New Quantum Monte Carlo Algorithms to Efficiently Utilize Massively Parallel Computers

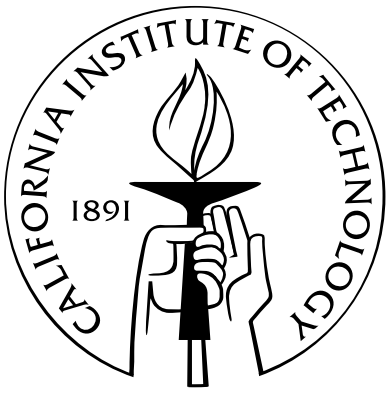
Thesis by

David Randall "Chip" Kent IV

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2003

(Defended March 10, 2003)

This work is dedicated to my wife, parents, grandparents, and brother. They have supported me in every way possible and have been a constant encouragement during my education. Special recognition is given to my grandparents, Joe and LaNell Lanning, who taught me that there are many things much more important than work. They did not live to see me graduate.

We are perhaps not far removed from the time when we shall be able to submit the bulk of chemical phenomena to calculation. [1]

J. L. Gay-Lussac, 1809

Acknowledgements

This work would not have been possible without help from a number of people and organizations.

My wife, Brooke, has sacrificed the most for this text. She has tolerated the insane work hours of a Caltech graduate student while working hard so that we can afford to live in Pasadena. She has also tolerated my many, sometimes expensive, hobbies used to relieve the stress of long days at Caltech.

The majority of the intellectual content within this work has resulted from numerous, sometimes heated, brainstorming sessions with Michael T. Feldmann (now Dr. Feldmann). Mike is a great colleague. He's packed with fantastic ideas, and we're always friends at the end of the day, no matter how much "head-butting" has gone on about QMC during the day. In addition to the intellectual content, Mike has contributed many thousands of hours of his time and tens of thousands of lines of C++ to make QMcBeaver possible. What can you say, he's from Iowa.

Ever since I did undergraduate research at Caltech, Dr. Richard P. Muller has been a tremendous scientific mentor and motivator for me. Day in and day out, Rick is excited because he gets to wake up and do science. He's always willing to give a new idea a try no matter how crazy it initially sounds. Without Rick's motivation, I

would not have stayed at Caltech to complete my degree.

Professor John D. Roberts has also been an invaluable mentor. Working with him as an undergraduate convinced me to attend graduate school.

The major financial contributor to this work has been the Fannie and John Hertz Foundation. I have been a Hertz Fellow from 1999 to 2003. During this time, the Hertz Foundation has contributed many hundreds of thousands of dollars towards my education. This support has covered my tuition and provided me with a generous stipend to live on. Caltech tuition is not cheap!

Financial support for my undergraduate education, from my parents, grandparents, and the Welch Foundation, has allowed me to attend graduate school. Without this support, I would be working to pay for school instead of attending graduate school.

The majority of the computing resources used in the presented calculations was provided by the Department of Energy's (DOE) Accelerated Strategic Computing Initiative (ASCI) project. Additional computing resources were provided by the Material and Process Simulation Center (MSC) at Caltech and the National Energy Research Scientific Computing Center (NERSC) Parallel Distributed Systems Facility (PDSF).

Last, but not least, this work would not have been possible without my research advisors, Professors William A. Goddard III and Harry B. Gray.

Abstract

The exponential growth in computer power over the past few decades has been a huge boon to computational chemistry, physics, biology, and materials science. Now, a standard workstation or Linux cluster can calculate semi-quantitative properties of moderately sized systems. The next step in computational science is developing better algorithms which allow quantitative calculations of a system's properties.

A relatively new class of algorithms, known collectively as Quantum Monte Carlo (QMC), has the potential to quantitatively calculate the properties of molecular systems. Furthermore, QMC scales as $O(N^3)$ or better. This makes possible very high-level calculations on systems that are too large to be examined using standard high-level methods.

This thesis develops (1) an efficient algorithm for determining "on-the-fly" the statistical error in serially correlated data, (2) a manager-worker parallelization algorithm for QMC that allows calculations to run on heterogeneous parallel computers and computational grids, (3) a robust algorithm for optimizing Jastrow functions which have singularities for some parameter values, and (4) a proof-of-concept demonstrating that it is possible to find transferable parameter sets for large classes of compounds.

Contents

Acknowledgements **v**

Abstract **vii**

1 Overview **1**

2 Introduction **4**

2.1 Introduction to Quantum Mechanics 4

2.2 Wave Function Particle-Interchange Symmetry 7

2.3 Cusp Conditions 8

2.4 Quantum-Mechanical Variational Principle 9

2.5 Quantum Mechanics of Atoms and Molecules 10

2.5.1 Born-Oppenheimer Approximation 12

2.5.2 Approximate Solution of the Nuclear Schrödinger Equation . . 14

2.5.3 Summary 16

3 Random Number Generation **18**

3.1 Uniform Random Numbers 19

3.2 Transformation Method 21

3.3 Rejection Method 22

3.4 Metropolis Algorithm 23

4 Introduction to Quantum Monte Carlo 26

4.1 Variational Quantum Monte Carlo 27

4.1.1 Variational Quantum Monte Carlo Wave Functions 29

4.1.2 Jastrow Functions 30

4.2 Diffusion Quantum Monte Carlo 32

4.2.1 DMC Energy Evaluation 34

4.2.2 DMC Random Point Generation 37

5 Efficient Algorithm for "On-the-fly" Error Analysis of Local or Distributed Serially Correlated Data 43

5.1 Introduction 44

5.2 Theory 45

5.3 Algorithms 49

5.3.1 Flyvbjerg-Petersen Algorithm 49

5.3.2 Dynamic Distributable Decorrelation Algorithm (DDDA) . . . 50

5.3.2.1 Statistic Class 50

5.3.2.2 Decorrelation Class 50

5.3.3 Analysis of DDDA 52

5.4 Computational Experiments 54

5.4.1 "On-the-fly" Variance Determination for a Single Processor Variational QMC Particle-in-a-Box Calculation 54

5.4.2 "On-the-fly" Variance Determination for a Massively Parallel Variational QMC Calculation of RDX 57

5.5 Conclusions 60

5.6 Statistic Class Pseudocode 63

5.6.1 Pseudocode for `Statistic.initialize()` 63

5.6.2 Pseudocode for `Statistic.addData(new_sample)` 63

5.6.3 Pseudocode for `Statistic.addition(A,B)` 64

5.7 Decorrelation Class Pseudocode 64

5.7.1 Pseudocode for `Decorrelation.initialize()` 64

5.7.2 Pseudocode for `Decorrelation.addData(new_sample)` 65

5.7.3 Pseudocode for `Decorrelation.addition(A,B)` 66

5.8 Simple Example Calculation Pseudocode 70

6 Manager–Worker-Based Model for Parallelizing Quantum Monte Carlo on Heterogeneous and Homogeneous Networks 72

6.1 Introduction 73

6.2 Theory 75

6.2.1 Pure Iterative Parallelization Algorithm 76

6.2.2 Manager–Worker-Parallelization Algorithm 80

6.2.3 Initialization Catastrophe 83

6.3 Experiments 84

6.3.1 Experiment: Varying Levels of Heterogeneity 85

6.3.2 Experiment: Heterogeneous Network Size 88

6.3.3 Experiment: Large Heterogeneous Network 90

6.3.4 Experiment: Homogeneous Network 92

6.3.5 Experiment: Initialization Catastrophe 94

6.4 Conclusion 96

6.5 Pure Iterative Algorithm (QMC-PI) 98

6.6 Manager–Worker Algorithm (QMC-MW) 98

7 Robust Jastrow-Function Optimization Algorithm for Variational Quantum Monte Carlo 100

7.1 Introduction 100

7.2 Algorithm 103

7.3 Example: Padé-Jastrow Function 106

7.4 Conclusion 107

8 Generic Jastrow Functions for Quantum Monte Carlo Calculations on Hydrocarbons 108

8.1 Introduction 108

8.2 Computational Experiments 111

8.2.1 Generation of Transferable Hydrocarbon Parameters 112

8.2.2 Generic Jastrow for DMC 115

8.3 Conclusion 116

9 QMcBeaver 118

9.1 QMcBeaver Copyright Statement 119

Bibliography **121**

List of Figures

3.1 Example comparison function, $f(x)$, to generate random points distributed with respect to $\rho(x)$. 22

5.1 $\sqrt{\chi}$ (Equation 5.17) as a function of block size for a variational QMC "particle-in-a-box" calculation using uncorrelated data points (Method 1). The Flyvbjerg-Petersen algorithm and DDDA yield exactly the same results. 56

5.2 $\sqrt{\chi}$ (Equation 5.17) as a function of block size for a variational QMC "particle-in-a-box" calculation using serially correlated data points (Method 2). The Flyvbjerg-Petersen algorithm and DDDA yield exactly the same results. 57

5.3 Standard deviation as a function of number of Monte Carlo steps for a variational QMC "particle-in-a-box" calculation. The standard deviation was evaluated "on-the-fly" using DDDA. 58

5.4 The RDX molecule, cyclic $[CH_2\text{-}N(NO_2)]_3$. 59

5.5 Evolution of $\sqrt{\chi}$ as a function of block size as a variational QMC calculation of the RDX ground state progresses. Shown here are the results for five cases with 62122, 2137179, 6283647, 14566309, and 31163746 total Monte Carlo steps. . 61

5.6 Standard deviation as a function of number of Monte Carlo steps for a 1024 processor variational QMC calculations of RDX. The standard deviation was evaluated "on-the-fly" using DDDA. 62

6.1 Time required to complete an 8-processor variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. The theoretical optimal performance for a given configuration of processors is provided by the curve. . 86

6.2 Number of variational QMC steps completed during an 8-processor calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) parallelization algorithms. The pure iterative algorithm always calculates the same number of steps, but the manager–worker algorithm dynamically determines how many steps to take. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. . 88

6.3 Percentage of total calculation time devoted to each component in the pure iterative parallelization algorithm (QMC-PI) during an 8-processor variational QMC calculation of Ne. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. . . . 89

6.4 Percentage of total calculation time devoted to each component in the manager–worker-parallelization algorithm (QMC-MW) during an 8-processor variational QMC calculation of Ne. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. 90

6.5 Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on a heterogeneous Linux cluster. The theoretical optimal performance for a given configuration of processors is provided by the line. The specific processor configuration is discussed in Section 6.3.2. 91

6.6 Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on the ASCI Blue Pacific homogeneous supercomputer. The theoretical optimal performance for a given configuration of processors is provided by the line. . . 93

6.7 Ratio of wall time for QMC-MW/QMC-PI on ASCI Blue Pacific. 94

6.8 Efficiency of a variational QMC calculation of RDX as a function of the number of processors used. The calculations were performed using the manager–worker-parallelization algorithm (QMC-MW) on the ASCI-Blue Mountain supercomputer, which has 250 MHz MIPS 10000 processors connected by HIPPI networking. A similar result is produced by the Pure Iterative parallelization algorithm. The data is fit to $\epsilon(N_{Processors}) = a/(a + N_{Processors})$ with $a = 104.203$. 95

7.1 Poles of the Jastrow function. If any poles fall within the region of size ϵ, the Jastrow function is considered to be singular. 103

7.2 Distance of poles from the positive real axis. 104

8.1 Correlation energy (Hartree) recovered divided by total nuclear charge. 113

8.2 Reduction of the QMC variance for a wave function containing a Generic Jastrow compared to a Hartree-Fock wave function. 114

8.3 Variances of DMC calculations using Generic Jastrow functions. 116

List of Tables

5.1 Comparison of computational costs. N is the number of data points analyzed, m is the number of times the variance is evaluated during a calculation, and P is the number of processors. 53

5.2 Padé-Jastrow correlation function parameters for RDX. 59

5.3 Total energies (Hartree) for the various calculations on RDX. The HF results were obtained from Jaguar 4.1 with the 6-31G** basis set Variational Quantum Monte Carlo based on 3×10^7 points. 60

8.1 Compounds used to determine transferability of hydrocarbon Generic Jastrow parameters. 113

Chapter 1

Overview

> *The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. [2]*
>
> P. A. M. Dirac, 1929

The ultimate goal of computational quantum chemistry and computational molecular physics is to quickly and quantitatively predict the properties of molecular systems before performing any experiments. This will allow the design and optimization of new materials and catalysts before investing in expensive and time-consuming laboratory work.

