Discussion .......................................................... 147
  7.6.1 Water Hexamer .............................................. 147
  7.6.2 Parallel Performance ....................................... 152
7.7 Future Enhancements .............................................. 156
7.8 Conclusions ......................................................... 157
7.9 Bibliography ......................................................... 159
PART III THEORY PROJECT

CHAPTER 8. Selected Orbital Subspace Methods .......................... 167

8.1 Introduction ......................................................... 168
8.2 Method ............................................................... 170
8.3 Comparison to Current Methods ..................................... 174
8.4 Extended Features .................................................. 176
8.5 Conclusions .......................................................... 181
8.6 Bibliography ......................................................... 183

GENERAL CONCLUSIONS ............................................ 186

ACKNOWLEDGEMENTS ................................................. 188
LIST OF TABLES

Table 2.1 Relative Energies in kcal/mol for Au$_6$ Isomers ................. 25
Table 2.2 Relative Energies in kcal/mol for Au$_8$ Isomers .................. 26
Table 3.1 Relative energies (kcal/mol) for neutral Au$_8$ Isomers. The relative triples contributions to the CCSD(T) method is listed as (T). .......... 42
Table 3.2 Estimated large basis set CCSD(T) relative energies using Eq. 3.1. CCSD(T)/uSBKJC(3f,2g) was estimated using SBKJC(1f) as the small basis set and CCSD(T)/cc-pVTZ-PP was estimated using cc-pVDZ-PP as the small basis set. .................................................. 44
Table 3.3 Relative energies (kcal/mol) and vertical ionization potential (vIP measured in eV) for Au$_8^+$ calculations. ................................. 45
Table 4.1 Binding energies (eV) using the SBKJC basis set. ................. 55
Table 4.2 M = Ag: total energies in Hartrees, binding energies in eV. ....... 56
Table 4.3 M = Au: total energies in Hartrees, binding energies in eV. ........ 57
Table 5.1 DDI benchmark platforms. ............................................ 77
Table 5.2 DDI communication breakdown showing the usage of shared memory transfers for the socket, fast and full versions of DDI as a function of number of processors and nodes on the HP SC. Values have been averaged over all compute processes. Results given under “% of All” represent the total number of DDI calls and as such these values are the same for all three DDI implementations. .......................... 79
Table 5.3 Elapsed times (sec) for the entire GAMESS benchmark run using the various versions of DDI on a variety of platforms with a range of different numbers of CPUs and nodes.

Table 6.1 The division of $N_{nod}$ nodes among $N_{gr}$ groups, the number of groups $M_k$ is given for the group consisting of $k$ nodes.

Table 6.2 Analysis of the Parallelization Costs: Dependence of the Average Wait and Data Exchange Time in Seconds at the MX1 Step, for One Monomer SCF Iteration (the Total Time for All Iterations Is Shown in Parentheses).

Table 6.3 Scalability Tests for $(\text{H}_2\text{O})_{256}$ FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

Table 6.4 Comparison of DDI vs. GDDI Scalability Tests for $(\text{H}_2\text{O})_{256}$ FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster (DDI is the First Row in the Table, $N_{gr} = 1$)

Table 6.5 Comparison of DDI vs. GDDI Scalability Tests for $(\text{H}_2\text{O})_{256}$ FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster (DDI is the First Row in the Table, $N_{gr} = 1$)

Table 6.6 Scalability Tests for $(\text{H}_2\text{O})_{1024}$ FMO-RHF/STO-3G, Evenly Divided into 128 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

Table 6.7 Scalability Tests for Lyso (2036 Atoms) FMO-RHF/STO-3G, Divided into 128 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

Table 6.8 The Total Scalability ($S_{tot}$) Dependence upon Load Balancing (LB) Type at the Upper (Intergroup) Level: Dynamic (D), Static (S) with the Possible Addition of Job Reordering According to the Subtask Size (Indicated by + Where Used).
Table 6.9 Scalability Tests for (H$_2$O)$_{64}$ RHF 6-31G* (Numeric Gradient), on the 7-Hub Star-Topology Fast Ethernet PC Cluster. 109

Table 7.1 General list of data types that describes what type of memory the quantity will be stored in and how the quantity scales as a function of $N_o$ and $N_v$. 130

Table 7.2 The maximum size in gigabytes (GB) of the array to hold the $T_2$ amplitudes, the [VV|OO] integrals or the [VO|VO] MO integrals. The rows correspond to values of $N_o$ and the columns correspond to values of $N_v$. The bold faced values correspond to arrays less than or equal to 6GB. 132

Table 7.3 The actual size of the [VV|VO] integral class as stored in distributed memory. The values are in gigabytes (GB). The rows correspond to values of $N_o$ and the columns correspond to values of $N_v$. The bold face values correspond to those values of $N_o$ and $N_v$ for which the size of the $T_2$ amplitudes array is less or equal to 6GB (Table 7.2). 133

Table 7.4 The size of $N^4_v$ arrays in gigabytes (GB). 134

Table 7.5 Size of $N^3_v$ arrays in megabytes (MB). 135

Table 7.6 Total energies (Hartree), binding energies (kcal/mol) and relative binding energies (kcal/mol) using the aug-cc-pVTZ basis set on the MP2/DH(d,p) optimized structures of Day et al.[43]. The binding energies represent the energy difference between the water hexamer isomer and six isolated water molecules. MP2* represents the MP2/aug-cc-pVTZ calculations from Xantheas et al. [41] who did not examine the boat isomer. 148

Table 7.7 Total energies (Hartree), binding energies (kcal/mol) and relative binding energies (kcal/mol) using the aug'-cc-pVTZ basis set on the MP2/DH(d,p) optimized structures of Day et al.[43]. The binding energies represent the energy difference between the water hexamer isomer and six isolated water molecules. 149
Table 7.8 The difference in relative binding energies between the CCSD(T), CCSD and MP2 methods with respect to basis set measured in kcal/mol. The first column [CCSD(T)-MP2] shows how the relative binding energies differ between the CCSD(T) and MP2 method. The second column [CCSD(T)-CCSD] shows the effect of the triples correction on the relative binding energies. And the last column [CCSD-MP2] shows the differences between CCSD and MP2 on the relative binding energies. The section subtitled “Difference” subtracts the values from the aug-cc-pVTZ set from the corresponding aug'-cc-pVTZ set.

Table 7.9 Estimated CCSD(T)/aug-cc-pVTZ binding energies (kcal/mol) using Eq. (7.29) compared to the actual CCSD(T)/aug-cc-pVTZ at the MP2/DH(d,p) optimized geometries. The differences were rounded up.

Table 7.10 Parallel speedup (S) and parallel efficiency (E) for the MP-CCSD(T) algorithm as a function of the number of processors per node (PPN) and the number of nodes for calculations performed on the prism isomer using the aug'-cc-pVTZ basis set. CCSD-AO represents the AO-driven four virtual term of the MP-CCSD algorithm; CCSD-MO represents all the other MO-based terms of the MP-CCSD algorithm. CCSD-Total is the overall scalability for MP-CCSD algorithm. The speedup and efficiency is also given for the triples correction. Intra-node trends are observed across rows, while inter-node trends are observed down the columns. The benchmark calculations are based on MP-CCSD(T) calculations of the water hexamer (prism isomer) with \( N_o=24, N_v=408 \) and \( N_{bf}=510 \) run on nodes containing a total of 8 processors.
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Au$_6$ isomers.</td>
<td>25</td>
</tr>
<tr>
<td>2.2</td>
<td>Au$_8$ isomers.</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Four low-energy energy isomers for Au$_8$ from Olson et al.[5].</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>The virtual shared-memory model. Each large box (grey) represents the memory available to a given CPU. The inner boxes represent the memory used by the parallel processes (rank in lower right). The gold region depicts the memory reserved for the storage of distributed data. The arrows indicate memory access (through any means) for the distributed operations: get, put and accumulate.</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>The implementation hierarchy for a DDI application; in this case GAMESS. Native implementations of DDI fully support the communication requirements of DDI; non-native implementations require explicit programming or special models to achieve full functionality.</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Data server model on two 2-CPU SMP nodes. Each large (grey) box represents the available memory on the node and the inner boxes represent the memory used by the parallel processes (rank in lower right). The memory reserved for distributed-data (gold) is held in the local address space of the data servers. Thin arrows indicate data transfers via the message passing library, while thick arrows indicate direct memory copies.</td>
<td>70</td>
</tr>
</tbody>
</table>
Figure 5.4  The FAST DDI model is a modification on the original data-server model (Figure 5.3) in which the memory used for distributed data storage is removed from the local address space of the data servers and implemented as a shared-memory segment. As shown, only one compute process/data server pair is attached to a given shared-memory segment. In this model, the get operation can be accomplished entirely through a memory copy, while the accumulate operation can be partially completed through shared-memory; the remaining portion must use the message passing library and run through the data server.

Figure 5.5  The FULL shared-memory model develops from the FAST model (Figure 5.3), but now all DDI processes within a node attach to all the shared-memory segments. The accumulate operation shown can now be completed directly through memory.

Figure 6.1  Comparison of notation for DDI and GDDI. Individual nodes are denoted by rectangular boxes, marked below as master and slaves.

Figure 6.2  One hundred twenty-eight node PC cluster with star topology, connected by Fast Ethernet. Each rectangular block denotes a 24-port Fast Ethernet hub. The numbers near brackets show the number of nodes connected to the particular hub. The reason the nodes are divided in this way is that the 128 node cluster used for the tests is part of a 400-node cluster, and some ports on the above hubs are used by other clusters.
Figure 6.3  The diagram for the GDDI parallelization of the FMO method. “init” stands for initialization, $Y_{LB}n$ is the $n$-th level load balancing of $Y = I$ initial guess, $Y = M$ monomer SCF, $Y = D$ dimer single point runs, $n = 1$ (intergroup) and 2 (intragroup). For all $n = 2$ parts the diagram is simplified not showing loops occurring during those steps (such as, e.g., SCF iterations). “W” is some work performed by one node (typically, some fraction of integrals). $YXn$ is first ($n = 1$) and second ($n = 2$) level data exchange. “conv.” stands for the test of convergence of the monomer SCF. “prop.” stand for properties, currently computed (redundantly) by all group masters. The load-balancing part of the diagram shows the total load that is divided among available nodes, whose number is arbitrary. .......................................................... 96

Figure 7.1  Images of the five geometric isomers used in this study. The geometries correspond to MP2 optimized structures using the DH(d,p) basis set obtained by Day et al. [43]. .......................................................... 120

Figure 7.2  A diagrammatic description of the four-virtual term in the MP-CCSD(T) algorithm. The left hand portion of the diagram is pseudo-code, while the right hand portion is figure illustrates a distributed array. The columns of the distributed array correspond to two occupied indexes, where the total number of columns is $(N_ON_O)^*$. $(N_ON_O)^*$ refers to the lower triangular portion (including diagonal elements) of an $N_O^2$ matrix. The number of rows in the distributed matrix is $N_O^2$. The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms the half-transformed intermediate ($I_{ij}^{\nu\sigma}$) in distributed memory. A global synchronization is used to ensure $I_{ij}^{\nu\sigma}$ is complete before the second parallel task begins. The second parallel task transforms $I_{ij}^{\nu\sigma}$ into $I_{ij}^{ab}$ for all “local” i-j columns.140
Figure 7.3  A diagrammatic description of the Fletcher [50] four-index integral transformation for the [VV\texttext|OO] integral class. The left hand portion of the diagram is pseudo-code, while the right hand portion illustrates a distributed array. The columns of the distributed array correspond to two occupied indexes, where the total number of columns is \((N_o N_o)^\ast\). \((N_o N_o)^\ast\) refers to the lower triangular portion (including diagonal elements) of an \(N_o^2\) matrix. The number of rows in the distributed matrix is \(N_{bf}^2\). The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms half-transformed integrals over two occupied indexes for a given set of two AO shell indexes. The half-transformed integrals are stored in the distributed array. A global synchronization is used to ensure the first task is complete before the second parallel task begins. The second parallel task transforms the final two AO indexes into virtual MO indexes.
Figure 7.4 A diagrammatic description of the four-virtual term of Kobayashi et al. [29]. The left hand portion of the diagram is pseudo-code, while the right hand portion illustrates the distributed arrays. The columns of the distributed arrays correspond to two AO indexes where the total number of columns is $(N_{bf} N_{bf})^*$. $(N_{bf} N_{bf})^*$ refers to the lower triangular portion (including diagonal elements) of an $N_{bf}^2$ matrix. The number of rows in the distributed matrix is $(N_o N_o)^*$ corresponding to the lower triangular portion of an $N_o^2$ matrix. The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms the half-transformed intermediate ($I_{ij}^{\nu\sigma}$) in distributed memory. A global synchronization is used to ensure $I_{ij}^{\nu\sigma}$ is complete before the second parallel task begins. The second parallel task transforms $I_{ij}^{\nu\sigma}$ into $I_{ij}^{ob}$ for all “local” $i$-$j$ columns.

Figure 7.5 A more detailed overview of the first parallel task in Figure 7.2 describing the use of a temporary buffer to store half-transformed integrals. The left-hand portion of the figure is the pseudo-code describing how the buffer is filled. The right-hand portion describes the “Contract and PUT” operation on the buffer. The description of the distributed array is the same as in Figure 7.2.
Figure 8.1 The larger blue region represents the entire molecular complex. Within the QM/MM model, the entire complex is treated with molecular mechanics (MM). The green sections correspond schematically to regions of the molecular complex which are treated using quantum mechanics: (A) represents the largest possible region in which the Hartree-Fock method can be used, (B) represents the largest QM region in which the MP2 method can be used, and (C) represents the largest QM region in which CCSD can be used. This figure represents the inverse relationship between the size and the accuracy for the QM region within the QM/MM model.

Figure 8.2 Using the SOS QM/QM method to enhance the QM/MM model to evaluate a portion of the Hartree-Fock (green) QM region using CCSD (grey). The remainder of the system is treated with molecular mechanics.

Figure 8.3 ONIOM method for describing the energy system in Figure 2. The shapes represent the specific regions of the system and the colors are used to describe the method used (blue = MM, green = HF, grey = CCSD to calculate the energy. The equation is $E_{ONIOM} = E_{MM}(All) + E_{HF}(B) - E_{MM}(B) + E_{CCSD}(A) - E_{HF}(A)$.

Figure 8.4 An illustration of the orbital subdivision. The entire orbital space can be subdivided into an arbitrary set of disjoint orbital subspaces, where each orbital subspace has a set of occupied and virtual LMOs.
GENERAL INTRODUCTION
CHAPTER 1. Introduction

1.1 Forward

The role of modern quantum chemistry is to answer questions about chemical systems by performing calculations on computers that solve particular theoretical models based on first principles and quantum mechanics. In this regard, the field of quantum chemistry can be divided into three primary areas of research: chemical applications, computational methodology, and theory/method development.

The principle application of quantum chemistry is to solve chemistry related problems, e.g. to predict thermodynamic properties, spectroscopic properties, and other experimentally measurable chemical properties. In order to provide meaningful results, one must understand the chemical system and then choose an appropriate theoretical method to calculate properties of the system. Properly choosing the theoretical methods means having an awareness of the strengths and limitations of the variety of theoretical models that are available, and choosing the correct model based on the chemical system in question. Essentially, the question must be asked: can the theoretical model provide a reliable treatment of the chemistry in the system of interest, e.g. Are bonds being made or broken?; Are highly accurate energies for the electronic ground and/or excited states required?

In addition to understanding the strengths and weaknesses of the theoretical models, one must also be aware of the practical limitations on evaluating these models. An understanding of how the models were programmed into a usable method that can be solved on a computer is necessary to assess how a particular system can be studied and what types of resources will be required. A full understanding of the practical limits of modern quantum chemistry methods is critically important when deciding what compromises in chemical accuracy need to be made
to achieve computational feasibility.

In many ways the computer, as the platform upon which the theoretical calculations are performed, is equivalent to the experimental equipment (spectrometers, lasers, chromatograms, and other laboratory instrumentation) found in any experimental chemistry laboratory. In this regard, the rapid evolution of the microprocessor, whose trends over the last forty years have been governed by Moore’s Law, which states that the transistor density on integrated circuits doubles every 18-24 months, has been extremely important in expanding the frontiers of modern quantum chemistry. The increase in computer power and a similar increase in the available memory and disk storage has not only greatly expanded the scope and complexity of our theories, but has also expanded the range of chemical applications that can be studied using those theories. The modern quantum chemist can now study a large range of systems from extremely accurate calculations on diatomic and triatomic molecules to large proteins using first principles \textit{(ab initio)} methods.

Quantum chemists, who have engineered the most promising theoretical methods (some of which are discussed herein) into highly efficient computer codes for general use, have brought the use of quantum chemistry into the mainstream. Creating software packages that are both easy to use and highly efficient on the available computer hardware has been and continues to be an important area of research. This is especially true for researchers developing new methods.

Since many of the high-level methods are very computationally demanding, one means of achieving practical results in a reasonable timeframe involves using multiple processors to simultaneously evaluate unique portions of a complex computational problem, thereby achieving an overall reduction in time to complete the calculation. This type of execution is called parallel computing. Over the last two decades, the use and development of parallel quantum chemistry methods have grown considerably. With the current direction in which computer processors are adding processing density (multiple cores), rather than continuing to significantly increase the power of the individual computing cores, it seems inevitable that the developers of highly computational demanding algorithms such as those used in quantum chemistry will have to
embrace parallel computing at some level in order to advance the applications and methods of
the next generation.

A strong focus of my graduate work has been in the study of chemical systems using high-
level \textit{ab initio} methods to calculate accurate energies of chemical systems as well as improving
the tools and the algorithms needed to apply these high-level methods on current and future
parallel computing platforms.

\section*{1.2 Methods and Theories}

\textit{Ab initio} quantum chemistry involves solving some set of approximations to the Schrödinger
equation for a many-electron and many-nuclear system. The time-dependent Schrödinger
equation \cite{1} is:

\begin{equation}
\hat{H}(t)|\Psi(t)\rangle = -i\hbar \frac{d}{dt}|\Psi(t)\rangle
\end{equation}

where $\hat{H}(t)$ represents the Hamiltonian that describes the total energy of the system, $|\Psi(t)\rangle$ is
the wavefunction whose square, $|\Psi^*(t)\Psi(t)|$, is a probability density function that is capable of
completely describing the physical system, and $t$ is time. Many chemical properties of interest
are not a function of time. For example, the relative equilibrium energies of a set of gas-
phase Au$_8$ isomers do not change over time unless acted on by an outside force. Therefore the
time-independent Schrödinger equation,

\begin{equation}
\hat{H}|\Psi\rangle = E|\Psi\rangle
\end{equation}

where $\hat{H}$ is the time independent Hamiltonian operator and $E$ is the eigenvalue (energy) of
the system, is commonly used as the starting point for many quantum chemistry methods.

Several additional approximations to the time-independent Schrödinger equation are still
required before a practical means of applying this equation to molecules can be obtained. The
first of these approximations deals with the Hamiltonian operator. The Hamiltonian operator
describes the total energy of a system of electrons and nuclei and can be represented by

\[ \hat{H} = \hat{T}_i + \hat{T}_A + \hat{V}_{ij} + \hat{V}_{AB} + \hat{V}_{iA} \]  

(1.3)

where the first two terms represent the single particle kinetic energies of the electrons \((i, j)\) and the nuclei \((A, B)\) and the remaining three terms represent the electron-electron, nuclear-nuclear and electron-nuclear potential energy expressions, respectively. Based on large differences \((\sim 4-5 \text{ orders of magnitude})\) between the mass of the electron and the mass of the atomic nuclei, Born and Oppenheimer [2] proposed that motions of the electrons and the nuclei could be solved separately. This decoupling of electron and nuclear motion leads to the electronic Hamiltonian,

\[ \hat{H}_{\text{elec}} = \hat{T}_i + \hat{V}_{ij} + \hat{V}_{iA} \]  

(1.4)

which describes the energy of the electrons moving in a fixed nuclear field, i.e. the kinetic energy of the atomic nuclei (term 2 of Equation 1.3) is set to zero and the nuclear-nuclear repulsion term becomes a constant potential energy \((V_{nn})\). The electronic Schrödinger equation becomes

\[ \hat{H}_{\text{elec}} |\Psi_{\text{elec}}\rangle = E_{\text{elec}} |\Psi_{\text{elec}}\rangle, \]  

(1.5)

where \(|\Psi_{\text{elec}}\rangle\) is the electronic wavefunction whose square, \(|\Psi_{\text{elec}}^* \Psi_{\text{elec}}|\), is a probability density function for the electrons in a fixed nuclear field and \(E_{\text{elec}}\) is the electronic energy of the system. The electronic energy in the fixed nuclear field, \(E_{\text{elec}}\), can be combined with the constant nuclear-nuclear repulsion energy, \(V_{nn}\), to describe a potential energy function in which the motion of the nuclei can be solved. The first derivative (gradient) and the second derivative (Hessian) of the energy, each with respect to nuclear positions, can be used to minimize the energy of the system with respect to nuclear coordinates (optimize the geometries) and determine the vibrational structure within the harmonic approximation.

The Born-Oppenheimer approximation [2] leads to the definition of the potential energy surface (PES): If the electronic and nuclear motions can be separated, then the electronic
energy of a chemical system can be examined solely as a function of the nuclear coordinates of the system. Particularly important sets of coordinates, i.e. points on the PES of a chemical system, include the local and global minimum energy stationary points, as well as saddle points that connect the minima. The calculations in Chapters 2-4 examine snapshots of the potential energy surfaces for $Au_6$, $Au_8$, $Au^+(CH_2 = CH_2 - CH_3)$ and $Ag^+(CH_2 = CH_2 - CH_3)$ at certain minimum energy points. For $Au_6$ and $Au_8$ (Chapters 2,4), low-energy stationary points on the potential energy surfaces, corresponding to various geometric isomers, are examined to determine which structure is the lowest-energy (global minimum) structure. Similarly, the relative energies between points on the potential energy surfaces correspond to the relative stabilities of the various geometric isomers. In Chapter 3, the energies of a free gold (or silver) cation and propene (reactants) are compared with the energies for the bound complexes to determine the binding energies of these coinage metal ions to propene. The examination of potential energy surfaces, similar to those in Chapters 2-4, forms the basis for most quantum chemical studies.

Because of the importance of describing the potential energy surface of a chemical system, the accuracy with which the electronic Schrödinger equation is solved for the energy and wavefunction at each point on the PES is critically important. The $1/r_{ij}$ electron-electron repulsion term (Equation 1.4), which connects the coordinates of two electrons $i$ and $j$ in the electronic Schrödinger equation, results in a many-body problem that cannot be solved analytically for three or more non-separable particles. There are only approximate solutions to the many-body problem for three or more interacting particles. Therefore, for quantum chemical systems, the accuracy of the electronic energies at points on the PES depend on: 1) the degree of completeness of the wavefunction within a chosen finite basis set approximation, and 2) the approximate method with which the energy and representative wavefunction are solved. It is important to understand and study how these two remaining approximations affect the landscapes of potential energy surfaces for chemical system.

The ground state electronic energy, $E_g$, of the time independent electronic Schrödinger equation (Equation 1.5) can be expressed as an expectation value of the electronic Hamiltonian
and the exact ground state wavefunction,

$$\langle \Psi_g | \hat{H}_{\text{elec}} | \Psi_g \rangle = E_g \langle \Psi_g | \Psi_g \rangle$$

(1.6)

Because the electronic Hamiltonian, $\hat{H}_{\text{elec}}$, is a self-adjoint or Hermitian operator, the variational principle states that the exact ground state wavefunction, $| \Psi_g \rangle$, can be approximated by an arbitrary function, $| \Phi \rangle$, and regardless of how $| \Phi \rangle$ is chosen, the following property holds true:

$$E_g \leq E_\Phi = \frac{\langle \Phi | \hat{H}_{\text{elec}} | \Phi \rangle}{\langle \Phi | \Phi \rangle}$$

(1.7)

The inequality in Equation 1.7 is an equivalence relation if and only if $| \Phi \rangle = | \Psi_g \rangle$. This is a very powerful relation, because $| \Psi_g \rangle$ is not known exactly. Therefore, as long as the exact Hamiltonian is used, the expectation energy for any arbitrary wavefunction, $| \Phi \rangle$, is capped by the lower bound of the true ground state electronic energy. Methods that are based on the variational principle can drive improvements in the wavefunction by systematically adjusting the trial wavefunction such that the energy is minimized, i.e. energy is minimized, i.e

$$\frac{dE_{\text{elec}}}{d\Phi} = 0.$$  

(1.8)

In modern quantum chemistry, the many-electron molecular wavefunction, $| \Phi \rangle$, is commonly expressed as a product (or a linear combination of products) of finite one-electron functions called molecular orbitals,

$$| \Phi \rangle = | \phi_1 \phi_2 \phi_3 \cdots \phi_n \rangle$$

(1.9)

The molecular orbitals are formed from linear combinations of atom centered basis functions $(\chi_n)$,

$$\phi_i = \sum_n c_i^j \chi_n$$

(1.10)

where each atom in the molecule is associated with a particular set of basis functions, $\chi_n$. The
basis functions, $\chi_n$, themselves are formed from a linear combination of functions. Typically these functions take the form of either Gaussian type orbitals (GTOs) [3],

$$N x^p y^q z^r e^{-\alpha r^2},$$

where $N$ is the normalization constant, $x, y, z$ are Cartesian coordinates, $\alpha$ is the orbital exponent and the angular part of the orbital is defined by integers $p, q, r$ such that $p+q+r = l$ where $l$ is the angular quantum number, or Slater type orbitals (STOs) [4], Slater type orbitals (STOs)

$$N r^{n-1} e^{-\zeta r} Y_l^m(\theta, \phi),$$

where $n$ is the principle quantum number, $\zeta$ is the orbital exponent and $Y_l^m(\theta, \phi)$ is the angular part of the orbital. Gaussian type orbitals have the advantage over Slater type orbitals in terms of computational efficiency for polyatomic molecules, since the integrals of GTOs over the electron repulsion operator have an analytic form, whereas the corresponding integrals over STOs must be calculated numerically. However, a larger number of GTOs are required to provide similar accuracy to the physically more sensible form of an STO. In general, GTOs have been predominately used to describe molecular orbitals since researchers began to perform calculation on polyatomic molecules.

To provide a reliable and reproducible representation of molecular orbitals, collections of atomic basis functions are organized by atomic number into libraries called atomic basis sets. For each type of atom in a molecule, the atomic basis set maps the same set of atomic basis functions to the nuclear coordinates of the atom. The smallest chemically sensible atomic basis sets, usually called minimal basis sets, assign basis functions to atoms that resemble the atomic orbital structure of the atom. As an example, a minimal basis set for carbon would have 5 atomic basis functions corresponding to the 1s, 2s, 2p_x, 2p_y and 2p_z atomic orbitals. Increasing the number of atomic basis functions (if chosen well) can improve the description of the molecular orbitals, and therefore the representation of the electronic wavefunction within the finite basis set approximation.
The current state-of-the-art atomic basis sets are generally considered to be the correlation consistent basis sets [5]. These basis sets were developed so that the energy of a system, obtained at a correlated level of theory, improves systematically as one expands the basis set from double-$\zeta$ (D) to triple-$\zeta$ (T) to quadruple-$\zeta$ (Q), etc. Because these basis sets systematically improve the energy, extrapolation techniques can be used to obtain the electronic energy of the system at the complete basis set (CBS) limit. An example of this procedure is discussed in Chapter 4 for the binding energies of gold and silver cations with propene.

A series of increasingly accurate methods have been developed, leading ultimately to the exact solution of the electronic Schrödinger equation within the confines of a given atomic basis set. However, these exact solutions are not computationally feasible for molecular systems with more than 2-4 atoms using reasonably sized basis sets. Fortunately, one frequently only requires methods that reasonably and consistently approximate the exact solution at various points on the potential energy surface.

The starting point for most \textit{ab initio} studies begins with the Hartree-Fock (HF) method. The Hartree-Fock method approximates the $1/r_{ij}$ problem by solving for single electron wavefunctions within the average field of all the remaining electrons. To properly treat the electrons as indistinguishable fermions, the Hartree-Fock description of the wavefunction must be antisymmetric. This is achieved by using a single Slater determinant of the one-electron molecular orbitals.

$$|\Phi_{HF}\rangle = \det \begin{pmatrix} \phi_1(1) & \cdots & \phi_n(1) \\ \vdots & \ddots & \vdots \\ \phi_1(n) & \cdots & \phi_n(n) \end{pmatrix} = |\phi_1\phi_2\phi_3\cdots\phi_n| \quad (1.13)$$

in which the orbitals, as previously described, are a linear combination of atomic basis functions,

$$\phi_i = \sum_n c_i^\alpha \chi_n \quad (1.14)$$

The use of an antisymmetric wavefunction guarantees that electrons of the same spin cannot overlap in space. This explicit correlation of electron motion is called exchange correlation. Using the variational principle, the single determinant Hartree-Fock wavefunction can be op-
timized with respect to the expectation energy, i.e.

\[
\frac{dE_{\text{HF}}}{dc_i^j} = 0, \forall i, j. \tag{1.15}
\]

The \(c_i^j\) are iteratively solved until the change in the Hartree-Fock energy is sufficiently stable and unchanging. This process is called the self-consistent field (SCF) method \[6\]. The computational effort required to solve the Hartree-Fock equations scales as \(O(N^4)\), where \(N\) represents the number of atomic basis functions in the calculation.

The resulting variationally optimized Hartree-Fock wavefunction is the best single determinant wavefunction within the finite basis set approximation. However, the Hartree-Fock method is not adequate for calculating energies or properties with chemical accuracy (< 1kcal/mol). The mean field approximation used in the Hartree-Fock method to reduce the many-electron wavefunction to a product of one-electron wavefunctions results in an incorrect treatment of explicit electron-electron interactions, i.e. electron correlation. The energy difference between the Hartree-Fock energy and the true electronic ground state energy is called the electron correlation energy. While this electron correlation energy is only a small fraction of the total electronic energy (\(\sim 1\%\)), it is absolutely essential in order to achieve chemical accuracy.

To recover the electron correlation energy missing in the Hartree-Fock method, \textit{a posteriori} corrections to the HF wavefunction can be calculated at the cost of additional computational effort. Two methods that are heavily used in this work include second-order Møller-Plesset perturbation theory (MP2) \[7\] and the popular CCSD(T) coupled-cluster method \[8][9\]. The MP2 method has been estimated \[10\] to recover approximately 80% of the correlation energy, while CCSD(T) methods have been estimated \[10\] to recover 95-98% of the electron correlation energy. These methods, especially CCSD(T), are capable of achieving chemical accuracy provided sufficiently large atomic basis sets are used.

The MP2 method \[7\] for calculating a correction to the Hartree-Fock energy is a derivative of Rayleigh-Schrödinger perturbation theory. In this type of theory, a perturbation \(\hat{V}\) is applied
to the unperturbed Hamiltonian, $\hat{H}_0$. The sum $\hat{H}_0 + \hat{V}$ is the exact Hamiltonian, i.e.,

$$\hat{H} = \hat{H}_0 + \hat{V}.$$  \hspace{1cm} (1.16)

The MP2 method is used to calculate a one-time correction to the Hartree-Fock energy, which scales computationally as $O(N^5)$. Due to very efficient implementations, this method is commonly used to calculate energies and optimized geometries. Since the MP2 method does not use the variational principle, there is no guaranteed lower bound on the MP2 energy. The accuracy of the method greatly depends on quality of the reference as represented by the unperturbed Hamiltonian ($\hat{H}_0$). For some chemical systems, e.g. water clusters in Chapter 7, the MP2 method is capable of chemical or near chemical accuracy; whereas, for other types of systems, such as $Au_8$ in Chapters 2 and 3, the method can break down. For truly accurate energies, more sophisticated levels of theory need to be applied.

Coupled cluster (CC) theory [11], in particular the iterative coupled cluster singles and doubles method [8] with perturbative triples [9] [CCSD(T)], has become very popular in recent years because improvements in the algorithm and computational methodologies (Chapters 5 and 7) now allow for calculations to be performed on significantly larger systems than allowed in the past. The coupled cluster wavefunction may be expressed as

$$|\Psi_{CC}\rangle = e^{\hat{T}}|\Phi_0\rangle,$$  \hspace{1cm} (1.17)

where $\hat{T}$ is the cluster operator and $|\Phi_0\rangle$ is the reference single determinant Hartree-Fock wavefunction. The coupled cluster wavefunction is capable of describing the exact electronic wavefunction within the confines of the finite basis set approximation as long as the cluster operator, $\hat{T}$, includes all possible excitation operators $\hat{T}_i$ for a given number of particles, i.e.

$$\hat{T} = \hat{T}_1 + \hat{T}_2 + \hat{T}_3 + \cdots$$  \hspace{1cm} (1.18)

Note, the general cluster excitation operators, $\hat{T}_i$, represent sets of excitations from the refer-
ence determinant, e.g. $\hat{T}_1$ corresponds to all single excitations, $\hat{T}_2$ corresponds to all double excitations, etc. While the full cluster operator is capable of describing the exact wavefunction, the full CC treatment on the reference wavefunction is computationally impractical for the majority of chemical systems. However, the coupled cluster operator can be truncated to an arbitrary level of excitations to produce a variety of size consistent quantum chemical methods whose accuracy depends on where the truncation occurs. CCSDT ($\hat{T} \simeq \hat{T}_1 + \hat{T}_2 + \hat{T}_3$) [12] and CCSD [8] ($\hat{T} \simeq \hat{T}_1 + \hat{T}_2$) are two common methods used to recover electron correlation from restricted Hartree-Fock reference wavefunctions. These methods require solving sets of coupled non-linear equations in an iterative manner. The CCSDT method is capable of achieving chemical accuracy (< 1 kcal/mol); however, even this truncated method, which scales as $O(N^8)$, is computationally impractical. Unlike the CCSDT method, the CCSD method is not expected to achieve chemical accuracy. The difference in these methods is based on the treatment or lack of treatment of the $\hat{T}_3$ (triples) cluster operator or “full T”. In the CCSD(T) method, perturbation theory is used to treat the effects of the triples cluster operator [9] based on a converged solution of the CCSD equations. The perturbative triples correction, (T), is a one-time $O(N^7)$ step after the iterative $O(N^6)$ CCSD iterations are converged. In a study of a variety of approximate CCSDT methods [13], CCSD(T) was found to be the most accurate and stable approximation to the full triples. The CCSD(T) method is currently viewed as the premier single determinant method for recovering electron correlation in small to mid-sized chemical systems.

The discussion of quantum chemistry methods to this point has been based on the non-relativistic Schrödinger equation. The non-relativistic Schrödinger equation does not account for energy-mass-velocity relationship that arises due to Einstein’s Theory of Special Relativity [14]. For the majority of chemical systems this is a reasonable approximation; however, for systems that include heavy elements such as gold, relativistic effects are expected to play an important role. Based on the relativistic Dirac equation, all-electron relativistic four-component Hartree-Fock, MP2 and CCSD(T) methods have been developed. However, these methods are extremely computationally expensive and are not practical for systems like $Au_6$ or $Au_8$ that are
studied in Chapters 2-4. In term of the molecular orbital description of the HF wavefunction, relativistic effects are mainly associated with the high-velocity inner core electrons. While these relativistic effects of the inner core orbitals can significantly affect the size and shape of the valence orbitals, the electrons in the valence orbitals (where the chemistry takes place) do not necessarily require the full relativistic treatment. Therefore, those electrons in the valence orbitals can be reasonably approximated by the non-relativistic Schrödinger equation and the various relativistic corrections that can be applied to the non-relativistic Schrödinger equation. Typically what is done to study atoms with important inner-core relativistic effects is to replace the description of the inner core electrons in the basis set with relativistically correct pseudopotentials that are specifically designed to maintain the proper size and possibly the proper shape, i.e. nodal structure, of the Gaussian type atomic orbital description of the valence. These basis sets using relativistic pseudopotentials for the inner-core electrons and Gaussian type orbitals for the valence can be combined with standard quantum chemistry methods based on the non-relativistic Schrödinger equation; examples can be seen in Chapters 2-4 for studying chemical systems containing gold.

1.3 Thesis Organization

The study of small gold clusters using high level ab initio techniques are discussed in Part I (Chapters 2-4) of this thesis. One of the primary motivations for studying gold clusters was to gain a greater understanding of the catalytic processes that occur for some oxidation reactions in the presence of metal oxide supported gold nanoparticles. These catalytic processes were first discovered by Haruta [15] for CO oxidation [15]-[17] and propene epoxidation [18] reactions. An understanding of the processes and mechanisms involved in the catalytic cycle combined with some novel insights could lead to newly engineered heterogeneous catalysts tuned for the industrial production of products such as propylene oxide, a major industrial product. Several factors have been identified as particularly important in the catalytic process: the size [17] and structure of the gold nanoparticles, the role of the supporting metal oxide and the detailed catalysis mechanism.
The first two papers (Chapters 2-3) discuss calculations on gold hexamer and gold octamer, in an effort to determine the planar to non-planar transition for neutral gold species in the gas-phase using several theoretical models. The methods used in this study include density functional theory (DFT), MP2 and CCSD(T). In addition to examining $\text{Au}_8$ as a possible crossover point at which the lowest energy minimum structures begin to favor non-planar structures, these chapters also compare the trends of the DFT and MP2 methods against the CCSD(T) method to gain some understanding on how the potential energy surfaces of the methods differ with regard to the approximate method and the basis set. This knowledge is valuable for understanding the losses in accuracy of the lower level methods and the potential errors for larger systems where CCSD(T) methods are computationally infeasible.

Chapter 4 details the use of correlation consistent basis sets and coupled cluster theory to extrapolate to the complete basis set (CBS) limit for the binding energies of gold and silver cations with propene using the CCSD(T) method. In this work, the calculations involving silver are in good agreement with the experimental values; however, the calculations on gold are only predictive in nature, as the experimental binding energy has not yet been determined.

Part II (Chapters 5-7) describes research to improve the performance and usability of quantum chemical methods through the use of parallel computing. Parallel computing provides a means to perform large scale \textit{ab initio} calculations like MP2 on large chemical systems (1,000-10,000 basis functions) or highly accurate calculations like CCSD(T) on mid-sized chemical systems (500-1000 basis functions).

In the GAMESS (General Atomic and Molecular Electronic Structure System) package, the Distributed Data Interface (DDI) [20] is used as the middle layer computer code that separates the quantum chemistry (QC) code from the underlying computer science (CS) code that is needed to execute the program in a parallel environment. By removing the CS layer from the QC layer, a consistent set of tools needed by the quantum chemistry code can be maintained, while those tools can be adapted and reengineered in the CS layer to transparently optimize them for new parallel hardware. Chapter 5 of this dissertation describes research designed to improve the performance of DDI for massively parallel architectures that are built using
networked multi-processor computers. Improvements in the DDI code have a wide reaching effect in that all the parallel quantum chemistry codes are enhanced with each improvement to DDI.

Chapter 6 of this dissertation describes the development and use of communication subgroups within the GAMESS/DDI model for the application of the fragment molecular orbital (FMO) method. Without the use of subgroups, the FMO method has limited scalability on a large number of processors; however, using the subgroup division of parallel processes, the execution of the FMO method achieves near perfect parallel performance. The author’s contribution to the work described in Chapter 6 focused on the development of DDI groups for general use; this chapter describes the implementation of the FMO method and numeric gradients using DDI groups.

Combining the new methodologies that originated in the earlier improvements to DDI (Chapter 5), Chapter 7 discusses the implementation of a massively parallel distributed memory CCSD(T) program. As discussed earlier, CCSD(T) is one of the most accurate single-reference methods used in modern quantum chemistry. The focus of the MP-CCSD(T) program is to make the CCSD(T) method applicable to a wider variety of chemical systems by reengineering the method to run in a massively parallel environment.

While improvements in computational methodologies and parallel computing continually extend and enhance the quantum chemistry that can be performed, there will always be limits on the size of chemical systems that can be modeled with high accuracy. In some large chemical systems, the chemistry of interest may take place in a spatially localized area; such areas are typically called active sites. To calculate properties of chemical reactions that occur in active sites of large chemical systems, one must accurately model the active region, while preserving at least a qualitatively correct description of the surrounding chemical environment. In Part III of this dissertation, Chapter 8 discusses a new method for the recovery of the correlation energy in the action region of large chemical systems based on a hierarchical approach for recovering electron correlation in different spatial regions of the molecule. This method called the Selected Orbital Subspace or “SOS” method. The fundamental principle behind
this method is that the molecular orbital space of a large chemical system can be divided into disjoint spatially continuous subspaces. A variety of standard methods, including perturbation and coupled cluster theory, can be used to recover correlation energy within individual orbital subspaces and/or between orbital subspaces. The active region(s) can be treated with high levels of theory, while the surrounded bulk regions are treated with less computationally intensive methods.
Bibliography


PART I

GOLD CLUSTERS
CHAPTER 2. Where does the planar-to-nonplanar turnover occur in small gold clusters?

A paper published in The Journal of the American Chemical Society. 1

Ryan M. Olson², Sergey Varganov², Mark S. Gordon², Horia Metiu³, Steeve Chretien³, Piotr Piecuch⁴, Karol Kowalski⁴, Stanislaw A. Kucharski⁵, Monika Musial⁵

Abstract

Several levels of theory, including both Gaussian-based and plane wave density functional theory (DFT), second-order perturbation theory (MP2), and coupled cluster methods (CCSD(T)), are employed to study Au₆ and Au₈ clusters. All methods predict that the lowest energy isomer of Au₆ is planar. For Au₈ both DFT methods predict that the two lowest isomers are planar. In contrast, both MP2 and CCSD(T) predict the lowest Au₈ isomers to be nonplanar.

2.1 Introduction

Since the discovery that small (2 nm ≤ diameter ≤ 4 nm) gold clusters Auₙ can selectively catalyze reactions, such as the epoxidation of propene [1], there has been a flurry of interest from both experimentalists and theorists in developing an understanding of the origin of this catalytic activity. It appears that several factors play a role in this activity, including the

---

²Department of Chemistry, Iowa State University, Ames, Iowa 50011
³Department of Chemistry and Biochemistry, University of California-Santa Barbara, Santa Barbara, California 93106
⁴Department of Chemistry, Michigan State University, East Lansing, Michigan 48824
⁵Institute of Chemistry, University of Silesia, Szkolna 9, 40-006 Katowice, Poland
presence of a metal oxide (e.g., TiO$_2$) support [2] and the presence of molecular hydrogen [2]. We have previously explored both bare Au$_n$ clusters [3] and the interactions of these clusters with both molecular oxygen [4, 5] and molecular hydrogen [6] to explore these first two factors. It has also been proposed [7] that surface roughening plays an important role in the catalytic activity, since nonplanarity (e.g., corners) in Au$_n$ clusters localizes the electron density and promotes reactivity.

Given the apparent important role of surface roughening in determining the catalytic activity of gold clusters, it is important to determine the value of $n$ at which nonplanar structures begin to dominate as the lowest energy isomers. There have been a number of papers dedicated to the structure of both neutral and ionic gold clusters over the past decade or so [8]-[28], but there appears to be little consensus regarding the “turnover point” from clusters in which planar isomers are lowest in energy to those in which nonplanar isomers dominate. There now does seem to be agreement that the lowest energy structure of Au$_6$ is planar. In early papers, Balasubramanian and Liao proposed, based on “restricted” multireference CI calculations, that the Au$_6$ global minimum is a nonplanar pentagonal bipyramid [8, 9]. Michaelian, Rendon, and Garzón [12], based on an $n$-body Gupta potential predicted that the lowest energy Au$_6$ isomer is a nonplanar square bipyramid. On the other hand, Bravo-Perez, Garzón, and Novarro [13, 14] used second-order perturbation theory (MP2) [29] with a relativistic effective core potential (RECP) and concluded that the lowest energy isomer of Au$_6$ has a planar $D_{3h}$ geometry, with the lowest energy nonplanar isomer ($C_{3v}$ pentagonal pyramid) 0.47 eV higher in energy. These authors also speculated that the transition from planar to nonplanar occurs between $n=6$ and $n=7$ and that nonadditive effects play an important role in favoring planarity. On the other hand, Wilson and Johnston, using a Murrell-Motttram model potential including 2- and 3-body terms, predict the Au$_6$ global minimum to be octahedral [15].

Häkkinen and Landman [16] used the generalized gradient approximation (GGA) density functional theory (DFT) approximation with the PBE functional and molecular dynamics simulations to probe the potential energy surfaces of small gold clusters. These authors predicted a planar capped W structure for Au$_6$. Au$_7$ was predicted to be planar as well. Furche et al. [18]
studied small Au\textsubscript{n}\textsuperscript{−} anions using the BP86 functional and molecular dynamics and predicted a planar structure for both \( n = 6 \) and \( n = 8 \). These same authors examined the corresponding cations and found Au\textsubscript{6}\textsuperscript{+} to be planar and Au\textsubscript{8}\textsuperscript{+} to be nonplanar \[22\]. Wang, Wang, and Zhao \[23\] used DFT/LDA with an RECP basis (with 11 explicit valence electrons on each Au atom) to predict that Au\textsubscript{6}\textsuperscript{−} and Au\textsubscript{8}\textsuperscript{−} have planar \( D_{3h} \) and \( D_{4h} \) structures.

There have been only a few theoretical studies of neutral Au\textsubscript{8}. Wilson and Johnston \[15\] used the Murrell-Mottram model potential to predict that the lowest energy isomer is a \( D_{2d} \) dodecahedron, while H"akkinen and Landman \[16\] used the GGA DFT method with an RECP to predict a \( T_d \) capped tetrahedron. Wang, Wang, and Zhao \[23\] used a different functional with the LDA/DFT/RECP approach to predict Au\textsubscript{8} to be a distorted bicapped octahedron. Most recently, Xiao and Wang used a plane wave DFT basis set and the PW91 functional for Au\textsubscript{14} and Au\textsubscript{20} to predict by interpolation that the crossover from planar to nonplanar gold clusters occurs between 14 and 15 gold atoms \[30\].

One can conclude, based on the foregoing brief historical summary, that there appears to be a consensus that the Au\textsubscript{6} global minimum is planar, although most of the previous calculations were performed at a fairly low level of theory and the actual structure has not obviously been resolved. The nature of the Au\textsubscript{8} global minimum structure remains unresolved.

The present work examines the global and local minimum structures for closed shell singlet states of Au\textsubscript{6} and Au\textsubscript{8} using several levels of theory that include DFT, MP2, and the coupled cluster method \[31\] with singles, doubles, and noniterative perturbative triples (CCSD(T)) \[32\]. The latter is generally considered to be the state of the art in electronic structure theory calculations. The following section summarizes the methods used for the calculations. This is followed in section 2.3 by a presentation of the results and discussion of them. Conclusions are drawn in section 2.4.

### 2.2 Computational Approach

Two sets of calculations were carried out, one using Gaussian basis sets and the other using plane waves. In the former, the SBKJC effective core potential (ECP) \[33\], augmented
in the valence basis set by a set of $f$ functions (exponent = 0.89) was used for geometry optimizations and single-point coupled cluster calculations. To probe basis set effects on the predicted relative energies, a much larger basis set, consisting of the completely uncontracted SBKJC ECP valence basis, augmented by three sets of $f$ functions (exponents = 2.0, 0.84, 0.31) and two sets of $g$ functions (exponents = 1.90, 0.69), was used. Using the smaller basis set, geometries were fully optimized, employing analytic gradients, with both DFT methods using the B3LYP functional [34] and MP2 [29]. The initial structures correspond to some of the structures optimized with the plane wave PW91 method (see below). For Au$_8$, only those structures located within a 10 kcal/mol window relative to the lowest energy structure have been considered. In each case, the nature of the stationary point was determined by calculating and diagonalizing the matrix of energy second derivatives (Hessian): A minimum (first-order saddle point) is characterized by zero (one) imaginary frequencies. Single-point calculations at the MP2 geometries were performed using the standard CCSD(T) approach [32]. In addition to the foregoing calculations with the smaller basis set, single-point MP2 calculations were also performed using the larger basis set. CCSD(T) calculations for Au$_8$ with the larger basis sets are currently beyond the available computational resources. All of the reported MP2 and CCSD(T) calculations were performed using the GAMESS (general atomic and molecular electronic structure system) [35] suite of programs, enhanced by the recently implemented coupled-cluster options [36, 37].

Periodic Kohn-Sham density functional theory calculations have been performed with the Vienna \textit{ab initio} simulation package (VASP) program (version 4.4.5) [38]. The potential energy surfaces of the closed shell singlet states of Au$_6$ and Au$_8$ were initially sampled with the combination of the Perdew and Wang 1991 (PW91) [39] functional and an ultrasoft pseudopotential of 11 “valence” electrons [40]. Relativistic effects were partially taken into account through the use of a relativistic scalar pseudopotential. The Brillouin zone has been sampled at the $\Gamma$-point only. The energy cutoff for the plane-waves expansion was 180 eV, the default for the VASP Au soft pseudopotential. This value is usually set to obtain an error that is less than 10 meV for the bulk cohesive energy. Tests performed with the PAW (projector augmented wave)
pseudopotential, for which a larger cutoff (230 eV) is used, produce no major differences in the predictions presented here. Dipole and quadrupole corrections to the energy were taken into account (to avoid interaction between the cluster and its periodic replicas) by using a modified version of the method proposed by Makov and Payne [41]. A correction to the forces similar to the Harris-Foulkes correction were included. The convergence criterion was $10^{-4}$ eV for the self-consistent electronic minimization and for the change of the total energy between two consecutive ionic steps. Fractional occupancies of the bands were allowed at the beginning of a geometry optimization, using a window of 0.05 eV and the Methfessel-Paxton (first-order) method [42], but all the equilibrium structures were converged to integer occupation numbers.

It is very important to systematically sample the configuration space, especially as the number of atoms in the cluster increases. More than 50 starting structures have been fully optimized for Au$_6$ and Au$_8$ without symmetry constraints to gain a good sampling of the potential energy surfaces. All the starting structures were optimized in a 15 and 16 Å$^3$ supercell for Au$_6$ and Au$_8$, respectively. The clusters were aligned along the diagonal of the box in order to maximize the separation between the clusters and their replica. This ensures a separation larger than 9.5 Å between the cluster and its replicas in all the starting structures studied.

The main focus of the present work is on the lowest energy singlet states of Au$_6$ and Au$_8$. It is important to make certain the singlets are indeed the ground states. This was accomplished by performing spin restricted open shell second-order perturbation theory energy calculations at each of the MP2 singlet geometries. In each case, the triplet state is at least 1 eV (23 kcal/mol) higher in energy than the corresponding singlet. Further, for both Au$_6$ and Au$_8$ the lowest energy triplet isomer is higher in energy than the highest energy singlet isomer. These observations justify the focus on singlet states. Spot checks on the lowest energy Au$_8$ isomers with MCSCF wave functions also suggest that these states are essentially closed shell with little configurational mixing.
Table 2.1 Relative Energies in kcal/mol for Au$_6$ Isomers

<table>
<thead>
<tr>
<th>Method</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCSB PW91</td>
<td>0.0</td>
<td>18.6</td>
<td>10.5</td>
</tr>
<tr>
<td>B3LYP</td>
<td>0.0</td>
<td>19.8</td>
<td>a</td>
</tr>
<tr>
<td>MP2</td>
<td>0.0</td>
<td>10.1</td>
<td>10.3</td>
</tr>
<tr>
<td>CCSD(T)</td>
<td>0.0</td>
<td>15.0</td>
<td>12.4</td>
</tr>
</tbody>
</table>

Optimizes to S1.

![S1, Planar](image1) ![S2, Nonplanar](image2) ![S3, Planar](image3)

Figure 2.1 Au$_6$ isomers.

2.3 Results and Discussion

The key geometric parameters for the three isomers (local minima) found for Au$_6$ are summarized in Table 2.1, and the structures and their relative energies are given in Figure 2.1. All of the methods agree that the lowest energy isomer for Au$_6$ is the planar S1 structure. All methods except B3LYP and MP2 predict the second lowest isomer to be S3, which is also planar. B3LYP optimization of S3 results in a rearrangement to S1. The only nonplanar isomer found at the MP2 level of theory is the pentagonal pyramid, S2. This structure is predicted by the two density functional theory methods and by CCSD(T) to be the highest of the three isomers, by 15-20 kcal/mol. According to MP2, S2 and S3 are essentially isoenergetic. Two other structures were identified by the plane wave PW91 geometry optimizations, for which Hessian calculations cannot be performed. Both B3LYP and MP2 find these structures to be saddle points (first or second order), so they are not considered here.

The salient geometric parameters for the isomers found for Au$_8$ are summarized in Table 2.2, and the structures and their relative energies are illustrated in Figure 2.2. All of the methods employed here predict similar structures. However, in contrast to Au$_6$, the four methods used in this work display marked differences for the relative energies of the Au$_8$ isomers. Even
Table 2.2 Relative Energies in kcal/mol for Au$_8$ Isomers

<table>
<thead>
<tr>
<th>Method</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCSB PW91</td>
<td>0.0</td>
<td>5.5</td>
<td>7.0</td>
<td>7.4</td>
<td>7.4</td>
<td>9.6</td>
<td>9.6</td>
<td>9.9</td>
<td>10.4</td>
</tr>
<tr>
<td>B3LYP</td>
<td>0.0</td>
<td>10.1</td>
<td>15.3</td>
<td>15.4</td>
<td>12.2</td>
<td>18.9</td>
<td>12.4</td>
<td>16.4</td>
<td>14.6</td>
</tr>
<tr>
<td>MP2</td>
<td>30.8</td>
<td>32.4</td>
<td>5.7</td>
<td>7.4</td>
<td>25.4</td>
<td>0.0</td>
<td>21.8</td>
<td>24.4</td>
<td>41.7</td>
</tr>
<tr>
<td>MP2 (large)</td>
<td>26.0</td>
<td>26.6</td>
<td>5.9</td>
<td>5.7</td>
<td>22.9</td>
<td>0.0</td>
<td>20.1</td>
<td>20.9</td>
<td>36.3</td>
</tr>
<tr>
<td>CCSD (T)</td>
<td>4.7</td>
<td>11.7</td>
<td>0.0</td>
<td>2.2</td>
<td>8.5</td>
<td>1.5</td>
<td>9.4</td>
<td>10.3</td>
<td>19.8</td>
</tr>
<tr>
<td>TRIPLES</td>
<td>0.0</td>
<td>-4.0</td>
<td>-8.0</td>
<td>-8.6</td>
<td>-4.5</td>
<td>-10.7</td>
<td>-6.3</td>
<td>-5.5</td>
<td>-3.4</td>
</tr>
</tbody>
</table>

Figure 2.2 Au$_8$ isomers.
though the two DFT methods differ in the type of basis set, type of core potential, and type of functional, they both predict that the two lowest energy species are the planar S1 and S2 isomers. The two DFT methods differ quantitatively, since PW-PW91 and B3LYP find S1 to be 5 and 10 kcal/mol, respectively, higher than S2. Similarly, PW-PW91 finds several nonplanar isomers that are only 2-5 kcal/mol higher in energy than S2, whereas the energy spread predicted by B3LYP is much larger. Nonetheless, the two methods agree that the lowest energy isomers are planar.

In contrast, the two ab initio methods, MP2 and CCSD(T), predict the lowest energy isomers to be the nonplanar species S3 and S6. These two methods also differ quantitatively, as the predicted MP2 energy spread is much larger than that predicted by CCSD(T). This spread in relative MP2 energies contracts somewhat when the larger basis set is used, but the changes are all small. CCSD(T) predicts S6 to be slightly higher in energy than S3, while the order of these two isomers is reversed by MP2, with S3 being higher by 6 kcal/mol. MP2 finds all six nonplanar isomers to be lower in energy than any of the planar ones (S1, S2, S9), while CCSD(T) predicts the planar S1 structure to be the fourth isomer in energy order and the other two planar structures, S2 and S9, to be higher in energy than all of the nonplanar species. Still, these two methods predict that the lowest energy Au$_8$ isomers are nonplanar. One can describe S3 as a capped tetrahedron and S6 as a bicapped octahedron, similar to the two nonplanar species predicted by two earlier DFT studies.

Although all structures in Figure 2.2 are predicted to be local minima by B3LYP, MP2 finds one very small imaginary frequency for each of the S1, S5, S7, and S8 structures of 3, 11, 12, and 20 cm$^{-1}$, respectively. These imaginary frequencies are so small that either they could result from numerical noise or the structures could be first-order saddle points. In the latter case, the imaginary mode in S1 corresponds to an out of plane motion that would clearly lead to one of the nonplanar structures. So, this would not alter any conclusions drawn here. The other three structures are much higher in energy and therefore not central to the main issue addressed here.

To understand the origin of the relative stability of the nonplanar isomers, Table 2.2 also
presents the contribution of the connected triples excitations for each isomer, relative to the planar isomer S1, as estimated by the CCSD(T) approach. Clearly, the triples make an essential contribution of this stabilization, since their most favored isomers are S6, S4, and S3 in that order.

2.4 Conclusions

The primary conclusion to be drawn from the present work is that although both density functional methods, one based on Gaussian basis sets and one based on plane waves, predict that the crossover from planarity to nonplanarity occurs at clusters larger than Au$_8$, both correlated ab initio methods predict that this crossover occurs between Au$_6$ and Au$_8$. It is likely that the fundamental difference between the DFT and correlated ab initio results is that the DFT calculations cannot account for longrange interactions such as dispersion. It is also clear that the inclusion of triple excitations is critical for the CCSD(T) predictions of nonplanarity.

Because this issue of crossover from planar to nonplanar structures may be important in the determination of catalytic activity, it is necessary to consider the remaining limitations in the ab initio calculations. The most reliable method employed here is certainly CCSD(T). Due to the high computational demands of this method (the $N^7$ scaling with the system size), it is not currently possible to optimize the geometries of the Au$_8$ species at this level of theory. So, the impact such CCSD(T) geometry optimizations may have on the predicted relative energies is unknown. Additionally, basis set effects (always a potential factor) cannot be assessed for similar reasons. Finally, the catalytic activity of small gold clusters is observed when such clusters sit on metal oxide surfaces, and the impact of the surface on the structure of the clusters is not yet known.

Acknowledgements

This work was supported by a DURINT grant from the Air Force Office of Scientific Research (M.S.G., H.M.). Additional support by the U.S. Department of Energy, Office of
Basic Energy Sciences, SciDAC Computational Chemistry Program (P.P., Grant No. DE-FG02-01ER15228) is gratefully acknowledged. P.P. also thanks the Alfred P. Sloan Foundation for financial support. Some calculations were performed on a 32-processor Origin 3400 system at Michigan State University obtained through a grant from the National Science Foundation, Chemical Instrumentation Program (Grant No. CHE-9974834), on a 16-processor Origin 2400 system at Michigan State University funded by Michigan Center for Biological Information, and on an IBM Power3 II cluster funded jointly by an IBM SUR grant, the Department of Energy and the Air Force Office of Scientific Research.
Bibliography


CHAPTER 3. Theory and Basis Set Effects on Au$_8$

A paper will be submitted to the Journal of Chemical Physics.

Ryan M. Olson$^1$, Sergey Varganov$^1$, Mark S. Gordon$^1$, Horia Metiu$^2$

Abstract

Using newly developed correlation-consistent basis sets for gold, the relative energies for the neutral Au$_8$ geometric isomers have been re-evaluated and the vertical ionization potentials calculated. The results using the correlation consistent basis sets show that second order perturbation theory (MP2) calculations strongly favor non-planar Au$_8$ structures for all basis sets that were employed. However, the general trend at the coupled cluster [CCSD(T)] level of theory is to increasingly favor planar structures as the basis set is improved. The effects of basis set and the effects of core-valence correlation are discussed.

3.1 Introduction

There has been a booming interest in gold and gold clusters since the discovery of Haruta and coworkers [1] that metal oxide supported gold nanoparticles are active catalysts for a variety of oxidation reactions. One important feature of small gold clusters is the structures of these species. One reason that the structural details are of interest is that it has been proposed [2] that surface roughening can serve to localize the electron density and therefore increase the reactivity of the gold system. In terms of small to moderate size clusters, “roughening” can be

$^1$Department of Chemistry, Iowa State University, Ames, Iowa 50011
$^2$Department of Chemistry and Biochemistry, University of California-Santa Barbara, Santa Barbara, California 93106
interpreted as non-planarity. Therefore, the cluster size, i.e., \( n \) in \( \text{Au}_n \), at which the clusters first begin to be predominantly non-planar, is important.

There have therefore been numerous structural studies [3]-[24] using a variety of theoretical methods to examine the relative stability of \( \text{Au}_n \) isomers, with particular attention given to the “crossover point” in which non-planar isomers become the energetically preferred geometry. One of the most intriguing and perhaps unexpected results of these structural studies of \( \text{Au}_n \) isomers is the prediction by most calculations that planar structures are lower in energy than non-planar structures for clusters with \( n \) much larger than 6. This is found mainly for the neutral [11]-[18][20, 24] and anionic [14, 22, 23] \( \text{Au}_n \) species.

In a study of neutral \( \text{Au}_n \) for \( n=3-6 \), Bravo-Perez and coworkers [3, 4] used second-order perturbation theory (MP2) and a valence only eleven electron (5d\(^{10}\)6s\(^1\)) Hay-Wadt basis set and relativistic effective core potential (RECP) [25] to predict that the global minima for clusters up to \( \text{Au}_6 \) are planar. Based on non-additive effects, these authors concluded [4] that the planar to non-planar turnover point for the MP2 method occurs between \( n=6 \) and \( n=7 \). This conclusion is consistent with the study by Olson et al. [5] who employed MP2 calculations using the 19-electron (5s\(^2\)5p\(^6\)5d\(^{10}\)6s\(^1\)) SBKJC RECP and associated basis set [26], augmented with one set of \( f \) functions (exponent=0.89) [27], to predict that whereas the \( \text{Au}_6 \) global minimum is planar, the non-planar \( D_{2d} \) dodecahedron \( \text{Au}_8 \) structure is favored over the \( D_{4h} \) and \( C_{2v} \) planar structures by more than 20 kcal/mol. Other non-planar structures, including one with \( T_d \) symmetry, are also predicted to be lower in energy than the planar isomers. Olson and coworkers also found [5] that single point energies at the MP2 geometries, calculated using the coupled-cluster singles and doubles with perturbative triples method [CCSD(T)] and the same basis set, favor the non-planar \( T_d \) and \( D_{2d} \) structures. However, the range of relative energies among the planar and non-planar structures was found to be significantly smaller using CCSD(T) than at the MP2 level of theory.

Several other studies [6, 7, 8, 9, 10] have predicted non-planar structures as the lowest energy isomer for \( n \leq 8 \). Using a cubic three-body Murrell-Mottram potential parameterized by Cox et al. [28] to fit experimental properties such as lattice and vacancy formation energies,
lattice spacing, elastic constants, and phonon frequencies, Wilson and Johnson [6] predicted non-planar gold clusters starting at $n = 4$ based on molecular dynamics/simulated annealing techniques. Michaelian et al. [7] used a genetic algorithm for global optimizations and the $n$-body Gupta potential to predict non-planar structures as the lowest energy minima for Au$_6$ and Au$_7$. Balasubramanian and Liao [8] proposed, based on “restricted” multi-reference CI calculations, that the Au$_6$ global minimum is a non-planar pentagonal bipyramid; however, no planar structures were examined in that study. More recently, DFT [9]-[15][18], MP2 [3, 4, 5, 19] and CCSD(T) [5, 19] studies have generally concluded that neutral Au$_n$ up to $n = 6$ are planar; however, depending on the methods used, there remains a wide range of predicted “crossover” points.

Wang and coworkers [9] have reported a density functional theory (DFT) study using the local density approximation (LDA) and an exchange-correlation potential parameterized by Perdew and Wang [29] with a RECP and a double numerical basis set including polarization functions only as large as $d$ functions. This study predicted the first non-planar (pentagonal bipyramidal) structure to occur at $n = 7$. Häkkinen and coworkers [10] used a DFT based molecular dynamics approach using the PBE form [30] of the generalized gradient approximation (GGA) for the exchange-correlation potential to predict planar structures up to $n = 7$, and non-planar structures for $n \geq 8$.

However, other DFT studies [11]-[19][24], all using RECPs, have shown the opposite trend in which a planar minimum is predicted for Au$_8$. Remacle and Kryachko [11] have used DFT with the B3LYP hybrid exchange-correlation functional [31] to predict a planar global minimum for Au$_8$. These authors also suggest that the planar to non-planar crossover begins at $n = 9$, where 2-D and 3-D structures are very similar in energy. Bonacic-Koutecky et al. [15] predicted planar lowest energy minima for Au$_n$ clusters with $n \leq 10$. These researchers used both a 1-electron RECP basis set with the BLYP functional [32] and a 19-electron RECP basis set with the BP86 functional [33]. Clusters larger than $n = 10$ were not examined by Bonacic-Koutecky and coworkers. Walker [12] used a series of basis sets including the Stuttgart 1997 [34], LANL2DZ [25, 35] and CRENBL [36] basis sets with a variety of exchange-correlation
functionals (PW91PW91-GGA [37], B3LYP-Hybrid [31] and SVWN5-LDA [38]) to predict the crossover point for neutral \( \text{Au}_n \) to occur at \( n = 11 \). Fernandez et al. [13, 14] also predict the crossover point to occur between \( n = 10 \) and \( n = 11 \) using GGA DFT methods. Plane wave DFT studies by Xiao and Wang [17] using PW91 [39] have predicted the crossover point in \( \text{Au}_n \) clusters to occur between \( n = 14 \) and \( n = 15 \). This study was later updated by Xiao and coworkers [16] to predict the crossover to occur between \( n = 13 \) and \( n = 14 \), with \( \text{Au}_{13} \) predicted to be the last cluster for which the global minimum is planar.

Most recently, an all-electron DFT/GGA study by Wei et al. [24] concluded that there is a wide co-existence range for gold clusters between \( n = 8 \) and \( n = 15 \), with energy differences between planar and non-planar structures predicted to be less than 0.15 eV. These authors also found that the vibrationally active radial breathing mode is very sensitive to the planar to non-planar transition. Wei and coworkers [24] predicted \( \text{Au}_{13} \) to be the first cluster that has a non-planar global minimum structure.

In a recent study comparing \( \text{Au}_8 \) and \( \text{Cu}_8 \), Gronbeck and Broqvist [18] have shown that the explicit treatment of semi-core electrons \((5s^25p^6)\) can have a significant impact on the relative energies of geometric isomers. They determined \( \text{Au}_8 \) to be planar, noting strong relativistic effects [40]. Han [19] recently recalculated the MP2 and CCSD(T) energies from the work of Olson et. al. [5] to include the effects of core-valence correlation, i.e. the semi-core electrons were correlated. The results of the latter two studies were very similar, in that:

1. The single point energies calculated using MP2 and CCSD(T) with the augmented SBKJC basis sets all predict 3-D minimum energy structures.

2. Increasing the size of the basis set used for MP2 stabilizes the 2-D isomers with respect to the lowest energy 3-D isomer, both with and without core-valence contributions.

3. There are significant differences in the relative energies of 2-D structures relative to 3-D structures for MP2 vs. CCSD(T).

While the trends and the calculated values from the studies by Han [19] and Olson et al.[5] are very similar, the conclusions drawn based on their calculated results are quite different.
Olson et al. [5] acknowledged deficiencies in the basis set; however, computational limitations at that time prevented CCSD(T) calculations with larger basis sets. Han [19] increased the number of electrons correlated in the MP2 and CCSD(T) calculations by including core-valence correlation; however, no improvements to the one-electron basis sets were made. The additive formula:

$$CCSD(T)/\text{Large} = CCSD(T)/\text{Small} + [MP2/\text{Large} - MP2/\text{Small}],$$

was used to estimate the large basis set CCSD(T) energy from the difference in MP2 correlation energy between large [SBKJC(3f,2g)] and small [SBKJC(1f)] basis sets. Using Eq. (3.1), Han concluded [19] that the lowest energy isomer of neutral Au$_8$ is the planar D$_{4h}$ structure. However, although schemes like that embodied in Eq. (1) are commonly used in methods such as G3 [41] and ONIOM [42], they rely on modest changes in relative energies, whereas for the Au$_8$ system, the basis set correction in Eq. (3.1) stabilizes the planar structures by as much as 14 kcal/mol relative to the non-planar structures. In view of the large magnitude of this correction, it is unclear how reliable corrections due to the improvements in the basis set measured for MP2 will be for estimating the corresponding improvement for CCSD(T). In the three years since the paper by Olson et al. [4] appeared, improvements in CCSD(T) algorithms have enabled the use of larger basis sets for CCSD(T) calculations applied to this problem. Therefore, the present work will examine the impact of systematically improving the basis set on the predictions of structures and relative energies for Au$_n$ clusters. This will in turn facilitate an analysis of the validity of additive schemes such as that embodied in Eq. (3.1).

The goal of this work is to systematically study the effects of the basis set on MP2 and CCSD(T) calculations for Au$_8$ geometric isomers using the correlation consistent basis sets recently developed by Peterson and Puzzarini [43], combined with the RECP developed by Figgen and coworkers [44]. Core-valence effects as a function of structure and as a function of basis set are also studied. In addition, the vertical ionization potential for Au$_8$ isomers are calculated and examined with regard to basis set and core-valence effects.
3.2 Computational Details

*Ab initio* calculations on Au$_8$ and Au$_8^+$ isomers were performed using several Gaussian basis sets to calculate the relative stability and vertical ionization potential of Au$_8$ isomers. The one-electron Gaussian-type basis sets for Au used small-core relativistic effective core potentials (RECPs) to describe the 60 core electrons; the remaining 19 electrons, 8 semi-core ($5s^25p^6$) and 11 valence ($5d^{10}6s^1$) electrons, were explicitly treated in the variational space of the reference Hartree-Fock (HF) wavefunction [45]. First, the standard SBKJC basis set and ECP [26] for gold augmented with one set of $f$ functions (exponent = 0.89) [SBKJC(1f)] [27] were used. Second, a fully uncontracted version of the SBKJC valence and semi-core functions augmented with three sets of $f$ (exponents = 2.0, 0.84, 0.31) and two sets of $g$ functions (exponents = 1.90, 0.69) [uSBKJC(3f,2g)] [27] were used. Next, in order to systematically improve the basis set, the recently published correlation consistent basis sets, cc-pVDZ-PP and cc-pVTZ-PP [43], developed by Peterson and Puzzarini were employed. The ECP for the correlation consistent basis sets was developed by Figgen and coworkers [44]. Post-Hartree-Fock methods, including second-order Møller-Plesset perturbation theory (MP2) [46] and the singles and doubles coupled cluster method [47] with perturbative triples [48] [CCSD(T)] were used to recover electron correlation energy from the valence molecular orbitals and, in some cases, the $5s^25p^6$ semi-core orbitals. The partially spin-restricted [49] [RCCSD(T)] open-shell variant of the closed shell CCSD(T) method was used to calculate single point energies for open-shell Au$_8^+$ isomers. Electron correlation energy from the excitation of the semi-core orbitals ($5s^25p^6$ orbitals) will be referred to as core-valence (CV) contributions. The calculations were performed using the GAMESS [50] and MOLPRO [51] electronic structure packages.

The focus here is on the four lowest energy Au$_8$ isomers (shown in Figure 3.1 predicted in the previous study [4] by CCSD(T) calculations with the smaller basis set. The geometries were optimized at the MP2 level of theory with the SBKJC(1f) basis set, without CV correlation contributions. All coupled cluster energies in this study were calculated at these MP2 optimized geometries. Since the optimized structure S4 has nearly $C_{2v}$ symmetry, the geometry used in the CCSD(T) calculations on this isomer was symmetrized from $C_s$ to $C_{2v}$. It was confirmed
Figure 3.1 Four low-energy energy isomers for Au$_8$ from Olson et al. [5].
that this symmetrization resulted in no significant change in energy for the smallest basis set. Valence-only electron correlation energies and CV correlation contributions were calculated for MP2, CCSD and CCSD(T) using SBKJC(1f), cc-pVDZ-PP, and cc-pVTZ-PP. CV contributions to the open shell RCCSD(T)/cc-pVTZ-PP energy for the cations could not be calculated due to computational limitations; however, CV contributions to the RCCSD(T)/cc-pVDZ-PP energies were calculated.

### 3.3 Discussion

The important low energy neutral Au\(_8\) structures (S1, S3, S4 and S6) reported by Olson et al. [5] were reevaluated using the correlation consistent basis sets. The relative energies of the four isomers in Figure 3.1, with respect to the lowest energy minimum structure for the MP2, CCSD and CCSD(T) methods, along with the relative contributions from the perturbative triples correction and the relative core-valence (CV) correlation effects for neutral Au\(_8\) are given in Table 3.1. The results for the three levels of theory are discussed in turn in the following paragraphs.

**MP2.** The S6 structure is the lowest energy MP2 neutral Au\(_8\) isomer, regardless of basis set or contributions from core-valence correlation (Table 3.1). MP2 strongly favors the non-planar S6 structure over the planar S1 structure by more than 20 kcal/mol for all cases. The relative energy of the planar S1 structure with respect to the S6 structure decreases by 8.5 kcal/mol (11.4 kcal/mol with CV effects) upon improving the basis set from SBKJC(1f) to cc-pVDZ-PP and 0.6 kcal/mol (2.2 kcal/mol with CV effects) upon improving the basis set from cc-pVDZ-PP to cc-pVTZ-PP. The relative MP2 energies of the S3 and S4 non-planar structures are 5.5-7.4 kcal/mol higher than S6 over the range of MP2 calculations. CV correlation effects were found to destabilize the planar S1 structure with respect to the non-planar S6 structure by 6.1, 3.2 and 1.6 kcal/mol as the basis set is increased from SBKJC(1f) to cc-pVDZ-PP to cc-pVTZ-PP, respectively. However, for the for uSBKJC(3f,2g) basis set, CV correlation has the opposite effect - it lowers the energy of S1 by 1.7 kcal/mol relative to S6.

**CCSD.** The planar isomer S1 is predicted to be the minimum energy CCSD structure for all
Table 3.1 Relative energies (kcal/mol) for neutral Au₈ Isomers. The relative triples contributions to the CCSD(T) method is listed as (T).