New algorithms for electronic structure calculations, coupled with the continued exponential growth in computing power, have converted the previously esoteric field of computational quantum chemistry into a useful tool for laboratory scientists. For example, Density Functional Theory (DFT) [3, 4, 5] has made possible semi-quantitative calculations [6] on a huge range of important systems [7].

Though much progress has been made, standard methods, such as DFT, coupled cluster, and many-body perturbation theory, are still unable to quantitatively calculate properties of molecular systems [8, 9, 10]. Furthermore, the linear algebra involved in standard methods limits the number of processors that ultimately could be utilized in a single calculation, which limits in turn the system size which can be examined.

A relatively new class of algorithms for performing electronic structure calculations shows promise in quantitatively calculating properties of molecular systems. The algorithms in this class all fall under the broad category of Quantum Monte Carlo (QMC). Instead of using a linear algebra-based approach to solve the Schrödinger equation, as is done in standard electronic structure algorithms, QMC algorithms utilize a stochastic (also known as Monte Carlo) approach.

Presented in this work is a collection of new algorithms which:

- allow the convergence of a QMC calculation to be examined as the calculation progresses (Chapter 5);
- allow QMC to fully utilize the next generation of supercomputers (Chapter 6);
- facilitate the stable, robust optimization of potentially singular variational QMC wave functions (Chapter 7); and
- reduce the time required to optimize variational QMC wave functions before performing diffusion QMC calculations (Chapter 8).

Each of these new algorithms enable QMC calculations to utilize computational re-

sources more efficiently. Additionally, the new algorithms allow more processors to be used than has previously been possible. These improvements allow the examination of larger systems while consuming less computational time.

Furthermore, the new algorithms permit the utilization of less expensive computer hardware. The algorithms require very little inter-processor communication, very little RAM, and often no hard drive. This eleminates expensive components such as Myrinet, fast switches, and large RAM. Furthermore, the most fault-prone component of a computer, the hard drive, is eliminated, making such a QMC cluster far more reliable than a standard cluster. These improvements make the hardware necessary to perform QMC calculations affordable to many more researchers.

Chapter 2 provides a very basic introduction to quantum mechanics as it applies to electronic structure calculations. Chapter 3 covers the standard algorithms used to generate random numbers, which will later be used in QMC calculations. Finally, Chapter 4 is an introduction to QMC algorithms. This chapter focuses on variational QMC and diffusion QMC, which are the most important for electronic structure calculations.

All Quantum Monte Carlo calculations presented here were performed using QMcBeaver. The source code for this software, developed by Michael T. Feldmann and myself, is listed in Chapter 9.

Chapter 2

Introduction

> *It turns out to be very difficult to predict precisely what will happen in a chemical reaction: nevertheless, the deepest part of theoretical chemistry must end up in quantum mechanics. [11]*
>
> R. P. Feynman, 1965

This chapter provides an extremely elementary introduction to quantum mechanics. For a more detailed coverage of this material, see References [12], [13], [11], [14], [15], and [16]. Standard algorithms for quantum-mechanical calculations of molecular systems are covered in References [17] and [3].

2.1 Introduction to Quantum Mechanics

To correctly describe the physics of atomic and molecular systems, quantum mechanics must be used in place of classical, Newtonian mechanics. In quantum mechanics, the state of a system is completely defined by an abstract vector, $|\Psi(t)\rangle$, known as the wave function or state of the system, where t is time. $|\Psi(t)\rangle$ can be represented as a function of variable $\mathbf{x}$ as $\Psi(\mathbf{x}, t) = \langle \mathbf{x}|\Psi(t)\rangle$.

To calculate the value of an experimentally measurable quantity, a Hermitian operator, $\hat{\mathbf{O}} = \hat{\mathbf{O}}^{\dagger}$, must be constructed which corresponds to the quantity. $\hat{\mathbf{O}}^{\dagger}$ is the adjoint of $\hat{\mathbf{O}}$ and is equal to the complex conjugate of the transpose of $\hat{\mathbf{O}}$ in the matrix representation. The details of constructing such an operator are beyond the scope of this text [13, 12]. Using the operator, the expectation value of the calculated property for a system in state $|\Psi(t)\rangle$, $\left\langle \hat{\mathbf{O}}(t) \right\rangle$, is

$$\left\langle \hat{\mathbf{O}}(t) \right\rangle = \frac{\langle \Psi(t)| \, \hat{\mathbf{O}} \, |\Psi(t)\rangle}{\langle \Psi(t)|\Psi(t)\rangle} = \frac{\int \Psi(\mathbf{x},t)^{*} \hat{\mathbf{O}} \Psi(\mathbf{x},t) d\mathbf{x}}{\int \Psi(\mathbf{x},t)^{*} \Psi(\mathbf{x},t) d\mathbf{x}} \tag{2.1}$$

where $\langle \Psi(t)| = |\Psi(t)\rangle^{\dagger}$ and $\Psi(\mathbf{x},t)^{*}$ is the complex conjugate of $\Psi(\mathbf{x},t)$.

As an example, the probability of finding a particle within $d\mathbf{x}$ of $\mathbf{x}$, $P(\mathbf{x},t)d\mathbf{x}$, can be calculated using $\hat{\mathbf{O}} = |\mathbf{x}\rangle \langle \mathbf{x}| \, d\mathbf{x}$.

$$P(\mathbf{x},t)d\mathbf{x} = \frac{\langle \Psi(t)| \, \mathbf{x} \rangle \langle \, \mathbf{x} \, |\Psi(t)\rangle \, d\mathbf{x}}{\langle \Psi(t)|\Psi(t)\rangle} = \frac{\Psi(\mathbf{x},t)^{*} \Psi(\mathbf{x},t) d\mathbf{x}}{\int \Psi(\mathbf{x},t)^{*} \Psi(\mathbf{x},t) d\mathbf{x}} \tag{2.2}$$

Similarly, the expectation value of the total energy for the system can be calculated using the Hamiltonian operator, $\hat{\mathbf{H}}$, for the system.

$$\langle E(t) \rangle = \frac{\langle \Psi(t)| \, \hat{\mathbf{H}} \, |\Psi(t)\rangle}{\langle \Psi(t)|\Psi(t)\rangle} = \frac{\int \Psi(\mathbf{x},t)^{*} \hat{\mathbf{H}} \Psi(\mathbf{x},t) d\mathbf{x}}{\int \Psi(\mathbf{x},t)^{*} \Psi(\mathbf{x},t) d\mathbf{x}} \tag{2.3}$$

For the above formalism to be useful, it must be possible to calculate the wave function for a system. This is done using the time-dependent Schrödinger Equation,

the wave function's deterministic equation of motion,

$$i\frac{\partial}{\partial t}\left|\Psi(t)\right\rangle = \hat{\mathbf{H}}\left|\Psi(t)\right\rangle \tag{2.4}$$

where i is $\sqrt{-1}$ and t is time. The solution to the time-dependent Schrödinger Equation can be expanded as

$$\left|\Psi(t)\right\rangle = \sum_j c_j e^{-iE_j t}\left|\Phi_j\right\rangle \tag{2.5}$$

where $c_j = \langle\Phi_j|\Psi(0)\rangle$ are complex coefficients and E_j and $|\Phi_j\rangle$ are the eigenvalues and eigenvectors, respectively, of the time-independent Schrödinger Equation.

$$\hat{\mathbf{H}}\left|\Phi_j\right\rangle = E_j\left|\Phi_j\right\rangle \tag{2.6}$$

The $|\Phi_j\rangle$ are a set of special wave functions, known as stationary states, eigenstates, or eigenfunctions, which do not change in time. Each eigenfunction of $\hat{\mathbf{H}}$, $|\Phi_j\rangle$, has an associated eigenvalue, E_j, which is a constant and can be interpreted as the total energy of the stationary state.

Because $\hat{\mathbf{H}}$ is a Hermitian operator, its eigenvalues, E_j, are real numbers, and the eigenfunctions are orthogonal to one another, $\langle\Phi_i|\Phi_j\rangle = \delta_{i,j}$. $\delta_{i,j}$ is the Kronecker delta and equals 1 for $i = j$ and 0 otherwise.

From this point forward, it is assumed that the stationary states are ordered so that $E_0 \leq E_1 \leq E_2 \leq \cdots$. By convention, the lowest energy state, $|\Phi_0\rangle$, is called the *ground state*.

2.2 Wave Function Particle-Interchange Symmetry

All known subatomic particles can be divided into two classes: fermions and bosons. Bosons are particles with spins of 0, 1, etc., such as photons and deuterium atoms, while fermions are particles with spins of 1/2, 3/2, etc., such as electrons and protons.

The quantum-mechanical behavior of bosons and fermions is very different. Wave functions for bosons are totally symmetric so that interchanging the positions of any two identical particles does not alter the wave function.

$$\Phi_{boson}(\ldots, x_i, \ldots, x_j, \ldots) = \Phi_{boson}(\ldots, x_j, \ldots, x_i, \ldots) \tag{2.7}$$

On the other hand, the wave function for fermions is totally antisymmetric so interchanging the position of any two identical particles changes the wave function's sign.

$$\Phi_{fermion}(\ldots, x_i, \ldots, x_j, \ldots) = -\Phi_{fermion}(\ldots, x_j, \ldots, x_i, \ldots) \tag{2.8}$$

The Pauli exclusion principle (no two electrons in a system can be at the same time in the same state or configuration) is a direct result of the antisymmetry of fermionic wave functions.

2.3 Cusp Conditions

The time-independent Schrödinger Equation for an N-particle Coulombic system is

$$\left[-\frac{1}{2}\sum_{i=1}^{N}\frac{1}{m_i}\nabla_i^2+\sum_{i=1}^{N}\sum_{j=1}^{j<i}\frac{q_i q_j}{r_{ij}}\right]\Phi = E\Phi \tag{2.9}$$

where m_i is the mass of particle i, r_{ij} is the distance between particles i and j, and q_i and q_j are the charges on particles i and j. The Coulomb terms in the potential energy diverge as two particles approach one another; therefore, for the total energy of the system, E, to be finite, divergence in the kinetic energy must exactly cancel the divergence in the potential energy. Satisfying the cusp conditions achieves this exact cancellation.

The cusp condition for particles i and j approaching one another is

$$\lim_{r_{ij}\to 0}\frac{\partial\bar{\Phi}}{\partial r_{ij}} = \frac{\mu_{ij}q_i q_j}{l+1}\lim_{r_{ij}\to 0}\Phi \tag{2.10}$$

where $\bar{\Phi}$ is the average of Φ over an infinitesimally small sphere centered at $r_{ij}=0$, $\mu_{ij} = m_i m_j/(m_i+m_j)$ is the reduced mass of particles i and j, and l results from the symmetry of the wave function (Section 2.2). l is 1 for identical fermions and 0 otherwise. Derivations of this result can be found in References [18] and [19].

Using Equation 2.10, it is straightforward to show that the electron-nuclear cusp condition is $-Z$, in atomic units, where Z is the atomic number of the nucleus. The electron-electron cusp condition, in atomic units, is 1/2 for opposite-spin electrons and 1/4 for same-spin electrons.