<table>
<thead>
<tr>
<th>Method</th>
<th>S1</th>
<th>S3</th>
<th>S4</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP2/SBKJC(1f)</td>
<td>30.8</td>
<td>5.7</td>
<td>7.4</td>
<td>0.0</td>
</tr>
<tr>
<td>MP2/SBKJC(1f) + CV</td>
<td>36.9</td>
<td>6.1</td>
<td>7.7</td>
<td>0.0</td>
</tr>
<tr>
<td>MP2/uSBKJC(3f,2g)</td>
<td>26.0</td>
<td>5.9</td>
<td>5.7</td>
<td>0.0</td>
</tr>
<tr>
<td>MP2/uSBKJC(3f,2g) + CV</td>
<td>24.3</td>
<td>5.9</td>
<td>5.1</td>
<td>0.0</td>
</tr>
<tr>
<td>MP2/cc-pV</td>
<td>22.3</td>
<td>5.4</td>
<td>6.5</td>
<td>0.0</td>
</tr>
<tr>
<td>MP2/cc-pVDZ-PP + CV</td>
<td>24.3</td>
<td>5.9</td>
<td>5.1</td>
<td>0.0</td>
</tr>
<tr>
<td>CCSD/SBKJC(1f)</td>
<td>0.0</td>
<td>3.3</td>
<td>6.1</td>
<td>7.5</td>
</tr>
<tr>
<td>CCSD/SBKJC(1f) + CV</td>
<td>1.1</td>
<td>0.0</td>
<td>2.6</td>
<td>4.0</td>
</tr>
<tr>
<td>CCSD/cc-pVDZ-PP</td>
<td>0.0</td>
<td>11.6</td>
<td>13.8</td>
<td>16.4</td>
</tr>
<tr>
<td>CCSD/cc-pVDZ-PP + CV</td>
<td>0.0</td>
<td>9.1</td>
<td>10.9</td>
<td>13.7</td>
</tr>
<tr>
<td>CCSD/cc-pVTZ-PP</td>
<td>0.0</td>
<td>15.6</td>
<td>16.4</td>
<td>20.3</td>
</tr>
<tr>
<td>CCSD/cc-pVTZ-PP + CV</td>
<td>0.0</td>
<td>14.9</td>
<td>15.3</td>
<td>19.2</td>
</tr>
<tr>
<td>CCSD(T)/SBKJC(1f)</td>
<td>4.7</td>
<td>0.0</td>
<td>2.2</td>
<td>1.5</td>
</tr>
<tr>
<td>CCSD(T)/SBKJC(1f) + CV</td>
<td>10.0</td>
<td>0.0</td>
<td>2.1</td>
<td>1.1</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVDZ-PP</td>
<td>0.0</td>
<td>3.6</td>
<td>5.2</td>
<td>5.6</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVDZ-PP + CV</td>
<td>0.0</td>
<td>0.1</td>
<td>1.3</td>
<td>1.8</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP</td>
<td>0.0</td>
<td>7.5</td>
<td>7.5</td>
<td>9.1</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP + CV</td>
<td>0.0</td>
<td>6.0</td>
<td>5.7</td>
<td>7.2</td>
</tr>
<tr>
<td>(T)/SBKJC(1f)</td>
<td>0.0</td>
<td>-8.0</td>
<td>-8.5</td>
<td>-10.8</td>
</tr>
<tr>
<td>(T)/SBKJC(1f) + CV</td>
<td>0.0</td>
<td>-11.1</td>
<td>-11.6</td>
<td>-14.0</td>
</tr>
<tr>
<td>(T)/cc-pVDZ-PP</td>
<td>0.0</td>
<td>-8.0</td>
<td>-8.6</td>
<td>-10.8</td>
</tr>
<tr>
<td>(T)/cc-pVDZ-PP + CV</td>
<td>0.0</td>
<td>-9.0</td>
<td>-9.5</td>
<td>-11.9</td>
</tr>
<tr>
<td>(T)/cc-pVTZ-PP</td>
<td>0.0</td>
<td>-8.1</td>
<td>-8.9</td>
<td>-11.2</td>
</tr>
<tr>
<td>(T)/cc-pVTZ-PP + CV</td>
<td>0.0</td>
<td>-8.8</td>
<td>-9.6</td>
<td>-11.9</td>
</tr>
</tbody>
</table>
calculations, except for CCSD/SBKJC(1f) with CV correlation effects. The CCSD/SBKJC(1f) + CV calculation predicts the S3 structure to be the lowest energy isomer, although the energy difference between the S1 and S3 structures is only 1.1 kcal/mol (Table 3.1). Improving the basis set for the CCSD calculations has a similar effect to that described above for MP2: Upon improving the basis set from SBKJC(1f) to cc-pVDZ-PP, the planar S1 structure is stabilized with respect to the non-planar structures by 7.7-8.9 kcal/mol (9.4-10.8 kcal/mol with CV correlation effects included). Likewise, the relative energy lowering of S1 is 2.6-4.0 kcal/mol (4.4-5.8 kcal/mol with CV correlation effects included) upon increasing the basis set from cc-pVDZ-PP to cc-pVTZ-PP. The core-valence correlation effects, on the other hand, destabilize S1 (increase its relative energy) with respect to the non-planar isomers by 4.4-4.7 kcal/mol [SBKJC(1f)], 2.5-2.9 kcal/mol (cc-pVDZ-PP) and 0.8-1.2 (cc-pVTZ-PP).

CCSD(T). The non-planar S3 structure was previously reported to be the lowest energy Au₈ isomer at the CCSD(T)/SBKJC(1f) level of theory. In contrast, the planar S1 structure is the lowest energy CCSD(T) isomer when the correlation consistent basis sets are employed (Table 3.1). For all basis sets, the perturbative triples correction in the CCSD(T) method preferentially lowers the energy of the non-planar Au₈ structures, relative to the planar isomers. This may be seen most directly in the Table 3.1 entries labeled “(T)/(basis)”. When the CV correlation effects are excluded, the triples correction for each isomer is essentially independent of basis set. The preferential energy lowering of the non-planar structures due to the triples correction is 8.0-8.1 kcal/mol for S3, 8.5-8.9 for S4 and 10.8-11.2 kcal/mol for S6. This suggests that one might be able to reasonably estimate the triples corrections for the larger basis sets, for which the full calculations may be too computationally demanding.

Core-valence contributions to the triples are also found to have a stabilizing (relative energy lowering) effect on non-planar structures. However, as may be seen in Table 3.1, this effect decreases as the basis set is improved: The energy lowering due to the CV contributions to the triples correction for the non-planar structures S3, S4 and S6 is 3.1-3.3 kcal/mol [SBKJC(1f)], 0.9-1.1 kcal/mol (cc-pVDZ-PP) and 0.7-0.8 (cc-pVTZ-PP).

Basis Set + Correlation Additivity. As noted earlier [see Eq. (3.1)], one might assume
Table 3.2 Estimated large basis set CCSD(T) relative energies using Eq. 3.1. CCSD(T)/uSBKJC(3f,2g) was estimated using SBKJC(1f) as the small basis set and CCSD(T)/cc-pVTZ-PP was estimated using cc-pVDZ-PP as the small basis set.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S3</th>
<th>S4</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSD(T)/uSBKJC(3f,2g) est</td>
<td>0.0</td>
<td>0.3</td>
<td>0.6</td>
<td>1.5</td>
</tr>
<tr>
<td>CCSD(T)/uSBKJC(3f,2g) (CV) est</td>
<td>0.0</td>
<td>2.4</td>
<td>2.2</td>
<td>3.7</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP est</td>
<td>0.0</td>
<td>5.2</td>
<td>4.8</td>
<td>6.1</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP (CV) est</td>
<td>0.0</td>
<td>3.4</td>
<td>2.7</td>
<td>3.9</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP actual</td>
<td>0.0</td>
<td>7.5</td>
<td>7.5</td>
<td>9.1</td>
</tr>
<tr>
<td>CCSD(T)/cc-pVTZ-PP (CV) actual</td>
<td>0.0</td>
<td>6.0</td>
<td>5.7</td>
<td>7.2</td>
</tr>
</tbody>
</table>

that the effects of basis set improvement and correlation are additive, in order to estimate the CCSD(T)/large basis relative energies. The large basis set CCSD(T) relative energies estimated in this manner (with and without CV contributions) for uSBKJC(3f, 2g) using SBKJC(1f) as the “small” basis set, along with estimated cc-pVTZ-PP using cc-pVDZ-PP as the “small” basis set are presented in Table 3.2, together with the explicitly computed CCSD(T)/cc-pVTZ-PP values. Although this was not mentioned by Han [19], the estimated CCSD(T)/uSBKJC(3f,2g) values without CV contributions suggest that S1, S3 and S4 are separated by as little as 0.6 kcal/mol, with the S1 structure only narrowly favored. These estimated values are in poor agreement (6.9-7.6 kcal/mol error) with the actual CCSD(T)/cc-pVTZ-PP values which are considered to be the most reliable calculations for Au$_8$ in the current study. Including core-valence contributions, the estimated CCSD(T)/uSBKJC(3f,2g) (CV) relative energies differ from the explicitly calculated CCSD(T)/cc-pVTZ-PP (CV) relative energies by 3.5-3.6 kcal/mol. The estimated CCSD(T)/cc-pVTZ-PP relative energies differ from the explicitly calculated values by 2.3-3.0 kcal/mol (2.6-3.3 kcal/mol with CV effects). It is interesting to note, that the core-valence effects on the estimated CCSD(T)/uSBKJC(3f,2g) relative energies preferentially stabilize the planar structures. This is opposite to the trends observed for both the explicitly calculated correlation consistent basis sets and the estimated cc-pVTZ values. On the other hand, the estimated CV correlation effects for the cc-pVTZ-
Table 3.3 Relative energies (kcal/mol) and vertical ionization potential (vIP measured in eV) for Au$_8^+$ calculations.

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S3</th>
<th>S4</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCCSD/cc-pVDZ-PP</td>
<td>5.6</td>
<td>13.4</td>
<td>0.0</td>
<td>31.0</td>
</tr>
<tr>
<td>RCCSD/cc-pVDZ-PP + CV</td>
<td>10.3</td>
<td>13.4</td>
<td>0.0</td>
<td>30.3</td>
</tr>
<tr>
<td>RCCSD(T)/cc-pVDZ-PP</td>
<td>11.5</td>
<td>11.9</td>
<td>0.0</td>
<td>25.7</td>
</tr>
<tr>
<td>RCCSD(T)/cc-pVDZ-PP + CV</td>
<td>17.2</td>
<td>11.8</td>
<td>0.0</td>
<td>24.6</td>
</tr>
<tr>
<td>RCCSD/cc-pVTZ-PP</td>
<td>3.6</td>
<td>running</td>
<td>0.0</td>
<td>32.0</td>
</tr>
<tr>
<td>RCCSD(T)/cc-pVTZ-PP</td>
<td>9.7</td>
<td>running</td>
<td>0.0</td>
<td>26.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>vIP(CCSD)/cc-pVDZ-PP</th>
<th>vIP(CCSD)/cc-pVDZ-PP + CV</th>
<th>vIP(CCSD(T))/cc-pVDZ-PP</th>
<th>vIP(CCSD(T))/cc-pVDZ-PP + CV</th>
<th>vIP(CCSD)/cc-pVTZ-PP</th>
<th>vIP(CCSD(T)/cc-pVTZ-PP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.90</td>
<td>7.74</td>
<td>7.06</td>
<td>8.30</td>
<td>8.02</td>
<td>running 7.15</td>
</tr>
<tr>
<td></td>
<td>8.03</td>
<td>7.77</td>
<td>7.11</td>
<td>8.31</td>
<td>8.10</td>
<td>running 7.35</td>
</tr>
<tr>
<td></td>
<td>7.96</td>
<td>7.82</td>
<td>7.23</td>
<td>8.33</td>
<td>8.10</td>
<td>running 7.35</td>
</tr>
<tr>
<td></td>
<td>8.10</td>
<td>7.86</td>
<td>7.29</td>
<td>8.34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PP basis set are good agreement with the explicitly calculated values for this basis set, for example 1.8 vs. 1.5 kcal/mol relative stabilization for S3, 2.1 vs. 1.9 (S4) and 2.2 vs. 1.9 (S6). Overall, the additivity assumption is not very reliable for the SBKJC basis sets. The approach is more reliable for the correlation consistent basis sets, but even here the errors are as large as 3 kcal/mol. The inability of MP2 to accurately estimate the basis set effect for CCSD(T) for Au$_8$ is likely due to the notably large differences between the two methods for predicting relative energies (as discussed earlier).

Au$_8^+$. The vertical ionization potentials for the four neutral Au$_8$ isomers that were studied in this work were calculated by taking the energy difference between an Au$_8^+$ cation and the corresponding neutral isomer at the same geometry. The relative energies of the RCCSD(T)//MP2/SBKJC(1f) structures and the vertical ionization potentials are given in Table 3.3. All levels of theory predict that the S4 vertical species is the lowest of the four
isomers that were studied. At the highest level of theory, RCCSD(T)/cc-pVTZ-PP, the isomer that is closest in energy to S4 is the planar S1 isomer, at $\sim 10$ kcal/mol. It is emphasized, however, that other $\text{Au}_8^+$ isomers that are much higher on the neutral potential energy surface may well be competitive on the cation surface [22], and that geometry optimization of the cations, at least at the second order perturbation level of theory, to generate the adiabatic ionization potential, may also be revealing. So, the results presented in Table 3.3 should be viewed as preliminary. In this context, note that the values of the vertical ionization potential slightly increase ($\sim 0.1$ eV) as the size of the basis set is increased. Similar trends are found when core-valence effects are included. It remains unclear what the effects on the ionization potential will be if the geometries of both the neutral and the cation structures are allowed to relax. As demonstrated above for the neutral isomers, there are significant differences between the MP2 and CCSD(T) potential energy surfaces. The agreement between CCSD and CCSD(T) is much better, except for the planar S1 isomer. Therefore, to accurately measure the adiabatic ionization potential, the geometries of the structures should be optimized using at least CCSD, and at minimum the cc-pVDZ-PP basis set.

3.4 Conclusions

This study illustrates a remarkable difference between the MP2 method and the coupled cluster methods, CCSD and CCSD(T), for predicting the lowest energy structure of $\text{Au}_8$. The tendency of MP2 to over-bind strongly favors the non-planar structures. Increasing the basis set stabilizes the planar structure at both the MP2 and CCSD levels of theory, while there is little basis set effect on the CCSD(T) triples correction. Core-valence effects are predicted to stabilize the non-planar $\text{Au}_8$ structures relative to the planar structures for all correlation consistent basis sets, however, the effects of CV stabilization on non-planar structures diminish as the basis set increased. The CCSD(T)/SBKJC(1f) level of theory predicts S3 to be the minimum energy structure. However, as the basis set is increased to cc-pVDZ-PP and cc-pVTZ-PP, the triples and core valence contributions that favor the non-planar structures remain the same or diminish, while the CCSD contributions that favor the planar structure increase. As the basis
set is increased, the CCSD(T) crossover point at which the planar Au$_8$ structure becomes the global minimum occurs at the cc-pVDZ-PP basis set. Estimates of larger basis set CCSD(T) values based on MP2 basis set effects may be qualitatively correct, but these estimates are quantitatively highly inaccurate. If one wishes to estimate relative Au$_8$ CCSD(T) energies for larger basis sets (e.g. CCSD(T)/cc-pVTZ-PP), the smallest reliable reference basis set appears to be cc-pVDZ-PP. In any event, MP2 predictions of the relative energies using the correlation consistent basis sets are qualitatively incorrect. This is unfortunate, given the high computational demands of CCSD(T) calculations.

Using the correlation consistent basis sets, increasing the basis set and including core-valence correlation both generally contribute to the stabilization of the neutral structure with respect to the cation, i.e. the value of the vertical ionization potential increases slightly (~ 0.1 eV or less).

Finally it should be noted that due to differences between the MP2 and the CCSD(T) methods for predicting relative energies of the neutral isomers, it is possible that the optimized MP2 geometries do not represent good candidate structures for single point calculations to describe the CCSD(T) potential energy surface. Relaxation of the neutral and cationic Au$_8$ cluster geometries by optimizations at the CCSD or the CCSD(T) levels of theory could still have a significant impact on the relative energies and ionization potentials of Au$_8$ isomers; however, such calculations, especially with a reliable basis set, are impractical at the present time.
Bibliography


CHAPTER 4. The binding of the noble metal cations Au$^+$ and Ag$^+$ to propene

A paper published in Chemical Physics Letters.  \(^1\)

Ryan M. Olson\(^2\), Sergey Varganov\(^2\), Mark S. Gordon\(^2\), Horia Metiu\(^3\)

Abstract

The binding energies for the reaction M$^+ + \text{CH}_2=\text{CHCH}_3 \rightarrow M^+\text{[CH}_2=\text{CHCH}_3]\), for M = Ag, Au, are predicted using both second order perturbation theory (MP2) and coupled cluster theory (CCSD(T)). Systematically improving the quality of the atomic basis set to the complete basis set limit produces a monotonic improvement in the predicted binding energy for M = Ag relative to the experimentally determined value; the final predicted binding energy is within experimental error. Conversely, the same systematic improvement in the atomic basis set results in successively worse agreement at the MP2 level of theory.

4.1 Introduction

The recent interest in the structure and reactivity of small gold and silver clusters, both experimentally and theoretically, has been driven by the unique uses of these clusters in catalysis [1]. The binding of gold or silver cationic clusters to propene is a possible first step in the catalytic process of forming propene oxide. For a single noble metal cation, the binding energy


\(^2\)Department of Chemistry, Iowa State University, Ames, Iowa 50011

\(^3\)Department of Chemistry and Biochemistry, University of California-Santa Barbara, Santa Barbara, California 93106
E\textsubscript{B} of the metal cation M to propene (prop) is calculated according to

\[ E\textsubscript{B}(M) = E\textsubscript{M} + E\textsubscript{prop} - E\textsubscript{M-prop}. \] (4.1)

This 0 K binding energy has been determined experimentally for M=Ag\textsuperscript{+} to be 1.70 ± 0.13 eV in an equilibrium experiment [2]. Briefly, Ag\textsuperscript{+} ions are formed by laser ablation, mass selected and injected into a drift/ reaction cell containing 5 Torr of He and about 0.1 Torr propene. An equilibrium distribution of Ag(prop)\textsuperscript{n+} adducts is formed. A second quadrupole mass spectrometer after the cell records the individual ion intensities which are used to calculate equilibrium constants. The experiment is repeated at temperatures from 150 to 800 K and a van’t Hoff type analysis done to give ΔH and ΔS values for the sequential propene additions [2].

Recently, binding energies for the interaction of propene with gold and silver clusters [3, 4] using plane-wave density functional theory (DFT) methods have been reported by Chretien and co-workers [5]. Their DFT studies show that the predicted binding energies depend on the functional used, however, the difference between calculated binding energies for the two functionals of interest, PW91 and r-PBE, is less variable (difference = 0.30 ± 0.02 eV). Their r-PBE predicted binding energy, expected to be the most reliable based on recent comparisons [5], is 1.72 eV, in good agreement with the experimental result. The PW91 binding energy is 2.01 eV.

This Note reports the binding energies for Ag(C\textsubscript{3}H\textsubscript{6})\textsuperscript{+} and Au(C\textsubscript{3}H\textsubscript{6})\textsuperscript{+} calculated using state-of-the-art ab initio methods and new highly accurate basis sets. These calculations should serve as a benchmark to determine how well other theoretical methods, such as DFT and MP2, can predict relative energies for these types of systems.

4.2 Computational Details

All theoretical results reported here were obtained using the GAMESS [6] and MOLPRO [7] electronic structure packages. The variational space for the single determinant Hartree-Fock
Table 4.1 Binding energies (eV) using the SBKJC basis set.

<table>
<thead>
<tr>
<th></th>
<th>MP2</th>
<th>CCSD(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M=Ag⁺</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBKJC(1f)</td>
<td>1.56</td>
<td>1.43</td>
</tr>
<tr>
<td>Uncontracted SBKJC(3f,2g)</td>
<td>1.74</td>
<td>1.55</td>
</tr>
<tr>
<td>M=Au⁺</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBKJC(1f)</td>
<td>2.84</td>
<td>2.52</td>
</tr>
<tr>
<td>Uncontracted SBKJC(3f,2g)</td>
<td>3.23</td>
<td>2.84</td>
</tr>
</tbody>
</table>

[8] reference wavefunction explicitly treated all the electrons for the main group elements (C,H) and the 19 semi-core and valence electrons for Ag and Au. The remaining core electrons were described using relativistic effective-core potentials (ECP). Second-order Möller-Plesset perturbation theory [9] (MP2) and the singles and doubles coupled cluster method [10], augmented by perturbative triples, CCSD(T) [11], were used to recover energy contributions from electron correlation in the valence orbitals. Only 11 Ag or Au electrons were explicitly correlated in the post-Hartree-Fock methods; i.e., the 8 semi-core electrons were not correlated.

Geometries were optimized with MP2 using the 6-31G+(d) basis set for main group elements [12] and the SBKJC effective core potential (ECP) basis set [13] augmented with one f-function for gold (exponent = 0.89 [14]) and silver (exponent = 1.30). Core electrons were treated using the SBKJC ECP. Nuclear Hessians (energy second derivatives with respect to Cartesian coordinates) were calculated to ensure that each optimized structure is a stationary point on the potential energy surface (PES). CCSD(T) single point energies were calculated for each stationary point using the previously described basis set.

4.3 Discussion

To examine the effect of the atomic basis set on the predicted binding energies, a larger basis set, cc-pVTZ [15] for carbon and hydrogen and a fully uncontracted SBKJC valence basis set, augmented with three f-functions and two g-functions for gold (f exponents = 2.00, 0.84, 0.31; g exponents = 1.90, 0.69) [14] and silver (f exponents = 4.55, 1.47, 0.49; g exponents = 2.49, 0.90), was used. The binding energies calculated using MP2 and CCSD(T) are given in
Table 4.2 M = Ag: total energies in Hartrees, binding energies in eV.

<table>
<thead>
<tr>
<th></th>
<th>Ag⁺</th>
<th>C₃H₆</th>
<th>Ag(C₃H₆)+</th>
<th>E_B (eV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DZ-HF</td>
<td>145.820491</td>
<td>117.081858</td>
<td>262.937672</td>
<td>0.96</td>
</tr>
<tr>
<td>DZ-MP2</td>
<td>146.083753</td>
<td>117.500097</td>
<td>263.643773</td>
<td>1.63</td>
</tr>
<tr>
<td>DZ-CCSD(T)</td>
<td>146.075857</td>
<td>117.557342</td>
<td>263.687682</td>
<td>1.48</td>
</tr>
<tr>
<td>TZ-HF</td>
<td>145.821659</td>
<td>117.115444</td>
<td>262.972357</td>
<td>0.96</td>
</tr>
<tr>
<td>TZ-MP2</td>
<td>146.182039</td>
<td>117.625606</td>
<td>263.872380</td>
<td>1.76</td>
</tr>
<tr>
<td>TZ-CCSD(T)</td>
<td>146.168240</td>
<td>117.682396</td>
<td>263.908646</td>
<td>1.58</td>
</tr>
<tr>
<td>QZ-HF</td>
<td>145.821839</td>
<td>117.123118</td>
<td>262.980230</td>
<td>0.96</td>
</tr>
<tr>
<td>QZ-MP2</td>
<td>146.221697</td>
<td>117.664541</td>
<td>263.952865</td>
<td>1.81</td>
</tr>
<tr>
<td>QZ-CCSD(T)</td>
<td>146.206042</td>
<td>117.716969</td>
<td>263.982833</td>
<td>1.63</td>
</tr>
<tr>
<td>CBS-HF</td>
<td>145.821873</td>
<td>117.125390</td>
<td>262.982536</td>
<td>0.96</td>
</tr>
<tr>
<td>CBS-COR-MP2</td>
<td>0.428665</td>
<td>0.564235</td>
<td>1.025622</td>
<td></td>
</tr>
<tr>
<td>CBS-COR-CCSD(T)</td>
<td>0.411657</td>
<td>0.613480</td>
<td>1.050994</td>
<td></td>
</tr>
<tr>
<td>CBS-MP2</td>
<td>146.25054</td>
<td>117.68936</td>
<td>264.00816</td>
<td>1.85</td>
</tr>
<tr>
<td>CBS-CCSD(T)</td>
<td>146.23353</td>
<td>117.73887</td>
<td>264.03353</td>
<td>1.66</td>
</tr>
</tbody>
</table>

Table 4.1 In general, MP2 predicts larger binding energies than does CCSD(T). This is not surprising as MP2 has a tendency to overbind [14, 16]. For both metals, the binding energy increases as the quality of the basis set is improved.

To systematically account for the basis set effects, the binding energies were re-calculated at the original MP2 geometries using correlation consistent basis sets [15] which systematically improve as the quality increases from double to triple to quadruple-zeta. The correlation consistent basis sets cc-pVXZ (X = D, T, Q) for main group elements [15] are widely used, however, the extension of cc-pVXZ (X = D, T, Q) to include gold and silver is new work soon to be published by Peterson and co-workers [17]. Relativistic pseudopotentials developed by Figgen et al. [18] were used to represent the core electrons for Au and Ag. Using these new basis sets, the CCSD(T) binding energy of Au(C₃H₆)+ and Ag(C₃H₆)+ were extrapolated to the complete basis set (CBS) limit.

The extrapolation to the CBS limit for the binding energy was performed separately for the Hartree-Fock (HF) reference energy and the correlation energy. It has previously been shown
Table 4.3 M = Au: total energies in Hartrees, binding energies in eV.

<table>
<thead>
<tr>
<th>Method</th>
<th>( \text{Au}^+ )</th>
<th>( \text{C}_3\text{H}_6 )</th>
<th>( \text{Au(C}_3\text{H}_6)^+ )</th>
<th>( E_B ) (eV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DZ-HF</td>
<td>134.49840</td>
<td>117.08186</td>
<td>251.63568</td>
<td>1.51</td>
</tr>
<tr>
<td>DZ-MP2</td>
<td>134.75129</td>
<td>117.50010</td>
<td>252.36314</td>
<td>3.04</td>
</tr>
<tr>
<td>DZ-CCSD(T)</td>
<td>134.74461</td>
<td>117.55734</td>
<td>252.40050</td>
<td>2.68</td>
</tr>
<tr>
<td>TZ-HF</td>
<td>134.49946</td>
<td>117.11544</td>
<td>251.67110</td>
<td>1.53</td>
</tr>
<tr>
<td>TZ-MP2</td>
<td>134.83069</td>
<td>117.62561</td>
<td>252.57443</td>
<td>3.21</td>
</tr>
<tr>
<td>TZ-CCSD(T)</td>
<td>134.82037</td>
<td>117.68240</td>
<td>252.60651</td>
<td>2.82</td>
</tr>
<tr>
<td>QZ-HF</td>
<td>134.49951</td>
<td>117.12312</td>
<td>251.67903</td>
<td>1.53</td>
</tr>
<tr>
<td>QZ-MP2</td>
<td>134.86332</td>
<td>117.66454</td>
<td>252.64819</td>
<td>3.27</td>
</tr>
<tr>
<td>QZ-CCSD(T)</td>
<td>134.85156</td>
<td>117.71697</td>
<td>252.67441</td>
<td>2.88</td>
</tr>
<tr>
<td>CBS-HF</td>
<td>134.49952</td>
<td>117.12539</td>
<td>251.68132</td>
<td>1.54</td>
</tr>
<tr>
<td>CBS-COR-MP2</td>
<td>0.38758</td>
<td>0.56424</td>
<td>1.01719</td>
<td></td>
</tr>
<tr>
<td>CBS-COR-CCSD(T)</td>
<td>0.37477</td>
<td>0.61348</td>
<td>1.03915</td>
<td></td>
</tr>
<tr>
<td>CBS-MP2</td>
<td>134.88710</td>
<td>117.68963</td>
<td>252.69851</td>
<td>3.31</td>
</tr>
<tr>
<td>CBS-CCSD(T)</td>
<td>134.87429</td>
<td>117.73887</td>
<td>252.72047</td>
<td>2.92</td>
</tr>
</tbody>
</table>

that the underlying HF energies and the correlation energies are best extrapolated separately using different formulae [19, 20]. Therefore, the Hartree-Fock energy was extrapolated using a first-order exponential fit [19] including the three data points from the double, triple and quadruple-zeta basis sets. The correlation energy was fit using an inverse cubic method, in which only the triple and quadruplezeta energies were included [20]. A summary of the individual CBS energies and the final CBS binding energies are given in Tables 4.2 and 4.3, respectively.

Using the correlation consistent basis sets MP2 again predicts larger binding energies than does CCSD(T). Indeed, as the size of the basis set increases, the agreement with experiment for \( M = \text{Ag} \) worsens. One can assume the same is true for \( M = \text{Au} \). On the other hand, the CCSD(T) \( E_B \) steadily improves for \( M = \text{Ag} \). Similar trends (deterioration of predicted energetics by MP2 and improved agreement with experiment for CCSD(T) as the basis set is improved) were observed previously [14, 16].

The CCSD(T) CBS binding energy for \( \text{Ag(C}_3\text{H}_6)^+ \) is 1.66 eV (Table 4.2). Using the MP2
Hessian, the calculated $D_0$, including vibrational zero point energy (ZPE) corrections is 1.61 eV. This is within the error bounds of the experimental value ($1.70 \pm 0.13$ eV). The excellent agreement between theory and experiment is very satisfying, and lends credence to using the same methodology for the corresponding Au complex, for which experimental binding energies are not available. The CCSD(T)/CBS $E_B$ (Table 4.3) for $\text{Au(C}_3\text{H}_6)^+$ is predicted to be 2.92 eV (2.85 eV when ZPE corrections are included). This is more than 1 eV greater than the binding predicted (and observed) for the Ag complex and reflects the stronger Au binding observed previously [4]. The rPBE binding energy for $\text{Au}^+$ reported in [4] is 3.13 eV.

Future work will address the binding of larger gold and silver clusters with propene. The comparison of theory to experiment will be made as the results of both become available.

**Acknowledgements**

The authors are grateful to Professor Kirk Peterson for making the cc-pVXZ basis sets available to us prior to publication. This work was supported by a DURINT grant from the Air Force Office of Scientific Research.
Bibliography


PART II

PARALLEL COMPUTATIONAL CHEMISTRY
CHAPTER 5. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model


Ryan M. Olson, Michael W. Schmidt, Mark S. Gordon, Alistair P. Rendell

Abstract

An important advance in cluster computing is the evolution from single processor clusters to multi-processor SMP clusters. Due to the increased complexity in the memory model on SMP clusters, new approaches are needed for applications that make use of distributed-memory paradigms. This paper presents new communications software developments that are designed to take advantage of SMP cluster hardware. Although the specific focus is on the central field of computational chemistry and materials science, as embodied in the popular electronic structure package GAMESS (General Atomic and Molecular Electronic Structure System), the impact of these new developments will be far broader in scope. Following a summary of the essential features of the distributed data interface (DDI) in the current implementation of GAMESS, the new developments for SMP clusters are described. The advantages of these new features are illustrated using timing benchmarks on several hardware platforms, using a typical computational chemistry application.
5.1 Introduction

High performance computing has played a major role in the advancement of science and the application of scientific theory to solving modern problems. The evolution of the supercomputer from megaflops, to gigaflops, to teraflops and beyond has stimulated improvements in both the size and accuracy of the theoretical models than can be solved. In the field of computational chemistry, the smallest molecular systems can now be studied at levels equaling or even surpassing the abilities of measurable experiments. However, due to the computational cost of these state-of-the-art methods, even on small systems and using the best available hardware, calculations may take weeks or even months to complete. Inevitably chemists are forced to balance the accuracy of their calculations by the need to obtain the results in a timely fashion. For this reason, coupled with the desire to perform ever larger and more accurate calculations, computational chemists were motivated to become early proponents of high performance computing (HPC).

GAMESS is a widely used computational chemistry package [1] that has constantly sought to exploit the latest HPC platforms, in particular massively parallel processors (MPPs). As well as achieving high performance, the GAMESS developers have been equally keen to ensure that their code is also portable across a wide range of sequential and parallel platforms, thus enabling the community to use the same code on platforms that range from PC to MPP.

The parallel model used in GAMESS is constantly evolving to take into account advances in HPC hardware and software. Initially the parallel code was based on a replicated data message passing approach that used TCGMSG[2], but this moved to MPI-1[3] as that standard became widely accepted and available. Then to make better use of the large aggregate memory available on parallel machines the replicated data approach has been progressively replaced by algorithms that use the Distributed Data Interface (DDI)[4]. The latter essentially provides an interface by which all processes in a parallel job can independently access and modify any data element in a distributed data array. This is true even when the array in question is stored across the memory of a physically distributed-memory computer. In this respect DDI is similar to, and was in part inspired by the early work of Nieplocha et al. on Global Arrays (GA)[5, 6].
The initial version of DDI was developed for Cray T3D systems, and used the one-sided communication capabilities of Cray’s SHMEM library[7] to implement the distributed arrays. However, to provide portability to other systems in which native one-sided communications were not supported, a data server model was developed[4]. In this scheme two processes are (nominally) assigned to each CPU, with one process performing the traditional computational tasks, while the other exists solely to store and service requests for the data associated with the distributed arrays. Depending on the platform, communications between the compute and data server processes occur either via TCP/IP socket connections or MPI.

With the GAMESS/DDI data server model much useful computational chemistry has been performed. This has been particularly true in recent years with the emergence and widespread availability of inexpensive Beowulf cluster systems[8]. Evidence for this is provided by the increasing number of literature citations of GAMESS, now averaging 30-50/month. The data server model was well suited to early Beowulf clusters as these were constructed from single CPU PCs, and the pairs of compute and data server processes could be naturally constrained to run on each PC node. More recently, however, clusters are increasingly being assembled using symmetric multi-processor (SMP) nodes containing a few (typically $\geq 4$) CPUs coupled with high performance and potentially intelligent interconnect networks like Gigabit Ethernet[9], Myrinet[10], SCI[11], or Infiniband[12]. A similar trend is also evident in dedicated super-computers, where, for example, large-scale IBM SP and HP SC systems now use SMP nodes. Indeed, very large shared memory computers, like the SGI Origin 3000 or HP GS, usually have Non-Uniform Memory Access (NUMA) architectures that can be viewed as a cluster of uniform memory SMPs linked via a network, albeit a very good network.

With this move away from single processor toward multi-processor based clusters we are confronted with a considerably more complicated memory model than that which was present when DDI was originally conceived. Now small groups of processes have equally fast access to chunks of memory, while accessing memory between groups of processes is slower. Recognizing this plus the success and popularity of distributed-memory programming models, it is pertinent to consider how these models might be extended to better exploit SMP clusters. The aim of
this paper is to begin to address this issue, presenting an enhanced version of DDI that includes new functionality specifically targeting SMP clusters. Using both the new and original versions of DDI, performance results are presented and discussed for a typical GAMESS computation run on a variety of MPP systems. First, however, we begin with a brief discussion of the existing DDI data server model used in GAMESS.

5.2 The Distributed Data Interface

The Distributed Data Interface (DDI) developed by Fletcher et al.[4] became the parallel interface for the GAMESS computational chemistry program in 1999. Prior to this, as mentioned above, parallelization used a replicated data approach and the TCGMSG message passing library[2]. The initial parallelization permitted a reduction in execution times. However, the problem size was still limited by the resources available on the smallest node. DDI was developed as a means to support distributed-memory programming, while also providing a level of abstraction between the user program (i.e., GAMESS) and the underlying communication libraries and hardware.

5.2.1 DDI Programming Model

The DDI programming model is based on the idea of “virtual shared-memory”, where a portion of the physical memory available to each processor is designated for the storage of distributed data (Figure 5.1). In this model, there are two types of distributed-memory: local and remote. Local distributed-memory is defined as the memory a given process uses to store its portion of the distributed data, while remote distributed-memory is the memory reserved by all the remaining parallel processes for their portions of the distributed data. Every process in a parallel job is allowed to access/modify any element in the distributed memory segment (regardless of its physical location); however, access to local distributed-memory is assumed to be faster than access to remote distributed-memory. Thus the DDI programming strategy aims to maximize the use of local distributed data while minimizing remote data requests. Note that the performance penalty for accessing distributed-memory (local or remote) is completely
Figure 5.1 The virtual shared-memory model. Each large box (grey) represents the memory available to a given CPU. The inner boxes represent the memory used by the parallel processes (rank in lower right). The gold region depicts the memory reserved for the storage of distributed data. The arrows indicate memory access (through any means) for the distributed operations: get, put and accumulate.
Figure 5.2 The implementation hierarchy for a DDI application; in this case GAMESS. Native implementations of DDI fully support the communication requirements of DDI; non-native implementations require explicit programming or special models to achieve full functionality.

dependent on the underlying machine architecture and the parallel library/libraries used in the implementation of DDI.

5.2.2 DDI Implementation

The DDI framework consists of a small set of functions required for replicated and distributed-memory programming. These include common point-to-point and collective operations, along with distributed-data operations, such as one-sided gets, puts and accumulates. There are essentially two types of DDI implementations: native and non-native, as illustrated in Figure 5.2. A native implementation implies that the underlying parallel library (or libraries) fully supports all the necessary communications operations required by DDI, i.e. DDI becomes a wrapper to this library. A non-native implementation of DDI implies the underlying parallel libraries are deficient in some manner and that further explicit programming is required to reach full DDI functionality. Any implementation must support all of the required DDI functions.

In the following sections we discuss the evolution from the native SHMEM implementation of DDI developed for the Cray T3D/T3E systems to the nonnative data server model used on early clusters. We then discuss the benefits and limitations of the data server model, as this
relates directly to the improvements made for DDI on SMP clusters.

5.2.3 Native SHMEM Implementation

The DDI “virtual shared-memory” model (Figure 5.1) most resembles the SHMEM programming model that originated on the Cray T3D. On these systems, memory is physically distributed and processes do not migrate between CPUs, however the address space is shared and by using the SHMEM library any process can read, write or accumulate to any memory location. In a parallel job, a distributed array is created with the array divided into N disjoint sub-patches that are stored in the memory associated with each of the parallel processes. In order to perform distributed-memory operations (get/put/acc), each process maintains a mapping of the distributed array across the set of parallel processes. To guard against simultaneous accesses to the same memory location SHMEM locks are used.