2.4 Quantum-Mechanical Variational Principle

The eigenstates of the Hamiltonian operator span the space of all possible wave functions for the system. Therefore, a wave function $|\Psi\rangle$ can be expanded in terms of the eigenstates of the Hamiltonian for the system, $|\Phi_i\rangle$,

$$|\Psi\rangle = \sum_i a_i |\Phi_i\rangle \tag{2.11}$$

where $a_i = \langle\Phi_i|\Psi\rangle$ are complex numbers.

Evaluating the expected energy of the wave function gives

$$\begin{aligned}
\langle E\rangle &= \frac{\langle\Psi|\,\hat{\mathbf{H}}\,|\Psi\rangle}{\langle\Psi|\Psi\rangle} && (2.12)\\
&= \frac{\sum_{i,j} a_i^* a_j \langle\Phi_i|\,\hat{\mathbf{H}}\,|\Phi_j\rangle}{\sum_{i,j} a_i^* a_j \langle\Phi_i|\Phi_j\rangle} && (2.13)\\
&= \frac{\sum_{i,j} a_i^* a_j E_j \langle\Phi_i|\Phi_j\rangle}{\sum_{i,j} a_i^* a_j \langle\Phi_i|\Phi_j\rangle} && (2.14)\\
&= \frac{\sum_i |a_i|^2 E_i}{\sum_i |a_i|^2} && (2.15)
\end{aligned}$$

It is then trivial to show that

$$\langle E\rangle \geq E_0 \tag{2.16}$$

where E_0 is the energy of the ground state.

The variational theorem provides a means by which to approximate the ground state wave function of a system. First, a parameterized wave function is constructed. Then the wave function parameters are adjusted to give the lowest expected energy. This approximates the ground state wave function.

Approximations for excited state wave functions can be obtained by requiring that the parameterized wave function is orthogonal to all lower energy states. In this case, the expected energy of the approximate wave function is greater than the true energy of the excited state. Because the exact wave functions for lower energy states are typically unknown, approximate wave functions must be used. Thus, the expected energy of the approximate excited state wave function is not guaranteed to be greater than the true energy of the excited state.

2.5 Quantum Mechanics of Atoms and Molecules

Atoms and molecules are composed of nuclei and electrons, where the position of nucleus A is represented by $\mathbf{X}_A$ and the position of electron i is represented by $\mathbf{x}_i$. The distance between electron i and nucleus A is $r_{iA} = |\mathbf{x}_i - \mathbf{X}_A|$. The distance between electrons i and j is $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$. Finally, the distance between nuclei A and B is $R_{AB} = |\mathbf{X}_A - \mathbf{X}_B|$. Using these coordinates, the non-relativistic Hamiltonian for a system of N electrons and M nuclei, in atomic units, is

$$\hat{\mathbf{H}} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 - \frac{1}{2}\sum_{A=1}^{M}\frac{1}{M_A}\nabla_A^2 + \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}} \quad (2.17)$$

where M_A is the ratio of the mass of nucleus A to the mass of an electron, Z_A is the atomic number of nucleus A, and Laplacian operators ∇_i^2 and ∇_A^2 respectively involve differentiation with respect to the coordinates of electron i and nucleus A.

This Hamiltonian operator can be broken into kinetic energy, $\hat{\mathbf{T}}$, and potential

energy, $\hat{\mathbf{V}}$, operators

$$\hat{\mathbf{H}} = \hat{\mathbf{T}} + \hat{\mathbf{V}} \tag{2.18}$$

which can be further broken down into

$$\hat{\mathbf{T}} = \hat{\mathbf{T}}_e + \hat{\mathbf{T}}_n \tag{2.19}$$

and

$$\hat{\mathbf{V}} = \hat{\mathbf{V}}_{ee} + \hat{\mathbf{V}}_{en} + \hat{\mathbf{V}}_{nn} \tag{2.20}$$

where $\hat{\mathbf{T}}_e$ and $\hat{\mathbf{T}}_n$ are the electronic and nuclear kinetic energy operators and $\hat{\mathbf{V}}_{ee}$, $\hat{\mathbf{V}}_{en}$, and $\hat{\mathbf{V}}_{nn}$ are the potential energy operators for electron-electron, electron-nuclear, and nuclear-nuclear interactions.

$$\hat{\mathbf{T}}_e = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 \tag{2.21}$$

$$\hat{\mathbf{T}}_n = -\frac{1}{2}\sum_{A=1}^{M}\frac{1}{M_A}\nabla_A^2 \tag{2.22}$$

$$\hat{\mathbf{V}}_{ee} = \sum_{i=1}^{N}\sum_{j>i}^{N}\frac{1}{r_{ij}} \tag{2.23}$$

$$\hat{\mathbf{V}}_{en} = -\sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} \tag{2.24}$$

$$\hat{\mathbf{V}}_{nn} = \sum_{A=1}^{M}\sum_{B>A}^{M}\frac{Z_A Z_B}{R_{AB}} \tag{2.25}$$

The time-independent Schrödinger equation for atoms and molecules (Equations 2.6 and 2.17) is an inseparable $(3N + 3M)$-dimensional partial differential equation. Because the Hamiltonian (Equation 2.17) is real and Hermitian, the eigenvalues are

real constants, and the eigenfunctions can be chosen to be real functions. This property greatly aids in numerically solving the Schrödinger equation.

For a "simple" case such as benzene, C_6H_6, the differential equation is 162-dimensional ($M = 12$, $N = 42$). If a standard grid-based PDE solver was applied to this problem using an absurdly coarse grid with only 2 points in each dimension, $2^{162} \approx 10^{49}$ grid points would be required for the calculation. Due to the grid's coarseness, such a calculation is both computationally infeasible and would yield extremely poor-quality results. As a result, standard PDE solvers are not applicable to high-accuracy solutions of the Schrödinger equation for general atomic and molecular systems, so other algorithms must be used. Quantum Monte Carlo is one such algorithm and is the focus of this work.

2.5.1 Born-Oppenheimer Approximation

A nucleus's mass is much greater than that of an electron. For the lightest nucleus, hydrogen, the ratio of the mass of the nucleus to the mass of the electron is 1836. Because nuclei are so heavy relative to electrons, electrons move much more quickly than nuclei. This situation allows the employment of an *adiabatic approximation*.

First, the positions of the nuclei are fixed and the eigenstate of the electrons, $|\Phi_j(\mathbf{X})\rangle$, corresponding to the fixed configuration of nuclei, is calculated using

$$\left[\hat{\mathbf{T}}_e + \hat{\mathbf{V}}_{ee} + \hat{\mathbf{V}}_{en} + \hat{\mathbf{V}}_{nn}\right] |\Phi_j(\mathbf{X})\rangle = E_{e,j}(\mathbf{X}) \, |\Phi_j(\mathbf{X})\rangle \tag{2.26}$$

where $E_{e,j}(\mathbf{X})$ is the energy of the j^{th} electronic eigenstate at fixed nuclear coordi-

nates, $\mathbf{X}$. The eigenstate for the entire system, $|\Psi\rangle$, is then a tensor product of a nuclear eigenstate, $|\Upsilon_k\rangle$, and an adiabatic electronic eigenstate, $|\Phi_j(\mathbf{X})\rangle$.

$$|\Psi\rangle = |\Upsilon_k\rangle \otimes |\Phi_j(\mathbf{X})\rangle \tag{2.27}$$

Substituting into the Schrödinger equation for the entire system, $\hat{\mathbf{H}}\,|\Psi\rangle = E\,|\Psi\rangle$,

$$\left[\hat{\mathbf{T}}_n + E_{e,j}(\mathbf{X})\right]\left[|\Upsilon_k\rangle \otimes |\Phi_j(\mathbf{X})\rangle\right] = E_{jk}\left[|\Upsilon_k\rangle \otimes |\Phi_j(\mathbf{X})\rangle\right] \tag{2.28}$$

is obtained, where E_{jk} is the total energy for the system, which is in the j^{th} electronic state and k^{th} nuclear state. $\hat{\mathbf{T}}_n$ operates on both the nuclear state, $|\Upsilon_k\rangle$, and the electronic state, $|\Phi_j(\mathbf{X})\rangle$. As long as the amplitude of the relative motion of pairs of nuclei is small compared to the distance between them, $\hat{\mathbf{T}}_n\,|\Phi_j(\mathbf{X})\rangle \approx 0$. The *Born-Oppenheimer approximation* assumes that $\hat{\mathbf{T}}_n\,|\Phi_j(\mathbf{X})\rangle = 0$. This approximation yields an eigenvalue equation for the nuclear wave function which is not coupled with the electronic wave function.

$$\left[\hat{\mathbf{T}}_n + E_{e,j}(\mathbf{X})\right]|\Upsilon_k\rangle = E_{jk}\,|\Upsilon_k\rangle \tag{2.29}$$

The adiabatic potential, $E_{e,j}(\mathbf{X})$, determines the motion of the nuclei. Its local-minimum values corresponds to the system's equilibrium geometries.

2.5.2 Approximate Solution of the Nuclear Schrödinger Equation

Using the Born-Oppenheimer approximation (Section 2.5.1), it is possible to separate the electronic and nuclear degrees of freedom in a quantum-mechanical calculation of a molecular system. The nuclear degrees of freedom correspond to the molecule's translations, rotations, and vibrations. This separation produces a Schrödinger equation for the nuclear part of the system (Equation 2.29).

The adiabatic potential, $E_{e,j}(\mathbf{X})$, can be expanded in a Taylor series around a particular set of equilibrium nuclear coordinates, $\mathbf{X}_{eq}$,

$$\mathbf{X}_{eq} = \arg\left\{\min_{\mathbf{X}} E_{e,j}(\mathbf{X})\right\} \tag{2.30}$$

such that

$$E_{e,j}(\mathbf{X}) = E_{e,j}(\mathbf{X}_{eq}) + \Delta\mathbf{X}^T \nabla_n E_{e,j}(\mathbf{X}_{eq}) + \frac{1}{2}\Delta\mathbf{X}^T \left[\nabla_n : \nabla_n E_{e,j}(\mathbf{X}_{eq})\right] \Delta\mathbf{X} + O\left(\Delta\mathbf{X}^3\right) \tag{2.31}$$

where $\Delta\mathbf{X} = \mathbf{X} - \mathbf{X}_{eq}$ and ∇_n is the gradient with respect to all of the nuclear coordinates.

For Equation 2.30 to be true, $\nabla_n E_{e,j}(\mathbf{X}_{eq})$ must vanish. Using this fact and assuming that nuclear displacements are small ($\Delta\mathbf{X} \approx 0$), the adiabatic potential can

be simplified.

$$E_{e,j}(\mathbf{X}) = E_{e,j}(\mathbf{X}_{eq}) + \frac{1}{2}\Delta\mathbf{X}^T \left[\nabla_n : \nabla_n E_{e,j}(\mathbf{X}_{eq})\right] \Delta\mathbf{X} \tag{2.32}$$

$E_{e,j}(\mathbf{X}_{eq})$ is the energy of the j^{th} electronic state calculated with the nuclei fixed at the equilibrium geometry. Because this is a constant, it only shifts the final eigenvalue of Equation 2.29 and can therefore be subtracted from the Hamiltonian. This yields a 3M-dimensional Schrödinger equation for the nuclear portion of the system

$$\left[\hat{\mathbf{T}}_n + \frac{1}{2}\Delta\mathbf{X}^T \left[\nabla_n : \nabla_n E_{e,j}(\mathbf{X}_{eq})\right] \Delta\mathbf{X}\right] |\Upsilon_k\rangle = E_{n,jk} |\Upsilon_k\rangle \tag{2.33}$$

where M is the number of nuclei and $E_{n,jk}$ is the nuclear energy for the system which is in the j^{th} electronic sate and k^{th} nuclear state.