5.2.4 Non-Native Implementations

To support the “virtual shared-memory” model on clusters in which one-sided access to remote memory is not directly available, DDI adopts a data server model. In this model a portion of the memory that is nominally associated with each CPU is designated for the storage of a distributed data object. However, in contrast to the SHMEM model there is no requirement that this memory be directly accessible to the process executing the parallel task (the compute process). Instead, to simulate one-sided communications this memory is associated with a second, data serving process. Moving the local patch of the distributed-memory segment to a separate process means that, subject to the scheduling policy of the underlying operating system, this process will always be available to service requests for its data. An illustration of this model is given in Figure 5.3.

As with the SHMEM model each compute process maintains a mapping of the disjoint sub-patches of each distributed array to the process responsible for that subpatch. However, in contrast to the SHMEM model, what was previously the “local” patch of a distributed array no longer resides in the address space of that compute process, but must be fetched from the
Figure 5.3 Data server model on two 2-CPU SMP nodes. Each large (grey) box represents the available memory on the node and the inner boxes represent the memory used by the parallel processes (rank in lower right). The memory reserved for distributed-data (gold) is held in the local address space of the data servers. Thin arrows indicate data transfers via the message passing library, while thick arrows indicate direct memory copies.
associated data server process. Thus when running on an N processor machine, each CPU executes one compute process and one data server process. For convenience the ranks of the compute processes are \([0,N-1]\) while the ranks of the data servers are \([N,2N-1]\), with the rank of the data server associated with compute process \(A\) being \(A+N\).

Technically, the data server processes are the same executable as the user’s parallel program, but specialized to become data servers on DDI initialization. Thus the code maintains a Single Process Multiple Data (SPMD) model. The role of the data server is to respond to data requests initiated by the compute processes. To do this each data server process waits for incoming requests to arrive. In the TCP/IP implementation while waiting for a request, each data server process is put to sleep, thus essentially yielding full CPU access to the compute process. Achieving the same effect with MPI is not usually possible, since most MPI implementations continuously poll for incoming receive calls and in so doing compete directly with the compute process for CPU cycles. In this respect the TCP/IP data server implementation is usually found to outperform the MPI implementation.

In the TCP/IP data server model when the data server receives a request from a compute process it is woken up and responds appropriately. If a get operation is issued, the requested portion of the distributed array is packed into a contiguous message and sent back to the requesting compute process. Similarly, if a put or accumulate operation is requested, the data server receives the incoming data segment from the compute process, unpacks it, and places it in the relevant location. In this respect the data server model also has some advantages over the SHMEM model. In the SHMEM model accessing a remote patch of memory is likely to give rise to many SHMEM calls corresponding to accesses to different rows or columns of the distributed array. In contrast in the data server model a single message can be sent to the remote process with the contents of the remote memory request packed into a single message.

The dual process data server model guarantees exclusive access to distributed-memory during onesided operations, since a data server can only handle a single request at a time. The model was well suited to early Beowulf clusters, since each node usually comprised a single processor PC and this naturally constrained each compute and data server process to
execute on the same CPU. In contrast, on a multiprocessor SMP node, in the absence of process to processor binding, the possibility exists for the compute and data server processes to execute on any available CPU. The dual model is also very portable, since all inter-process communications rely only on point-to-point operations and, in the absence of anything better, TCP/IP sockets are widely supported.

Forcing all inter-process communication to run through some sort of message-passing library makes for a simple and portable model, but seriously degrades the rate at which a compute process can access its own subpatch of a distributed array. That is, unless the message passing library makes use of the fact that both the compute process and its associated local data server are co-located in the same memory space and uses this fact to achieve improved performance, the advantage of local data over remote data is greatly reduced. Similarly this model also ignores the improved availability each compute process has to memory associated with any of the data servers located within its own SMP node - because each CPU is effectively treated as a separate node, with the total memory divided equally amongst the processes in that node. In the following section we discuss how we have used shared memory segments and semaphores to substantially enhance the performance of the DDI data server model on both single processor and SMP based clusters.

5.3 Improved DDI Data Server Model

As mentioned above, the DDI programming model encourages the user to maximize the use of local distributed memory while minimizing the use of remote distributed memory. However, the original data server implemented all communications using message passing, and this is likely to substantially degrade the advantage of the local over remote distributed data. When this model was first developed, this was a workable option since for most GAMESS calculations the time spent communicating was a small fraction of the total wall time. The increasing use of highly correlated calculations means this is no longer true. Our first improvement has been to use shared memory segments to store all distributed data quantities, while maintaining one memory segment per data server. Each shared memory segment is then attached to both the
Figure 5.4 The FAST DDI model is a modification on the original data-server model (Figure 5.3) in which the memory used for distributed data storage is removed from the local address space of the data servers and implemented as a shared-memory segment. As shown, only one compute process/data server pair is attached to a given shared-memory segment. In this model, the get operation can be accomplished entirely through a memory copy, while the accumulate operation can be partially completed through shared-memory; the remaining portion must use the message passing library and run through the data server.
compute and data server processes. Using this model, the compute process now has direct access to its local distributed-memory, and only needs to use message passing when accessing remote portions of the distributed array. This model is illustrated in Figure 5.3 and will be referred to as the FAST implementation. This model provides a substantial performance boost over the original data server model even on single processor clusters, since it enhances the speed at which a compute process can access its local portion of a distributed array.

In this new model the data server process behaves in the same manner as in the original model in responding to data requests from compute processes. However, exclusive access to the distributed-memory during data requests is no longer guaranteed as both the data server and the associated compute process can now compete to access data from the shared memory segment to which they are both attached (Figure 5.3). Thus, to control access to the shared memory segments we have used an array of general semaphores. One semaphore is used to independently control access to each distributed array with two types of access permitted: read-access and write-access. Thus before either the DDI compute or data server process is allowed to operate (get/put/acc) on a portion of a distributed array, it must first gain access via the appropriate semaphore. A process that is granted read-access is not guaranteed exclusive access. This is useful to permit multiple get operations from the same segment at the same time. Write-access is exclusive access and needs to be acquired before a process can put or accumulate to a distributed array. Since access is granted based on array handles rather than entire shared memory segments, simultaneous access to different arrays is allowed, i.e. the compute process could be accumulating to one array while the data server is accumulating to another. This multi-level access provides another significant advantage over the original implementation which effectively locks the entire local distributed-memory segment (all arrays) whenever any type of distributed data operation is performed.

The next step is to realize that use of shared memory segments permits intra-node communication on multiprocessor systems to be considerably improved by allowing all compute and data servers within a given node to attach to all shared memory segments on that node. This enables the data in all of these segments to be accessed at a rate that is equivalent to it
Figure 5.5 The FULL shared-memory model develops from the FAST model (Figure 5.3), but now all DDI processes within a node attach to all the shared-memory segments. The accumulate operation shown can now be completed directly through memory.

being local. While to date the DDI programming philosophy has been to assume that only one block of the distributed array is local to any given compute process, it is possible that future DDI based algorithms may benefit from knowing that on SMP clusters more than one block is effectively local. To this end we have extended the DDI functionality to include two new functions; DDL_NDISTRI and DDL_NNODE. The former provides the user with information as to what portion of a distributed array is local to that SMP node, while the latter returns the total number of SMP nodes being used and the rank of the SMP node that the calling process is running on.

The strategy taken to develop full SMP support for the data server model is a small, but important, step forward from that used in the FAST DDI scheme (Figure 5.3) in which shared-memory segments are shared only between pairs of compute and data server processes. The new scheme is illustrated diagrammatically in Figure 5.3, and will be referred to as FULL SMP support. Data access in the FULL SMP model is no different from that in the FAST model,
because each shared memory segment already has its own array of semaphores to control access. The only (non-trivial) difference is that now the possibility for multiple simultaneous access has increased further.

The role of the data servers in the FULL SMP implementation of DDI is now considerably different from the original implementation. There are two important implications that result when each data server is given access to all the shared-memory segments on that node. First, each data server can now handle data requests for any patch of the distributed array that resides on that node, not just the data owned by its associated compute process. The second implication is that all the data servers on a node are equivalent.

The equivalence of the data servers within a node means that we can now change how a compute process on one node requests distributed-data from a remote node, i.e. how inter-node distributed data requests are handled. In the original model, a compute process would send a data request to each data server that was responsible for part of the distributed data it was seeking. On SMP nodes this could mean multiple requests being sent to the same node. In the FULL SMP implementation, all compute processes now maintain a mapping of each distributed array both by processor and by node. Thus when performing a distributed memory operation, the compute process checks the node mapping rather than the processor mapping and sends one request to just one of the remote node data servers. This data server then returns all the requested data even if it was gathered from multiple shared memory segments. The implication is that there can be a significant reduction in the total number of point-to-point operations needed in order to receive the same amount of remote data.

Finally we note that the current decision to use multiple shared-memory segments in the full model instead of one large shared memory segment is partly prompted by recent trends in NUMA shared memory machines, like the HP GS1280, SGI Origin 3000, and 64-bit AMD Opteron. In these architectures, each CPU has a dedicated memory controller and a local bank of memory. Shared-memory between processors is accomplished via high-speed communications links in the processors and some means of cache coherency. Because of the disjoint nature of the memory in NUMA machines, the creation of a single large shared memory segment re-
lies too heavily on the OS to correctly distribute the shared-memory segment throughout the memory of the machine. The creation of multiple shared-memory segments helps ensure that the memory that is meant to be local for a given CPU is actually present in that CPU’s local memory.

### 5.4 Results / Timings

To test the performance of the two new DDI data server implementations, the new codes were benchmarked against the original data server and, when available, the SHMEM code. For reasons noted above, the underlying message-passing library for all data server implementations is TCP/IP sockets. The results (discussed below) labeled “DDI-Socket” and “DDISHMEM” refer to the original DDI implementations, while those labeled “DDI-Fast” and “DDI-Full” correspond to the two new DDI implementations described in Section 3.

Performance data was obtained for a variety of platforms that used various high-performance interconnects. Details of these platforms are given in Table 5.1. The HP SC and the IBM Power3-II platforms were chosen because they represent the building blocks for modern cluster based supercomputers, but differ in their network capabilities. The IBM uses commodity Gigabit Ethernet while the HP SC uses the much more expensive Quadrics interconnect. The single processor node Compaq cluster built on Myrinet is included to test the improvements gained by having fast access to local distributed-memory in the new shared-memory model. Finally the HP GS1280 is a NUMA shared-memory machine that is scalable up to 64 CPUs. We include this system to assess the suitability of the new FULL shared-memory data server model for a genuine large scale shared memory system.

The GAMESS calculation that we have used to assess performance is a second order pertur-
bation theory (MP2) gradient evaluation on a benzoquinone derivative used in the synthesis of hongconin, a cardioprotective natural product. This is the same benchmark used by Fletcher et al.\cite{4} in previous work. It uses 245 atomic basis functions and requires a total of \(~1250\) MB of distributed-memory to run. The computational cost of this type of calculation, as implemented in GAMESS, scales as \(O(N^5)\), where \(N\) is the number of atomic basis functions. The memory requirements for the distributed arrays scale as \(O(N^4)\). The MP2 method is widely used in quantum chemistry to provide accurate information on the energetics, kinetics and infrared spectra of molecules.

The distributed-memory MP2 gradient\cite{13, 14} was chosen as a benchmark because the algorithm performs a significant number of local and remote distributed data operations, thus stressing the underlying DDI implementation. By current standards in the chemistry community, the MP2 gradient for benzoquinone is a relatively small calculation, but for benchmarking purposes it is necessary that the calculation can fit into memory available on a single CPU on each of the various platforms available to us. Even though this benchmark is regarded as relatively small, the calculation still requires approximately 62 GB of DDI data transfers, i.e. over 50 times more data is transferred to and from the DDI arrays than is stored in them. The detailed profiling information for the distributed-memory operations in the benzoquinone benchmark is given in Table 5.2. The data, although similar to that presented previously by Fletcher et al.\cite{4}, augments their results by showing how the profile varies with numbers of processors used.

Consider first the combined socket and shared memory data transfers on 4 CPUs and 1 node of the HP SC, i.e. the rows entitled “All (via socket + shared memory)”. This shows that based on the number of calls, the MP2 gradient is dominated by get operations; these exceed the sum of all accumulate and put operations by \(~9:1\). However, based on the total volume of data transferred, get operations account for only \(~30\)%, while accumulates correspond to roughly \(67\)%.

The implication of this is that get operations involve a large number of relatively short data transfers (average 0.05MB), while accumulates involve a much smaller number of larger transfers (average 1.6MB). Put operations, on the other hand, are much smaller in
Table 5.2 DDI communication breakdown showing the usage of shared memory transfers for the socket, fast and full versions of DDI as a function of number of processors and nodes on the HP SC. Values have been averaged over all compute processes. Results given under “% of All” represent the total number of DDI calls and as such these values are the same for all three DDI implementations.

<table>
<thead>
<tr>
<th>Number of Processors / Number of Nodes</th>
<th>4 / 1 Calls</th>
<th>Mbytes</th>
<th>% of All</th>
<th>8 / 2 Calls</th>
<th>Mbytes</th>
<th>% of All</th>
<th>16 / 4 Calls</th>
<th>Mbytes</th>
<th>% of All</th>
</tr>
</thead>
<tbody>
<tr>
<td>All (via socket + shared memory)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDI_Get</td>
<td>90329</td>
<td>4567</td>
<td>28.6</td>
<td>45165</td>
<td>2284</td>
<td>28.6</td>
<td>22582</td>
<td>1142</td>
<td>28.4</td>
</tr>
<tr>
<td>DDI_Acc</td>
<td>6480</td>
<td>10736</td>
<td>67.4</td>
<td>6480</td>
<td>5369</td>
<td>67.4</td>
<td>6480</td>
<td>2684</td>
<td>67.4</td>
</tr>
<tr>
<td>DDI_Put</td>
<td>3238</td>
<td>637</td>
<td>4.0</td>
<td>1619</td>
<td>319</td>
<td>4.0</td>
<td>809</td>
<td>159</td>
<td>4.0</td>
</tr>
<tr>
<td>Local (via shared memory)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDI_Get</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>DDI_Fast</td>
<td>71650</td>
<td>4207</td>
<td>92.1</td>
<td>33603</td>
<td>2073</td>
<td>90.8</td>
<td>16009</td>
<td>1028</td>
<td>90.0</td>
</tr>
<tr>
<td>DDI_Full</td>
<td>90329</td>
<td>4567</td>
<td>100.0</td>
<td>39374</td>
<td>2164</td>
<td>94.7</td>
<td>17912</td>
<td>1052</td>
<td>92.1</td>
</tr>
<tr>
<td>DDI_Acc</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>DDI_Fast</td>
<td>1620</td>
<td>2684</td>
<td>25.0</td>
<td>810</td>
<td>671</td>
<td>12.5</td>
<td>404</td>
<td>168</td>
<td>6.3</td>
</tr>
<tr>
<td>DDI_Full</td>
<td>6480</td>
<td>10736</td>
<td>100.0</td>
<td>2684</td>
<td>2684</td>
<td>50.0</td>
<td>1620</td>
<td>671</td>
<td>25.0</td>
</tr>
<tr>
<td>DDI_Put</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>DDI_Fast</td>
<td>3238</td>
<td>573</td>
<td>90.0</td>
<td>1619</td>
<td>281</td>
<td>88.1</td>
<td>809</td>
<td>139</td>
<td>87.4</td>
</tr>
<tr>
<td>DDI_Full</td>
<td>3238</td>
<td>637</td>
<td>100.0</td>
<td>1619</td>
<td>298</td>
<td>93.4</td>
<td>809</td>
<td>143</td>
<td>89.9</td>
</tr>
</tbody>
</table>
number and have an average size (0.2MB) that lies between that of the get and accumulate operations.

For the get and put operations, increasing the number of processors from 4 to 16 results in a linear decrease in both the number of calls and volume of data transferred, indicating good scalability. For the accumulates, however, only the volume of data decreases while the number of calls remains constant at 6480. Thus, accumulates scale in that the data transfers decrease with the number of processors, but do not scale in the sense that regardless of the number of processors used, each GAMESS process will always require a fixed number of accumulates. This will ultimately limit scalability, which will also be impacted by the increasingly smaller average data transfer sizes making less efficient use of the underlying communication hardware.

Table 5.2 further breaks down the distributed-data operations by listing those portions that can be accomplished via shared-memory. The results given as “calls” correspond to the total number of GAMESS DDI calls that give rise to at least one shared memory data transfer, while the corresponding “mbytes” relates only to that fraction of the total data transfers that use shared memory. That is, if we consider a one to one mapping between compute and data sever processes, DDI calls can be classified as giving rise to exclusively local, exclusively remote, or mixed (global) data transfers[4]. The number of calls given in Table 5.2 when using shared memory is an aggregate of all the DDI calls that are either exclusively local or mixed, but the mbytes transferred is just that fraction of the total data transferred by these calls that occurs via shared memory.

From the shared memory breakdown for DDI-Fast on 4 CPUs and 1 node of the HP SC it is evident that gets are heavily biased towards local transfers, with over 92% of the transfers (by volume) occurring via shared memory. A similar conclusion is reached for put operations, although in this case the difference between the number of DDI-Fast and DDI-Full calls (3238-637) shows that there are a substantial number of DDI put operations that give exclusively remote transfers but with relatively little data (637-573MB). Meanwhile accumulates are more evenly balanced, with DDI-Fast recording exactly one quarter of all accumulate calls and transferring one quarter of the total accumulate data transfers.
The profile results obtained with increased processor count further reflect the locality of the data transfers. That is, for both the get and put operations the fraction of transfers that can be achieved via shared memory remains relatively flat, while for accumulates it decreases linearly. This would suggest that the biggest advantage of DDI-Full over DDI-Sockets will be on one SMP node, and that this performance difference will decrease as SMP nodes are added. However, due to the locality of the get and put operations, there will always be some advantage in using DDI-Full over DDI-Socket regardless of the number of SMP nodes. Thus, for this benchmark as SMP nodes are added, the percentage of distributed-data operations that can be accomplished via shared-memory reaches a practical limit corresponding to ~90% of all get and put operations and essentially none of the accumulate operations; this translates to roughly 25-30% of all DDI operations.

Timing data on the various platforms is given in Table 5.3. This shows that the FAST and FULL implementations consistently outperform the original DDI implementations on SMP systems. The advantage of intra-node data transfers through shared-memory vs. the TCP/IP layer is best measured in the 1 CPU benchmark, where one sees an improvement of 727 seconds (~15%) on the HP SC. This improvement results from both faster transfer rates (via shared-memory copies) and lower latency, i.e. the time to acquire access via a semaphore vs. the time required by the OS to wake the data server. The importance of shared-memory transfers is further emphasized by the 4 CPU DDI-Full results which show a time reduction of ~25% compared to the original socket implementation. In this case, the original implementation has 4 compute processes stressing the TCP/IP layer (on the same machine) which are also competing for access to the 4 data servers. In the DDI-Full implementation, the data servers remain inactive and all the data is transferred via shared-memory.

On SMP nodes, where multiple distributed-memory segments exist within the same node, DDI-Full has two distinct advantages over DDI-Fast: i) the ability to perform operations directly on all intra-node shared-memory segments and ii) the reduction in the number of communications needed to perform remote inter-node data operations. On the HP SC and in comparison with the original socket code, an improvement of 25%-35% in the total execution is
Table 5.3 Elapsed times (sec) for the entire GAMESS benchmark run using the various versions of DDI on a variety of platforms with a range of different numbers of CPUs and nodes.

<table>
<thead>
<tr>
<th>DDI Model</th>
<th>Number of CPUs / Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP SC</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td>4/1</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
</tr>
<tr>
<td></td>
<td>16/4</td>
</tr>
<tr>
<td></td>
<td>32/8</td>
</tr>
<tr>
<td>DDI-Full</td>
<td>3568</td>
</tr>
<tr>
<td></td>
<td>1915</td>
</tr>
<tr>
<td></td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>658</td>
</tr>
<tr>
<td></td>
<td>432</td>
</tr>
<tr>
<td></td>
<td>267</td>
</tr>
<tr>
<td>DDI-Fast</td>
<td>3532</td>
</tr>
<tr>
<td></td>
<td>2018</td>
</tr>
<tr>
<td></td>
<td>1081</td>
</tr>
<tr>
<td></td>
<td>759</td>
</tr>
<tr>
<td></td>
<td>520</td>
</tr>
<tr>
<td></td>
<td>354</td>
</tr>
<tr>
<td>DDI-Socket</td>
<td>4259</td>
</tr>
<tr>
<td></td>
<td>2284</td>
</tr>
<tr>
<td></td>
<td>1272</td>
</tr>
<tr>
<td></td>
<td>863</td>
</tr>
<tr>
<td></td>
<td>597</td>
</tr>
<tr>
<td></td>
<td>422</td>
</tr>
<tr>
<td>DDI-SHMEM</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>2340</td>
</tr>
<tr>
<td></td>
<td>1502</td>
</tr>
<tr>
<td></td>
<td>868</td>
</tr>
<tr>
<td></td>
<td>491</td>
</tr>
<tr>
<td></td>
<td>281</td>
</tr>
<tr>
<td>DDI-Full (Modified)†</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>925</td>
</tr>
<tr>
<td></td>
<td>688</td>
</tr>
<tr>
<td></td>
<td>501</td>
</tr>
<tr>
<td></td>
<td>348</td>
</tr>
<tr>
<td>HP GS1280</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td>4/1</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
</tr>
<tr>
<td></td>
<td>16/4</td>
</tr>
<tr>
<td>DDI-Full</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>1550</td>
</tr>
<tr>
<td></td>
<td>757</td>
</tr>
<tr>
<td></td>
<td>387</td>
</tr>
<tr>
<td></td>
<td>239</td>
</tr>
<tr>
<td>DDI-Fast</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>1847</td>
</tr>
<tr>
<td></td>
<td>1407</td>
</tr>
<tr>
<td></td>
<td>1166</td>
</tr>
<tr>
<td></td>
<td>458</td>
</tr>
<tr>
<td>DDI-Socket</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>2196</td>
</tr>
<tr>
<td></td>
<td>1796</td>
</tr>
<tr>
<td></td>
<td>1058</td>
</tr>
<tr>
<td></td>
<td>732</td>
</tr>
<tr>
<td>IBM Power3</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td>4/1</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
</tr>
<tr>
<td></td>
<td>16/4</td>
</tr>
<tr>
<td>DDI-Full</td>
<td>7860</td>
</tr>
<tr>
<td></td>
<td>4033</td>
</tr>
<tr>
<td></td>
<td>2059</td>
</tr>
<tr>
<td></td>
<td>1161</td>
</tr>
<tr>
<td></td>
<td>635</td>
</tr>
<tr>
<td>DDI-Fast</td>
<td>7933</td>
</tr>
<tr>
<td></td>
<td>4145</td>
</tr>
<tr>
<td></td>
<td>2180</td>
</tr>
<tr>
<td></td>
<td>1238</td>
</tr>
<tr>
<td></td>
<td>707</td>
</tr>
<tr>
<td>DDI-Socket</td>
<td>8670</td>
</tr>
<tr>
<td></td>
<td>4434</td>
</tr>
<tr>
<td></td>
<td>2353</td>
</tr>
<tr>
<td></td>
<td>1351</td>
</tr>
<tr>
<td></td>
<td>774</td>
</tr>
<tr>
<td>Compaq</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>2/1</td>
</tr>
<tr>
<td></td>
<td>4/1</td>
</tr>
<tr>
<td></td>
<td>8/2</td>
</tr>
<tr>
<td></td>
<td>16/4</td>
</tr>
<tr>
<td>DDI-Fast/Full</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>4650</td>
</tr>
<tr>
<td></td>
<td>2521</td>
</tr>
<tr>
<td></td>
<td>1352</td>
</tr>
<tr>
<td></td>
<td>739</td>
</tr>
<tr>
<td>DDI-Socket</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>4338</td>
</tr>
<tr>
<td></td>
<td>2365</td>
</tr>
<tr>
<td></td>
<td>1393</td>
</tr>
<tr>
<td></td>
<td>688</td>
</tr>
</tbody>
</table>

† Specially modified version of DDI-Full to show effects of coalescing inter-node messages and data transfers. See text for details.
measured for DDI-Full over a range of 1-8 nodes (4-32 processors), whereas DDI-Fast achieves only 10-15% improvement.

To ascertain the speedup associated with each of the above factors we modified DDI-Full so that all intranode distributed-data operations are performed via shared-memory, while all remote data operations use separate socket connections for each remote data sever. The timing difference between this “modified” DDI-Full and DDI-socket is due to the use of shared-memory for all intra-node transfers, while that with DDI-Fast is due to the performance gained from using shared-memory for all intra-node shared-memory segments rather than just one. Meanwhile the difference between DDI-Full (Modified) and the original DDI-Full is due to both a reduction in inter-node communications (from one message per remote data server to just one) and coalescing of the associated data transfers into one long message instead of multiple shorter ones.

Timing results on the HP SC for DDI-Full (Modified) are also given in Table 5.3. On 2 nodes and 8 CPUs DDI-Full outperforms DDI-Sockets by 205s. Half of this performance difference (104s) is accounted for by moving to DDI-Fast and allowing fast transfers to the local DDI sever. Comparing DDI-Fast with DDI-Full (Modified) shows that permitting fast transfers to all DDI servers on the same SMP node gives a further 71s improvement, while coalescing intra-node messages and data transfers gives rise to a 30s performance gain. Moving to 8 nodes and 32 processors changes this picture somewhat. Now the difference between DDI-socket and DDI-Full is 155s; 68s of this difference is due to fast local data transfers, 6s to using fast data transfers to all shared memory segments on the same SMP node and 81s due to the coalescing of inter-node messages and data transfers. This result is probably not surprising, as reducing inter node data traffic will naturally become more important as the number of nodes being used is increased.

Examining the benefit of DDI-Full over DDI-socket on different platforms, we find improvements of 9, 9 and 12% on 1, 2 and 4 processors of the IBM 375 MHz Power3 system compared to 16, 16 and 27% on the HP SC. The larger percentage improvement on the SC is due to its significantly better CPU performance. This emphasizes the benefits that arise
from communication improvements. This effect is even more apparent on the HP GS system where the performance improvements are 29, 58, 63 and 67% on 2, 4, 8 and 16 processors respectively. The HP GS has slightly faster processors than the SC, but considerably better memory bandwidth (12.6 GB/s per CPU compared to 1.6 GB/s per CPU) and also lower memory latency.

In some respects it can be argued that the FULL SMP data server model reduces to the SHMEM model on single node SMP systems. That is, with one SMP node the data servers in DDI-Full are essentially idle, consuming clock cycles during DDI initialization but soon after becoming dormant and remaining so for the rest of the calculation. In future implementations, we plan to remove the data servers entirely for single node parallel tasks. It is therefore somewhat surprising to find that the performance of DDI-SHMEM on 2 CPUs of the HP SC is significantly worse than any of the other DDI implementations. Furthermore, this difference becomes greater on 4 CPUs and 1 node. We suspect that this difference is partly due to the SHMEM library accessing the network interface card even though all SHMEM transfers occur solely within an SMP node. Indeed it is also interesting to note that the performance of DDI-SHMEM only begins to approach that of DDIFull when using 32 CPUs across 8 nodes.

The one anomaly in Table 5.3 is the Compaq timings. This is the only cluster that is not an SMP system, and here the original DDI-socket code performs slightly better than the improved versions. At this writing it is not known whether this is a flaw in DDI-Fast/Full, or an anomaly that was caused by machine issues. This will be discussed in more detail at the conference.

5.5 Conclusions

In the paper we have presented the approach used by DDI to increase parallel efficiency on SMPs and clusters of SMPs for high-performance distributed data computations by maximizing the advantage of sharedmemory. Considerable reductions in the communication overhead have been achieved for many different hardware platforms and communication fabrics.

It is important to emphasize that the accomplishments reported here have broad impli-
cations. While the results are presented in context of computational chemistry using the GAMESS package with the Distributed Data Interface, the approach is easily extended to other distributed-memory models in all areas of computational science and engineering.

Acknowledgements

The authors thank Drs. Ricky Kendall and Dmitri Fedorov for their many helpful discussions throughout the course of this work.

Funding for this work was provided by the Australian Partnership for Advanced Computing (APAC), by the US Department of Energy (USDOE) via a SciDAC (Scientific Discovery Through Advanced Computing) grant administered by the Ames Laboratory, by the Australia-America Fulbright Association, and by a travel grant from the National Science Foundation (NSF).

Computational facilities were graciously provided by the APAC National Facility for the use of the HP SC and HP GS systems and the Scalable Computing Laboratory in the U.S. Dept. of Energy Ames Laboratory for the use of the IBM Power3 and Compaq clusters. The IBM Power3 cluster was provided by a Shared University Research grant to Iowa State University.
Bibliography


Hellwagner and Alexander Reinfeld, editors, SCI: Scalable Coherent Interface,
volume 1291 of Lecture Notes in Computer Science, pages 95–116. Springer-
Verlag, 1999.

[12] Infiniband Architecture Specification Volume 1, Release 1.0. Infiniband Trade


294.
CHAPTER 6. A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO)

A paper published in the Journal of Computational Chemistry.¹

Dmitri G. Fedorov ², Ryan M. Olson ³, Kazuo Kitaura², Mark S. Gordon³, Shiro Koseki ⁴

Abstract

A two-level hierarchical scheme, generalized distributed data interface (GDDI), implemented into GAMESS is presented. Parallelization is accomplished first at the upper level by assigning computational tasks to groups. Then each group does parallelization at the lower level, by dividing its task into smaller work loads. The types of computations that can be used with this scheme are limited to those for which nearly independent tasks and subtasks can be assigned. Typical examples implemented, tested, and analyzed in this work are numeric derivatives and the fragment molecular orbital method (FMO) that is used to compute large molecules quantum mechanically by dividing them into fragments. Numeric derivatives can be used for algorithms based on them, such as geometry optimizations, saddle-point searches, frequency analyses, etc. This new hierarchical scheme is found to be a flexible tool easily utilizing network topology and delivering excellent performance even on slow networks. In one of the

¹Reprinted from the Journal of Computation Chemistry, Vol. 25, "A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO)", p 872-880, Copyright 2004, with permission from Elsevier.
²National Institute of Advanced Industrial Science and Technology, 1-1-1 Umezono, Tsukuba, 305-6568 Ibaraki, Japan
³Ames Laboratory, US-DOE and Department of Chemistry, Iowa State University, Ames, IA 50011
⁴Department of Material Science, Osaka Prefecture University, 1-1 Gakuen-cho, Sakai, 599-8531 Osaka, Japan
typical tests, on 16 nodes the scalability of GDDI is 1.7 times better than that of the standard parallelization scheme DDI and on 128 nodes GDDI is 93 times faster than DDI (on a multihub Fast Ethernet network). FMO delivered scalability of 80-90% on 128 nodes, depending on the molecular system (water clusters and a protein). A numerical gradient calculation for a water cluster achieved a scalability of 70% on 128 nodes. It is expected that GDDI will become a preferred tool on massively parallel computers for appropriate computational tasks.

6.1 Introduction

Application of parallel programming has turned out to be a boon to quantum chemistry, greatly broadening the areas in which it can be practically applied. The early development of a distributed data approach for performing parallel electronic structure theory calculations with the GAMESS [1] (General Atomic and Molecular Electronic Structure System) code is based on the distributed data interface (DDI) developed by Fletcher et al. [2], initially for second-order perturbation theory energies and gradients. More recent applications of DDI include the work of Umeda et al. [3] to parallelize multiconfigurational quasi-degenerate perturbation theory (MCQDPT) [4], the distributed data SCF by Alexeev et al. [5], full configuration interaction (FCI) [6], multiconfigurational self-consistent field (MCSCF) [7], and perturbation theory energies and gradients for open shells [8]. Although the DDI interface remains a general tool applicable to any type of calculations GAMESS can perform, its efficiency is most useful for highly correlated calculations using large basis sets on hardware with very fast CPU interconnects. For the less efficient communication fabrics, such as Fast and Gigabit Ethernet, that are common on clusters, the efficiency of the current DDI implementation decreases.

In this work we propose a general hierarchical type of interface, presently with two levels, that works by dividing all nodes into groups. The two levels correspond to intergroup processing (higher level, coarse-grained parallel, when a group operates as one unit) and intragroup (lower level, finer grained when each group member constitutes one unit). This approach allows individual tasks to be assigned to groups that do them independently (or nearly so) of each other. This greatly increases efficiency, as the scalability at the group level is nearly
linear. This new interface can be applied to those types of computations where it is feasible to divide the work into smaller units. A commonly used parallel library MPI [9] also has the capability to divide nodes into groups; however, it does not provide the means to divide the work load between groups dynamically. That is, MPI is also a hierarchical approach but is static in nature. At present, two types of calculations can be performed with generalized DDI (GDDI): numerical derivatives and the fragment molecular orbital method (FMO) developed by Kitaura et al. [10], [11]. An implementation of the FMO method based on the MPI static two-level hierarchy is also available in the ABINIT-MP program [12]. FMO is a powerful method enabling quantum-mechanical calculations of large molecules, and it has been applied to molecules containing up to 4000 atoms [13]. However, the ability to handle such large systems incurs high computational costs and requires massively parallel computers and programs utilizing them with high efficiency. The hierarchical scheme proposed in this work may be easily applied to the FMO method.

Numeric derivatives, while potentially subject to numerical inaccuracy, remain an important tool in quantum chemistry, as it is commonplace to develop energy functions significantly ahead of the implementation of analytic gradients. Especially for the most sophisticated computational methods, the derivation and coding of analytic derivatives is a challenge that can take years to accomplish. Therefore, the new GDDI implementation presented here provides an efficient way to improve scalability and to flexibly use the network topology. GDDI can also be used with clusters of computer clusters.

6.2 Methodology and Implementation

The basic idea is illustrated in Figure 6.1. The implementation is based upon a socket library written in C for UNIX systems; there is also a MPI version with slightly reduced capabilities. In the original DDI, a given task (e.g., energy plus gradient) is divided as efficiently as possible among all available processors. In GDDI, each group is assigned a similar independent task. The task assigned to each such group may then be treated as in DDI.

In parallel computations using MPI libraries, the concept of communicator is introduced.
A communicator is defined as “an MPI object that describes the communication context and an associated group of processes” [9]. In other words, a subset of all nodes is assigned to a group that is labeled by an integer index (a communicator). Thereafter, one can do global sums or broadcasts within the nodes of a group by specifying its communicator. When the job starts, there is only one communicator defined that includes all available nodes and more communicators can easily be created when needed.

The MPI version of GDDI was very straightforward to construct using the MPI_COMM_SPLIT subroutine, by creating three levels of communicators: WORLD, GROUP, and MASTER. The first of these includes all compute nodes, the second includes all compute nodes in a group, and the third includes all master compute nodes. Note that WORLD as described above is not the default MPI communicator MPI_COMM_WORLD, as all three communicators are limited to compute nodes; data servers are not included (see the original DDI reference for the concepts of compute and data server processes [2]). In addition, a communicator that includes both compute and data server nodes in a group is also created. The switching from one
communicator to another is accomplished by assigning one of the three communicator values obtained from calling MPI_COMM_SPLIT to a common block variable that is then used as the current communicator in all MPI calls. The current communicator environment (that is, WORLD, GROUP, or MASTER) is called the \textit{scope}.

For the UNIX socket-based library the underlying MPI structure is emulated by storing the node IDs into arrays used in all communications, in analogy with the MPI communicators. For sockets, some work is needed to switch scopes. This includes changing the way communications that are restricted to a given scope occur, whereas for MPI communications can, in principle, occur freely by simply using a different index (communicator). Either way, all nodes within a scope must perform the same parallel communication synchronously lest a deadlock should occur; thus, there is no practical difference.

Regardless of which library is used, the outer wrapper subroutines in the library-dependent source file DDI.SRC free the quantum-mechanical part of the program from knowing all of these unnecessary details. In fact, even most of DDI.SRC does not know if DDI or GDDI is used as the changes required to the parallel routines were minimal, excluding the core part that creates and switches communicators. It is possible and useful, as shown below, to change the number of groups on the fly, to accommodate the needs of a particular step of some calculation. The implementation of this is straightforward: a global wait to synchronize groups, followed by updating node arrays that contain the group division (sockets). A similar approach to create new communicators can be followed for MPI.

After all nodes switch to the group scope, parallel communications occur independently of each other. Each group has its own synchronization points, global sums distributed memory operations, etc. This effectively localizes communications and greatly speeds up calculations on massively parallel computers. Although the original MPI provides group division as well, it is a very basic and static interface, which also localizes communications within groups. Furthermore, the available distributed memory implementation in GAMESS cannot use the original MPI group-divided nodes, whereas GDDI supports distributed memory operations localized within groups.
The basic usage of the three scopes is as follows. The program begins by running in the WORLD scope, then each group is assigned a task and the scope is switched to GROUP. After the group finishes its task, it gets another one until all tasks are completed. Next, the scope is switched either to MASTER or WORLD, and the nodes in the new scope exchange results (the choice between the two depends on the type of computation), accomplished by global sums and broadcasts. Finally, the scope is switched back to WORLD, so the whole computation may be repeated (e.g., during a geometry optimization based upon numeric gradients).