Because the potential energy for a molecular system (Equation 2.20) is invariant to rotations and translations of the entire system, it can be shown that $E_{e,j}(\mathbf{X}_{eq})$ and $\nabla_n : \nabla_n E_{e,j}(\mathbf{X}_{eq})$ are also invariant to rotations and translations of the entire system. Using this result, the nuclear Hamiltonian (Equation 2.33) can be broken down into translational, rotational, and vibrational Hamiltonians.

$$\hat{\mathbf{H}}_{n,j} = \hat{\mathbf{H}}_{t,j} \oplus \hat{\mathbf{H}}_{r,j} \oplus \hat{\mathbf{H}}_{v,j} \tag{2.34}$$

The nuclear wave function for the system is now the tensor product of a translational,

a rotational, and a vibrational wave function.

$$|\Upsilon_{jklm}\rangle = |\psi_{t,k}\rangle \otimes |\psi_{r,l}\rangle \otimes |\psi_{v,m}\rangle \tag{2.35}$$

This manipulation allows the translational, rotational, and vibrational portions of the nuclear Schrödinger equation to be solved independently.

$$\hat{\mathbf{H}}_{t,j} |\psi_{t,k}\rangle = E_{t,jk} |\psi_{t,k}\rangle \tag{2.36}$$

$$\hat{\mathbf{H}}_{r,j} |\psi_{r,l}\rangle = E_{r,jl} |\psi_{r,l}\rangle \tag{2.37}$$

$$\hat{\mathbf{H}}_{v,j} |\psi_{v,m}\rangle = E_{v,jm} |\psi_{v,m}\rangle \tag{2.38}$$

$$\tag{2.39}$$

These Schrödinger equations are 3-dimensional, 3-dimensional, and $(3M-6)$-dimensional, respectively, and can be solved analytically.

2.5.3 Summary

Sections 2.5.1 and 2.5.2 have shown the approximations necessary to break the $(3M+3N)$-dimensional molecular Schrödinger equation into a more manageable $3N$-dimensional electronic Schrödinger equation, a $(3M-6)$-dimensional vibrational Schrödinger equation, a 3-dimensional rotational Schrödinger equation, and a 3-dimensional translational Schrödinger equation. These approximations significantly reduce the effort of calculating molecular properties.

Of these Schrödinger equations, the electronic Schrödinger equation is *by far* the

most difficult to solve. Because of this, it will be used in examples of methods throughout this work.

Chapter 3

Random Number Generation

> *God not only plays dice. He also sometimes throws the dice where they cannot be seen. [20]*
>
> Stephen Hawking, 1975

Many types of numerical simulations, including Quantum Monte Carlo, require the generation of random numbers with respect to a given probability density function. This happens to be significantly more difficult on a computer than one might initially expect.

The result of an inherently random physical process, such as the decay of radioactive nuclei, yields *truly* random results. Computers, on the other hand, are precise and deterministic; therefore, "random" numbers generated by computers are often called pseudo-random numbers. Pseudo-random numbers are generated by deterministic computational processes, but the numbers satisfy one or more statistical tests for randomness. The more statistical tests for randomness a sequence of pseudo-random numbers passes, the higher the quality of the pseudo-random numbers. For many problems, high-quality pseudo-random numbers are overkill, but, for other problems,

high-quality pseudo-random numbers are critical to obtaining the correct results for a calculation.

3.1 Uniform Random Numbers

Virtually all schemes to generate random numbers with respect to a given probability density function rely on uniform random numbers. Uniform random numbers are random numbers that fall between 0 and 1, with all numbers having an equal probability of being generated.

The most commonly used algorithms for generating uniform pseudo-random numbers are based on linear congruential generators. A sequence $\{I_i\}$ of nonnegative integers is generated by means of the fundamental congruence relationship

$$I_{i+1} = aI_i + c \pmod{m}, \tag{3.1}$$

where the multiplier a, the increment c, and the modulus m are nonnegative integers.

From Equation 3.1, it is easy to show that $I_i < m$ for all i. Because of this, the sequence $\{I_i\}$ contains at most m distinct numbers. Using this result, a set of uniform pseudo-random numbers, $\{U_i\}$, can be obtained by letting

$$U_i = \frac{I_i}{m}. \tag{3.2}$$

Because Equation 3.1 is deterministic and because I_i is bounded, the sequence $\{I_i\}$ is composed of repeating subsequences. The period of the sequence $\{I_i\}$, p, is

equal to the length of the repeating subsequence. As an example, consider the case where $a = c = I_0 = 3$ and $m = 5$. Here the generator, $I_{i+1} = 3I_i + 3 \pmod 5$, produces the sequence $\{3, 2, 4, 0, 3, 2, 4, \cdots\}$. This sequence is composed of repetitions of the subsequence $\{3, 2, 4, 0\}$ and has a period of $p = 4$.

Obviously when generating pseudo-random numbers, a and c should be chosen so that the sequence $\{I_i\}$ has a maximum period ($p = m$). This ensures that the uniform pseudo-random number generator produces the maximum number of distinct pseudo-random numbers. This full period occurs if and only if [21]:

1. c is relatively prime to m (or equivalently $\gcd(c, m) = 1$).

2. $a \equiv 1 \pmod g$ for every prime factor g of m.

3. $a \equiv 1 \pmod 4$ if m is a multiple of 4.

Because current computers use binary numbers, m is typically chosen to be close to 2^β, where β is the length of a long integer on the computer.

The quality of sequences generated using linear congruential generators is determined by the period length and the results of standard statistical tests for pseudo-random numbers. Details of these tests will not be covered here but can be found in Reference [21]. Values of a, c, and m which perform well can be found in *Numerical Recipes* [22, 23] and the literature.

Modifications can be made to linear congruential generators to improve the algorithm's results in standard statistical tests [23]. One such modification simply shuffles the sequence generated by a linear congruential generator.

In addition to linear congruential generators, uniform random numbers can be created using multiplicative congruential generators. These generators are the same as the linear version except $c = 0$. In this case, it is not possible to choose a so that the sequence $\{I_i\}$ has a full period; however, to optimize the method, it is possible to choose a and I_0 so that the sequence has a maximum period. Because fewer operations are performed, multiplicative congruential generators are faster than linear congruential generators.

3.2 Transformation Method

The transformation method of generating random numbers transforms uniform random numbers to random numbers with a given probability distribution, $\rho(x)$.

The cumulative distribution function, $F(y)$, is defined as

$$F(y) = \int_{-\infty}^{y} \rho(x)dx \tag{3.3}$$

where $F(-\infty) = 0$ and $F(\infty) = 1$, because $\rho(x)$ is normalized.

To generate a random number, y, distributed with respect to $\rho(x)$, a uniform random number, ζ, is generated. Then $y = F^{-1}(\zeta)$, where F^{-1} is the inverse of F. Often it is impossible to determine F^{-1} either analytically or numerically. Other times, F^{-1} is prohibitively expensive to evaluate. In these cases, the transformation method is not applicable, and another algorithm must be used.

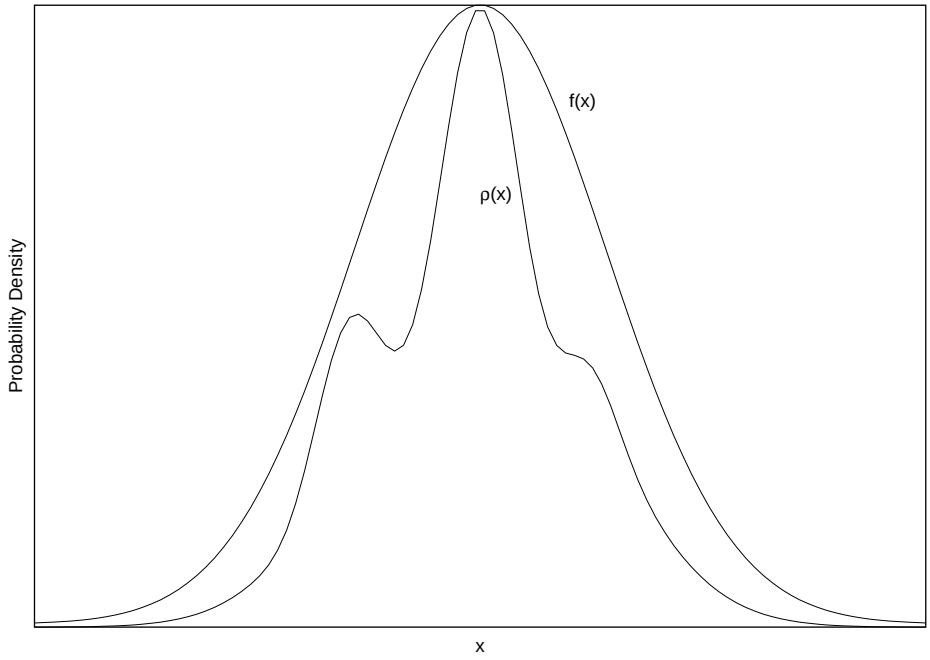


Figure 3.1: Example comparison function, $f(x)$, to generate random points distributed with respect to $\rho(x)$.

3.3 Rejection Method

The rejection method generates random numbers with respect to a given probability distribution, $\rho(x)$, which is known and computable. Unlike the transformation method (Section 3.2), evaluating the cumulative distribution function or its inverse is not required. This allows distributions of random numbers to be generated which were impossible using the transformation method.

A function $f(x)$, called the comparison function, is constructed so that it has a finite area and lies everywhere above $\rho(x)$ (Figure 3.1). The transformation method (Section 3.2) is then used to generate a random number, y, distributed with respect

to $\rho_f(x)$.

$$\rho_f(x) = \frac{f(x)}{\int_{-\infty}^{\infty} f(x)dx} \tag{3.4}$$

A uniform random number, ζ, is then generated. If $\zeta > \rho(y)/f(y)$, y is rejected; otherwise, y is accepted and is a random number distributed with respect to $\rho(x)$.

With the rejection method, it is possible to generate random numbers distributed with respect to essentially any distribution encountered in calculations of physical systems. Unfortunately, the algorithm is not always efficient. If it is impossible to construct a comparison function, $f(x)$, which closely approximates the probability distribution, $\rho(x)$, a large fraction of the random numbers generated with respect to $\rho_f(x)$ will be rejected, rendering the algorithm very inefficient.

3.4 Metropolis Algorithm

The Metropolis algorithm [24] begins by assuming the master equation:

$$\frac{\partial \rho(x,t)}{\partial t} = \int \left[T(y \to x)\rho(y,t) - T(x \to y)\rho(x,t)\right] dy \tag{3.5}$$

$$\int T(x \to y)dy = 1 \tag{3.6}$$

where $\rho(x,t)$ is the probability distribution at time t and $T(x \to y)$ is the transition probability for moving from x to y. From this, it is then assumed that the system is

in equilibrium ($\partial\rho(x,t)/\partial t \approx 0$), and the time dependence in $\rho(x,t)$ is dropped.

$$\int \left[T(y \to x)\rho(y) - T(x \to y)\rho(x)\right] dy = 0 \tag{3.7}$$

There are many possible solutions to Equation 3.7. The Metropolis solution [24] is the most simple and has proven to be the most efficient in actual use.

$$T(y \to x)\rho(y) = T(x \to y)\rho(x) \tag{3.8}$$

This is also known as the detailed balance solution.

Using Equation 3.8, the probability for accepting an attempted move from x to y is given by

$$A(y,x) = \min\left(1, \frac{T(y \to x)\rho(y)}{T(x \to y)\rho(x)}\right). \tag{3.9}$$

In Equation 3.9, it should be noted that the ratio $\rho(y)/\rho(x)$ is calculated, rather than $\rho(x)$ and $\rho(y)$ separately. As a result, ρ is not required to be normalized.

In the most simple implementation of the Metropolis algorithm, T is chosen so that $T(y \to x) = T(x \to y)$. More elaborate choices for $T(x \to y)$ can be used to increase the probability of accepting an attempted move and, therefore, to improve the algorithm's efficiency.