The load balancing can be chosen to be either static or dynamic. The former is a simple fixed division of work according to the work index, whereas the latter is a flexible scheme in which the computational units request the next work index upon completion of the current work. Load balancing is done at all hierarchical levels; that is, at the intergroup and intragroup levels. In the former case, the grand master keeps track of the job indices given out to groups, while in the latter case, it is the group master that does the same within its group.

It is important to stress that dynamic load balancing has overwhelming importance for parallel computations, unless only a few nodes are used. In practice, clusters can be expected to be heterogeneous, with nodes having varying capabilities. This means some nodes will finish their tasks ahead of others. Even for clusters with identical nodes, the mathematical nature of the tasks strongly affects the computational time they require, even if tasks are formally of the same size (due to, for instance, integral screening).

In the present work, a modified version of GAMESS was used in which the option to store two-electron integrals in memory was added. In most of the systems considered below the amount of memory present was sufficient to store all two-electron integrals, greatly speeding up the calculations. There was only one system (C, see below) where 0-20% (depending on the group size) of the total number of two-electron integrals were stored on disk. With this exception the amount of disk I/O was really small and limited mostly to storing one-electron integrals and the monomer densities for all fragments.

General first and second numeric derivative codes were added to GAMESS, both based on single-point energy runs with double differencing. These codes can be used to do geometry
optimizations, saddle point searches, and Hessians.

All applications below are of the restricted Hartree-Fock (RHF) type. Although GDDI can be used for any type of wave function that GAMESS supports, the parallel strategy that is making the decision on how to divide nodes into groups and how to set up other options, depends upon the wave function type. Some electron correlation methods, such as MP2, have substantial memory requirements that are divided among nodes in a group using the DDI interface. This places a constraint on the minimum number of nodes per group that is generally not an issue for RHF, for instance. In addition, parallel communication overhead is often larger if electron correlation is included. Nevertheless, general trends that are easily illuminated with RHF runs are applicable to other wave function types. Because RHF does not use distributed arrays, parallelization is based on manual memory division between nodes, using global sums and broadcast routines not involving DDI specific operations, such as data distribution.

The applications discussed below use group sizes that are as close to uniform as possible, although nonuniform distributions could conceivably be more efficient in some cases. It is hard or perhaps impossible to suggest a general rule of granulating group sizes to achieve maximum performance. The actual network topology and the details of the system affect scalability in a complex way as elucidated below. It is likely that granulating group sizes may become more important issues if the number of available nodes grows beyond 1024.

Network topology can be very simply utilized with GDDI. It seems best to localize all group members within one hub (or its equivalent if a different type of network is used). Because hubs can have different numbers of nodes attached, it is often not possible to maintain a constant group size. Therefore, both automatic and manual node division have been implemented. In the former case, the user only specifies the constant group size, while in the latter case each node is manually assigned to a group. Due to global broadcasts, communication between nodes does take place and hinders performance to some extent. Finally, for SMP machines all of the CPUs on the same machine naturally should be put into the same group (possibly with addition of other nodes), as the communication within one node should be very fast.
Figure 6.2 One hundred twenty-eight node PC cluster with star topology, connected by Fast Ethernet. Each rectangular block denotes a 24-port Fast Ethernet hub. The numbers near brackets show the number of nodes connected to the particular hub. The reason the nodes are divided in this way is that the 128 node cluster used for the tests is part of a 400-node cluster, and some ports on the above hubs are used by other clusters.

6.3 Applications

The cluster used for the tests contained 128 nodes, each equipped with a Pentium III 1 GHz CPU, 512-MB RAM and running RedHat 7.1 Linux with the 2.4.9-31 kernel. The nodes were connected by Fast Ethernet, as shown in Figure 6.2. GAMESS was compiled with the 7.0 version of the Intel compiler [14].

All timings are wall-clock and do not include system startup time taken to run rsh processes and copy the input files to corresponding nodes (this can take several minutes on 128 nodes). As indicated below, the error bounds for the timings are estimated to be less than 1%, due to slight differences in runs using dynamic load balancing and system conditions.
Figure 6.3 The diagram for the GDDI parallelization of the FMO method. “init” stands for initialization, YLBn is the n-th level load balancing of Y = I initial guess, Y = M monomer SCF, Y = D dimer single point runs, n = 1 (intergroup) and 2 (intragroup). For all n = 2 parts the diagram is simplified not showing loops occurring during those steps (such as, e.g., SCF iterations). “W” is some work performed by one node (typically, some fraction of integrals). YXn is first (n = 1) and second (n = 2) level data exchange. “conv.” stands for the test of convergence of the monomer SCF. “prop.” stand for properties, currently computed (redundantly) by all group masters. The load-balancing part of the diagram shows the total load that is divided among available nodes, whose number is arbitrary.
6.3.1 Fragment Molecular Orbital Method

To discuss parallelization issues, the main steps of the FMO method are briefly introduced here. A molecule (or a molecular cluster) is divided into fragments (also called monomers). A single-point RHF calculation is then performed on each fragment separately in the mean Coulomb field (electrostatic potentials, ESPs) of the other monomers. Each single-point is performed by one DDI group, nearly independently of the others. One cycle of single-point monomer SCF runs generates a new density, thus changing the Coulomb field, so these cycles are repeated until convergence is reached. This generates the monomer densities that are then used during the next step, namely, the dimer single-point SCF runs, also done independently by DDI groups. During this second step dimers are constructed from each pair of monomers, and the initial density is taken to be the sum of the two monomer densities. Then, an SCF calculation is performed once for each dimer. If two monomers are well separated, an approximation is used that essentially ignores exchange and self-consistency, and only the Coulomb interaction contribution is computed. All these steps are illustrated in Figure 6.3.

The computation of the Coulomb interaction requires one- and two-electron integrals involving both the \(n\)-mer \((n = 1, 2)\) and the external monomers, one at a time (that is, the total entity is a \((n + 1)\)-mer). The calculation of ESPs is a very time-consuming step, and it takes on the order of half of the total time. Several approximations have been developed for well-separated monomers; thus, in practice, most ESPs due to such separated monomers can be computed practically without any loss of accuracy by using point charges, reducing the costs from two-electron to effectively one-electron. Finally, note that breaking bonds in a molecule when defining fragments necessitates the computation of one-electron projection operator contributions. They are parallelized in a similar manner to other one-electron integrals, and require little time to compute; thus, they are not mentioned in the discussion below.

Returning to Figure 6.3, after initialization, the preparation of starting orbitals begins by a load-balancing scheme (ILB1) that assigns one fragment to one DDI group. As noted above, load balancing can be static or dynamic. Next, a group divides the work for one fragment via the ILB2 step. This includes computation of one-electron integrals, performed by each node.
(denoted by “W” in Fig. 3). After all work at the intragroup level is finished, data exchange (IX2: global sums of one-electron integrals) occurs. Finally, after all monomers have been treated, data exchange (IX1: exchange of monomer densities) is done. There is other minor communication (indexing arrays, interfragment distances, etc.), part of which involves only exchange between masters.

The other two steps, monomer SCF and single point dimer SCF runs, have exactly the same structure, with some differences in the type of work done (e.g., the addition of two-electron integrals, ESPs are computed once before each SCF). MX2 and DX2 thus involve global sums of ESPs, one-electron integrals, and Fock matrices; MX1 involves density and energy exchange (global convergence is based on energy). DX1 only involves dimer energy exchange. Finally, properties are computed (sequentially).

Now consider the general trends in this scheme, followed by timings for actual computations. Regarding the strategy for dividing nodes into groups, the following points are important: (a) the smaller the group size, the faster the intragroup runs (the more groups the better); (b) the larger the number of groups, the longer the synchronization wait at the exchange points IX1 and MX1. As explained below, DX1 is less affected (fewer groups is better); (c) the fewer the number of groups, the faster the density exchange (especially on networks with large latency, such as Fast and Gigabit Ethernets) (fewer groups is better).

The last point arises from the fact that the density exchange works through a broadcast to all nodes by a group master of all densities its group computed. Although the total amount of data is independent of the number of groups (it is equal to the sum of density sizes for all fragments, plus Mulliken charges and populations), due to latency it is faster to broadcast one large block compared to two smaller ones with the same total size. The reason DX1 is less affected by point (b) above is as follows. The number of dimers is generally many times larger than the number of DDI groups, because the number of dimers is \( N(N-1)/2 \), where \( N \) is the number of monomers. Then, the work load balancing (see below) changes the order in which dimers are done in such a way that separated dimers (that are treated with an approximation and take little time) are done at the end, so that there is little wait at DX1.
The balance between points (a)-(c) is what determines the node division strategy, and both depend on the network type and topology.

To reduce waiting time at the synchronization points IX1, MX1, and DX1, load balancing (both static and dynamic) is implemented in the descending order of the computational work load. The latter is determined by: (a) \( n \)-mer basis set size \((n = 1, 2)\), (b) usage of dynamic correlation, (c) applicability of the separated dimer approximation, and (d) in case of the initial guess, the lower triangular matrix of interfragment distances is computed at the same time and it usually requires more time than the Hückel guess; therefore, the lower part of the triangular matrix takes more time (one row is computed for each fragment).

It is easy to take all these points into account and assign a work load weight to each \( n \)-mer. By doing the work in descending order, the wait is reduced due to faster final runs before the synchronization.

Parallel performance can be fine-tuned (with input file options) as follows: (a) division of nodes into groups (implemented independently for each of the three major steps; group division is changed on the fly); (b) the choice of static/dynamic load balancing at each level (inter- and intragroup); (c) the choice of the SCF type, which is direct or conventional (for the sake of sensible analysis we used conventional SCF during all tests; however, for large group sizes direct SCF scales better and should be considered for practical applications); and (d) other fine tuning, such as whether to divide the MLB2, DLB2 work over fragments or shells during ESP calculations.

Although we find that dynamic load balancing definitely wins in general at the intergroup level, static load balancing can be faster at the intragroup level, especially on networks with large latency, due to significant communication to obtain the job index. Small basis sets seem to prefer static load balancing at the intragroup level. The choice of fragment or shell ESP parallelization is determined by the ratio of the number of fragments that are computed without approximations to the number of group members. If this ratio is large (roughly = 5-10), then the fragment option is faster, due to reduced communication.

The following systems were used for parallel tests: (A) \((\text{H}_2\text{O})_{256}\) FMO-RHF/6-31G*, di-
Table 6.1 The division of \( N_{nod} \) nodes among \( N_{gr} \) groups, the number of groups \( M_k \) is given for the group consisting of \( k \) nodes.

<table>
<thead>
<tr>
<th>( N_{nod} )</th>
<th>( N_{gr} )</th>
<th>( M_2 )</th>
<th>( M_3 )</th>
<th>( M_4 )</th>
<th>( M_5 )</th>
<th>( M_6 )</th>
<th>( M_7 )</th>
<th>( M_8 )</th>
<th>( M_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>17</td>
<td>4</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>17</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>28</td>
<td>4</td>
<td>4</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>34</td>
<td>8</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>45</td>
<td>7</td>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

vided into 64 fragments; (B) \((\text{H}_2\text{O})_{1024}\) FMO-RHF/STO-3G, divided into 128 fragments; and (C) lysozyme tri\([N\text{-acetyl-}\text{D-glucosamine}]\) complex, 2036 atoms, FMO-RHF/STO-3G, divided into 128 fragments.

All systems were run with conventional RHF and tightened accuracy (SCF convergence \(10^{-7}\), monomer SCF convergence \(10^{-7}\), and two-electron integral accuracy \(10^{-12}\)). Hückel orbitals were used for the initial guess and Cartesian d functions (6d option) were used (ISPHER = -1). The fragment size was uniform for tests A and B, and varied (one residue per fragment) for test C. Due to cluster topology, it was not possible to use group division with uniform size for 64 nodes and 128 nodes (it is not necessarily desired in any case). Therefore, the manual group division option was used with varying group size given in detail in Table 6.1. All nodes within a group were always connected to the same hub.

For each system, several sensible options (such as altering the node division among groups while fixing the total number of nodes) were considered. The fastest option is presented below. The general trends are explained for each system, and options used on 128 nodes are clarified. The superlinear scaling that is observed in some cases (verified on different clusters) must come from either CPU or system-level caching that was not explicitly controlled.

Initialization and initial guess are included in the monomer timings and scalabilities. The initial guess takes 7, 17, and 16 s on one node for the systems A, B, and C, respectively, so it is unnecessary to discuss its scalability separately. On 128 nodes the timings are, respectively, 15, 20, and 20 s. This is despite the gain from dividing work among nodes. The data exchange at IX1 makes the overall timing worse on 128 nodes compared to 1 on Fast Ethernet. For system
A, there is some loss of efficiency due to the single point initial guess scalability, because of larger group size.

Summarizing general trends prior to considering the details, in all cases the scalability of monomer SCF is somewhat low. This is explained by large communication to exchange density between nodes and the long wait at the synchronization point MX1. In contrast, the dimer efficiency is high. This is because there is little communication cost and smaller group size. Using fewer nodes per group during monomer runs reduces the efficiency due to the increased wait that overcomes the efficiency boost because of better intragroup scaling. The necessity to strive to form very small groups, thereby increasing the wait at MX1, is, however, the sole consequence of using a slow network. It should also be noted that dimer runs can also be saturated, meaning the group size has to be decreased if the ratio of the number of dimers to the number of groups is not large enough (as observed in case A on 128 nodes). In reality, it is not just a simple ratio but a more complicated relation involving the number of SCF and separated dimer calculations.

Table 6.2 provides a detailed analysis of the parallelization costs for the monomer SCF. Comparing the results for two nodes, note that system C has shorter waiting times than A and B. This is because system C fragments have varying sizes that are processed in decreasing order. So, the wait time is small because small fragments are done last. It appears to be typical to have such “coarse-grained” performance on a few nodes, when \( N_{nod} - 1 \) nodes wait for one node to finish. Wait times for system C are consistently shorter than those for the other two systems (with one small exception). The fragment basis set size is larger for system A; therefore, one often observes longer wait times for the same number of groups, compared to systems B and C.

It is interesting to notice that in almost all cases the wait time exceeds the exchange time, except for 64 and 128 nodes. No doubt the exchange time could improve with an improved network topology, for example, with more direct access between nodes that are now localized at different hubs. It can be computed from data presented below for individual systems that the total direct parallel communication on 128 nodes at the MX1 point consumes 38, 28, and 33%
Table 6.2 Analysis of the Parallelization Costs: Dependence of the Average Wait and Data Exchange Time in Seconds at the MX1 Step, for One Monomer SCF Iteration (the Total Time for All Iterations Is Shown in Parentheses).

<table>
<thead>
<tr>
<th>$N_{nod}$</th>
<th>$N_{gr,m}$</th>
<th>$N_{D}$</th>
<th>$N_{B}$</th>
<th>$N_{gr,m}$</th>
<th>$N_{D}$</th>
<th>$N_{B}$</th>
<th>$N_{gr,m}$</th>
<th>$N_{D}$</th>
<th>$N_{B}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.0 (0.)</td>
<td>0.0 (0.)</td>
<td>1</td>
<td>0.0 (0.)</td>
<td>0.2 (1.)</td>
<td>1</td>
<td>0.0 (0.)</td>
<td>0.0 (0.)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4.2 (33.)</td>
<td>0.4 (3.)</td>
<td>2</td>
<td>7.7 (46.)</td>
<td>0.0 (0.)</td>
<td>2</td>
<td>0.2 (4.)</td>
<td>0.1 (2.)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5.5 (44.)</td>
<td>0.5 (4.)</td>
<td>4</td>
<td>18.3 (110.)</td>
<td>0.3 (2.)</td>
<td>4</td>
<td>4.0 (68.)</td>
<td>0.5 (9.)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>13.8 (110.)</td>
<td>0.7 (5.)</td>
<td>8</td>
<td>15.4 (92.)</td>
<td>0.6 (4.)</td>
<td>8</td>
<td>2.1 (35.)</td>
<td>0.6 (11.)</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>5.7 (45.)</td>
<td>1.1 (8.)</td>
<td>16</td>
<td>8.5 (51.)</td>
<td>0.9 (6.)</td>
<td>16</td>
<td>4.2 (71.)</td>
<td>0.8 (13.)</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>14.0 (112.)</td>
<td>3.3 (26.)</td>
<td>16</td>
<td>8.1 (49.)</td>
<td>3.9 (23.)</td>
<td>16</td>
<td>2.0 (33.)</td>
<td>4.1 (69.)</td>
</tr>
<tr>
<td>64</td>
<td>17</td>
<td>3.9 (31.)</td>
<td>5.3 (42.)</td>
<td>17</td>
<td>6.4 (38.)</td>
<td>6.2 (37.)</td>
<td>17</td>
<td>1.6 (27.)</td>
<td>5.9 (100.)</td>
</tr>
<tr>
<td>128</td>
<td>17</td>
<td>2.7 (22.)</td>
<td>9.6 (77.)</td>
<td>28</td>
<td>4.5 (27.)</td>
<td>12.4 (74.)</td>
<td>34</td>
<td>3.3 (56.)</td>
<td>14.1 (239.)</td>
</tr>
</tbody>
</table>

Systems A, B, and C are defined in the text. $N_{nod}$ is the total number of nodes and $N_{gr,m}$ is the number of groups during the monomer SCF. $N_{D}$ is the amount of data transferred at each iteration, in words (1 word = 8 bytes). $N_{B}$ is the fragment basis set size.
of the monomer SCF wall time for systems A, B, and C, respectively. Without this overhead, the monomer SCF scalability would have been 82, 95, and 94%, respectively. The scaling is less for system A due to the larger group size. Other parallel communication occurs at the MX2 point, within single-point SCF calculations. In contrast, parallel communication for dimers is nearly negligible, if the group size of 1 is a sensible option and is usually quite small otherwise.

Note that the number of fragments into which a given molecule is divided has a dramatic effect on scalability. This number is usually determined from physical rather than computational considerations, as increasing the number of fragments decreases the accuracy of the FMO method. It is both customary and physically reasonable to cut proteins so that amino acid residues are not cut internally (e.g., by assigning one residue per fragment). For molecular clusters it is natural to assign a fixed number of molecules per fragment.

The scalability results for system A are given in Table 6.3. The scalabilities $S_A$ in Table 6.3 and subsequent tables refer to the wall time ratio $S_A(n) = t_A(1)/(n \times t_A(n))$, where $n$ is the number of nodes and A is either “mon” or “tot”, for monomer SCF and total, respectively, so that ideal linear scaling is equal to one. Parallel efficiency remains high up to eight nodes. Thereafter, the longer wait at MX1 forces the group size to be set to 2, 2, 4, and 8 for the monomer SCF calculation, on 16, 32, 64, and 128 nodes, respectively. On the other hand, the group size was fixed at 1 for the dimers, except for 128 nodes where the group size was 3. Shell parallelization is preferred over fragment parallelization, due to the small number of fragments that are computed without approximations. The low efficiency of monomer SCF calculations at 128 nodes comes from the low efficiency of single point SCF runs (including ESPs) due to the slow network. As can be seen by subtracting the monomer SCF part, the scalability of the rest of the code, even on 128 nodes, is 0.9252. This is actually a bit low, because groups with average size of 3 were used during the dimer runs, as explained above.

The efficiency of GDDI vs. DDI is presented in Table 6.4. DDI corresponds to having just one GDDI group, and for all cases both static and dynamic load balancing was tried and the fastest timing is shown (for large groups dynamic load balancing within a group somewhat suffers from high latency of FastEthernet). GDDI outperforms DDI in all cases. In the best
Table 6.3 Scalability Tests for (H₂O)₂₅₆ FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

<table>
<thead>
<tr>
<th>N_{nod}</th>
<th>N_{gr,m}</th>
<th>P_{ESP}</th>
<th>t_{mon}, s</th>
<th>S_{mon}</th>
<th>t_{tot}, s</th>
<th>S_{tot}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>14602.2</td>
<td>1</td>
<td>87818.9</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
<td>7225.0</td>
<td>1.0105</td>
<td>43493.1</td>
<td>1.0096</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-</td>
<td>3673.0</td>
<td>0.9939</td>
<td>21760.6</td>
<td>1.0089</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>-</td>
<td>1912.3</td>
<td>0.9545</td>
<td>10959.4</td>
<td>1.0016</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>S</td>
<td>1023.4</td>
<td>0.8918</td>
<td>5575.0</td>
<td>0.9845</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>S</td>
<td>596.5</td>
<td>0.7649</td>
<td>2885.8</td>
<td>0.9509</td>
</tr>
<tr>
<td>64</td>
<td>17</td>
<td>S</td>
<td>354.6</td>
<td>0.6434</td>
<td>1544.3</td>
<td>0.8885</td>
</tr>
<tr>
<td>128</td>
<td>17</td>
<td>S</td>
<td>258.6</td>
<td>0.4411</td>
<td>876.8</td>
<td>0.7825</td>
</tr>
</tbody>
</table>

\(N_{nod}\) is the number of nodes, equal to \(N_{gr,d}\), the number of groups during the dimer runs, \(N_{gr,m}\) is the number of groups during the monomer SCF. The load balancing was dynamic at the ILB1, MLB1, DLB1, and static at the ILB2, MLB2, and DLB2 levels. \(P_{ESP}\) stands for parallelization of ESPs in MLB2 and DLB2: “-” (none: due to one node in a group), “S” (over shells), and “F” (over fragments). \(t_{mon}\) and \(t_{tot}\) are the wall clock monomer SCF\(^a\) and total times, respectively. \(S_{mon}\) and \(S_{tot}\) are the monomer SCF and total scalabilities, respectively. See caption to Figure 6.3 for the FMO step notation.

\(^a\)Initial guess is included into monomer timings throughout all results.

\(^b\)The number of dimer groups was set to 45.
Table 6.4 Comparison of DDI vs. GDDI Scalability Tests for (H₂O)256 FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster (DDI is the First Row in the Table, N_{gr} = 1)

<table>
<thead>
<tr>
<th>N_{nod}</th>
<th>N_{gr,m}</th>
<th>LB</th>
<th>t_{mon}, s</th>
<th>S_{mon}</th>
<th>t_{tot}, s</th>
<th>S_{tot}</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>D</td>
<td>1516.8</td>
<td>0.6017</td>
<td>9616.6</td>
<td>0.5707</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>D</td>
<td>1177.6</td>
<td>0.7750</td>
<td>7315.2</td>
<td>0.7503</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>S</td>
<td>1055.4</td>
<td>0.8647</td>
<td>6218.9</td>
<td>0.8826</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>D</td>
<td>1009.9</td>
<td>0.9037</td>
<td>5777.9</td>
<td>0.9499</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td></td>
<td>1066.9</td>
<td>0.8554</td>
<td>5614.7</td>
<td>0.9776</td>
</tr>
</tbody>
</table>

N_{gr} is the number of groups, the same during both monomer and dimer steps. The number of nodes N_{nod} is fixed and equal to 16, shown to avoid confusion. LB denotes load balancing within each group (S for static, D for dynamic and none for a group with one node). See caption to Table 6.3 for other notation.

Table 6.5 Comparison of DDI vs. GDDI Scalability Tests for (H₂O)256 FMO-RHF/6-31G*, Evenly Divided into 64 Fragments, On the 7-Hub Star-Topology Fast Ethernet PC Cluster (DDI is the First Row in the Table, N_{gr} = 1)

<table>
<thead>
<tr>
<th>N_{nod}</th>
<th>N_{gr}</th>
<th>t_{iter}, s</th>
<th>S_{iter}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1892.5</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>2963.8</td>
<td>0.0049</td>
</tr>
<tr>
<td>128</td>
<td>17</td>
<td>32.0</td>
<td>0.4620</td>
</tr>
</tbody>
</table>

t_{iter} and S_{iter} are timings and scalability of the first monomer SCF iteration (S_{iter} is a very good estimate of S_{mon}).

See caption of Table 6.4 for other notation.
Table 6.6 Scalability Tests for \((\text{H}_2\text{O})_{1024}\) FMO-RHF/STO-3G, Evenly Divided into 128 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

<table>
<thead>
<tr>
<th>(N_{nod})</th>
<th>(N_{gr,m})</th>
<th>(P_{ESP})</th>
<th>(t_{mon}, s)</th>
<th>(S_{mon})</th>
<th>(t_{tot}, s)</th>
<th>(S_{tot})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>25,945.6</td>
<td>1</td>
<td>201,819.5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
<td>13,249.9</td>
<td>0.9791</td>
<td>102,192.0</td>
<td>0.9875</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-</td>
<td>6603.5</td>
<td>0.9823</td>
<td>50,738.6</td>
<td>0.9944</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>-</td>
<td>3351.0</td>
<td>0.9678</td>
<td>25,489.9</td>
<td>0.9897</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>-</td>
<td>1753.0</td>
<td>0.9250</td>
<td>12,821.1</td>
<td>0.9838</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>F</td>
<td>955.5</td>
<td>0.8486</td>
<td>6494.7</td>
<td>0.9711</td>
</tr>
<tr>
<td>64</td>
<td>17</td>
<td>S</td>
<td>542.1</td>
<td>0.7478</td>
<td>3312.6</td>
<td>0.9520</td>
</tr>
<tr>
<td>128</td>
<td>28</td>
<td>S</td>
<td>355.2</td>
<td>0.5707</td>
<td>1742.7</td>
<td>0.9048</td>
</tr>
</tbody>
</table>

See caption to Table 6.3 for notation.

set of options (given in Table 6.3, for 16 nodes), the total scalability with GDDI is 0.9845, whereas the DDI scalability is only 0.5707. It is instructive to observe the drop in monomer parallel efficiency when going from 8 to 16 groups. This is due to the increased wait at MX1, as discussed above. This difference becomes dramatic when the number of nodes increases further, as shown in Table 6.5. *For 128 nodes the scalability of the standard DDI becomes disastrous*: it is faster to run on 1 node than on 128, and the scalability of DDI is merely 0.0049. GDDI, on the other hand, shows scalability of 0.4620; not too high, but acceptable. The timing of GDDI is 93 times faster than DDI at the same number of nodes (128) and the same input, other than the group division. Of course, in practice, one would only grow the number of nodes under DDI in accord with the efficiency that hardware and the computational methods provide. Nonetheless, these examples emphasize the great utility of GDDI, even when DDI is not very useful.

Because dynamic load balancing is used throughout, small changes in timings are observed with every run. As can be seen, the \(t_{mon}\) timings for 16 nodes, eight groups in Table 6.3 and Table 6.4 differ by less than 1%. This is a typical error bound for the timings (note that \(t_{tot}\) for the aforementioned entries differs because the dimers are run with 16 and 8 groups, respectively).

The results for system B are summarized in Table 6.6. The trends are similar to those
Table 6.7 Scalability Tests for Lyso (2036 Atoms) FMO-RHF/STO-3G, Divided into 128 Fragments, on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

<table>
<thead>
<tr>
<th>$N_{nod}$</th>
<th>$N_{gr,m}$</th>
<th>$P_{ESP}$</th>
<th>$t_{mon},s$</th>
<th>$S_{mon}$</th>
<th>$t_{tot},s$</th>
<th>$S_{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>69,264.6</td>
<td>1</td>
<td>229,925.1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-</td>
<td>34,792.3</td>
<td>0.9954</td>
<td>115,430.4</td>
<td>0.9959</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-</td>
<td>17,438.7</td>
<td>0.9930</td>
<td>57,684.5</td>
<td>0.9965</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>-</td>
<td>8802.6</td>
<td>0.9836</td>
<td>28,958.7</td>
<td>0.9925</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>-</td>
<td>4434.6</td>
<td>0.9762</td>
<td>14,484.5</td>
<td>0.9921</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>S</td>
<td>2348.2</td>
<td>0.9218</td>
<td>7385.0</td>
<td>0.9729</td>
</tr>
<tr>
<td>64</td>
<td>17</td>
<td>S</td>
<td>1297.9</td>
<td>0.8339</td>
<td>3815.0</td>
<td>0.9417</td>
</tr>
<tr>
<td>128</td>
<td>34</td>
<td>S</td>
<td>893.4</td>
<td>0.6057</td>
<td>2154.9</td>
<td>0.8336</td>
</tr>
</tbody>
</table>

See caption to Table 6.3 for notation.

observed for system A. Due to a larger number of fragments (128 vs. 64), one can increase the group size to 16 nodes. Similarly, at 128 nodes the average group size is 4.5, increasing the scalability to 0.5707. Likewise, fragment parallelization of ESPs is slightly (by 21 s, not shown) preferred to shell parallelization, due to the increased number of fragments. Shell parallelization is preferred for larger group sizes. The total scalability on 128 nodes without including the monomer SCF calculation is 0.9902. The total scalability for system B is better than that for system A. In part, this is due to the decreased fraction of time the monomer SCF calculation takes compared to the total time (13 vs. 17%), because it takes fewer monomer SCF iterations to converge (6 vs. 8).

Table 6.7 summarizes the results for system C. The node division strategy closely resembles that for system B, because the two systems are similar in size and in the number of fragments. Shell parallelization is faster than fragment parallelization on 32 nodes, by 105 s. This may be due in part to varying fragment size (load balancing according to size is not implemented at present inside of ESPs) and possibly to the smaller number of fragments treated without approximation because of the larger spatial dimensions of the molecule. The overall scalability for C is lower than that for B, due to the increased fraction of the monomer SCF calculation in the total time: 30 vs. 13%. This reflects, at least in part, the 17 iterations it took to converge. The total scalability at 128 nodes excluding the monomer SCF times is 99.5%. It is typical to
Table 6.8 The Total Scalability ($S_{tot}$) Dependence upon Load Balancing (LB) Type at the Upper (Intergroup) Level: Dynamic (D), Static (S) with the Possible Addition of Job Reordering According to the Subtask Size (Indicated by + Where Used).

<table>
<thead>
<tr>
<th>$N_{nod}$</th>
<th>LB</th>
<th>$A, S_{tot}$</th>
<th>$B, S_{tot}$</th>
<th>$C, S_{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D+</td>
<td>1.0016</td>
<td>0.9897</td>
<td></td>
<td>0.9925</td>
</tr>
<tr>
<td>8</td>
<td>S+</td>
<td>0.9534</td>
<td>0.9753</td>
<td>0.9552</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>0.9445</td>
<td>0.9688</td>
<td>0.8790</td>
</tr>
<tr>
<td>D+</td>
<td>0.9845</td>
<td>0.9838</td>
<td></td>
<td>0.9921</td>
</tr>
<tr>
<td>16</td>
<td>S+</td>
<td>0.9307</td>
<td>0.9548</td>
<td>0.9010</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>0.8548</td>
<td>0.9198</td>
<td>0.7074</td>
</tr>
<tr>
<td>D+</td>
<td>0.9510</td>
<td>0.9711</td>
<td></td>
<td>0.9729</td>
</tr>
<tr>
<td>32</td>
<td>S+</td>
<td>0.8644</td>
<td>0.9373</td>
<td>0.8538</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>0.8184</td>
<td>0.8300</td>
<td>0.6793</td>
</tr>
<tr>
<td>D+</td>
<td>0.8885</td>
<td>0.9520</td>
<td></td>
<td>0.9417</td>
</tr>
<tr>
<td>64</td>
<td>S+</td>
<td>0.5240</td>
<td>0.8853</td>
<td>0.7771</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>0.6105</td>
<td>0.7336</td>
<td>0.5876</td>
</tr>
</tbody>
</table>

Molecular systems A, B, and C are described in the main text.

It takes 15-25 iterations to converge the monomer SCF calculation vs. 5-10 iterations to converge molecular clusters. Thus, in general, the scalability for single molecules is lower compared to molecular clusters.

Data presented in Table 6.8 demonstrate the importance of the dynamic load balancing and job scheduling (altering the execution order to reduce parallelization costs). It is seen that dynamic load balancing is always the best strategy, and its importance grows with the number of nodes. Approximately equally important is job reordering according to the job size. While for eight nodes the combined effect of the two factors is 5-12% (in terms of scalability), the effect grows with the number of nodes and reaches 27-36%, which in the most pronounced case of system C on 64 nodes translates into the total timing about 1.6 times slower if the two options (dynamic load balancing and job reordering) are not used. It should be emphasized that both options become vital to good scalability on clusters with mixed nodes; but even on identical nodes they significantly affect the performance.
Table 6.9 Scalability Tests for (H$_2$O)$_{64}$ RHF 6-31G* (Numeric Gradient), on the 7-Hub Star-Topology Fast Ethernet PC Cluster.

<table>
<thead>
<tr>
<th>$N_{nod} = N_{gr}$</th>
<th>$t_{tot}$, s</th>
<th>$S_{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51,831.7</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>25,697.3</td>
<td>1.0085</td>
</tr>
<tr>
<td>4</td>
<td>12,826.2</td>
<td>1.0103</td>
</tr>
<tr>
<td>8</td>
<td>6461.4</td>
<td>1.0027</td>
</tr>
<tr>
<td>16</td>
<td>3244.2</td>
<td>0.9985</td>
</tr>
<tr>
<td>32</td>
<td>1678.1</td>
<td>0.9652</td>
</tr>
<tr>
<td>64</td>
<td>900.5</td>
<td>0.8994</td>
</tr>
<tr>
<td>128</td>
<td>579.8</td>
<td>0.6984</td>
</tr>
</tbody>
</table>

$t_{tot}$ is the total time and $S_{tot}$ is the total scalability.

Finally, we note in passing that there is a factor altering the parallel performance that was not very relevant for the systems considered in this work. If large basis sets (and large fragments) are considered, two-electron integrals may not fit into memory, and have to be partially stored on disk. Using larger group size one can manage to fit all integrals in memory, reducing idle CPU time waiting for I/O. Correspondingly, the benefits of in-memory runs will add to the considerations of the most efficient group size and they will compete with other factors reducing performance for large groups.

6.3.2 Numeric Derivatives

Parallel tests employed the (H$_2$O)$_{64}$ system, with RHF/6-31G* two-point numeric gradient (naturally divided into $64^*3^*3^*2 * 1 = 1155$ single-point runs), using dynamic load balancing. Although a RHF analytic gradient is available, this example is a convenient tool for testing the numeric gradient scalability. The results are presented in Table 6.9. In all cases the most effective strategy is to create groups of size 1.

The scalability remains high as there is no intragroup communication, and it is hindered only by the wait at the final data exchange points, where $N_{nod} - 1$ nodes wait for the last node to finish its single energy calculation. If the number of nodes increases further, it is quite likely that a group size equal to 2 or larger will be preferred.

In this case, there is also nearly no difference between dynamic and static load balancing.
This arises because there is a single synchronization point at the end, and because every single-point run performed by each group differs very little from one another, physically and computationally, even when integral screening and other geometry dependent factors are taken into account. Such behavior is different from FMO runs of molecular clusters and large molecules. For instance, for the same \((\text{H}_2\text{O})_{64}\) cluster using the FMO method, each run deals with a different \text{H}_2\text{O} molecule that feels a physically and computationally different potential, whereas in case of the numeric derivatives it is a slightly displaced total molecule that is computed.

### 6.4 Conclusions

A general approach to efficiently use network topology has been proposed, based on a two-level hierarchical scheme. Practical implementations utilizing this approach are at present limited to the FMO method and numeric derivatives. A detailed analysis of the parallel implementation and its performance was given for both. The new approach was found to drastically outperform the standard DDI scheme, if many nodes are used (93 times on 128 nodes) for SCF calculations. Future work will be concerned with adding electron correlation to the FMO method and studying its scalability, as well as improving the inter-group communication paradigm.

Some problems remain to be solved in the parallel interface itself. One of these is the ability to survive sudden death of a node without interrupting the calculation. There is also a need to accommodate computers of different memory size (at present all are forced to use the same amount). It is likely that a smarter load balancing scheme will be required for electron correlation, as then the memory and computational requirements grow significantly.

Because the current implementation is based on either MPI or socket libraries, both available on most UNIX systems, and can be extended to most other types of computers, it is expected that this new approach will be readily available for efficient use on massively parallel computers.
Acknowledgments

All benchmark tests were performed on the PC cluster in Osaka Prefecture University. This work was supported in part by a Department of Energy Scientific Discovery through Advanced Computing (SciDAC) grant to MSG, administered by the Ames Laboratory. We would like to thank Dr. Michael W. Schmidt and Dr. Tamio Ikeshoji for helpful discussions and assistance.
Bibliography


CHAPTER 7. A Novel Approach to Parallel Coupled Cluster Calculations:
Combining Distributed and Shared Memory Techniques for Modern
Cluster Based Systems

A paper submitted to the Journal of Theoretical Computational Chemistry.¹

Ryan M. Olson², Jonathan L. Bentz³, Ricky A. Kendell⁴,³, Michael W. Schmidt², and Mark S. Gordon²

Abstract

A parallel coupled cluster algorithm that combines distributed and shared memory techniques for the CCSD(T) method (singles + doubles with perturbative triples) is described. The implementation of the massively parallel CCSD(T) algorithm uses a hybrid molecular and direct atomic integral driven approach. Shared memory is used to minimize redundant replicated storage per compute process. The algorithm is targeted at modern cluster based architectures that are comprised of multi-processor nodes connected by a dedicated communication network. Parallelism is achieved on two levels: parallelism within a compute node via shared memory parallel techniques and parallelism between nodes using distributed memory techniques. The new parallel implementation is designed to allow for the routine evaluation of mid- (500-750 basis function) to large-scale (750-1000 basis function) CCSD(T) energies. Sample calculations are performed on five low-lying isomers of water hexamer using the aug-cc-pVTZ basis set.

²Department of Chemistry, Iowa State University
³Department of Computer Science, Iowa State University
⁴Oak Ridge National Laboratory
7.1 Introduction

Coupled cluster (CC) methods [1] are now widely accepted as the premier single-reference electronic structure methods for small chemical systems at or near equilibrium geometries. One of the most popular CC methods is CCSD(T), which is based on an iterative solution of the single and double (SD) cluster amplitude equations [2] with a non-iterative perturbative correction for the triples (T) [3]. The CCSD(T) approach has been shown [4] to be a good compromise between the chemical accuracy of the higher-order CCSDT (full triples) method [5] and the computational efficiency of low order many-body perturbation theory (MBPT). Equation of motion (EOM) CC methods [6] have been developed for excited-state calculations. Spin flip [7] and method of moments CC methods [8], including the popular renormalized (R) [8], completely-renormalized (CR) [8] and CR-CCSD(T)$_L$ (CCL) methods [9] have extended formally single-reference CC methods into the regime of bond making and bond breaking, an area where traditional CC methods break down.

The biggest drawback of CC methods is the large computational demands required to perform such calculations. However, due to the popularity of methods like CCSD(T), considerable research has been carried out to generate highly efficient algorithms [2, 10] and their implementations. A variety of CC methods can be found in all of the major electronic structure programs available today, including GAMESS [11], MOLPRO [12], ACES II [13] Q-CHEM [14], PSI3 [15], NWChem [16] and GAUSSIAN94 [17]. Most CC programs are highly optimized to run sequentially. This usually means the calculation is performed on a single processor. The speed of the processor and the size of the associated memory and disk are limiting factors for sequential algorithms. CCSD(T) calculations, especially those run in C1 symmetry, reach the limit of most single processor workstations at around 400-500 basis functions (BF); even then, calculations of these sizes may require weeks of time on a dedicated workstation [18].

One means of evaluating computationally demanding problems such as large basis set (> 500 BF) CCSD(T) calculations is to make use of parallel computing. Parallel computing involves simultaneously evaluating multiple portions of a larger computational problem on multiple processors, in order to achieve an overall reduction in the real-time evaluation of the
problem. Equally important, parallel computing can extend computationally demanding methods like CCSD(T) to larger problems because of increased computational resources and also storage (memory/disk) resources. There is a wide range of parallel computing environments and methodologies, two examples of which are addressed herein. These are: 1) parallelism that is achieved by using multiple computers or *nodes* which are connected by a dedicated communication network, and 2) parallelism that is achieved by multiple processors within a single node that share local system resources including memory and I/O channels.

The tools and methodologies for these two traditional types of parallel computing environments are very different. Multi-node parallelism focuses on combining replicated and/or distributed memory techniques using parallel communication libraries such as TCGMSG [19], SHMEM [20], MPI [21], Global Arrays (GA) [22], and the Distributed Data Interface (DDI) [23, 24]. One advantage of multi-node models is that the aggregate system resources increase as the number of nodes increases, thereby facilitating more resource demanding calculations. However, since the nodes are distinct and inter-node communication must travel over a high-speed network, there are three factors that will strongly affect the performance for these types of calculations: 1) the performance (bandwidth and latency) of the network, 2) the total amount of inter-node communication required, and 3) the degree to which the necessary communication can be overlapped with computation.

On the other hand, single node multi-processor parallel schemes have traditionally focused on a relatively small number of compute processes (or threads), usually between 2-16, using shared resources as a means to reduce: 1) message passing communication and 2) replicated storage overhead, i.e. using the shared resources of the system to store certain data arrays only one time, rather than stored multiple identical copies for each process (or thread). A major focus of these techniques involves sharing portions of the system memory between all parallel processes (or threads) and providing tools to control access to this shared data. Examples of shared-memory based programming models include: the POSIX Pthreads model, OpenMP model [25] and the System V inter-process communication model.

In general, the multi-node and single node parallel strategies were developed separately
based on two different types of parallel architectures. However, it is the evolution of the node, specifically the use of multi-processor “shared-memory” nodes as the building blocks for multi-node cluster based systems, which is bringing about a convergence of these methodologies. That is, it is possible to embed the use of shared-memory programming techniques within each node of a cluster based system, yet retain the advantages of increased aggregate system resources from a multi-node platform. This becomes especially important when examining the roadmap for future generations of computers. The next generation(s) of processors are not expected to dramatically increase in frequency, which traditionally has accounted for 80% of the performance improvements. Rather, the current trend is to add multiple processing “cores” on each processor. This use of multi-core processors in multi-processor nodes further increases the computational density per node and further emphasizes the need to address different parallel strategies for intra- and inter-node computing and data management within current and future cluster based systems.

The focus of this work is to describe an algorithm for the CCSD(T) method that can utilize both intra-node and inter-node forms of parallelism. Algorithms for parallel CC methods [26, 27, 28, 29, 30] have been developed by other groups. These methodologies for the parallelization of CC methods and other correlation methods were divided into two categories: those aimed at shared memory machines (SMPs) and those aimed at distributed memory machines. Early work by Komornicki, Lee and Rendell [26] described a highly vectorized shared memory algorithm for evaluating the connected triples excitations (T) on the CRAY Y-MP. Vectorized shared-memory CCSD and CCSD(T) algorithms based on AO integrals stored on disk were later implemented by Koch and coworkers [31]. These early shared-memory vectorized algorithms primarily used optimized library calls to gain computational speedup (the libraries, not the programs themselves, were multi-process or multi-thread based), although some directives to parallelize the loops were employed. Rendell, Lee and Lindh [27] implemented the first distributed memory CCSD algorithm on an Intel i860 hypercube. In that work, asymptotic speedups were quickly reached due to I/O bottlenecks based on retrieval of the molecular integrals. The authors proposed the use of a “semi-direct” method in which
atomic integrals evaluated “on demand” could be used to alleviate the I/O bottleneck. Rendell, Guest, and Kendall [28] improved the previous MO-based distributed memory CCSD approach and extended the program to include CCSD(T). Later, Kobayashi and Rendell [29] implemented a “direct” AO-driven CCSD(T) algorithm which avoided the I/O bottlenecks of earlier MO-based distributed memory methods; this development formed the basis for the parallel CCSD(T) module within the NWChem package [16]. As another means of avoiding potential I/O bottlenecks, Rendell and Lee proposed [32] that some two-electron integrals can be approximated using the resolution of the identity (RI) technique. RI-based approaches can dramatically reduce the storage requirements needed for CCSD and CCSD(T) calculations, while maintaining $O(N^6)$ and $O(N^7)$ scaling for the computational effort where $N$ is a description of the size of the system being calculated; the number of atomic basis functions is an upper limit to $N$. MOLPRO [12] also offers a parallel implementation of its coupled cluster methods. Most recently, Janowski and coworkers [30] have presented a parallel algorithm for the CCSD method using the Array Files toolkit [33].

Another exciting advance in the development of parallel computer codes for high level ab initio quantum chemistry methods is the tensor contraction engine (TCE) [34], a program used for automatic code generation for a general set of high level ab initio methods, including coupled cluster methods. Hirata [35] has shown the utility of the TCE for deriving and implementing many common second-quantized many-electron theories including a variety of coupled cluster methods. The TCE also has the ability to automatically generate parallel computer codes. A recent study by Piecuch and coworkers [36] used the TCE to generate parallel code for the completely renormalized CCSD(T) method [8] which showed that a ten times execution speedup could be achieved using 64 processors. As illustrated by this example, parallel codes generated by the TCE are generally not as efficient as hand-tuned computer codes; however, the major benefits of using the TCE are its ease of use, the avoidance of errors in generating very complex codes, and its general applicability to higher order ab initio methods in which detailed hand tuning and parallelization can be very difficult. The contributions from a number of researchers [37] to the improvement of the generation of highly-efficient parallel codes via the
TCE program has extraordinary potential and could someday result in automatically generated code that is as good as or better than hand-tuned programs.

A major focus of this research is to describe an implementation of the CCSD(T) algorithm that combines distributed memory (inter-node) and shared-memory (intra-node) techniques in a massively parallel (MP) environment. The MP-CCSD(T) algorithm described here is an adaptation of the sequential algorithm, previously implemented in GAMESS [11] by Piecuch et al [38]. Because the MP-CCSD(T) method described here is based on the same spin-free equations used by the EOM and renormalized CC methods in GAMESS, the approach to closed shell CCSD(T) parallelism described here can be extended to the other types of coupled cluster methods in a straightforward manner.

To provide an example of the viability of the MP-CCSD(T) algorithm on modern cluster based architectures, CCSD(T) calculations on geometric isomers of water hexamer using the aug-cc-pVTZ basis set [39] are presented. These calculations are important, since there are five low lying isomers of water hexamer (Figure 7.1), some of which have three-dimensional structures, whose relative energies are very close to each other. Since these are the smallest 3-D water clusters, it is very important to be able to predict the correct energy order for the low-energy isomers with high accuracy. This means that one needs both large basis sets that approach the complete basis set limit, in order to avoid basis set superposition error (BSSE), and a high theoretical level, such as CCSD(T). A number of high-level ab initio studies [40, 41, 42, 43, 44] have been performed on the water hexamer. In a very thorough and systematic study of the potential energy surface for small water clusters using second-order Møller-Plesset perturbation theory [45] (MP2) and a series of augmented correlation consistent basis sets [39, 46] that are systematically improved (aug-cc-pVXZ ranging from X=D,T,Q,5) Xantheas and coworkers [41] have predicted that the prism structure is the global minimum. However, the predicted energy differences among the water hexamer isomers are very small (a range of less than 1.2 kcal/mol for the four isomers studied). Given the known tendency of MP2 to overbind clusters, it is important to employ a more sophisticated level of theory, e.g. CCSD(T), with a sufficiently large basis set such that BSSE approaches zero [47].
Figure 7.1 Images of the five geometric isomers used in this study. The geometries correspond to MP2 optimized structures using the DH(d,p) basis set obtained by Day et al. [43].
The calculations performed herein represent, to the authors’ knowledge, the most accurate CCSD(T) calculations on water hexamer to date.

This paper highlights the key features of the MP-CCSD(T) program and demonstrates that the algorithm is viable on modern cluster based MP platforms. The goal of the MP-CCSD(T) algorithm is to enable high-level CC calculations to provide accurate energies and potential energy surfaces for systems, like water hexamer, that are currently very difficult to achieve. As an illustration of the new method, the CCSD(T)/aug-cc-pVTZ energies for the five low-lying water hexamer isomers are calculated and the performance of the MP-CCSD(T) method is examined.

7.2 CCSD/CCSD(T) Theory

The MP-CCSD(T) method described in this work is an adaptation of the sequential CCSD(T) program previously implemented by Piecuch et al. [38], therefore, the same notation used in Ref [38] is followed here. The letters \(i, j, k, l, \ldots\) will be used to denote occupied spatial molecular orbitals, \(a, b, c, d, \ldots\) will be used to represent unoccupied (virtual) orbitals, \(\mu, \nu, \lambda, \sigma, \ldots\) are used to represent atomic orbital indices or atomic shell indices, and \(p, q, r, s, \ldots\) are a general set of indices. Details of the CCSD and perturbative (T) correction have been discussed in several reviews [48], so only a brief outline is given here.

The CCSD method derives from CC theory in the following manner. Let the exact CC wavefunction \(|\Psi_{CC}\rangle\) be defined as,

\[
|\Psi_{CC}\rangle = e^T|\Phi\rangle
\]  

(7.1)

where \(|\Phi\rangle\) is the reference wavefunction (for this work, \(|\Phi\rangle\) is the restricted closed-shell Hartree-Fock reference wavefunction) and \(T\) is the complete cluster operator containing all possible single \((T_1)\), double \((T_2)\), triple \((T_3)\) etc. excitation operators,

\[
T = T_1 + T_2 + T_3 + \cdots
\]  

(7.2)
The CCSD method results from the truncation of $T$ such that only single and double excitation operations are included,

$$T \simeq T_1 + T_2.$$  \hfill (7.3)

Projecting the connected-cluster form of the CCSD equation,

$$(H_{Ne}^{T_1+T_2})_C|\Phi\rangle = E_{CCSD}|\Phi\rangle,$$  \hfill (7.4)

where $H_N$ is the normal product Hamiltonian $(H - \langle \Phi|H|\Phi\rangle)$, onto the set of excited determinants defined by the truncated excitation operator (Eq. 7.3) gives rise to a set of coupled non-linear equations,

$$\langle \Phi^a_i|(H_{Ne}^{T_1+T_2})_C|\Phi\rangle = 0,$$  \hfill (7.5)

$$\langle \Phi^{ab}_{ij}|(H_{Ne}^{T_1+T_2})_C|\Phi\rangle = 0.$$  \hfill (7.6)

Eqs. (7.5) and (7.6), which are solved iteratively for the single and double excitations, respectively. In terms of amplitudes ($t^a_i, t^{ab}_{ij}$), Fock matrix elements ($f^a_i$) and two-electron molecular integrals ($v^{pq}_{rs} = \langle pq|\frac{1}{r_{12}}|rs\rangle$), the CCSD amplitude equations (Eqs. 7.5 and 7.6) are given (using the Einstein summation convention$^5$):

$$D^a_i t^a_i = f^a_i + I^{ae}_i t^e_i + I^{ma}_i t^a_m + I^m_i (2t^{ea}_i - t^{em}_i) + (2v^{ma}_i - v^{am}_e)t_i^e$$

$$- v^e_m (2t^{ea}_m - t^{em}_m) + v^{ma}_e (2t^{ej}_m - t^{em}_m)$$  \hfill (7.7)

$$D^{ab}_{ij} t^{ab}_{ij} = v^{ab}_{ij} + P(ij/ab)[t^{ae}_{ij} t^b_i - t^{ab}_{im} I^m_j + \frac{1}{2}v^{ab}_{ef} t^{ef}_{ij} + \frac{1}{2}c^{ab}_{mn} I^{mn}_{ij}]$$

$$- t^{ae}_{mj} I^{mb}_{ie} - I^{ma}_i t^{eb}_j + (2t^{ea}_i - t^{em}_i) I^{mb}_{ej} + t^{a}_{ie} I^{ab}_{mj} - t^{a}_{m} I^{mb}_{ij}$$  \hfill (7.8)

$^5$The Einstein summation convention implies a summation over all possible values of repeated indexes found in the lower or upper positions of a single term. For example, $I^a_{ik} t^i_k = \sum t^i_k I^a_{ik}$. 

In Eqs. 7.7 and 7.8, $c_{ij}^{ab}$ is defined as

$$c_{ij}^{ab} = t_{ij}^{ab} + t_{ij}^{a|b}.$$  \hspace{1cm} (7.9)

The permutation operator $P(ij/ab)$ acting on an arbitrary term $X$ has the following properties,

$$P(ij/ab)[X_{ij}^{ab}] = X_{ij}^{ab} + X_{ji}^{ba},$$  \hspace{1cm} (7.10)

and the general MBPT denominators are used to define $D_i^a$ and $D_{ij}^{ab}$ such that

$$D_{ij}^{ab}... = \varepsilon_i - \varepsilon_a + \varepsilon_j - \varepsilon_b \cdots,$$  \hspace{1cm} (7.11)

where,

$$\varepsilon_p = f_p^p$$  \hspace{1cm} (7.12)

are are the diagonal elements of the Fock matrix. The intermediates ($I$ and $I'$) of Eqs. 7.7 and 7.8 are:

$$I_{a}^i = f_i^a + 2v_i^{ae}t_{me}^e - v_i^{me}t_{ae}^e,$$  \hspace{1cm} (7.13)

$$I_{b}^i = (1 - \delta_{ab})f_i^a + 2v_i^{ma}t_{me}^e - 2v_i^{mn}v_{ea}^a + v_i^{mn}v_{ea}^a - t_{ma}f_{eb}^m,$$  \hspace{1cm} (7.14)

$$I_{j}^i = (1 - \delta_{ij})f_j^i + 2v_i^{me}t_{me}^e - v_{ej}^m + v_{ef}^m - v_{ef}^m,$$  \hspace{1cm} (7.15)

$$I_{j}^i = I_{j}^{i|} + I_{j}^{i|e},$$  \hspace{1cm} (7.16)

$$I_{ij}^{kl} = v_{ij}^{kl} + v_{ef}^k v_{e|}^l + P(ik/jl)[t_{e}^{k}v_{e|}^l],$$  \hspace{1cm} (7.17)

$$I_{ja}^{jb} = v_{ja}^{jb} - \frac{1}{2}v_{ja}^{ce}v_{ae}^{ce} - v_{ja}^{be},$$  \hspace{1cm} (7.18)

$$I_{bj}^{ia} = v_{bj}^{ia} + v_{be}^{ma}v_{ae}^{ce} - \frac{1}{2}v_{eb}^{me}v_{ae}^{ce} - \frac{1}{2}v_{be}^{me}v_{ae}^{ce} + v_{bj}^{ia},$$  \hspace{1cm} (7.19)

$$I_{ci}^{ab} = v_{ci}^{ab} - v_{ci}^{me}t_{me}^b - t_{ma}v_{ci}^m,$$  \hspace{1cm} (7.20)

$$I_{j}^{ia} = v_{j}^{ia} + v_{ef}^{ia}v_{e|}^a + v_{j}^{ia} + t_{j}^{a}v_{e|}^a,$$  \hspace{1cm} (7.21)

where $\delta_{pq}^*$ represents the standard Kronecker delta.
The CCSD correlation energy from the CCSD method is calculated after Eqn 7.7 and 7.8 are solved iteratively for $t^a_i$ and $t^{ab}_{ij}$ and is given by the following formula:

$$\Delta E_{CCSD} = 2f^a_i t^a_i + (2v^{ij}_{ab} - v^{ij}_{ba})c^{ab}. \quad (7.22)$$

Non-iterative solutions to the full CCSDT problem using only lower order excitation operators ($T_1$ and/or $T_2$) were first developed by Urban and coworkers [49]. These methods eventually led to the CCSD(T) method derived by Raghavachari and coworkers [3]. The (T) of CCSD(T) is an \textit{a posteriori} non-iterative correction to the CCSD energy. In a study analyzing a variety of different approximations to the full CCSDT treatment, Scuseria and Lee [4] found the CCSD(T) method to be the most accurate and the most computationally efficient of all the approximate methods examined. In terms of molecular integrals and amplitudes [38], the correction to the CCSD energy is given by:

$$E^{(T)} = \tilde{t}^{ijk}_{abc}(2)t^{abc}_{ijk}(2)D^{abc}_{ijk} + z^{ijk}_{abc}t^{abc}_{ijk}(2)D^{abc}_{ijk}, \quad (7.23)$$

where an arbitrary $\tilde{X}^{ijk}_{abc}$ term is expanded such that,

$$\tilde{X}^{ijk}_{abc} = 4X^{ijk}_{abc} - 2X^{ijk}_{acb} + 2X^{ijk}_{bca}. \quad (7.24)$$

The $t^{abc}_{ijk}(2)$ coefficients are defined in terms of $t^a_i$ and $t^{ab}_{ij}$,

$$D^{abc}_{ijk}t^{abc}_{ijk}(2) = P(ia/jb/kc)[t^{ae}_{ij}v^{bc}_{ek} - t^{ab}_{im}v^{mc}_{jk}], \quad (7.25)$$

where the permutation operator $P(ia/jb/kc)$ expands a quantity containing the (ia), (jb) and/or (kc) pair into a summation of up to six quantities:

$$P(ia/jb/kc)[\cdots]_{ijk}^{abc} = [\cdots]_{ijk}^{abc} + [\cdots]_{ijk}^{acb} + [\cdots]_{ijk}^{cab} + [\cdots]_{ijk}^{cba} + [\cdots]_{ijk}^{bca} + [\cdots]_{ijk}^{bac}. \quad (7.26)$$
The second term of Eqn (7.23) is the disconnected triples correction to \( E^{(T)} \) where

\[
z^{ijk}_{abc} = (z^{abc}_{ijk})^* = (t^i_{abc} v^{ijk}_{bc} + t^i_{bac} v^{ijk}_{ac} + t^i_{cab} v^{ijk}_{ab})/D^{abc}_{ijk},
\]

(7.27)

and the \( z^{ijk}_{abc} \) and \( z^{abc}_{ijk} \) are complex conjugates. The final CCSD(T) energy is given by:

\[
E^{CCSD(T)} = \langle \Phi | H | \Phi \rangle + \Delta E^{CCSD} + E^{(T)}.
\]

(7.28)

A detailed discussion of the individual terms in the CCSD and (T) equations is presented in Section 3 of Piecuch et al.[38]. The summary presented in Eqs. (1) - (28) provides a sufficient background for the following discussion of the implementation of the MP-CCSD(T) method.

### 7.3 Parallel Design

There are two primary issues that must be considered in order to perform large-scale CCSD(T) calculations in a massively parallel environment: How can the computational workload be divided among the available parallel processes?; How can the large data sets associated with such demanding calculations be stored and utilized efficiently by the available parallel processes?

The amount of computational effort associated with the CCSD and CCSD(T) algorithms scales asymptotically as \( \mathcal{O}(N^6) \) and \( \mathcal{O}(N^7) \) respectively. \( N \) is a measure of system size and can be broken down more specifically in terms of the number of occupied molecular orbitals \( (N_o) \) and the number of unoccupied (virtual) molecular orbitals \( (N_v) \). More generally (and more conservatively), one can use the number of one-electron atomic basis functions \( (N_{bf}) \).

In terms of molecular orbitals, the CCSD and CCSD(T) algorithms scale on the order of their most expensive terms, \( \mathcal{O}(N_o^2 N_v^4) \) and \( \mathcal{O}(N_o^3 N_v^4) \), respectively. Each of the terms in the sequential code [38] was parallelized, with specific attention paid to the terms which comprise the computational bottlenecks. However, the distribution of the computational work is very closely related to the distribution of the large data sets required by the CCSD(T) method. Therefore, before detailed examples of the manner in which the terms of the CCSD(T) method
were parallelized, an examination of data distribution is required.

The second major consideration addresses the storage requirements for large CCSD(T) calculations in a massively parallel environment. As mentioned in Section 7.2, the CCSD(T) equations are written in terms of cluster amplitudes and molecular (or atomic) integrals. The manner in which the integrals and amplitudes are stored on a large parallel computer has a direct effect on how the computational workload can be distributed. Equally important, the choice of how the amplitudes and integrals are stored will directly affect the storage bottlenecks of the algorithm.

The MP-CCSD(T) algorithm was designed to address these bottlenecks by first examining the data storage problem and then addressing the parallel work division based on a defined data distribution. In the following discussion, the storage bottlenecks are examined in the scope of the programming model and the available types of storage. Based on these ideas and an outlined storage model, Section 7.4 describes how the computational work is divided into inter-node and intra-node components.

7.3.1 Parallel Programming Model

The MP-CCSD(T) algorithm introduces and utilizes the third generation of the Distributed Data Interface (DDI) for communication and data storage in a massively parallel environment. The DDI model is a high-level abstraction of the virtual shared-memory model for use in the GAMESS quantum chemistry suite of programs. DDI was designed as a means to provide a consistent set of parallel programming tools for the quantum chemistry code, while maintaining enough generality to be implemented using a variety of existing parallel libraries that offer one-sided message passing, including: SHMEM, Global Arrays (GA) [22], MPI [21], and a native implementation based on point-to-point libraries such as MPI [21] and/or TCP/IP sockets. The DDI model was strongly influence by the structure and functionality of the Global Arrays (GA) Toolkit; however, to maintain a high degree of portability only a subset of the GA functionality is used within the DDI model.

The first generation of DDI [24, 50], DDI/1, provided a process-based implementation of
the distributed-memory programming model in which large arrays could be evenly divided over all available nodes, yet remain globally accessible via one-sided message operations. DDI/1 was modeled on the design of the Cray T3E in which the system image of each node contained a single processor and some associated system memory. The nodes formed the building blocks of the parallel computer and were connected to other nodes by a high-speed network. DDI/1 is a process-based model, because the data and the computational workload are divided over the parallel processes.

The second generation of DDI [23], DDI/2, introduced a greater awareness of the memory topology by recognizing that multiple parallel processes could co-exist within the same node, i.e. multiple processors in a single node sharing the same local system memory. This shared-memory awareness increases the amount of data that can be considered “local” and can significantly reduce the number of remote communication operations for calculations run using multi-processor nodes; this was recognized for point-to-point communication in many MPI implementations and also in the one-sided communications for both GA [22] and DDI [23].

The third generation of DDI, DDI/3, further enhances the shared-memory capabilities of DDI by providing the tools needed for multi-processor nodes to utilize shared-memory outside of the distributed-memory model. Specifically, DDI/3 provides the ability to create and control access to shared-memory segments, as well as the ability to perform point-to-point and collective operations within the node. The shared-memory model in DDI/3 is based on multiple processes using SystemV shared-memory and semaphores for inter-process communication rather than a thread-based model. This maintains the integrity of the former DDI models, whereas a shift to a thread-based model for intra-node parallelism would require a radical change to the DDI programming model. DDI/3 provides all the necessary tools for process-based and node-based parallelism.

Node-based parallelism differs from process-based parallelism in that the data and the computational work is first divided by node (inter-node), and then the work assigned to each node is further decomposed and parallelized over the “local” processes within each node. Node-based
parallel schemes have the advantage of being able to handle larger replicated data sets when compared to process-based schemes, because shared-memory can be used to store particular quantities \textit{once per node}, rather than \textit{once per process}. The MP-CCSD(T) algorithm described here utilizes both process-based and node-based parallel techniques.

### 7.3.2 Memory

DDI/3 supports three types of memory storage to be used in the MP-CCSD(T) algorithm: replicated, shared and distributed. Replicated memory is process-based, and the amount of memory needed for data stored in replicated memory scales linearly with the number of processes. Typically, arrays that scale as $O(N^2)$ and some that scale as $O(N^3)$ can be stored in replicated-memory. Shared memory is node-based and the amount of memory needed for data stored in shared memory scales linearly with the number of nodes. Shared-memory allows for the storage of larger arrays than does replicated-memory, since the arrays are only stored \textit{once per node}. In a shared-memory environment, every process within the node can access and modify the data in shared-memory segments. This feature provides a convenient means of parallelizing the computational work over a shared data set, since each process has direct access to the data in that memory (by physical address). However, allowing multiple processes to have access to shared resources means that special care must be taken to prevent possible race conditions, i.e. situations that occur during parallel execution in which one process seeks to modify data that is concurrently being used by another process. To handle these race conditions, DDI/3 uses SystemV semaphores and collective synchronizations over all intra-node processes to control access to shared resources and guarantee data integrity.

Distributed memory is the aggregate of portions of "local" system memory reserved by each process for the storage of distributed data. In the DDI framework, the number of columns of distributed two-dimensional matrix is divided evenly over the total number of parallel processes; the disjoint sets of columns are mapped in a one-to-one matter onto the set of parallel processes and the data associated with each set of columns is stored in the memory reserved by each process for distributed memory storage. In contrast to shared memory, access to distributed
memory requires calling subroutines from the DDI library. The amount of distributed-memory needed for a given calculation is defined solely by the parameters of the calculation and has no dependence on the number of parallel processes used for the calculation. The requirements for distributed memory can in some cases be very large; in those cases, the number of nodes must be chosen to accommodate the required distributed memory.

There are two types of distributed-memory: local and remote. All parallel processes are allowed to modify any element of an array stored in distributed-memory (regardless of physical location); however, due to the communication overhead of accessing remote distributed-memory, the programming strategy seeks to maximize the use of local distributed-memory and minimize the use of remote distributed-memory. In this regard, arrays stored in distributed-memory are not easily rearranged between distributed indexes. For example, when a transpose operation, i.e. the swapping of the rows and columns, is performed on a distributed matrix that is distributed evenly over the number of columns, every parallel process must communicate with all of the other parallel processes. Thus, for very large distributed matrices, this type of operation would require a large amount of communication overhead and would be an impediment to achieving good parallel speedups.

### 7.3.3 Molecular Integral Transformation

The MO integral classes use an “O” to denote an actively correlated occupied MO index and a “V” to denote a virtual MO index. In the present work, a modified version of the distributed-memory “direct” four-index integral transformation[51] previously implemented by Fletcher and coworkers[50] was used to calculate the MO integrals: [OO|OO], [VO|OO], [VV|OO], [VO|VO] and [VV|VO]. The original integral transformation was only able to calculate MO integrals with up to two virtual indexes, and was not able to exclude frozen-core MOs from the transformation for a general set of MO integral classes. Modifications were therefore made to allow for the formation of [VV|VO] integrals and to add an option to include or exclude frozen core MOs in the transformation. These modifications maintain the integrity of the original algorithm, i.e. the identical procedures are used; however, the starting indexes and ranges of
Table 7.1  General list of data types that describes what type of memory the quantity will be stored in and how the quantity scales as a function of $N_o$ and $N_v$.

<table>
<thead>
<tr>
<th>Class</th>
<th>Type</th>
<th>Size</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>amplitudes</td>
<td>$O(N_o N_v)$</td>
<td>replicated</td>
</tr>
<tr>
<td>$T_2$</td>
<td>amplitudes</td>
<td>$O(N_o^2 N_v^2)$</td>
<td>shared</td>
</tr>
<tr>
<td>[OO</td>
<td>OO]</td>
<td>integrals</td>
<td>$O(N_o^4)$</td>
</tr>
<tr>
<td>[VO</td>
<td>OO]</td>
<td>integrals</td>
<td>$O(N_o^3 N_v)$</td>
</tr>
<tr>
<td>[VV</td>
<td>OO]</td>
<td>integrals</td>
<td>$O(N_o^2 N_v^3)$</td>
</tr>
<tr>
<td>[VO</td>
<td>VO]</td>
<td>integrals</td>
<td>$O(N_o^2 N_v^2)$</td>
</tr>
<tr>
<td>[VV</td>
<td>VO]</td>
<td>integrals</td>
<td>$O(N_o N_v^3)$</td>
</tr>
<tr>
<td>[VV</td>
<td>VV]</td>
<td>integrals</td>
<td>$O(N_v^4)$</td>
</tr>
</tbody>
</table>

MO indexes that are transformed were modified.

The formation of the [VV|VO] integrals requires an additional distributed array to store the [NN|VO] integrals, where “N” is the total number of basis functions and the entries can be V or O. The same procedure that is used to complete the [VV|OO] integrals from the [NN|OO] set of half transformed integrals is used to complete the [VV|VO] integrals from the [NN|VO] set of half-transformed integrals. Exclusion of the frozen-core integrals is accomplished using a straightforward modification of the starting index and the range of MO orbitals that are defined as occupied (active). If one wishes to freeze the core molecular orbitals in the coupled cluster calculation, those core molecular orbitals are not correlated and therefore the MO integrals associated with the frozen-core MOs are not required. The option to exclude the frozen-core integrals can result in a significant reduction in computational effort and most importantly a reduction in the distributed-memory requirements for the integral transformation. Of course, for heavier elements such as Au, one must take care in defining those orbitals that are frozen, in order to avoid excluding orbitals that are important in the chemical process of interest [52].

7.3.4 Memory Requirements and Bottlenecks

The scaling of the storage requirements and how the data is stored within the MP-CCSD(T) algorithm is given in Table 7.1 in terms of the number of actively correlated occupied ($N_o$) and the number of virtual ($N_v$) molecular orbitals (MO). In the following discussion, the
memory requirements and the potential memory bottlenecks are examined over the range of:
\[ 10 \leq N_o \leq 60 \text{ and } 300 \leq N_v \leq 1000. \]

For mid-range to high-end dedicated supercomputers, the assumption is made that 4-8GB of memory per processor is available. For common 4-8 processor nodes, this means that typically 16-64GB of “local” system memory per node is generally available. For low-end commodity clusters, these assumptions would not necessarily hold at the present; however, it is assumed here that sufficient high-performance computer facilities are available.

Another working assumption is that access to quality disk storage, i.e. “local” multi-channeled striped disk arrays on every node, is not generally available. This is a conservative approach to minimize the performance dependence of the algorithm on the quality of the available disk I/O, which can vary greatly from cluster to cluster. In fact, some clusters do not even have local scratch disk storage and the only available file system may be a remote networked file server, or a parallel file system such as Lustre or PVFS2. The performance of the algorithm might be improved if one could assume that quality local disk storage per node is available. In this initial implementation of the MP-CCSD(T) algorithm, only minimal system requirements are assumed.

There are two storage bottlenecks in the MP-CCSD(T) algorithm as defined by the choice of data storage (Table 7.1). These are the storage of the \( T_2 (t_{ij}^{ab}) \) in shared-memory and the storage of the \([VV|VO]\) molecular integrals in distributed memory.

The storage of the \( T_2 (t_{ij}^{ab}) \) amplitudes in shared-memory is the first of two storage bottlenecks within the MP-CCSD(T) algorithm. The \( T_2 \) amplitudes require \( N_o^2 N_v^2 \) words of shared memory; however, two other intermediates of the same size must also be stored in shared memory. The actual size of the \( T_2 \) amplitudes measured in gigabytes (GB) is given in Table 7.2 (See Table 7.2 for a summary of all integral and amplitude types). The use of shared memory to store the \( T_2 \) amplitudes represents a compromise for the efficient use of the amplitudes, since the \( T_2 \) amplitudes are too large (in most cases >1GB) to be stored in replicated-memory, and these \( T_2 \) amplitudes are reordered and manipulated too frequently to be stored in distributed memory. At the limits of \( N_o = 60 \) and \( N_v = 1000 \), approximately 27GB of shared memory
The maximum size in gigabytes (GB) of the array to hold the $T_2$ amplitudes, the [VV|OO] integrals or the [VO|VO] MO integrals. The rows correspond to values of $N_o$ and the columns correspond to values of $N_v$. The bold faced values correspond to arrays less than or equal to 6GB.

<table>
<thead>
<tr>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>15</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.1</td>
<td>1.4</td>
<td>1.7</td>
</tr>
<tr>
<td>20</td>
<td>0.3</td>
<td>0.7</td>
<td>1.1</td>
<td>1.5</td>
<td>1.9</td>
<td>2.4</td>
<td>3.0</td>
</tr>
<tr>
<td>25</td>
<td>0.4</td>
<td>0.7</td>
<td>1.2</td>
<td>1.7</td>
<td>2.3</td>
<td>3.0</td>
<td>4.7</td>
</tr>
<tr>
<td>30</td>
<td>0.6</td>
<td>1.1</td>
<td>1.7</td>
<td>2.4</td>
<td>3.3</td>
<td>4.3</td>
<td>5.4</td>
</tr>
<tr>
<td>35</td>
<td>0.8</td>
<td>1.5</td>
<td>2.3</td>
<td>3.3</td>
<td>4.5</td>
<td>5.8</td>
<td>7.4</td>
</tr>
<tr>
<td>40</td>
<td>1.1</td>
<td>1.9</td>
<td>3.0</td>
<td>4.3</td>
<td>5.8</td>
<td>7.6</td>
<td>9.7</td>
</tr>
<tr>
<td>45</td>
<td>1.4</td>
<td>2.4</td>
<td>3.8</td>
<td>5.4</td>
<td>7.4</td>
<td>9.7</td>
<td>12.2</td>
</tr>
<tr>
<td>50</td>
<td>1.7</td>
<td>3.0</td>
<td>4.7</td>
<td>6.7</td>
<td>9.1</td>
<td>11.9</td>
<td>15.1</td>
</tr>
<tr>
<td>55</td>
<td>2.0</td>
<td>3.6</td>
<td>5.6</td>
<td>8.1</td>
<td>11.0</td>
<td>14.4</td>
<td>18.3</td>
</tr>
<tr>
<td>60</td>
<td>2.4</td>
<td>4.3</td>
<td>6.7</td>
<td>9.7</td>
<td>13.1</td>
<td>17.2</td>
<td>21.7</td>
</tr>
</tbody>
</table>

per node is required for the $T_2$ amplitudes. In such cases, the storage of the $T_2$ amplitudes and the other intermediates is not possible on modern SMP clusters, which, as noted above, typically have 16-64 GB of system memory per node. The present discussion focuses on the implementation for clusters of SMPs; therefore, the practical range of $N_o$ and $N_v$ is defined to be those values for which the size of the $T_2$ amplitudes is less than 6GB (the shaded region in Table 7.2). This practical range of $N_o$ and $N_v$ is defined to overcome the first major storage bottleneck. The same range will be used to examine the sizes for the remaining amplitude and integral classes.

The [VV|OO] and [VO|VO] integrals are similar in size (Table 7.2) to the $T_2$ amplitudes, thus over the practical range of $N_o$ and $N_v$ which define the shared-memory bottleneck of less than 6GB per array, these quantities are considered small when stored in distributed-memory. Like the $T_2$ amplitudes, the [VV|OO] and [VO|VO] MO integrals need to be reordered several times throughout the calculation. As mentioned earlier, the reordering of distributed arrays can be very inefficient due to the large amount of communication that is needed. However, unlike the $T_2$ amplitudes that get updated at the end of every CCSD iteration, the [VV|OO] and [VO|VO] MO integrals are constant for a fixed geometry and basis set. Therefore, instead
Table 7.3 The actual size of the [VV|VO] integral class as stored in distributed memory. The values are in gigabytes (GB). The rows correspond to values of $N_o$ and the columns correspond to values of $N_v$. The bold face values correspond to those values of $N_o$ and $N_v$ for which the size of the $T_2$ amplitudes array is less or equal to 6GB (Table 7.2).