The above machinery provides all of the components necessary to produce random numbers distributed with respect to a given distribution, $\rho(x)$, no matter how com-

plex the distribution. The random numbers distributed with respect to $\rho(x)$ are the numbers x_i. Begin by choosing an initial point x_0. To generate the $(i+1)^{th}$ random number, choose a new random point, y_i, and generate a uniform random number, ζ. If $\zeta > A(y_i, x_i)$, $x_{i+1} = x_i$; otherwise, $x_{i+1} = y_i$. This process is repeated until the desired number of random points have been calculated.

The Metropolis algorithm assumes that the system is in equilibrium ($\partial\rho(x,t)/\partial t \approx 0$). Because the initial point x_0 is arbitrary, this assumption is not necessarily valid for the first random points which are generated. For example, if random points are generated with respect to a Gaussian distribution and x_0 is chosen to be in the tail of the distribution, the next random points will likely be near to x_0. Since x_0 and its neighboring points have a low probability of occurring, because they are in the tail of the distribution, this region is oversampled compared to the distribution $\rho(x)$. As the algorithm iterates, it reaches equilibrium, and the random points are generated with respect to $\rho(x)$. Therefore, some number of initial random points must be discarded while the calculation reaches equilibrium. This is an initialization expense inherent in this algorithm. Intelligent choices for x_0 can shorten this equilibration period.

Chapter 4

Introduction to Quantum Monte Carlo

> *Insofar as the laws of quantum mechanics are correct, chemical questions are problems in applied mathematics. [25]*
>
> H. Eyring, J. Walter, and G. E. Kimball, 1944

Quantum Monte Carlo (QMC) is becoming the method of choice for high-accuracy quantum-mechanical calculations of atomic and molecular systems [26, 27, 28, 29, 30]. QMC scales as $O(N^3)$ while other very high-level methods, such as coupled-cluster, scale as $O(N^6)$ or worse. Additionally, a new algorithm by Williamson, Hood, and Grossman makes QMC scale as $O(N)$ for systems with localized electrons and more than about 20 electrons [31]. QMC's favorable scaling makes possible the high-accuracy examination of compounds too large to study with other methods. Furthermore, QMC is a stochastic method, so it is possible to parallelize a calculation over a large number of processors.

QMC refers not to one specific method but rather to an entire class of methods. These methods have been applied to problems covering everything from chemistry to

quantum field theory.

Two flavors of QMC are the most important to electronic structure theory calculations: variational QMC (VMC) and diffusion QMC (DMC). Sections 4.1 and 4.2 will discuss these methods.

4.1 Variational Quantum Monte Carlo

Variational Quantum Monte Carlo (VMC) is conceptually very simple. A parameterized wave function is constructed; the parameters are then adjusted to minimize the energy expectation value or the variance in this quantity. The variational theorem (Section 2.4) proves that minimizing the energy expectation value provides an approximation to the ground state wave function given the wave function's particular parameterization. Minimizing the energy expectation value's variance can be used to approximate any eigenfunction.

When using the Born-Oppenheimer approximation (Section 2.5.1), the energy expectation value for an atomic system, $\langle E \rangle$, is

$$\langle E \rangle = \frac{\langle \Psi | \hat{\mathbf{H}} | \Psi \rangle}{\langle \Psi | \Psi \rangle} \tag{4.1}$$

$$= \frac{\int \Psi^*(\mathbf{x}) \hat{\mathbf{H}} \Psi(\mathbf{x}) d\mathbf{x}^{3N}}{\int \Psi^*(\mathbf{x}) \Psi(\mathbf{x}) d\mathbf{x}^{3N}} \tag{4.2}$$

where Ψ is a wave function, $\hat{\mathbf{H}}$ is the electronic Hamiltonian operator for the system, N is the number of electrons in the system, and $\mathbf{x}$ is a $3N$-dimensional vector containing

the positions of all N electrons. Manipulating this expression yields

$$\langle E \rangle = \frac{\int \Psi^*(\mathbf{x}) \hat{\mathbf{H}} \Psi(\mathbf{x}) d\mathbf{x}^{3N}}{\int \Psi^*(\mathbf{x}) \Psi(\mathbf{x}) d\mathbf{x}^{3N}} \tag{4.3}$$

$$= \frac{\int |\Psi(\mathbf{x})|^2 \frac{\hat{\mathbf{H}} \Psi(\mathbf{x})}{\Psi(\mathbf{x})} d\mathbf{x}^{3N}}{\int |\Psi(\mathbf{x})|^2 d\mathbf{x}^{3N}} \tag{4.4}$$

$$= \int \rho_{VMC}(\mathbf{x}) E_{local}(\mathbf{x}) d\mathbf{x}^{3N} \tag{4.5}$$

where

$$\rho_{VMC}(\mathbf{x}) \equiv \frac{|\Psi(\mathbf{x})|^2}{\int |\Psi(\mathbf{x})|^2 d\mathbf{x}^{3N}} \tag{4.6}$$

is the probability for the electrons to have positions $\mathbf{x}$ and

$$E_{local}(\mathbf{x}) \equiv \frac{\hat{\mathbf{H}} \Psi(\mathbf{x})}{\Psi(\mathbf{x})} \tag{4.7}$$

is the energy for electrons with positions $\mathbf{x}$.

There are many approaches to solve Equation 4.5. Hartree-Fock uses an independent particle approximation to break the $3N$-dimensional integral into N easily evaluated 3-dimensional integrals. The accuracy of this approach suffers because it replaces explicit electron-electron interactions with average interactions. It is also possible to apply standard integration algorithms to Equation 4.5, but such approaches scale as $O(2^N)$, rendering them computationally infeasible for all but the simplest problems.

On the other hand, VMC employs Monte Carlo integration [15] to evaluate Equation 4.5. In Monte Carlo integration, M random vectors, x_i, distributed with respect

to $\rho_{VMC}(\mathbf{x})$, are generated. The energy expectation value is then found to be

$$\langle E \rangle = \frac{1}{M} \sum_{i=1}^{M} E_{local}(x_i) \pm O\left(\frac{1}{\sqrt{M}}\right). \tag{4.8}$$

Here the standard deviation in the calculated expected energy falls off with the square root of the number of random samples used. This error is independent of the problem's dimensionality; thus, Monte Carlo integration is faster than standard integration algorithms when the integral's dimensionality is greater than about 7 [15, 23]. For atomic and molecular systems, the Metropolis algorithm (Section 3.4) is used to generate x_i since $\rho_{VMC}(\mathbf{x})$ is a complicated $3N$-dimensional function.

Optimizing the wave function parameters is difficult. Because Monte Carlo integration is used to evaluate the energy expectation value and its variance, these quantities are stochastic, and therefore result in a hard to optimize noisy objective function. In order to minimize this noise, a correlated sampling optimization procedure [32] is often used.

4.1.1 Variational Quantum Monte Carlo Wave Functions

Any antisymmetric wave function may serve as the electronic wave function for VMC calculations of atomic and molecular systems. Nonetheless, the closer the wave function is to the desired, exact eigenfunction, the faster the VMC calculation converges and the less parameter optimization is required to obtain the optimal solution.

A good general purpose VMC wave function, Ψ_{VMC}, can be constructed with the

form

$$\Psi_{VMC} = \sum_i c_i \psi_i J \tag{4.9}$$

where c_i are constants, ψ_i is a determinantal wave function which is the product of a Slater determinant for the up-spin electrons and a Slater determinant for the down-spin electrons, and J is a symmetric function of the electron-electron and electron-nuclear distances called the Jastrow function (Section 4.1.2). This gives an overall wave function which is antisymmetric and includes explicit electron-electron correlations. $\{c_i\}$ and $\{\psi_i\}$ can be obtained from standard electronic structure calculations such as Hartree-Fock (HF), Density Functional Theory (DFT), Multi-Configuration Self Consistent Field (MCSCF), and Configuration Interaction (CI).

There are many adjustable parameters in this general-purpose wave function. These include the c_i, the parameters in ψ_i, and the parameters in J. Although optimization of the determinantal wave function parameters is possible, this is often a poor strategy because finding derivatives with respect to these parameters adds poorly scaling steps to the calculation.

4.1.2 Jastrow Functions

A Jastrow function (Equation 4.9) is a symmetric function of all of the electron-electron and electron-nuclear distances. This function introduces explicit particle-particle correlations into the wave function.

The Jastrow function can be expanded as a sum of 1-body, 2-body, etc., terms. It has been shown that the most important terms are the electron-nuclear and electron-

electron terms [26, 33]; therefore, the majority of calculations employ only these terms. Such a Jastrow function can be expressed as

$$J = \exp\left(\sum u_{ij}(r_{ij})\right) \tag{4.10}$$

where the sum is over all electron-electron and electron-nuclear pairs, r_{ij} is the distance between particles i and j, and $u_{ij}(r)$ is a function describing the correlations of particles i and j in the wave function.

If the determinantal wave functions are constructed using Gaussian orbitals, it is straightforward to show that the cusp condition (Section 2.3) for particles α and β simplifies to

$$\lim_{r\to 0} \frac{\partial u_{\alpha\beta}(r)}{\partial r} = -\frac{\mu_{\alpha\beta} q_\alpha q_\beta}{l+1} \tag{4.11}$$

where $\mu_{\alpha\beta} = m_\alpha m_\beta/(m_\alpha + m_\beta)$ is the reduced mass, q_α and q_β are the charges of the particles, and l is 1 for same-spin electrons and 0 otherwise. This simple result is obtained because the radial derivative of a Gaussian orbital, averaged over an infinitesimally small sphere centered at $r = 0$, is zero at $r = 0$.

Constructing the VMC wave function (Equation 4.9) to obey the cusp conditions removes all singularities from $E_{local}(x)$. This yields a smaller variance in Equation 4.8 and thus faster convergence of the VMC calculation.

Many functional forms for $u_{ij}(r)$ have been used. The most common form for

finite atomic and molecular simulations is the Padé-Jastrow function

$$u_{ij}(r) = \frac{\sum_{k=1}^{N} a_{ij,k} r^k}{1 + \sum_{k=1}^{M} b_{ij,k} r^k} \tag{4.12}$$

where $a_{ij,k}$ and $b_{ij,k}$ are constants. To satisfy the cusp conditions, $a_{ij,0}$ must be set to the value of the cusp condition for particles i and j; other parameters are not affected. If $N > M$, $\lim_{r\to\infty} u_{ij}(r) = \pm\infty$. This can cause problems with numerical stability when implemented on a computer. M and N are typically chosen so that $N \leq M$ to ensure that the limit is finite.

The wave function's symmetry (Section 2.2) can be used to simplify the Jastrow function. Because the Jastrow function is totally symmetric, interchanging the positions of two identical particles must not alter the wave function. Therefore, if particles i and j are identical, $u_{ik}(r) = u_{jk}(r)$.

4.2 Diffusion Quantum Monte Carlo

Diffusion Quantum Monte Carlo (DMC) has the potential to calculate "exact" expectation values for N-body quantum-mechanical problems. The increased accuracy, relative to VMC, comes at the expense of additional complexity and computational effort.