<table>
<thead>
<tr>
<th></th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.0</td>
<td>2.4</td>
<td>4.7</td>
<td>8.1</td>
<td>12.8</td>
<td>19.1</td>
<td>27.2</td>
<td>37.3</td>
</tr>
<tr>
<td>15</td>
<td>1.5</td>
<td>3.6</td>
<td>7.0</td>
<td>12.1</td>
<td>19.2</td>
<td>28.6</td>
<td>40.8</td>
<td>55.9</td>
</tr>
<tr>
<td>20</td>
<td>2.0</td>
<td>4.8</td>
<td>9.3</td>
<td>16.1</td>
<td>25.6</td>
<td>38.2</td>
<td>54.4</td>
<td>74.6</td>
</tr>
<tr>
<td>25</td>
<td>2.5</td>
<td>6.0</td>
<td>11.7</td>
<td>20.2</td>
<td>32.0</td>
<td>47.7</td>
<td>68.0</td>
<td>93.2</td>
</tr>
<tr>
<td>30</td>
<td>3.0</td>
<td>7.2</td>
<td>14.0</td>
<td>24.2</td>
<td>38.4</td>
<td>57.3</td>
<td>81.6</td>
<td>111.9</td>
</tr>
<tr>
<td>35</td>
<td>3.5</td>
<td>8.4</td>
<td>16.3</td>
<td>28.2</td>
<td>44.8</td>
<td>66.8</td>
<td>95.2</td>
<td>130.5</td>
</tr>
<tr>
<td>40</td>
<td>4.0</td>
<td>9.6</td>
<td>18.7</td>
<td>32.2</td>
<td>51.2</td>
<td>76.4</td>
<td>108.8</td>
<td>149.2</td>
</tr>
<tr>
<td>45</td>
<td>4.5</td>
<td>10.8</td>
<td>21.0</td>
<td>36.3</td>
<td>57.6</td>
<td>85.9</td>
<td>122.3</td>
<td>167.8</td>
</tr>
<tr>
<td>50</td>
<td>5.0</td>
<td>12.0</td>
<td>23.3</td>
<td>40.3</td>
<td>64.0</td>
<td>95.5</td>
<td>135.9</td>
<td>186.5</td>
</tr>
<tr>
<td>55</td>
<td>5.6</td>
<td>13.1</td>
<td>25.7</td>
<td>44.3</td>
<td>70.4</td>
<td>105.0</td>
<td>149.5</td>
<td>205.1</td>
</tr>
<tr>
<td>60</td>
<td>6.1</td>
<td>14.3</td>
<td>28.0</td>
<td>48.4</td>
<td>76.8</td>
<td>114.6</td>
<td>163.1</td>
<td>223.7</td>
</tr>
</tbody>
</table>

of reordering the distributed arrays throughout the calculation, two copies of the [VV|OO] integrals and five copies of the [VO|VO] integrals are stored in distributed-memory in the various orders in which they are needed throughout the algorithm. This requires a one-time sorting of the [VV|OO] and [VO|VO] integrals after the integral transformation, but prior to the start of the CCSD/CCSD(T) calculation.

The [VV|VO] class of MO integrals is the largest stored quantity in the MP-CCSD(T) algorithm. The distributed-memory needed to store the [VV|VO] integrals represents the second storage bottleneck in the present algorithm. The distributed-memory requirements for the [VV|VO] integrals are given in Table 7.3. Based on the practical limits of $N_o$ and $N_v$ as governed by the shared-memory bottleneck for the $T_2$ amplitudes, the largest [VV|VO] distributed-memory arrays can approach 96 GB. MP-CCSD(T) calculations of this size represent a significant computational challenge. If one employed 128 or more processors for this type of calculation, the storage requirement for the [VV|VO] integrals per processor would be less than 1 GB. This is easily attained. To decrease the storage requirement of the [VV|VO] integrals, the permutational symmetry of the bra is exploited such that the storage require-
Table 7.4 The size of $N_v^4$ arrays in gigabytes (GB).

<table>
<thead>
<tr>
<th>$N_v$</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>60</td>
<td>191</td>
<td>466</td>
<td>966</td>
<td>1789</td>
<td>3052</td>
<td>4888</td>
<td>7451</td>
</tr>
</tbody>
</table>

The $N_v^4$ arrays are of size $[(N_v^2 + N_v) \times N_v N_o]/2$ words. When required by the algorithm, the lower triangular $([N_v^2 + N_v]/2)$ rows are expanded to a square $(N_v^2)$ set of rows after the data has been received locally. This provides a nearly two-fold reduction in the storage and communication costs, at the cost of a slight increase in computational effort. This tradeoff is logical since the computational resources often cost much less than the memory storage or communication infrastructure.

The [$VV$|$VV$] integrals are the largest class of integrals needed for a CCSD calculation; however, due to the $O(N_v^4)$ scaling of the storage requirement for these integrals, the values cannot be practically stored in distributed-memory (Table 7.1). Only one term in the CCSD equations require the use of the [$VV$|$VV$] integrals. This four-index virtual integral term scales computationally as $O(N_v^4 N_o)$ and will be referred to here as the *four-virtual term*. An efficient implementation of the four-virtual term is absolutely essential for a CCSD(T) program, since the computational effort required to evaluate the four-virtual term scales as $O(N_v^4)$ with respect to increasing the basis set. Consequently, this is the same rate at which the perturbative triples computation increases using the same metric. Most CCSD programs store the [$VV$|$VV$] integrals on disk. However, some programs [11, 12, 16] provide the ability to calculate the four-virtual term directly from AO integrals that are calculated “on the fly” rather than stored, thereby avoiding the [$VV$|$VV$] integral storage requirement. Methods that avoid storage by calculating quantities “on the fly” are called “direct methods”. Table 7.4 shows the actual storage requirement in gigabytes for the [$VV$|$VV$] class of MO integrals. As $N_v$ increases, the memory requirements for the [$VV$|$VV$] integrals exceed the distributed-memory capabilities on the vast majority of available computers. The inability to store the [$VV$|$VV$] integrals in distributed memory and the general lack of quality disk I/O on large supercomputers, led to the implementation of a “direct” four-virtual algorithm that is calculated in parallel from AO
integrals. AO driven methods, both direct and disk-based, for CC methods have been studied in the past [29][31]. Further details about the direct AO driven four-virtual term are given in Section 7.4.2.

Finally, arrays of size $N_v^3$ are required in both the CCSD and triples correction. For the majority of calculations, $N_v^3$ arrays are smaller than $N_o^2 N_v^2$ arrays; however, as $N_v$ approaches or surpasses $N_o^2$, these $N_v^3$ arrays can be similar in size (Table 7.5) or surpass the size of the $T_2$ amplitudes array (Table 7.2). It is for that reason that arrays of size $N_v^3$ are stored in shared memory and not replicated memory. All other $O(N^3)$ arrays and those of lower order are sufficiently small that they can be stored in the replicated memory of each parallel process.

### 7.4 Parallel Implementation

#### 7.4.1 CCSD

Once the Hartree-Fock calculation has converged and the molecular integrals have been calculated and sorted, the CCSD iterations begin. The first part of each CCSD iteration is the evaluation of the direct four-virtual term. The details of this direct calculation are described in Section 7.4.2. As mentioned earlier, the direct evaluation of the four-virtual term eliminates the storage requirements of the $[VV|VV]$ integral class, because the integrals are calculated “on demand” during each iteration. After the four-virtual term has been completed, the MO-based terms (Eqs. 7.7 and 7.8) are evaluated in essentially the same order as in the sequential algorithm. The order in which the terms are evaluated has been designed to reduce the number of floating point operations by maximizing the use of intermediate quantities. The sequential algorithm relies heavily on double-precision general matrix-matrix multiplication, DGEMM\textsuperscript{6}, operations for the bulk of the computational effort. The node-based parallelization strategy

---

\textsuperscript{6}DGEMM is a level 3 BLAS (Basic Linear Algebra Subroutine) library function that performs matrix multiplications.
for the DGEMM operations of the CCSD algorithm involves partitioning the DGEMM evenly by the number of nodes. Each node gets one portion of the DGEMM to work on. Then each node divides the DGEMM into equal sized work portions for each process to evaluate.

Another challenging aspect of the parallelization of the CCSD algorithm involves the location of the data, i.e. whether the data for the matrices involved in the DGEMM operation are stored in replicated, shared or distributed memory. Since the CCSD terms involve contracting integrals and amplitudes via DGEMM operations, and since $T_1$ and $T_2$ amplitudes or temporary intermediates of the same size are stored locally on each node ($T_1$ sized arrays in replicated memory and $T_2$ sized arrays in shared-memory), the distribution of the MO integrals by node is used in the first partitioning of DGEMM operations. The subsequent intra-node partitioning divides the local work among the local processes, where “local” refers to processes within a given node.

For node-based strategies, special care must be taken to ensure the data integrity of shared quantities (both shared and distributed memory arrays); i.e., before a shared quantity can be used, modified or reordered, a collective synchronization of the processes that have access to the particular quantity must occur. These collective synchronizations, also known as barriers or fences, are points within the program in which all parallel processes of a collective set must enter before any are allowed to continue executing the parallel program. The MP-CCSD(T) algorithm uses the DDI_SYNC subroutine to synchronize the entire set of parallel processes, while the DDI_SMP_SYNC subroutine is used to synchronize all parallel processes that coexist on the same physical node. These collective synchronization routines help safeguard the integrity of shared resources by ensuring that all parallel processes requiring the use of a shared resource have completed a particular task before those processes are allowed to perform new tasks using the same shared resource. An example of this in terms of distributed memory arrays is found in the four-virtual term (Section 7.4.2) where a global synchronization is used to ensure the distributed intermediate ($I^{\mu\sigma}_{ij}$) is complete before second task of forming $I^{ab}_{ij}$ from $I^{\mu\sigma}_{ij}$ is allowed to begin. However, the most common need for process synchronization occurs when using shared-memory segments within a node. As an example, the evaluation
of two CCSD terms that use different orderings of the $T_2$ amplitudes requires an intra-node synchronization to ensure that all local processes have completed the evaluation of the first term and another intra-node synchronization to ensure that the entire set of $T_2$ amplitudes are in the proper order before the evaluation of the second term can begin. This kind of lock-step synchronization can reduce the parallel efficiency of an algorithm if the work between the synchronization points is not evenly balanced.

The following is an example of a node-based algorithm for evaluating the $v_{be}^i$ component of the $I_{a}^b$ intermediate (Eq. 7.14):

1. Divide $N_o$ by the number of nodes so as to assign each node an equal amount of work.

2. Each node obtains a complete $N_v^3$ portion of the $[VV|VO]$ integrals from a GET operation based on the index calculated in the previous step, resulting in a 4-index array with dimensions $(N_v, N_v, N_v, i)$ for a given $i^{th}$ index. This array $(v_{be}^i)$ is stored in shared memory. The GET operation is performed only by the master process on each node; therefore, an intra-node synchronization is needed before and after this step.

3. Each node performs the permutation of the first and third index, using a routine that allows all the processors on the node to do the permutation in parallel, without overwriting shared memory data. To ensure data integrity, an intra-node synchronization is needed after this step is complete.

4. Each node executes a DGEMM (as a $N_v^2 \times N_v$ matrix times a $N_v \times 1$ matrix resulting in a $N_v^2 \times 1$ matrix [$I_{a}^b = v_{ae}^bi_i$ for a fixed $i$]). This DGEMM is further split among the processes on the node, by dividing $N_v^2$ (the row dimension of the first matrix) by the number of processors. The actual DGEMM executed by each process consists of a portion of the first matrix, times the entire second matrix to yield the entire resultant matrix. In this way, each process works on a different portion of the array. The second matrix and the product matrix are stored in replicated memory on each parallel process.

5. If $N_o$ is greater than the number of nodes, then some (possibly all) nodes will execute steps 2-4 again with a different portion of the $[VV|VO]$ array until the entire matrix
6. Local synchronization: A local gather operation is performed to gather the disjoint set of $N_v^2$ rows of the product matrices into a single $N_v^2$ matrix on the master process of each node.

7. The term is completed by a global sum (executed by the master process on each node) over all $N_v^2$ partial product matrices. A global sum is a form of synchronization.

The remaining terms of the CCSD equations (Eqs. (7.7) and (7.8)) have been parallelized using similar techniques to those illustrated in the above example.

In the development of this node-based approach, a similar process-based model was also developed [53]. Depending on the available memory and the size of the calculation, the MP-CCSD(T) parallel algorithm may be evaluated as a process-based algorithm or a node-based algorithm. The more traditional process-based algorithm, which divides the work based on individual processors, may achieve better intra-node performance than the node-based model by removing many of the data synchronizations required; however, the process-based algorithm has a larger memory requirement due to the necessity of more replicated temporary memory, and this significantly limits the size of a molecular problem that can be addressed. Therefore, although both process-based and node-based algorithms have been developed and implemented [53], only the node-based algorithm is discussed here, as a primary focus of this discussion to extend the size and complexity of molecular species that can be studied with CCSD(T) methods.

7.4.2 “Direct” AO-driven Four-Virtual Term

The AO “direct” four-virtual term is a distributed-memory algorithm that makes use of both node-based and process-based parallel techniques. The parallel programming techniques used to implement the four-virtual term of the MP-CCSD(T) algorithm were inspired by both the implementation by Fletcher and coworkers [50] of the direct four-index integral transformation [51] and the direct CCSD algorithm of Kobayashi et al. [29]. Diagrammatic representations
of these algorithms are illustrated in Figure 7.2 (the new MP-CCSD(T) algorithm presented here), Figure 7.3 (the [VV|OO] integral class of the integral transformation of Fletcher et al.[50]) and Figure 7.4 (four-virtual term of Kobayashi et al.[29]). The four-virtual terms of each CCSD algorithm contracts AO integrals and amplitudes to calculate the contribution of the four-virtual term to each CCSD iteration.

The four-virtual term of the MP-CCSD(T) algorithm (Figure 7.2) “directly” calculates full sets of two virtual-indexed half-transformed MO integrals \( v_{\nu\sigma}^{ab} \) for the specific shell indices \( \nu \) and \( \sigma \). The half-transformed integrals are then contracted against the \( c_{ij}^{ab} \) amplitudes \( (c_{ij}^{ab} = t_{ij}^{ab} + t_{ij}^{a}t_{ij}^{b}) \) to form a partial contribution to the set of half-transformed intermediates \( I_{ij}^{\nu\sigma} \), which are complete for a given set of atomic shells \( \nu \) and \( \sigma \) (Figure 7.2) corresponding to the parallel task (a given \( \nu-\sigma \) pair). For each parallel task the half-transformed intermediates \( I_{ij}^{\nu\sigma} \) and \( I_{ij}^{\sigma\nu} \) for \( i \geq j \) are stored in distributed memory. After the first set of parallel tasks is complete, the full set of half-transformed intermediates \( I_{ij}^{\nu\sigma} \) (for \( i \geq j \)) is stored in distributed memory. To finalize the contributions of the four-virtual CCSD term, the two remaining AO indices are transformed to the virtual MO space.

The four-virtual term gains potential performance advantages over the integral transformation on which it is modeled in two ways: an improved computation vs. communication ratio and a reduction in the total number of communication calls. The first parallel task of the four-virtual term (Figure 7.2) evaluates a larger number of AO integrals and then forms a larger set of half transformed integrals than the integral transformation in the Fletcher algorithm (Figure 7.3). In addition, the extra \( \mathcal{O}(N_o^2 N_v^2) \) contraction step makes the first parallel task of the four-virtual term of the MP-CCSD(T) algorithm significantly more computationally challenging than the first parallel task of the integral transformation. However, both methods share a similar communication profile, which places all of the communication at the end of the parallel task. In fact, the PUT operations performed for the four-virtual term in the MP-CCSD(T) algorithm communicate and store the same amount of data in distributed-memory as the integral transformation does in the formation of the [NN|OO] set of half-transformed integrals. The PUT operations in the four-virtual term of the MP-CCSD(T) algorithm actually gain a
Figure 7.2 A diagrammatic description of the four-virtual term in the MP-CCSD(T) algorithm. The left hand portion of the diagram is pseudo-code, while the right hand portion is figure illustrates a distributed array. The columns of the distributed array correspond to two occupied indexes, where the total number of columns is \((N_oN_o)^*\). \((N_oN_o)^*\) refers to the lower triangular portion (including diagonal elements) of an \(N_o^2\) matrix. The number of rows in the distributed matrix is \(N_o^2\). The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms the half-transformed intermediate \((I_{ij}^{\nu\sigma})\) in distributed memory. A global synchronization is used to ensure \(I_{ij}^{\nu\sigma}\) is complete before the second parallel task begins. The second parallel task transforms \(I_{ij}^{\nu\sigma}\) into \(I_{ij}^{ab}\) for all “local” \(i-j\) columns.
Figure 7.3 A diagrammatic description of the Fletcher [50] four-index integral transformation for the [VV|OO] integral class. The left hand portion of the diagram is pseudo-code, while the right hand portion illustrates a distributed array. The columns of the distributed array correspond to two occupied indexes, where the total number of columns is \((N_o N_o)^*\). \((N_o N_o)^*\) refers to the lower triangular portion (including diagonal elements) of an \(N_o^2\) matrix. The number of rows in the distributed matrix is \(N_o^2\). The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms half-transformed integrals over two occupied indexes for a given set of two AO shell indexes. The half-transformed integrals are stored in the distributed array. A global synchronization is used to ensure the first task is complete before the second parallel task begins. The second parallel task transforms the final two AO indexes into virtual MO indexes.
slight edge over the PUT operations of the integral transformation in that in the former, only one PUT operation is performed at the end of the first parallel task of the four-virtual term. In contrast, potentially two PUT operations are performed in the integral transformation, except for a single PUT operation when $\nu = \sigma$. Due to the larger computational profile of the four-virtual term and a communication profile that is similar to the integral transformation, the four-virtual term of the MP-CCSD(T) algorithm is expected to be as good or better in terms of computational efficiency when compared to the integral transformation. The latter has previously been shown to be highly efficient up to 512 processors [54].

Both distributed-memory CCSD algorithms examined herein form the half transformed intermediate $I_{ij}^{\nu\sigma}$ of the four-virtual term in distributed memory (Figures 7.2 and 7.4). The major difference between the two algorithms is the communication profile. In the four-virtual term of Kobayashi et al. [29], the communication calls (GET and ACC) are performed on the inner most nested loop (Figure 7.4. This type of algorithm was shown to be very successful on the Cray T3E. However, the Cray T3E is very different from modern HPC platforms in that the performance of modern processors have increased by more than an order of magnitude, while the performance of the communication networks have at best doubled or tripled since the benchmarks on the T3E. Therefore the communication heavy inner loop $[O(N^4) \text{ communication calls}]$ is less favorable on modern MP platforms due to this growing discrepancy of the communication network compared to the available computational power. The main benefit of the MP-CCSD(T) routine is that the communication operations are performed at the end of the parallel task making the number of communication calls scale as $O(N^2)$ (Figure 7.2). The GET operation of the Kobayashi et al. [29] method is avoided completely by the storage of the $c_{ij}^{ab}$ amplitudes in shared-memory once on every node. The ACC operation of the Kobayashi et al. [29] method is replaced by a less expensive PUT operation, since the set of $I_{ij}^{\nu\sigma}$ are complete for each set of $\nu$ and $\sigma$.

The diagrammatic description of the four-virtual term in the MP-CCSD(T) method (Figure 7.2) is a general description of the algorithm. The actual algorithm as programmed in GAMESS incorporates an extra step to further optimize the first parallel task (see Figure 7.5). To
Figure 7.4 A diagrammatic description of the four-virtual term of Kobayashi et al.[29]. The left hand portion of the diagram is pseudo-code, while the right hand portion illustrates the distributed arrays. The columns of the distributed arrays correspond to two AO indexes where the total number of columns is \((N_{bf}N_{bf})^*\). \((N_{bf}N_{bf})^*\) refers to the lower triangular portion (including diagonal elements) of an \(N_{bf}^2\) matrix. The number of rows in the distributed matrix is \((N_oN_o)^*\) corresponding to the lower triangular portion of an \(N_o^2\) matrix. The columns are distributed evenly over the total number of parallel processes. The boxed portions of the pseudo-code represent load-balanced parallel tasks. The first half of the pseudo-code forms the half-transformed intermediate \((I_{ij}^{\nu\sigma})\) in distributed memory. A global synchronization is used to ensure \(I_{ij}^{\nu\sigma}\) is complete before the second parallel task begins. The second parallel task transforms \(I_{ij}^{\nu\sigma}\) into \(I_{ij}^{ab}\) for all “local” \(i-j\) columns.
Figure 7.5 A more detailed overview of the first parallel task in Figure 7.2 describing the use of a temporary buffer to store half-transformed integrals. The left-hand portion of the figure is the pseudo-code describing how the buffer is filled. The right-hand portion describes the “Contract and PUT” operation on the buffer. The description of the distributed array is the same as in Figure 7.2.
maximize the efficiency in the contraction step, a local buffer is used to store multiple sets of half-transformed integrals prior to the DGEMM operation. Without the use of the buffer, the size of the DGEMM operation is a function of the size of the basis set shells $\nu$ and $\sigma$. When $\nu$ and $\sigma$ are $s$-shells, the DGEMM contraction step reduces to a less than optimal DGEMV (matrix times vector) operation. By locally buffering sets of half-transformed integrals (Figure 5), the efficiency of the DGEMM operation is increased because larger more efficient DGEMM operations are calculated rather than multiple sets of smaller less efficient DGEMV operations. The PUT operation for each set of $\nu$ and $\sigma$ is then performed for each $\nu$-$\sigma$ pair in the contracted buffer.

7.4.3 Triples Correction, MP-(T)

The (T) portion of the MP-CCSD(T) algorithm is more straightforward to parallelize than the CCSD component. It consists of three nested loops, each of size $n_i$, $n_j$, $n_k$ with $i \geq j \geq k$, where $i$, $j$ and $k$ are actively occupied indexes. Within each loop, 36 DGEMM calls are made, the largest of which scales computationally as $O(N_v^4)$ and corresponds to DGEMM operation where a $N_v \times N_v$ matrix is multiplied by a $N_v \times N_v^2$ matrix. One feature of the (T) algorithm is that the loop iterations can be performed independently of each other, thus the algorithm can be easily partitioned into unique parallel tasks. The node-based (T) algorithm partitions these independent tasks in terms of sets based on unique values of $i$, $j$ and $k$ (occupied indexes), where each task is evaluated on a node. Two $N_v^3$ temporary arrays are stored one time per node in shared memory. Similar to the parallelization scheme of the MO-based MP-CCSD algorithm, when a computationally intensive routine (such as a permutation or DGEMM) is encountered, the work is partitioned equally among the intra-node processes, with strict control maintained to avoid overwriting shared memory array locations by multiple processors.

The intra-node scaling of the MP-(T) algorithm is expected to exhibit similar trends to those of the MP-CCSD algorithm, since the lock-step synchronization needed between the intra-node processes within the node-based tasks are similar. However, due to the larger amount of computational effort per parallel task, the MP-(T) algorithm is expected to perform better.
In general, the MP-(T) algorithm has a large number of independent tasks that are similar to the four-virtual algorithm; however, unlike the four-virtual algorithm, the MP-(T) algorithm does not evaluate the integrals it requires on demand. Rather, it fetches them via GET operations. This aspect of the MP-(T) algorithm increases the communication overhead of the algorithm; however, the $O(N^4_v)$ effort within each parallel task easily compensates to allow for a favorable computational vs. communication ratio. Therefore, good inter-node scalability is expected from the MP-(T) routine.

7.5 Computational Details

The starting set of geometries for the five water hexamer isomers (prism, cage, book, ring, and boat) were obtained from Day et al. [43] (Figure 7.1). In that work, the geometries of water hexamer were optimized with second-order Møller-Plesset perturbation theory (MP2) [45] using the double-zeta Dunning-Hay [55] [DH(d,p)] basis set. In the present work, single-point CCSD(T) energies were calculated at each previously optimized structure using the following one-electron basis sets: aug-cc-pVTZ [39] and aug'-'cc-pVTZ, where aug'-'cc-pVTZ is a mixed basis set that uses aug-cc-pVTZ on the oxygen atoms and cc-pVTZ [46] on the hydrogen atoms. The MP-CCSD(T) method in GAMESS was used for all CCSD(T) calculations. A cluster of three IBM Power4 compute nodes each containing eight 1.7 GHz Power4 processors and 32 GB of memory connected by TCP/IP over an InfiniBand network was used to perform the MP-CCSD(T) calculations.

To evaluate the performance of the MP-CCSD(T) algorithm, a series of CCSD(T)/aug'-'cc-pVTZ calculations were performed on the MP2/DH(d,p) optimized prism isomer, and the parallel execution times were measured. To test intra-node scalability, a set of CCSD(T)/aug'-'cc-pVTZ//MP2/DH(d,p) energies were calculated using a single node; the number of parallel processes was varied from 1 to 8 in powers of 2. Inter-node scalability measures the changes in parallel runtime as the number of nodes is increased, while the number of parallel processes per node (1, 2, 4 or 8) is fixed. In terms of $N_o$, $N_v$ and the number of Cartesian basis functions ($N_{bf}$), the size of the aug-cc-pVTZ calculation is $N_o = 24$, $N_v = 516$ and $N_{bf} = 630$. The size
of the aug'-cc-pVTZ calculation is \( N_o = 24 \), \( N_v = 408 \) and \( N_{bf} = 510 \).

### 7.6 Discussion

#### 7.6.1 Water Hexamer

Calculations performed on the isomers of the water hexamer were used to test the MP-CCSD(T) algorithm. In the first step of what will be a more extensive study of water clusters, CCSD(T) single point energies using the aug-cc-pVTZ and the aug'-cc-pVTZ basis sets were calculated at the MP2 optimized geometries of Day et al. [43]. The absolute energies, binding energies per \( \text{H}_2\text{O} \) and relative binding energies are given in Table 7.6 for calculations using the aug-cc-pVTZ basis set and for calculations using the aug'-cc-pVTZ basis set in Table 7.7. These calculations represent, to the authors’ knowledge, the largest CCSD(T) calculations performed on water hexamers to date.

The MP2/DH(d,p) geometries of Day et al. [43] used in this study may not be as accurate as the MP2/aug-cc-pVTZ geometries of Xantheas et al. [41]; however, the differences in the binding energies for the two sets of geometries (Table 7.6) are very small: < 0.1 kcal/mol for the prism, cage and cyclic isomers, and 0.4 kcal/mol for the book isomer. The latter suggests that calculations reported below based on MP2/DH(d,p) geometries for the book isomer may not be as accurate as those for the prism, cage and cyclic isomers. Xantheas and coworkers did not examine the boat structure.

A main point of interest in this study is the difference between the CCSD(T) and MP2 methods. Column 1 of Table 7.8 shows the difference in CCSD(T) vs. MP2 relative binding energies; positive values indicate an increase in the energy difference between an isomer and the lowest energy prism isomer, i.e. the value in which the prism isomer is stabilized by the CCSD(T) method. In general, CCSD(T) and MP2 predict very similar binding energies. CCSD(T) moderately stabilizes the prism structure with respect to the cage and other higher energy isomers. The prism isomer is stabilized by 0.2 kcal/mol over the next lowest-energy cage isomer. For higher energy isomers, the difference in relative binding energies is larger: 0.4 kcal/mol for the book isomer and 0.6 kcal/mol for the cyclic and boat isomers. While
Table 7.6 Total energies (Hartree), binding energies (kcal/mol) and relative binding energies (kcal/mol) using the aug-cc-pVTZ basis set on the MP2/DH(d,p) optimized structures of Day et al.[43]. The binding energies represent the energy difference between the water hexamer isomer and six isolated water molecules. MP2* represents the MP2/aug-cc-pVTZ calculations from Xantheas et al. [41] who did not examine the boat isomer.

<table>
<thead>
<tr>
<th></th>
<th>CCSD(T)</th>
<th>CCSD</th>
<th>MP2</th>
<th>MP2*</th>
</tr>
</thead>
<tbody>
<tr>
<td>prism</td>
<td>-458.13045167</td>
<td>-458.07282232</td>
<td>-458.05015535</td>
<td>-458.05035804</td>
</tr>
<tr>
<td>cage</td>
<td>-458.13001003</td>
<td>-458.07248668</td>
<td>-458.05001662</td>
<td>-458.05017138</td>
</tr>
<tr>
<td>cyclic</td>
<td>-458.12704114</td>
<td>-458.07054907</td>
<td>-458.04769324</td>
<td>-458.04785303</td>
</tr>
<tr>
<td>boat</td>
<td>-458.12514762</td>
<td>-458.06876898</td>
<td>-458.04579806</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CCSD(T)</th>
<th>CCSD</th>
<th>MP2</th>
<th>MP2*</th>
</tr>
</thead>
<tbody>
<tr>
<td>prism</td>
<td>-48.1</td>
<td>-44.6</td>
<td>-47.9</td>
<td>-47.9</td>
</tr>
<tr>
<td>cage</td>
<td>-47.8</td>
<td>-44.4</td>
<td>-47.8</td>
<td>-47.8</td>
</tr>
<tr>
<td>book</td>
<td>-46.9</td>
<td>-43.7</td>
<td>-47.1</td>
<td>-47.5</td>
</tr>
<tr>
<td>cyclic</td>
<td>-46.0</td>
<td>-43.2</td>
<td>-46.4</td>
<td>-46.4</td>
</tr>
<tr>
<td>boat</td>
<td>-44.8</td>
<td>-42.1</td>
<td>-45.2</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CCSD(T)</th>
<th>CCSD</th>
<th>MP2</th>
<th>MP2*</th>
</tr>
</thead>
<tbody>
<tr>
<td>prism</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>cage</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>book</td>
<td>1.2</td>
<td>0.9</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>cyclic</td>
<td>2.1</td>
<td>1.4</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>boat</td>
<td>3.3</td>
<td>2.5</td>
<td>2.7</td>
<td>n/a</td>
</tr>
</tbody>
</table>

† With respect to the lowest-energy isomer (prism).
Table 7.7 Total energies (Hartree), binding energies (kcal/mol) and relative binding energies (kcal/mol) using the aug'-cc-pVTZ basis set on the MP2/DH(d,p) optimized structures of Day et al.[43]. The binding energies represent the energy difference between the water hexamer isomer and six isolated water molecules.

<table>
<thead>
<tr>
<th>Isomer</th>
<th>Total Energies (Hartree)</th>
<th>Binding Energies (kcal/mol)</th>
<th>Relative Binding Energies (kcal/mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CCSD(T)</td>
<td>CCSD</td>
<td>MP2</td>
</tr>
<tr>
<td>prism</td>
<td>-458.12255430</td>
<td>-458.06558835</td>
<td>-458.04247161</td>
</tr>
<tr>
<td>cage</td>
<td>-458.12223107</td>
<td>-458.06536922</td>
<td>-458.04244064</td>
</tr>
<tr>
<td>cyclic</td>
<td>-458.11967703</td>
<td>-458.06381033</td>
<td>-458.04052628</td>
</tr>
<tr>
<td>boat</td>
<td>-458.11779063</td>
<td>-458.06203960</td>
<td>-458.03863436</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Isomer</th>
<th>CCSD(T)</th>
<th>CCSD</th>
<th>MP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>prism</td>
<td>-46.6</td>
<td>-43.2</td>
<td>-46.6</td>
</tr>
<tr>
<td>cage</td>
<td>-46.4</td>
<td>-43.1</td>
<td>-46.6</td>
</tr>
<tr>
<td>book</td>
<td>-45.6</td>
<td>-42.5</td>
<td>-45.9</td>
</tr>
<tr>
<td>cyclic</td>
<td>-44.8</td>
<td>-42.1</td>
<td>-45.4</td>
</tr>
<tr>
<td>boat</td>
<td>-43.6</td>
<td>-41.0</td>
<td>-44.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Isomer</th>
<th>CCSD(T)</th>
<th>CCSD</th>
<th>MP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>prism</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>cage</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>book</td>
<td>1.0</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>cyclic</td>
<td>1.8</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>boat</td>
<td>3.0</td>
<td>2.2</td>
<td>2.4</td>
</tr>
</tbody>
</table>

† With respect to the lowest-energy isomer (prism).
Table 7.8 The difference in relative binding energies between the CCSD(T), CCSD and MP2 methods with respect to basis set measured in kcal/mol. The first column [CCSD(T)-MP2] shows how the relative binding energies differ between the CCSD(T) and MP2 method. The second column [CCSD(T)-CCSD] shows the effect of the triples correction on the relative binding energies. And the last column [CCSD-MP2] shows the differences between CCSD and MP2 on the relative binding energies. The section subtitled “Difference” subtracts the values from the aug-cc-pVTZ set from the corresponding aug’-cc-pVTZ set.

<table>
<thead>
<tr>
<th></th>
<th>aug-cc-pVTZ</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CCSD(T)-MP2</td>
<td>(T)-CCSD</td>
<td>CCSD-MP2</td>
<td>prism</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>cage</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>book</td>
<td>0.39</td>
<td>0.33</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cyclic</td>
<td>0.60</td>
<td>0.71</td>
<td>-0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>boat</td>
<td>0.59</td>
<td>0.78</td>
<td>-0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>aug’-cc-pVTZ</td>
<td></td>
<td></td>
<td>prism</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>cage</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>book</td>
<td>0.39</td>
<td>0.32</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cyclic</td>
<td>0.58</td>
<td>0.69</td>
<td>-0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>boat</td>
<td>0.58</td>
<td>0.76</td>
<td>-0.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Difference</td>
<td></td>
<td></td>
<td>prism</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>cage</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>book</td>
<td>0.01</td>
<td>0.01</td>
<td>0.00</td>
<td>cyclic</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>boat</td>
<td>0.01</td>
<td>0.02</td>
<td>-0.01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 7.9 Estimated CCSD(T)/aug-cc-pVTZ binding energies (kcal/mol) using Eq. (7.29) compared to the actual CCSD(T)/aug-cc-pVTZ at the MP2/DH(d,p) optimized geometries. The differences were rounded up.

<table>
<thead>
<tr>
<th></th>
<th>prism</th>
<th>cage</th>
<th>book</th>
<th>cyclic</th>
<th>boat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Est. CCSD(T)/aug-cc-pVTZ</td>
<td>-47.9</td>
<td>-47.6</td>
<td>-46.7</td>
<td>-45.7</td>
<td>-44.6</td>
</tr>
<tr>
<td>Actual CCSD(T)/aug-cc-pVTZ</td>
<td>-48.1</td>
<td>-47.8</td>
<td>-46.9</td>
<td>-45.9</td>
<td>-44.8</td>
</tr>
<tr>
<td>Error</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

the differences in relative binding energies between CCSD(T) and MP2 (0.2-0.6 kcal/mol) are modest when the aug-cc-pVTZ basis set is used, it is unclear how basis set improvements will affect these energy differences.

Another interesting issue is the accuracy of the CCSD method with respect to CCSD(T) and MP2. The CCSD(T) and MP2 binding energies agree to within 0.5 kcal/mol for both the aug- and aug'-cc-pVTZ basis sets (Table 7.6 and Table 7.7). However, the CCSD binding energies differ from the CCSD(T) binding energies by 2.7-3.5 kcal/mol (Table 7.6 and Table 7.7). Assuming that CCSD(T) provides the most accurate binding energies, these calculations suggest that the MP2 method can more accurately predict the binding energies than the CCSD method. Kim et al. [44] reported such a difference between CCSD and MP2 for cyclic water hexamer. This is surprising, since the CCSD method is often considered to be more reliable than MP2.

Table 7.8 describes in more detail how the prism isomer is stabilized by the CCSD(T) method based on differences between CCSD(T) and CCSD (column 2) and differences between CCSD and MP2 (column 3). The triples correction to the CCSD energy (Column 2, Table 7.8) plays an increasingly larger role in stabilizing the prism structure relative to higher energy isomers. The difference between CCSD and MP2 (Column 3, Table 7.8) stabilizes the prism structure over the cage and book structures, but decreases the stability of the prism structure relative to the cyclic and boat structures. The effects of the triples approach 1 kcal/mol and should not be overlooked, especially for larger water clusters.

The two basis sets employed here exhibit very similar trends in the differences of relative
binding energies for all methods. Csonka and coworkers [42] have suggested that including diffuse functions in the oxygen atom basis set is important. In the present work, the aug'-cc-pVTZ basis set only includes diffuse functions on the oxygen atoms. The omitted hydrogen diffuse functions in the aug'-cc-pVTZ basis set were found to increase the binding energies of the water clusters by 1.0-1.4 kcal/mol (Table 7.6 and Table 7.7); therefore the diffuse functions on the hydrogen atoms do seem to be important for calculating the absolute binding energies. However, the relative binding energies (Table 7.8) predicted by the aug- and aug'-cc-pVTZ basis sets are very similar. For example, column 1 of Table 7.8 describes the differences in relative binding energies between CCSD(T) and MP2. These values are virtually identical for each isomer for both basis sets. This consistency in the differences between CCSD(T) and MP2 for the aug- and aug'-cc-pVTZ basis sets suggests that the CCSD(T)/aug-cc-pVTZ binding energies can be accurately estimated using computationally less intensive CCSD(T)/aug'-cc-pVTZ and MP2/aug-cc-pVTZ calculations. As illustrated in Table 7.9, the following additive scheme:

\[
CCSD(T)/\text{aug} = CCSD(T)/\text{aug}^{\prime} + [MP2/\text{aug} - MP2/\text{aug}^{\prime}]
\]  

(7.29)

where the -cc-pVTZ extension to the basis set is implied, estimates the actual CCSD(T)/aug-cc-pVTZ binding energies to within less than 0.2 kcal/mol. A future study will examine the extrapolation of the CCSD(T) binding energies of water hexamer isomers to the complete basis set limit (CBS).

7.6.2 Parallel Performance

The speedup and efficiency values for the four virtual term (CCSD-AO) and the remaining MO terms (CCSD-MO) from the MP-CCSD method on the benchmark calculation run on the IBM Power4 platform are given in Table 7.10. Speedup is defined as the ratio of the measured execution time to the execution time on a single processor; efficiency is the ratio of the measured speedup compared to the ideal speedup.

The intra-node scalability of the MP-CCSD method was measured by the speedup and efficiency of the benchmark calculation as the number of processors within a single node was
Table 7.10 Parallel speedup (S) and parallel efficiency (E) for the MP-CCSD(T) algorithm as a function of the number of processors per node (PPN) and the number of nodes for calculations performed on the prism isomer using the aug-cc-pVTZ basis set. CCSD-AO represents the AO-driven four virtual term of the MP-CCSD algorithm; CCSD-MO represents all the other MO-based terms of the MP-CCSD algorithm. CCSD-Total is the overall scalability for MP-CCSD algorithm. The speedup and efficiency is also given for the triples correction. Intra-node trends are observed across rows, while inter-node trends are observed down the columns. The benchmark calculations are based on MP-CCSD(T) calculations of the water hexamer (prism isomer) with $N_o=24$, $N_v=408$ and $N_{bf}=510$ run on nodes containing a total of 8 processors.

| Processes per node | 1 Node | | | | | | 2 Nodes | | | | | | 3 Nodes |
|-------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|                   | 1      | 2      | 4      | 8      | 1      | 2      | 4      | 8      | 1      | 2      | 4      | 8      | 1      | 2      | 4      | 8      |
|                   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   | S  E   |
| CCSD-AO           | 1.00   | 100%   | 1.90   | 95%   | 3.70   | 92%   | 6.18   | 77%   | 2.00   | 100%   | 3.76   | 94%   | 7.43   | 93%   | 12.31  | 77%   |
| CCSD-MO           | 1.00   | 100%   | 1.87   | 93%   | 3.11   | 78%   | 4.21   | 53%   | 1.38   | 69%    | 2.46   | 62%   | 4.10   | 51%   | 6.21   | 39%   |
| CCSD-Total        | 1.00   | 100%   | 1.86   | 93%   | 3.58   | 89%   | 5.68   | 71%   | 1.68   | 94%    | 3.34   | 84%   | 6.53   | 82%   | 9.56   | 60%   |
| Triples Correction (T) | 1.00 | 100% | 1.78 | 89% | 2.59 | 65% | 4.06 | 51% | 1.94 | 97% | 3.38 | 85% | 4.73 | 59% | 7.13 | 45% |
|                   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| CCSD-AO           | 2.00   | 100%   | 3.76   | 94%   | 7.43   | 93%   | 12.31  | 77%   | 3.00   | 100%   | 5.85   | 97%   | 11.07  | 92%   | 18.48  | 77%   |
| CCSD-MO           | 1.38   | 69%    | 2.46   | 62%   | 4.10   | 51%   | 6.21   | 39%   | 1.68   | 56%    | 2.96   | 49%   | 4.56   | 38%   | 6.91   | 29%   |
| CCSD-Total        | 1.88   | 94%    | 3.34   | 84%   | 6.53   | 82%   | 9.56   | 60%   | 2.55   | 85%    | 4.80   | 80%   | 8.28   | 69%   | 14.57  | 61%   |
| Triples Correction (T) | 1.94 | 97% | 3.38 | 85% | 4.73 | 59% | 7.13 | 45% | 2.95 | 98% | 5.24 | 87% | 7.63 | 64% | 11.82 | 49% |
increased. The intra-node scalability of the AO driven four-virtual term (CCSD-AO) is better than 90% of ideal over two and four processors within one node; however, the efficiency drops to approximately 77% when all eight processors within the node are used (Table 7.10). The drop in performance when using all eight processors with a single node is likely due to memory bandwidth limitations; i.e. all eight processors within the node were accessing and utilizing the same local system memory. The scalability of the MO based terms of the MP-CCSD algorithm is approximately 93%, 77% and 52% efficient when run on 2, 4 and 8 processors, respectively, within the same node (Table 7.10). The intra-node scalability of the MO based MP-CCSD terms suffers due to the high degree of synchronization needed between local processes; the lock-step manner in which the terms are calculated results in deviations from ideal speedup. The MO-based terms also require a significant number of cache unfriendly rearrangements of the $T_2$ amplitudes. These operations, similar to the four-virtual term, stress the memory bandwidth of the system and result in less than ideal scalability.

The inter-node scalability of the MP-CCSD method was measured as the number of nodes was increased, while the number of processors per node (PPN) was kept fixed. The inter-node scalability of the AO driven four-virtual term (CCSD-AO) is extremely good (Table 7.10), i.e. the parallel efficiency measured on one node stays approximately the same as the number of nodes is increased. This high degree of inter-node scalability is expected because very little communication is required relative to the amount of computational effort needed for the four-virtual term. The inter-node speedup is expected to extend well beyond three nodes, since the four-virtual term was modeled upon, and has a better computational vs. communication ratio than, the direct four-index integral transformation.

The inter-node scalability of the MO based terms suffers due to a low computation vs. communication ratio. As mentioned earlier, the MO terms of the MP-CCSD method require a high degree of synchronization. Some of these synchronization points in the MO based MP-CCSD algorithm are collective operations which require a considerable amount of network communication. The lower computation vs. communication ratio resulting from higher inter-node communication, combined with smaller computational workloads, significantly reduces
the inter-node scalability of the MO based terms in the MP-CCSD program.

Despite the poor scaling of the MO-based terms, reasonable overall scalability is achieved for the MP-CCSD algorithm due to the highly scalability and overwhelmingly dominant four-virtual term. On a single processor, 88% of the execution time of the MP-CCSD algorithm was spent calculating the four-virtual term in the benchmark calculation. The outlook for the MP-CCSD algorithm for larger calculations is good, since the four-virtual term becomes increasingly dominant for larger calculations.

The performance of the triples (T) correction in the MP-CCSD(T) algorithm falls in between that of the four-virtual term and the MO-based terms of the MP-CCSD algorithms. Similar to the four-virtual term, the MP-(T) algorithm scales well as the number of nodes is increased; i.e. the efficiency does not change considerably as the number of nodes is increased for a given number of processes per node (PPN). This is expected due to the general independence of the work distribution. The intra-node scaling of the (T) part suffers in the benchmark calculation from intra-node synchronization and a relatively small value of $N_v$. That is $N_v = 408$ is at the low end of the range of $N_v$ for which the algorithm was designed $(300 \leq N_v \leq 1000)$, and as such, the computational effort needed to evaluate the intra-node parallel DGEMMs does not scale well because the subdivided DGEMMs evaluated per process are too small in size to gain a significant advantage from the highly optimized BLAS library. In the benchmark calculations, the parallel efficiency to drops to just under 90% for PPN=2, 65-70% for PPN=4 and approximately 50% for PPN=8 (Table 7.10). Larger values of $N_v$ would provide a greater amount of computational work and better scaling is expected.

Overall, the MP-CCSD(T) routine is dominated by two key terms: the four-virtual term and the (T) term. The intra-node scaling of both of these terms is the major performance limiting variable. However, based on a fixed number of processors per node, the inter-node scaling is very good. Therefore, in general, a constant speedup is expected as the number of nodes is increased, even though this speed up is less than ideal due to the less than desirable intra-node scaling.
7.7 Future Enhancements

The four-virtual term was designed based on the premise that quality disk I/O would not be generally available, so the method, as presented, is a fully direct algorithm. This decision was deliberate since many of the next-generation MP platforms may not have local scratch disks. However, a considerable saving in the cost of recalculating the AO integrals might be achieved by making use of local scratch disk to store integrals or intermediates. One way in which local scratch disk could be utilized to reduce the computational cost of recalculating the AO integrals would be to selectively store those sets of half-transformed integrals that are the most expensive to recalculate. Using the angular momentum quantum number \( l \) for the basis set shells \( \nu \) and \( \sigma \), then for each set of \( \nu \) and \( \sigma \) in which the sum \( l(\nu) + l(\sigma) \) is larger than a user defined input parameter, the half-transformed integrals for the set of \( \nu \) and \( \sigma \) are saved on the local disk during the first CCSD iteration. Subsequent CCSD iterations process all “local” sets of half-transformed integrals stored on disk before processing the remaining sets of half-transformed integrals that must be calculated directly. There are a variety of ways in which load balancing might be achieved in such a scheme. One method would be to statically distribute the disk-based tasks while dynamically distributing the direct tasks. This would ensure that a similar amount of scratch disk is used on each node while the dynamically distributed direct task would compensate for any potential load imbalances from the disk-based portion of the algorithm.

The limited scalability of the MO-based terms of the MP-CCSD method represents one of the major limitations in the current MP-CCSD algorithm. Each CCSD iteration performs the computationally demanding four-virtual term using every parallel process and when complete, every parallel process is then used to calculate the MO-based terms. Since the four-virtual term scales extremely well with the number of parallel processes, it is desirable to utilize a large number of CPUs to gain significant computational speedup. However, since the MO-based terms reach asymptotic scaling with significantly fewer processes, performing these operations sequentially results in a loss of efficiency due to the MO-based terms. To compensate for this limitation, the MO-based terms could be calculated concurrently with the AO-based terms.
Using \( n \) nodes to calculate the MO-based terms, where \( n \) is the maximum number of nodes for which the MO-based terms achieve better than 50-75% parallel efficiency, the remaining nodes would then immediately begin work on the computationally dominant AO-based terms. Since the AO-based tasks are so computationally dominant, the \( n \) nodes used to calculate the MO-based terms could potentially finish before the AO-based terms are completed. In that case, those nodes would assist in the completion of the AO-based terms. This scheme would maximize the efficiency of the MP-CCSD algorithm.

Finally, improvements in the intra-node scaling would benefit every step in the MP-CCSD(T) algorithm. However, in terms of an overall reduction in wall time, the biggest computational saving could be gained by improving the intra-node performance of the MP-(T) algorithm. One means of improving the intra-node performance of the MP-(T) algorithm (and also the intra-node MP-CCSD algorithm) is to explore the use of a shared-memory model based on threads rather than processes. Thread-based models like OpenMP [25] and/or POSIX threads (Pthreads) offer a greater set of tools which are generally more robust and better performing than the limited capabilities of the System V model. Improved synchronization routines and better tuning of the intra-node portion of the (T) algorithm should result in the biggest overall performance improvements.

7.8 Conclusions

The MP-CCSD(T) algorithm was shown to achieve reasonable scalability for chemically interesting systems, i.e. water hexamer. The most computationally challenging portions of the algorithm, the four-virtual term and the triples corrections, achieve good inter-node scalability, which implies that the performance will scale well up to a large number of nodes. In general, the intra-node scalability for both the MP-CCSD and MP-(T) was found to be less than optimal. However, it was only due to the use of the node-based model that provided the ability to perform these calculations by making it possible to store all of the various data structures. Careful consideration of the data and storage model is as crucial to the algorithm design as is CPU scaling.
The CCSD(T) calculations on isomers of water hexamer show good agreement between the CCSD(T) and MP2 methods, while the CCSD method predicts significantly worse binding energies than either CCSD(T) or MP2. While the differences between the CCSD(T) and MP2 methods are small, these differences could be important at the CBS limit or for larger water clusters, since the geometric isomers are themselves very similar in energy. Diffuse functions on the hydrogen atoms are important for calculating accurate binding energies; however, the contributions of these diffuse functions to the total energy of higher level methods, like CCSD(T), can be accurately estimated using energy differences from calculations performed at a lower level of theory, e.g. MP2.

Overall, the MP-CCSD(T) algorithm offers a node-based parallel algorithm designed to take advantage of modern cluster of SMPs. With the ever increasing trend towards more intra-node compute power, mostly notably with advent of multi-core processors, the distinction between inter-node and intra-node parallelism will become more important. The present work provides an initial analysis of how effectively this dual-level parallelism can be applied to modern state-of-the-art ab initio methods. While further optimizations to improve the algorithm, especially the intra-node portions, should be considered, the MP-CCSD(T) method presented here is capable of calculating CCSD(T) energies for system up to approximately 1,000 basis functions in a massively parallel environment.

Acknowledgements

The authors are grateful for many illuminating discussions with Professor Alistair Rendell. The authors thank the Scalable Computing Laboratory of the Ames Laboratory for the generous allotment of computer resources allocated for use on this project. The authors also thank Professor Donald Truhlar and Ms. Erin Dalke for helpful discussion regarding basis sets, in particular the aug-cc-pVTZ basis set. This work was supported by an ITR (Information Technology Research) grant from the National Science Foundation.
Bibliography


PART III

THEORY PROJECT
CHAPTER 8. Selected Orbital Subspace Methods

A paper to be submitted for publication at a later date.

Ryan M. Olson, Mark S. Gordon

Abstract

The focus of this research is to develop a method that can be used to calculate accurate energies for chemical processes that occur in spatially localized active regions of “large” molecular systems, where “large” is defined as a size that is greater than the largest chemical system for which the \textit{ab initio} method needed to achieve the desired accuracy can be computationally evaluated. The general principle of this method is to use a highly accurate \textit{ab initio} method in the chemically important “active” region(s) to achieve some level of desired accuracy, while maintaining the important properties of the “bulk” regions, which are treated using a lower level of theory. A key component of this method that differentiates it from other similar methods is the use of localized molecular orbitals as a means to spatially select and treat the active region in such a manner that (a) the integrity of the “local” wave function of the active region is accurately preserved and (b) important influences from the bulk regions are included. Using the selected localized molecular orbitals for the active region(s), electron correlation methods, including many popular methods such as second order perturbation theory (MP2) or coupled cluster with singles and double excitations (CCSD), can be employed to calculate accurate energies for reactions that occur within the active region(s).
8.1 Introduction

For many molecular systems, chemical reactions occur in a well-defined spatially localized region. An example is insertion of some molecular species into an alkene homologous series, in which the substituents range from H to t-Bu to mesityl. The “active” region, in this case the double bond in the alkene plus the relevant bonds in the attacking species, is generally distinct from the “bulk” substituent region, even though the two regions influence each other. The relationship between the active regions and the bulk can vary greatly. If a chemical reaction that occurs at the active region can be accurately described without directly including the interaction of the surrounding bulk regions, the active region is chemically independent of the bulk. This will generally be an unusual circumstance, since the role of the bulk will often be important. The bulk region, for example, might serve as an interface between the active region and some surrounding environment, it might sterically distinguish between possible reactants which are allowed, or not allowed, to reach the active region, or it might exert electronic or electrostatic influence on the behavior that occurs in the active region, in either a promotional or inhibitory manner. Therefore, in general, the bulk regions need to be properly accounted for in order to ensure an accurate prediction of the chemistry that occurs in the active region. The need to directly include the bulk effects within the theoretical model of the active region can have significant ramifications, especially in terms of the compromise between chemical accuracy and computational feasibility that must be addressed as the size of the model increases.

Because of the growing interest in accurately studying the chemistry that occurs at active regions within larger chemical systems, e.g. biomolecules, polymers and large inorganic complexes, this research describes a new method for utilizing high-level *ab initio* electronic structure theory to calculate accurate energies for chemical processes occurring at spatially localized active regions. Traditionally, the use of high-level *ab initio* electronic structure methods including configuration interaction (CI), perturbation theory (PT) and/or coupled cluster (CC) approaches to treat electron correlation have been limited to considerably smaller systems than those of interest in this study, because the computational effort needed to compute electron correlation using these methods scales rapidly with system size.
Electron correlation is essential for calculating accurate energies for chemical processes. However, one can argue that electron correlation is a spatially local phenomenon [1]. For large chemical systems in which a reaction in the active region does not induce “significant” conformational changes to the bulk region, one can anticipate that the important changes in the correlation energy between the reactant and the product can be approximated by the differences in the correlation energy within the spatially localized active region between reactant and product, and perhaps the transition structure that connects them. That is, the difference in the correlation energy of the bulk is conserved if there is little to no change in the geometry of the bulk regions.

This does not imply that the interactions between the bulk and the active regions are not important. It simply means that changes in correlation effects across the two regions may often be negligible. The reference wave function from the Hartree-Fock method [2] accounts for many of the important interactions between the active and the bulk regions, including charge transfer, exchange repulsion and polarizability. Therefore, it remains important to maintain a complete description of the reference wave function within the active regions for a posteriori corrections involving electron correlation. However, because the full treatment of the electron correlation can be computationally demanding, it is imperative to design methods that can take advantage of the spatial locality of correlation effects. This can be accomplished by limiting the region in which the correlation treatments are applied, thus circumventing the prohibitive computational scaling associated with the full treatment of electron correlation, while maintaining the desired accuracy within the important active region.

In this work, localized molecular orbitals [3] are used as a means to partition the reference wave function into spatially localized subspaces. Provided the transformation is unitary, the spatially localized description of a closed shell wave function maintains the observable properties of the delocalized reference wave function; i.e., the electron density and all properties derived from it, such as the electronic energy, are invariant to the change in the molecular orbital basis. Using the localized molecular orbitals associated only with atoms in the active region, orbitally invariant correlation methods can be applied to calculate corrections to the
reference energy to a desired degree of accuracy. The computational cost of the correlation energy is a function of the size of the active region, so an active region that is substantially smaller than the full space increases the computational feasibility. This method will be referred to as the Selected Orbital Subspace (SOS) model. The use of the SOS model in conjunction with a high-level correlation method in an active region (QM') within a larger quantum mechanical (QM) region will be referred to as the SOS QM/QM' method.

The following outlines the key components of the SOS method. Appropriate descriptions of the reference wave function, orbital localization and orbital selection for correlation treatments are presented. The relation between the SOS QM/QM' method and existing popular methods for treating large molecular systems, e.g. QM/MM [4], ONIOM [5] and local electron correlation methods [6]-[11] are discussed.

8.2 Method

The following is a step-wise overview of the SOS method for calculating accurate energies for spatially localized active regions within large molecular systems:

1. Determine the reference wave function, commonly Hartree-Fock.

2. Localize the molecular orbitals.

3. Orbital Selection - Define the “local” wave function in the active region.

4. Correlation Treatment.

1. Reference Wave Function. The first step in the SOS method is the calculation of the reference wave function for the entire quantum mechanical (QM) portion of the molecular system. The reference wave function would typically be calculated at the traditional Hartree-Fock (HF) level of theory. However, for some molecular systems that are too large in size to be treated even with standard Hartree-Fock methods, the fragment molecular orbital (FMO) [12] method at the HF level of theory can be used as long as the active region is fully self-contained within an FMO monomer. The FMO method divides a large molecular system into monomers
and dimers in a novel manner that retains the accuracy of the full HF calculation. In general, the reference wave function must explicitly describe the entire active region and enough of the surrounding bulk region such that all of the important interactions between the bulk and the active region are included. In principle, the method could be extended to multi-reference approaches, with the initial reference taken to be an MCSCF wave function [13]. However, the present work has a Hartree-Fock reference as its primary focus.

Once the reference wave function for the entire molecular system has been determined, the next step involves defining a “local” reference wave function for the active region. One possibility is to simply cut the active region out of the full molecular system, by breaking any bonds that connect the active region with the bulk region, and “capping” those broken bonds with some appropriate atoms or groups of atoms. However, this is not sufficient if one wishes to maintain the properties and the presence of the bulk effects within the reference wave function for the active region.

2. Localized Molecular Orbitals. The goal of the SOS QM/QM’ method is to maintain as many of the properties of the reference wave function (QM) as possible, within the truncated wave function for the active region (QM’). One of the key features of the SOS method is to bridge this gap between the artificially pristine truncation of the “local” wave function within the active regions and the complete reference wave function, such that the “local” wave function of the active region can provide a realistic treatment of the influence of the surrounding bulk. One approach to maintain a nearly exact representation of the electron density of the reference calculation within the “local” reference wave function of the active region in terms of a subset of localized molecular orbitals (LMOs) obtained from the complete reference wave function.

The description of the local reference wave function for the active region must be defined in terms of sets of both occupied and unoccupied localized molecular orbitals, since all correlation methods involve excitations from the occupied space into the unoccupied (virtual) space. A variety of popular methods exist for the localization of occupied molecular orbitals to generate localized molecular orbitals (LMOs). The most popular methods include the Edminston-Ruedenberg method [14], the Boys method [15] and the Pipek-Mezey method [16].
The occupied LMOs from these methods generally correspond to core, bonding and lone-pair orbitals. More recently, a method to localize the virtual orbitals have been developed by Subotnik and coworkers [17]. This method provides a complete set of orthonormal occupied and virtual LMOs, where the occupied LMOs are generated first from standard methods, and the virtual LMOs are then formed from the remainder of the reference orbital subspace. The virtual LMOs correspond to valence virtual LMOs and hard-centered LMOs. The valence virtual LMOs may be regarded as antibonding orbitals that correspond to the occupied LMOs, while the hard-centered LMOs are atom-centered functions that correspond to the remaining functions in the reference orbital subspace.

3. Orbital Selection. After obtaining the localized orbitals for the reference wave function, one must select the LMOs that are to be actively correlated. This step defines the active region. The optimal active region for a given molecular system is not always easily defined. Analyzing the structure of the molecular system and the atoms and chemical functional groups in the region in which a reaction occurs may provide helpful insights for defining the active space. The goal is to choose the smallest possible region that includes all of the interactions needed to calculate accurate properties of the active region, i.e. increasing the size of the active region should have little or no effect on the calculated properties. It is suggested that multiple definitions of the active region should be evaluated until one is confident that a proper description of the active region has been found.

The local wave function for the active region must include all LMOs associated with each atom within the active region. Important orbitals from boundary atoms must also be included in the local wave function for the active region. Boundary atoms are defined as those atoms that are not themselves in the active region but share one or more bonding orbitals with atoms in the active region. The important orbitals from the boundary atoms that must be included in the local wave function correspond to these bonding orbital(s) that bridge the active and bulk regions and their associated anti-bonding LMO(s). Extensive testing will be required to determine if it is necessary to also include the hard-centered virtual LMOs associated with boundary atoms within the local wave function.
4. Correlation The final step in the method is to apply a correlation method to the local reference wave function. A variety of methods can be used. The most applicable methods are orbital invariant MP2 and coupled cluster approaches, which include both local correlation methods (local MP2 [LMP2] and local coupled cluster [LCC]) as well as full treatments. Most electronic structure programs allow for excluding selected occupied and selected virtual orbitals, commonly referred to as frozen-core or frozen-virtual orbitals, from the correlation treatment. While some extra coding might be required for sorting and orbital selection, applying these orbitals to orbital invariant correlation methods should be straightforward.
8.3 Comparison to Current Methods

In the QM/MM approach, there are two regions: a QM region that can be treated with any applicable electronic structure method and a region in which molecular mechanics (MM) is used to treat the remainder of the atoms. In general, the interaction between the QM and the MM region is one sided, in the sense that the forces from the surrounding MM atoms affect the QM region, while the charge density from the MM region is not represented in the QM wave function. Unless polarizable MM methods [18] are used, there is no effect of the QM region on the MM region besides the molecular mechanics treatment of the QM region. The QM/MM model has an inverse relationship between the size of the QM region and the accuracy with which the QM region can be modeled. Increasing the size of the QM region leads to a better description of the influence of the bulk within the active region, yet lowers the highest level of QM theory that can be used, due to the rapidly scaling computational cost of correlation methods as a function of system size. Decreasing the size of the QM region to allow for higher level corrections within the active region artificially separates the reference description of the active region and the bulk, which could have negative consequences on the accuracy of the calculation even though higher level methods are being used. This is illustrated schematically in Figure 8.1.

The SOS QM/QM' method can be used as a means to decouple this inverse relation between size and accuracy of the QM region within the QM/MM model. In order to use the SOS QM/QM' method in conjunction with the QM/MM model, one would choose the largest computationally feasible QM region, thereby including the important bulk effects within the reference wave function of the active region. Using the SOS QM/QM' method of localization and orbital selection, high-level correlation can be calculated in the specific QM' active region within the QM reference that is in turn surrounded by a larger MM region. The calculated energy for this model would be a traditional QM/MM energy,

\[ E_{QM/MM} = E_{MM}(QM \cup MM) - E_{MM}(QM) + E_{QM}(QM), \quad (8.1) \]
Figure 8.2 Using the SOS QM/QM method to enhance the QM/MM model to evaluate a portion of the Hartree-Fock (green) QM region using CCSD (grey). The remainder of the system is treated with molecular mechanics.
where \( E_X(A) \) represents the energy of region \( A \) evaluated with method \( X \), and QM is defined as the quantum mechanical region and MM is defined as the molecular mechanics region. The SOS QM/QM’ method can be used to simply replace the traditional QM energy evaluation. This is illustrated in Figure 8.2.

The ONIOM (Our own N-layered Integrated molecular Orbital and molecular Mechanics) method [5] uses an additive correction scheme based on energy differences between a series of model calculations. In general, ONIOM uses a hierarchical set of layers in which the entire system is treated at some base, low level, of theory and each subsequent layer uses an increasingly higher-level method by evaluating a smaller model region that is a subset of the region from the previous layer. Figure 8.3 describes a possible ONIOM calculation based on the example in Figure 8.2. This example shows the additive scheme involved in an ONIOM calculation. A drawback of the ONIOM model is that the layers treated with higher-level methods do not explicitly “feel” the lower-level layers. The energies of the lower-level layers are implicitly included in the energy differences; however, some accuracy is lost, since the higher-level wave functions are not directly impacted by the lower levels of theory. The SOS QM/QM’ method avoids this problem by incorporating the essence of the reference wave function for any higher-level correction.

At present, the implementation of the SOS QM/QM’ method is in the early coding stages. The SOS QM/QM’ model represents the first step to a larger series of SOS models. These models and their relation to current local correlation methods are discussed in the following subsection.

8.4 Extended Features

It is possible that one could employ local coupled cluster (LCC) or local MP2 (LMP2) methods for the treatment of electron correlation within the QM’ region of an SOS QM/QM’ calculation, or potentially even for the treatment of the entire reference region. However, it is not yet clear that these low-order scaling local methods accurately reproduce the full (CC or MP2) treatment of the parent method for general chemical systems. This presents another
Figure 8.3 ONIOM method for describing the energy system in Figure 2. The shapes represent the specific regions of the system and the colors are used to describe the method used (blue = MM, green = HF, grey = CCSD to calculate the energy. The equation is \[ E_{ONIOM} = E_{MM}(\text{All}) + E_{HF}(B) - E_{MM}(B) + E_{CCSD}(A) - E_{HF}(A). \]
interesting use of the SOS method that could potentially bridge the gap between the full CC or MP2 treatment of the reference region and a local CC or a local MP2 treatment of the reference region. In the SOS model, this could be accomplished by dividing the orbital space of the reference wave function into orbital subspaces such that each subspace contains a set of occupied orbitals and a corresponding set of virtual orbitals (Figure 8.4). The evaluation of the correlation energy for the system would then be obtained as a sum of the correlation energies from within the orbital subspaces and from the correlation energy between selected subspaces. The computational cost of these SOS “cross correlation” methods depends on how the subspaces are divided and the manner for which the cross correlation is accounted. Whereas the basic SOS QM/QM’ involves actively correlating only one orbital subspace of the reference wave function, these cross correlation treatments within the SOS model involve multiple subspaces and will be referred to as SOS XCOR. There are several possibilities for employing local correlation methods within the SOS XCOR model.

One type of SOS XCOR approach would be an FMO-like treatment of the correlation energy using orbital subgroups as fragments and calculating one-, two-, and three-body interaction
energies between orbital subspaces.

\[ E^{[1]}(\text{Cor}) = \sum_i E^{\text{Cor}}_i \] (8.2)

\[ \Delta E^{[2]}(\text{Cor}) = \sum_{i<j} [E^{\text{Cor}}_{ij} - E^{\text{Cor}}_i - E^{\text{Cor}}_j] \] (8.3)

\[ \Delta E^{[3]}(\text{Cor}) = \sum_{i<j<k} [(E^{\text{Cor}}_{ijk} - E^{\text{Cor}}_i - E^{\text{Cor}}_j - E^{\text{Cor}}_k) - (E^{\text{Cor}}_{ij} - E^{\text{Cor}}_i - E^{\text{Cor}}_j) - (E^{\text{Cor}}_{ik} - E^{\text{Cor}}_i - E^{\text{Cor}}_k) - (E^{\text{Cor}}_{jk} - E^{\text{Cor}}_j - E^{\text{Cor}}_k)] \] (8.4)

\[ E^{[2]}(\text{Cor}) = E^{[1]}(\text{Cor}) + \Delta E^{[2]}(\text{Cor}) \] (8.5)

\[ E^{[3]}(\text{Cor}) = E^{[1]}(\text{Cor}) + \Delta E^{[2]}(\text{Cor}) + \Delta E^{[3]}(\text{Cor}). \] (8.6)

This method treats each orbital subspace as a monomer. The correlation energy for each monomer is calculated using the appropriate level of theory; e.g. MP2, CCSD, etc. Based on previous results using the FMO method, the extension to two-body interactions should recover most of the correlation energy that is lost in the monomer subdivision boundaries. The two-body correlation energy is evaluated for the union of all spatially “local” monomer pairs. Three-body interactions will further increase the accuracy, at a computational cost, by evaluating trimers formed from the union of all spatially “local” monomer triplets. Extending this analysis to the union of all \( n \)-mers converges to the full correlation energy of the entire orbital subspace. The computational savings within the FMO-based SOS XCOR scheme arise from excluding select unions of subspaces based on spatially proximity. As an example, the selection of important orbital subspaces could be determined during the evaluation of the dimer \((\Delta E^{[2]}_{ip})\) contributions. For each orbital subspace \( i \), one would evaluate

\[ \Delta E^{[2]}_{ip} = E_{ip} - E_i - E_p, \forall p \in \{j, \ldots, n\}, \] (8.7)

where the set of orbital subspaces \( j, n \) is sorted by their distances from \( i \). Using a numeric cutoff based on the value of \( \Delta E^{[2]}_{ip} \), one could ignore the remaining set of dimers once the
cutoff has been reached. The information gathered in evaluating the two-body interactions is inherited and used as a guide for selecting important three-body interactions and beyond. This type of treatment allows for a very systematic study of the locality of electron correlation based on the distribution of orbital subspaces. The error is well controlled. In the limit in which all $n$-body interactions are included, the FMO SOS XCOR is equivalent to treating the entire region with the given electron correlation method used.

Using this FMO SOS XCOR analysis, one could envision evaluating a series of orbital subdivisions, where in each progressive series, the number of orbital subdivisions systematically increases, i.e. each of the orbital subspaces within a given subdivision of the full space becomes systematically smaller. This type of study would examine the importance of the size of the orbital subspaces and provide a means of systematically determining the errors that occur when one truncates the full orbital subspace.

One might also consider using a decreasingly accurate series of electron correlation methods for one-, two-, three-body interaction, etc. in the FMO SOS XCOR method. As an example, one might treat the monomers with CCSDTQ, CCSDT, CCSD, and MP2; the dimers with CCSDT, CCSD, MP2, the trimers with CCSD and MP2 and the tetramers with only MP2. Using this scheme the approximate energy would be:

$$E(\text{Cor}) = E^{[1]}(\text{CCSDTQ}) + \Delta E^{[2]}(\text{CCSDT}) + \Delta E^{[3]}(\text{CCSD}) + \Delta E^{[4]}(\text{MP2}).$$

The FMO SOS XCOR method is similar to the Cluster In Molecule Coupled Cluster (CIMCC) approaches pioneered by Förner et al [8] and later refined by Li et al. [9]. Both methods maintain the global Hartree-Fock reference and used orbital subspaces as the means to recover correlation energy. However, the FMO SOS XCOR differs from the CIMCC method in the manner in which the unions of orbital subspaces are chosen and how the correlation energy from the unions of these orbital subspaces is summed. Another similar model is the Natural Linear Scaling Coupled Cluster (NLSCC) method by Flocke and Bartlett [7]. The NLSCC method differs from the SOS XCOR and the CIMCC methods in that the global reference wave function is not maintained. Rather, the NLSCC uses a series of truncated calculations.
Another way in which the electron correlation between orbital subgroups can be evaluated is to analyze excitations from across orbital subgroups. This differs from the FMO SOS XCOR method above, in that the complete unions of the orbital subspaces are never evaluated. Instead, correlations across subspace boundaries can be recovered by allowing excitations to virtual orbitals in neighboring virtual subspaces. In fact, one could also examine the use of different electron correlation methods between orbital subgroups as a means to better select important excitations. This concept has recently been studied by Auer and Nooijen [10] in their dynamically screened local CCSD method. In that work, MP2 amplitudes are used as a means to screen and avoid the computationally expensive CCSD term; this method is designed to approximate the exact CCSD energy. Sherrill and coworkers [11] have also studied the use of mixing MP2 and CCSD to understand why the MP2 method seems to fail as bonds are dissociated and what contributions from the CCSD method correct for that breakdown. In terms of the SOS method, one could envision extending the SOS QM/QM' method by performing an SOS XCOR calculation, such that the QM region is treated with MP2 and the QM' region is treated with a CC method. This approach would treat all excitations from occupied orbitals in the QM region with the MP2 method, excitations from occupied orbitals in the QM' region with MP2 for excitations to the virtual space of the QM region and a CC method for excitations to the virtual space of the QM' region. The SOS XCOR methods represent a potential next step after the initial SOS QM/QM' method is coded and evaluated.

8.5 Conclusions

Because the SOS QM/QM' method maintains the essence of the reference wave function within the description of the active region, this method potentially offers a fundamentally more satisfactory model for describing the quantum mechanical treatment of electron correlation within these active regions than the other available methods. The SOS QM/QM' method can be used in place of any traditional quantum mechanical treatment of a chemical system, provided the system contains spatially localized regions of interest. The SOS QM/QM' method
is one means of avoiding the steep scaling computational costs of post Hartree-Fock correlation methods, yet maintaining a high-level description of the active region. The SOS XCOR methods also provide a means of dramatically reducing the potential computational cost of post Hartree-Fock methods for the recovery of the correlation energy for the entire system. These methods are systematic and should provide a means of evaluating the accuracy of local electron correlation methods.
Bibliography


GENERAL CONCLUSIONS

The electron correlation problem in high-level electronic structure theory is a challenging, yet fundamentally important problem. The research presented in this thesis examines the electron correlation problem from three perspectives. These perspectives correspond to Parts I, II, and III of this document.

In Part I entitled “Gold Clusters”, the role of electron correlation with respect to determining accurate energies for Au$_6$ and Au$_8$ was examined. The two primary electron correlation methods studied, MP2 and CCSD(T), were found to yield qualitatively different results regardless of the basis set employed. Therefore, the lower-level MP2 method was found to be insufficient for estimating basis set improvement for the higher-level CCSD(T) method. At present, the highest level single-point energy calculations, CCSD(T)/cc-pVTZ-PP, calculated at MP2 optimized geometries suggest that Au$_8$ prefers a planar structure. However, because of the significant differences found between the MP2 and CCSD(T) methods, future work, in which the structures for the Au$_8$ isomers are allowed to relax on the CCSD(T) potential energy surface, is necessary for a conclusive end to the Au$_8$ problem.

In Part II, tools and mythologies for parallel computing were discussed and the parallel implementation of an important electron correlation method, CCSD(T), was examined. The applicability of electron correlation methods has been closely tied to the availability of sufficient computational resources. For many years, parallel computing has offered a means for the computational chemist to gain a real-time speedup for many popular electronic structure methods. However, the limits of sequential processing are becoming apparent, as the trend in the computer industry is shifting away from rapidly improving single processing units (or “cores”) and moving towards adding multiple processing units within their products. A consequence of
this technology shift is that sequential programs will not receive the same performance bonus that they have traditionally enjoyed from the rapidly improving single processing units. Thus, parallel computing for challenging scientific problems, including electron correlation methods, will be essential for continuing and advancing future development in this area.

In Part III of this thesis, the Selected Orbital Subspace (SOS) method for examining electron correlation was examined. Because of the significant computational challenges faced by applying traditional electron correlation methods to large chemical systems, local electron correlation methods, like the SOS method, are very important because the computation cost associated with these methods does not scale as rapidly as traditional methods. The SOS method provides a means to systematically control the errors associated using local correlation methods. This provides a means to relate the local SOS treatment of electron correlation to traditional methods. Future work on the SOS method includes examining how the boundary regions between orbital subspaces must be chosen to ensure accurate results. Another important aspect is to determine the minimum and optimal size of orbital subspaces for general use and provide a means of automating the selection of orbital subspaces for general chemical systems. The use of orbital subspaces for evaluating electron correlation methods at chemically important active sites has potential for studying biomolecules, polymer complexes and inorganic complexes.
ACKNOWLEDGEMENTS

I would like to thank:

Prof. Mark Gordon for allowing me the academic freedom to pursue research topics that I felt most strongly about and the guidance to make sure I did not get lost along the way.

Dr. Mike Schmidt for our many illuminating discussions on a broad range of topics from chemistry, physics and mathematics to far more geeky topics including computer hardware, programming languages, vintage systems and of course, CVS.

Prof. Alistair Rendell for a very rewarding collaborative partnership which greatly influenced the work in Part II of this document.

Members of the Gordon Group, past and present, for a fun and always interesting atmosphere.

Family and friends for all the love and support.

And to Andrea, who has learned about or at least has politely listened to more quantum chemistry and computer science than I’m sure she’d care to admit. Your friendship and love has been inspirational.

This work was performed at Ames Laboratory under Contract No. W-7405-Eng-82 with the U.S. Department of Energy. The United States government has assigned the DOE Report number IS-T 2207 to this thesis.