Beginning with the time-dependent Schrödinger equation

$$i\frac{\partial}{\partial t}\left|\Psi(t)\right\rangle = (\hat{\mathbf{H}} - E_T)\left|\Psi(t)\right\rangle \tag{4.13}$$

where $\hat{\mathbf{H}}$ is the Hamiltonian operator for the system and E_T is an arbitrary constant that changes the phase of the wave function, a change of variables to "imaginary time", $t \rightarrow -i\tau$, gives a diffusion equation.

$$-\frac{\partial}{\partial\tau}\left|\Psi(\tau)\right\rangle = (\hat{\mathbf{H}} - E_T)\left|\Psi(\tau)\right\rangle \tag{4.14}$$

As long as $\hat{\mathbf{H}}$is time-independent, the formal solution to Equation 4.14 can be written as

$$\left|\Psi(\tau)\right\rangle = e^{-(\hat{\mathbf{H}}-E_T)\tau}\left|\Psi(0)\right\rangle. \tag{4.15}$$

This solution can be expanded in terms of the eigenfunctions of the Hamiltonian operator as

$$\left|\Psi(\tau)\right\rangle = \sum_j c_j e^{-(E_j-E_T)\tau}\left|\Phi_j\right\rangle \tag{4.16}$$

where $c_j = \langle\Phi_j|\Psi(0)\rangle$ are constant coefficients and E_j and $|\Phi_j\rangle$ are the j^{th} eigenvalue and eigenfunction of $\hat{\mathbf{H}}$. Because $E_0 < E_1 < E_2 < \cdots$ (Section 2.1),

$$\lim_{\tau\to\infty}\left|\Psi(\tau)\right\rangle \rightarrow \lim_{\tau\to\infty} c_\alpha e^{-(E_\alpha-E_T)\tau}\left|\Phi_\alpha\right\rangle \tag{4.17}$$

where α is the lowest energy state that is not orthogonal to $|\Psi(0)\rangle$. Furthermore, if E_T is chosen to equal E_α

$$\lim_{\tau\to\infty}\left|\Psi(\tau)\right\rangle = c_\alpha\left|\Phi_\alpha\right\rangle. \tag{4.18}$$

It is clear from the above analysis that contributions to $|\Psi(\tau)\rangle$ from excited states higher in energy than α decay exponentially with τ. DMC is built upon this mathe-

matics.

Because it is easy to construct a wave function which is not orthogonal to the ground state by using a standard method, such as HF, DFT, MCSCF, CI, etc., and because the ground state is a system's lowest energy state, DMC calculations on the ground state are relatively straightforward to perform. Calculations of excited state properties for atomic and molecular systems are possible, but they are beyond the scope of this text. For details on such methods, see Reference [34].

The rest of this section discusses details on the implementation and convergence of DMC.

4.2.1 DMC Energy Evaluation

The DMC energy, E_{DMC}, is evaluated using a mixed estimator

$$E_{DMC} = \frac{\langle \Phi_\alpha | \hat{\mathbf{H}} | \Psi \rangle}{\langle \Phi_\alpha | \Psi \rangle} \tag{4.19}$$

$$= \frac{\int \Phi_\alpha(\mathbf{x}) \hat{\mathbf{H}} \Psi(\mathbf{x}) d\mathbf{x}^{3N}}{\int \Phi_\alpha(\mathbf{x}) \Psi(\mathbf{x}) d\mathbf{x}^{3N}} \tag{4.20}$$

where $|\Psi\rangle$ is an approximation to the desired eigenstate $|\Phi_\alpha\rangle$ and $\mathbf{x}$ is a $3N$-dimensional vector of the coordinates of all N particles. Because $\hat{\mathbf{H}}$ is Hermitian and $|\Phi_\alpha\rangle$ and $|\Psi\rangle$ are real, $\langle \Phi_\alpha | \hat{\mathbf{H}} | \Psi \rangle = \langle \Psi | \hat{\mathbf{H}} | \Phi_\alpha \rangle$, and it is easy to show that $E_{DMC} = E_\alpha$. Further rearrangement of Equation 4.20 yields

$$E_{DMC} = \frac{\int \Phi_\alpha(\mathbf{x}) \Psi(\mathbf{x}) \frac{\hat{\mathbf{H}}\Psi(\mathbf{x})}{\Psi(\mathbf{x})} d\mathbf{x}^{3N}}{\int \Phi_\alpha(\mathbf{x}) \Psi(\mathbf{x}) d\mathbf{x}^{3N}} \tag{4.21}$$

$$= \int \rho_{DMC}(\mathbf{x}) E_{local}(\mathbf{x}) d\mathbf{x}^{3N} \tag{4.22}$$

which is of the same form as the VMC energy expression (Equation 4.5). Just as in VMC,

$$E_{local}(\mathbf{x}) \equiv \frac{\hat{\mathbf{H}}\Psi(\mathbf{x})}{\Psi(\mathbf{x})} \tag{4.23}$$

but now

$$\rho_{DMC}(\mathbf{x}) \equiv \frac{f(\mathbf{x})}{\int f(\mathbf{x}) d\mathbf{x}^{3N}} \tag{4.24}$$

where

$$f(\mathbf{x}) \equiv \Phi_\alpha(\mathbf{x})\Psi(\mathbf{x}). \tag{4.25}$$

In VMC, interpreting $\rho_{VMC}(\mathbf{x})$ as a probability distribution is very straightforward, but the same interpretation in DMC has technicalities. Using symmetry (Section 2.2), it is easy to prove that the ground state wave function for a system of bosons, $\Phi_0(\mathbf{x})$, is positive (or negative) for all $\mathbf{x}$. Then, if $\Psi(\mathbf{x})$ is constructed to have the same sign as $\Phi_0(\mathbf{x})$ for all $\mathbf{x}$, $f(\mathbf{x})$ will be positive for all $\mathbf{x}$, and $\rho_{DMC}(\mathbf{x})$ can be interpreted as a probability distribution.

Excited states of the bosonic ground state wave function have nodes and thus regions of positive and negative values. If, somehow, $\Psi(\mathbf{x})$ is constructed to have the same nodal structure as $\Phi_\alpha(\mathbf{x})$, $f(\mathbf{x})$ will be non-negative for all $\mathbf{x}$ and can be interpreted as a probability distribution. Unfortunately, the paucity of mathematical analysis of the nodal structure of many-particle wave functions in the literature renders the task of constructing $\Psi(\mathbf{x})$ with the same nodal structure as $\Phi_\alpha(\mathbf{x})$ nearly

impossible at this point.

If $\Psi(\mathbf{x})$ and $\Phi_\alpha(\mathbf{x})$ have different nodal structures, $f(\mathbf{x})$, and thus $\rho_{DMC}(\mathbf{x})$, will posses both positive and negative regions, so $\rho_{DMC}(\mathbf{x})$ can not be interpreted as a probability distribution. This is known as the *nodal problem*. Because the ground state of a system of fermions is the lowest-energy totally-antisymmetric state of a system of bosons (Section 2.2), the nodal problem is very important in calculating atomic and molecular properties.

The most simple and commonly used solution to the nodal problem is the *fixed-node approximation*. In this approximation, $\Phi_\alpha(\mathbf{x})$ is assumed to have the same nodal structure as $\Psi(\mathbf{x})$. $\rho_{DMC}(\mathbf{x})$ can then be interpreted as a probability distribution. The energy resulting from fixed-node calculations lies above the exact energy and is variational in the nodal structure of $\Psi(\mathbf{x})$ [35]. Furthermore, the difference in fixed-node energy from the exact energy is second order in $(\Phi_\alpha(\mathbf{x}) - \Psi(\mathbf{x}))$ [36]. *A posteriori* comparisons with experimental and known, exact results show that standard wave functions (e.g., HF, DFT, MCSCF, CI), and therefore standard VMC wave functions, typically have "good-enough"-quality nodes for DMC calculations of small molecular systems to have errors significantly less than 1 kcal/mol. In some cases, such as Be, multi-configuration wave functions must be used to obtain high-quality nodes.

Other solutions to the nodal problem exist [37, 38, 39, 40], but thus far, they have proven neither to scale well nor to be robust enough for routine calculations.

Assuming that $\rho_{DMC}(\mathbf{x})$ can be interpreted as a probability distribution, Monte

Carlo integration [15] can be used to evaluate Equation 4.22

$$E_{DMC} = \frac{1}{M}\sum_{i=1}^{M} E_{local}(x_i) \pm O\left(\frac{1}{\sqrt{M}}\right) \tag{4.26}$$

where x_i are M random $3N$-dimensional points distributed with probability density $\rho_{DMC}(\mathbf{x})$. Section 4.2.2 covers the generation of x_i.

To calculate the expectation value of an operator which does not commute with $\hat{\mathbf{H}}$, $[\hat{\mathbf{O}}, \hat{\mathbf{H}}] \neq 0$, a correction must be applied to the mixed estimator calculated using DMC [41, 38, 35].

$$\frac{\langle \Phi_\alpha | \hat{\mathbf{O}} | \Phi_\alpha \rangle}{\langle \Phi_\alpha | \Phi_\alpha \rangle} = 2\frac{\langle \Phi_\alpha | \hat{\mathbf{O}} | \Psi \rangle}{\langle \Phi_\alpha | \Psi \rangle} - \frac{\langle \Psi | \hat{\mathbf{O}} | \Psi \rangle}{\langle \Psi | \Psi \rangle} + O\left([\Phi_\alpha - \Psi]^2\right) \tag{4.27}$$

Here the desired result is two times the DMC result minus the VMC result.

4.2.2 DMC Random Point Generation

For a DMC calculation (Section 4.2.1), it is necessary to generate random points with respect to a probability distribution $\rho_{DMC}(\mathbf{x})$ (Equations 4.24 and 4.25), where care has been taken to address the nodal problem (Section 4.2.1).

The non-dimensionalized Hamiltonian for a system of N identical particles is

$$\hat{\mathbf{H}} = -\frac{1}{2}\nabla^2 + V \tag{4.28}$$

where V is the potential energy and the derivatives are with respect to all of the

particles' $3N$ coordinates. From the Hamiltonian in Equation 4.28, it is possible to construct, with a significant amount of algebra, a new Hamiltonian, $\hat{\mathbf{L}}$, which has eigenvalue-eigenfunction pairs of E_i and $\Phi_i(\mathbf{x})\Psi(\mathbf{x})$; E_i and $\Phi_i(\mathbf{x})$ are the eigenvalue-eigenfunction pairs from Equation 4.28, and $\Psi(\mathbf{x})$ is the approximate wave function discussed in Section 4.2.1.

$$\hat{\mathbf{L}} = -\frac{1}{2}\nabla^2 + \nabla \bullet (\nabla \ln |\Psi(\mathbf{x})|) + E_{local}(\mathbf{x}) \tag{4.29}$$

$E_{local}(\mathbf{x}) \equiv \hat{\mathbf{H}}\Psi(\mathbf{x})/\Psi(\mathbf{x})$ is the local energy of a given configuration of electrons for the approximate wave function $\Psi(\mathbf{x})$.

Just as was discussed in Section 4.2, the time-dependent Schrödinger equation for $\hat{\mathbf{L}}$

$$i\frac{\partial}{\partial t}|f(t)\rangle = (\hat{\mathbf{L}} - E_T)|f(t)\rangle \tag{4.30}$$

can undergo a change of variables to "imaginary time", $t \rightarrow -i\tau$, to give

$$-\frac{\partial}{\partial \tau}|f(\tau)\rangle = (\hat{\mathbf{L}} - E_T)|f(\tau)\rangle \tag{4.31}$$

which, if $\hat{\mathbf{H}}$is time-independent, has the formal solution

$$|f(\tau)\rangle = e^{-(\hat{\mathbf{L}}-E_T)\tau}|f(0)\rangle \tag{4.32}$$

where E_T is an arbitrary constant that changes the phase of the "real time" wave function. The formal solution can be expanded in terms of the eigenfunctions of $\hat{\mathbf{L}}$ to

give

$$f(\mathbf{x},\tau) = \sum_j c_j e^{-(E_j - E_T)\tau} \Phi_j(\mathbf{x}) \Psi(\mathbf{x}) \tag{4.33}$$

where $c_j = \int \Phi_j(\mathbf{x}) \Psi(\mathbf{x}) f(\mathbf{x},0) d\mathbf{x}^{3N}$. As was the case in Section 4.2, the high-energy components die out exponentially with τ. Once again, because $E_0 < E_1 < E_2 < \cdots$,

$$\lim_{\tau\to\infty} f(\mathbf{x},\tau) \to \lim_{\tau\to\infty} c_\alpha e^{-(E_\alpha - E_T)\tau} \Phi_\alpha(\mathbf{x}) \Psi(\mathbf{x}) \tag{4.34}$$

where α is the smallest value for which $c_\alpha \neq 0$. If E_T is chosen to equal E_α,

$$\lim_{\tau\to\infty} f(\mathbf{x},\tau) = c_\alpha \Phi_\alpha(\mathbf{x}) \Psi(\mathbf{x}). \tag{4.35}$$

This is proportional to $\rho_{DMC}(\mathbf{x})$. Therefore, random points generated with the distribution $f(\mathbf{x},\tau)$, as $\tau \to \infty$, are also distributed with respect to $\rho_{DMC}(\mathbf{x})$. This is what is required to evaluate the DMC energy using Monte Carlo integration (Equation 4.26).

Equation 4.32 can be expressed in the functional representation as

$$f(\mathbf{y},\tau) = \int G(\mathbf{y},\mathbf{x},\tau) f(\mathbf{x},0) d\mathbf{x}^{3N} \tag{4.36}$$

where

$$G(\mathbf{y},\mathbf{x},\tau) = \langle y | \, e^{-(\hat{\mathbf{L}} - E_T)\tau} \, | x \rangle \tag{4.37}$$

is the Green's function for the problem. For nearly all quantum-mechanical problems of physical importance, it is impossible to efficiently evaluate $G(\mathbf{y},\mathbf{x},\tau)$ for arbitrary

τ. Fortunately, for small τ, $\delta\tau$, $G(\mathbf{y},\mathbf{x},\tau)$ can be factored into easy to evaluate pieces

$$G(\mathbf{y},\mathbf{x},\delta\tau) = G_{diffusion}(\mathbf{y},\mathbf{x},\delta\tau)G_{branching}(\mathbf{y},\mathbf{x},\delta\tau) + O(\delta\tau^2). \tag{4.38}$$

$G_{diffusion}(\mathbf{y},\mathbf{x},\delta\tau)$ is a function describing the probability of the point $\mathbf{x}$ moving to $\mathbf{y}$ in $\delta\tau$ imaginary time

$$G_{diffusion}(\mathbf{y},\mathbf{x},\delta\tau) = (2\pi\delta\tau)^{-3N/2}\, e^{-[\mathbf{y}-\mathbf{x}-\delta\tau\nabla \ln|\Psi(\mathbf{x})|]^2/2\delta\tau}, \tag{4.39}$$

and $G_{branching}(\mathbf{y},\mathbf{x},\delta\tau)$ is a function describing how the value of f changes in going from $(\mathbf{x},\tau)$ to $(\mathbf{y},\tau+\delta\tau)$

$$G_{branching}(\mathbf{y},\mathbf{x},\delta\tau) = e^{-\delta\tau(E_{local}(\mathbf{y})+E_{local}(\mathbf{x})-2E_T)/2}. \tag{4.40}$$

Using the small τ approximation, Equation 4.36 is

$$f\,(\mathbf{y},(n+1)\delta\tau) = \int G_{diffusion}(\mathbf{y},\mathbf{x},\delta\tau)G_{branching}(\mathbf{y},\mathbf{x},\delta\tau)f(\mathbf{x},n\delta\tau)d\mathbf{x}^{3N} + O(\delta\tau^2). \tag{4.41}$$

For large τ, $f(\mathbf{x},\tau)$ can be obtained by iteratively applying Equation 4.41.

Using the above results, it is now possible to produce an algorithm which generates random points distributed with respect to $\rho_{DMC}(\mathbf{x})$. Because Equation 4.41 is $3N$-dimensional, for most interesting problems, Monte Carlo integration is the fastest way to evaluate the integral. In the stochastic evaluation of this integral, a correspondence

can be established where $f(\mathbf{x}, \tau)$ is represented by

$$f(\mathbf{x}, \tau) \rightarrow \sum_k w_{k,\tau} \delta(\mathbf{x} - \mathbf{x}_{k,\tau}) \tag{4.42}$$

where $w_{k,\tau}$ is a statistical weight and $\delta(\mathbf{x} - \mathbf{x}_{k,\tau})$ is a Dirac delta function centered at $\mathbf{x}_{k,\tau}$. The pair $(\mathbf{x}_{k,\tau}, w_{k,\tau})$ is known as a *walker*. Combining Equations 4.41 and 4.42 gives

$$f\left(\mathbf{y}, (n+1)\delta\tau\right) = \sum_k w_{k,n\delta\tau} G_{diffusion}(\mathbf{y}, \mathbf{x}_{k,n\delta\tau}, \delta\tau) G_{branching}(\mathbf{y}, \mathbf{x}_{k,n\delta\tau}, \delta\tau). \tag{4.43}$$

Equation 4.43 can be returned to the delta function form (Equation 4.42). To do this, each of the new delta function locations, $\mathbf{x}_{k,(n+1)\delta\tau}$, is randomly chosen from the distribution $G_{diffusion}(\mathbf{y}, \mathbf{x}_{k,n\delta\tau}, \delta\tau)$. The new weights are then

$$w_{k,(n+1)\delta\tau} = G_{branching}(\mathbf{x}_{k,(n+1)\delta\tau}, \mathbf{x}_{k,n\delta\tau}, \delta\tau) w_{k,n\delta\tau}. \tag{4.44}$$

This new set of walkers is a stochastic representation of $f(\mathbf{x}, (n+1)\delta\tau)$. The new set of random points, $\mathbf{x}_{k,(n+1)\delta\tau}$, given the appropriate statistical weights, $w_{k,(n+1)\delta\tau}$, are random points distributed with respect to $f(\mathbf{x}, (n+1)\delta\tau)$.

By choosing $f(\mathbf{x}, 0)$ to be $|\Psi(\mathbf{x})|^2$, a stochastic representation of $f(\mathbf{x}, 0)$ can be generated by setting $w_{k,0} = 1$ and $x_{k,0}$ equal to random points generated with respect to $|\Psi(\mathbf{x})|^2$ using the Metropolis algorithm. After many applications of Equation 4.41, the walkers will provide a stochastic representation of $f(\mathbf{x}, \infty)$, which Equation 4.35

showed to be proportional to the distribution we are trying to sample, $\Phi_\alpha(\mathbf{x})\Psi(\mathbf{x})$. This produces the random numbers needed to evaluate the DMC energy.

As the calculation progresses, it is possible to improve upon the initial guess for E_T. This will ensure that the sum of the statistical weights remains relatively constant and does not exponentially decay or grow. Should the weights exponentially decay, they will become smaller than the machine precision on the computer used to calculate them and will contribute little or no information (due to the negligible statistical weights) to subsequent iterations. On the other hand, if the weights exponentially grow, a situation will be reached where the computer used for the calculation does not have enough memory to hold all of the walkers, or the statistical weights for the walkers will become larger than the machine's floating points. Neither situation results in a numerically stable, accurate calculation.

Because the small τ approximation has been made (Equation 4.38), it is necessary to extrapolate the DMC results to $\delta\tau = 0$ to compensate for the approximation. Unfortunately, small values of $\delta\tau$ yield inefficient calculations since the generated random numbers are highly serially correlated.

Other schemes exist for factoring the Green's function (Equations 4.37 and 4.38) and for recovering the delta function representation of $f(\mathbf{y}, (n+1)\delta\tau)$ from Equation 4.43. The details of these algorithms and their advantages and disadvantages are covered in the literature [38, 27, 35]. In my experience, Umrigar's algorithm [27] is the most stable, robust, and has the smallest time-step bias.

Chapter 5

Efficient Algorithm for "On-the-fly" Error Analysis of Local or Distributed Serially Correlated Data

The Dynamic Distributable Decorrelation Algorithm (DDDA), which efficiently calculates the true statistical error of an expectation value obtained from serially correlated data "on-the-fly," as the calculation progresses, is presented [42]. DDDA is an improvement on the Flyvbjerg-Petersen renormalization group blocking method [43]. This "on-the-fly" determination of statistical quantities allows dynamic termination of Monte Carlo calculations once a specified level of convergence is attained. This is highly desirable when the required precision might take days or months to compute, but cannot be accurately estimated prior to the calculation. Furthermore, DDDA allows for a parallel implementation which requires very low communication, $O(\log_2 N)$, and can also evaluate the variance of a calculation efficiently "on-the-fly." Quantum Monte Carlo calculations are presented to illustrate "on-the-fly" variance calculations for serial and massively parallel Monte Carlo calculations.

5.1 Introduction

Monte Carlo methods are becoming increasingly important in calculating the properties of chemical, biological, materials, and financial systems. The underlying algorithms of such simulations (e.g., Metropolis algorithm [24]) often involve Markov chains. The data generated from the Markov chains are serially correlated, meaning that the covariances between data elements is non-zero. Because of this, care must be taken to obtain the correct variances for observables calculated from the data.

Data blocking algorithms to obtain the correct variance of serially correlated data have been part of the lore of the Monte Carlo community for years. Flyvbjerg and Petersen were the first to formally analyze the technique [43], but at least partial credit should be given to Wilson [44], Whitmer [45], and Gottlieb [46] for their earlier contributions.

A new blocking algorithm, Dynamic Distributable Decorrelation Algorithm (DDDA), which gives the same results as the Flyvbjerg-Petersen algorithm but allows the underlying variance of the serially correlated data to be analyzed "on-the-fly" with negligible additional computational expense, is proposed. DDDA is also ideally suited for parallel computations because only a small amount of data must be communicated between processors to obtain the global results. Furthermore, an efficient method is presented for combining results from individual processors in a parallel calculation that allows fast "on-the-fly" result analysis for parallel calculations. Example calculations showing "on-the-fly" variance calculations for serial and massively parallel calculations are also presented.

All current blocking algorithms require $O(mN)$ operations to evaluate the variance m times during a calculation of N steps. DDDA only requires $O(N + m \log_2 N)$. Furthermore, current algorithms require communicating $O(N)$ data during a parallel calculation to evaluate the variance. DDDA requires only $O(\log_2 N)$.

5.2 Theory

Computer simulations of physical systems often involve the calculation of an expectation value, $\langle f \rangle$, using a complicated high-dimensional probability distribution function, $\rho(x)$.

$$\langle f \rangle \equiv \int \rho(x) f(x) dx \tag{5.1}$$

This expression is simple and elegant, but in many physical systems, $\rho(x)$ is too complex for Equation 5.1 to be useful computationally. Instead, simulations typically calculate the "time average" of f, $\bar{f}$.

$$\bar{f} \equiv \frac{1}{n} \sum_{i=1}^{n} f(x_i) \tag{5.2}$$

Here i is related to the Monte Carlo step number, and x_i is sampled from the distribution $\rho(x)$. Then, assuming ergodicity, $\langle f \rangle$ and $\bar{f}$ can be related through

$$\langle f \rangle = \lim_{n \to \infty} \bar{f} = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) \tag{5.3}$$

On modern computers, very large samplings are used to approach this limit. How-

ever, since such sampling is necessarily always finite, $\bar{f}$ will fluctuate as the calculation progresses because it has a non-zero variance, $\sigma^2(\bar{f})$. This variance can be expressed as

$$\sigma^2(\bar{f}) = \frac{1}{n^2}\sum_{i,j=1}^{n}\left[\langle f(x_i)f(x_j)\rangle - \langle f(x_i)\rangle\langle f(x_j)\rangle\right] \tag{5.4}$$

$$= \frac{\sigma^2(f)}{n} + \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j>i}^{n} cov(f(x_i), f(x_j)) \tag{5.5}$$

When the $\{f(x_i)\}$ are uncorrelated, the covariance terms are zero, and Equation 5.5 reduces to the typical variance relation.

$$\sigma^2(\bar{f}) = \frac{\langle f^2\rangle - \langle f\rangle^2}{n} = \frac{\sigma^2(f)}{n} \tag{5.6}$$

Calculations which use Markov chains to generate $\{x_i\}$, such as Metropolis algorithm [24] based calculations, produce $\{f(x_i)\}$ with non-zero covariances. This results because the probability of picking x_i is dependent on the value of x_{i-1}. If Equation 5.6 is used to calculate the variance of such systems, the result will be incorrect because the covariances between samples are not included.

Without loss of generality, Equations 5.2 and 5.5 can be expressed in terms of the random variate x_i instead of the random variate $f(x_i)$. This gives

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.7}$$

$$\sigma^2(\bar{x}) = \frac{1}{n^2}\sum_{i,j=1}^{n}\gamma_{i,j} \tag{5.8}$$

where $\gamma_{i,j} = cov(x_i, x_j)$.

Then, if it is assumed that a Markov chain method with stationary transition probabilities, such as Monte Carlo or molecular dynamics at equilibrium, was used to generate $\{x_i\}$,

$$\sigma^2(\bar{x}) = \frac{1}{n}\xi_0 + \frac{2}{n}\sum_{t=1}^{n-1}(n-t)\xi_t \tag{5.9}$$

where ξ_t is the covariance between data points t steps apart.

$$\xi_t \equiv \gamma_{i,j} \qquad t = |i-j| \tag{5.10}$$

In this representation, it is possible to define a blocking transform that takes $\{x_i\} \rightarrow \{x_i'\}$.

$$x_i' = \frac{1}{2}\{x_{2i-1} + x_{2i}\} \tag{5.11}$$

$$n' = \frac{1}{2}n \tag{5.12}$$

In performing this transform, it can be shown [43] that

$$\bar{x'} = \bar{x} \tag{5.13}$$

$$\sigma^2(\bar{x'}) = \sigma^2(\bar{x}) \tag{5.14}$$

$$\xi_t' = \begin{cases} \frac{1}{2}\xi_0 + \frac{1}{2}\xi_1 & \text{for } t = 0 \\ \frac{1}{4}\xi_{2t-1} + \frac{1}{2}\xi_{2t} + \frac{1}{4}\xi_{2t+1} & \text{for } t > 0 \end{cases} \tag{5.15}$$

Furthermore, from Equation 5.9, we see

$$\sigma^2(\bar{x}) \geq \frac{\xi_0}{n}, \tag{5.16}$$

and from Equations 5.12 and 5.15, it can be shown that ξ_0/n increases as blocking transforms are applied, unless $\xi_1 = 0$, in which case ξ_0/n is invariant. Further analysis shows that with repeated application of the blocking transforms in Equations 5.11 and 5.12 a fixed point is reached where $\sigma^2(\bar{x}) = \xi_0/n$. Therefore, the variance of a data set can be evaluated by performing blocking operations until ξ_0/n remains constant with further blocking operations.

During a calculation, χ can be used to estimate ξ_0/n.

$$\chi = \frac{\frac{1}{n}\left(\sum_{i=1}^{n} x_i^2\right) - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2}{n-1} \tag{5.17}$$

When enough blocking transforms have been applied to reach the fixed point, the blocked variables are independent Gaussian random variables making χ also a Gaussian random variable with standard deviation $\chi\sqrt{2/(n-1)}$.

The above analysis deals with serially correlated data from Markov processes. Branching processes, such as diffusion or Green's function QMC, also generate data that have parallel correlation. This can be removed by averaging the data from each iteration to make new data elements [47]. These new data elements are still serially correlated and must then be analyzed appropriately to obtain the true variance of the calculation.

5.3 Algorithms

5.3.1 Flyvbjerg-Petersen Algorithm

The Flyvbjerg-Petersen algorithm [43] is conceptually very simple. The average, $\bar{x}$, and χ, for the data, $\{x_i\}$, are calculated using Equations 5.7 and 5.17. A new blocked data set is generated from this data using the block transforms described in Equations 5.11 and 5.12. The average and χ of these data are then evaluated. This process is repeated until no more blocking operations can be performed. The true variance is the value of χ obtained when further blocking operations do not change the value.

Overall, this algorithm requires $O(N)$ operations, where N is the number of unblocked data points, to evaluate the true variance of the calculation. The state of the algorithm is given by an array of all unblocked data elements and is therefore of size $O(N)$. For many calculations, N is very large ($> 10^9$) forcing the state to be saved to disk because it does not fit in the machine's RAM. Because of this, an additional slow $O(N)$ cost is often incurred from reading the data in from disk in order to analyze it.

The Flyvbjerg-Petersen algorithm is an inherently serial algorithm. To use it for a parallel calculation, all of the unblocked data must be sent to one processor where it is concatenated and analyzed as above. Such an operation requires an $O(N)$ communication, where N is the number of unblocked data elements. Furthermore, the entire burden of error analysis is placed on one processor, making the variance calculation expensive for very large samplings. Also, the large amount of data communicated to

one processor can potentially saturate the bandwidth available to this processor.

During a stochastic simulation, it is desirable to evaluate the variance of calculated quantities periodically to determine when the calculation is converged and can be terminated. If the variance is to be evaluated m times during the calculation, the Flyvbjerg-Petersen algorithm requires $O(mN)$ operations, to accomplish this. This can be prohibitively expensive for large N or m.

A summary of the computational costs is listed in Table 5.1.

5.3.2 Dynamic Distributable Decorrelation Algorithm (DDDA)

The equations implemented by DDDA are *exactly* the same as those presented by Flyvbjerg and Petersen. DDDA is a new algorithm to evaluate these equations. The new algorithm involves two classes:

5.3.2.1 Statistic Class

(Pseudocode is listed in Section 5.6)

The Statistic class stores the number of samples, n, running sum of x_i, and running sum of x_i^2 for the data that is entered into it, $\{x_i\}$. This allows straightforward calculation of the average, $\bar{x}$, (Equation 5.7) and χ (Equation 5.17).

5.3.2.2 Decorrelation Class

(Pseudocode is listed in Section 5.7)

The Decorrelation class stores a vector of Statistic objects, where the i^{th} element of the vector corresponds to data that has been partitioned into blocks 2^i long, and

a collection of data samples waiting to be added to the vector. The variance and average for the i^{th} element of the vector can be evaluated by calling the corresponding functions in the appropriate Statistic object.

As data is generated during the calculation, it is added to a Decorrelation object. It is first added to the 0^{th} element of the vector of Statistic objects (vectors numbered from 0). If there is no sample waiting on the 0^{th} level, this sample is stored as the waiting sample for the 0^{th} level; otherwise, this sample and the waiting sample for the 0^{th} level are averaged to create a new sample, and the waiting sample is removed. This new sample is then added to the 1^{st} level in the same fashion as above.

This process repeats until a level is reached with no waiting samples. By adding data this way, new data blocks are generated as soon as there is enough data to create them. Furthermore, because the newly generated data blocks are added to Statistic objects as they are generated, the variance for a particular block size can be immediately evaluated with very few operations ($O(\log_2 N)$). Using these data, it is straightforward to evaluate the true variance as is done with standard blocking methods.

During a parallel calculation, each processor will have a Decorrelation object to which it adds data. The global results are then obtained by combining the Decorrelation objects from each processor into a global Decorrelation object. This can be accomplished using a binary operator to add two Decorrelation objects together. The first step in this process adds the Statistic vectors, from the two Decorrelation objects, element by element to form a new Statistic vector. Then, beginning with the 0^{th}

level, the waiting samples are combined to create either new waiting samples or new averaged samples to be added to the new Statistic vector and combined with waiting samples from the next higher level. Evaluating this binary addition requires only $O(\log_2 N)$ operations, where N is the number of samples.

5.3.3 Analysis of DDDA

The equations implemented by DDDA are *exactly* the same as those presented by Flyvbjerg and Petersen; both require $O(N)$ operations to evaluate the variance of N data samples. In contrast to Flyvbjerg and Petersen, the state (minimal set of data an algorithm must store) of DDDA is only of size $O(\log_2 N)$. The small size of this state ($\log_2 10^9 \approx 30$) means that all necessary data can be stored in RAM, avoiding the read-in expense often encountered with the Flyvbjerg-Petersen algorithm. Also, the small state yields a very small checkpoint from which calculations can be restarted. If an upper bound is known on the block size, then the algorithm can be modified slightly to give a state size of only $O(1)$.

One major advantage of DDDA over the Flyvbjerg-Petersen algorithm, is its ability to efficiently evaluate the true variance of a calculation "on-the-fly." If the variance is to be evaluated m times during the calculation, the Flyvbjerg-Petersen algorithm requires $O(mN)$ operations to accomplish this while DDDA requires only $O(N + m \log_2 N)$. The improved computational complexity makes convergence based termination practical to implement.

DDDA's other major advantage over the Flyvbjerg-Petersen algorithm is its per-

	Flyvbjerg-Petersen Algorithm	Dynamic Distributable Decorrelation Algorithm (DDDA)
Operations	$O(mN)$	$O(N + m \log_2 N)$
State Size	$O(N)$	$O(\log_2 N)$
Parallel Communications	$O(N)$	$O(\log_2 N)$
Parallel Variance Evaluation	$O(N)$	$O(\log_2 N \log_2 P)$

Table 5.1: Comparison of computational costs. N is the number of data points analyzed, m is the number of times the variance is evaluated during a calculation, and P is the number of processors.

formance on parallel calculations. Because the state of DDDA is so compact, only $O(\log_2 N)$ data elements must be communicated between processors. Furthermore, because two Decorrelation objects can be added with $O(\log_2 N)$ operations, a binary tree can be used to evaluate the global variance of the parallel calculation in only $O(\log_2 N \log_2 P)$ operations, where P is the number of processors. The expense of the additions is distributed over a large number of processors. This low complexity evaluation makes possible "on-the-fly" variance determination for massively parallel calculations.

A summary of the computational costs is listed in Table 5.1.

5.4 Computational Experiments

5.4.1 "On-the-fly" Variance Determination for a Single Processor Variational QMC Particle-in-a-Box Calculation

To illustrate DDDA, variational quantum Monte Carlo (VMC) [26] is used to calculate the energy for a one-dimensional particle-in-a-box of length one. For this illustration, the exact ground state wave function, $\Psi_{Exact} = \sqrt{2}\sin(\pi x)$, is approximated by a normalized wave function, Ψ_T.

$$\Psi_T = \sqrt{30}\left(x - x^2\right) \tag{5.18}$$

The expected energy of the system is given by

$$\begin{aligned} \langle E \rangle &= \int_0^1 \Psi_T \hat{H} \Psi_T dx \\ &= \int_0^1 \Psi_T^2 \left(\frac{\hat{H}\Psi_T}{\Psi_T} \right) dx \\ &= \int_0^1 \rho_T(x) E_L(x) dx, \end{aligned} \tag{5.19}$$

where $\hat{H}$ is the Hamiltonian for the system, $E_L(x)$ is the local energy, and $\rho_T(x)$ is the probability distribution of the particle. Since the Ψ_T is not an eigenfunction for this system, the local energy will not be constant and the calculated energy expectation value will fluctuate as the calculation progresses.

Equation 5.19 can be evaluated in two ways:

- One option (Method 1) is to generate points distributed with respect to $\rho_T(x)$ by directly inverting $\rho_T(x)$ and use these points to sample $E_L(x)$. Because $\rho_T(x)$ is directly inverted, this method will produce uncorrelated data.

- A second option (Method 2) is to generate points distributed with respect to $\rho_T(x)$ using the Metropolis algorithm [24] and use these points to sample $E_L(x)$. Because the Metropolis algorithm employs a Markov chain, this method will produce serially correlated data.

Performing 10^6 Monte Carlo steps gives expected energy values of 4.9979(23) for Method 1 and 4.9991(59) for Method 2. Both values agree with the analytic value of 5. Also note that the error estimates of the correlated and uncorrelated calculations are different. These error estimates illustrate that serially correlated data does not provide as much information as uncorrelated data, resulting in a larger standard deviation for the correlated case (Method 2) than the uncorrelated case (Method 1) when using the same number of samples.

Figures 5.1 and 5.2 show the calculated standard deviation vs. block size for uncorrelated (Method 1) and correlated (Method 2) VMC calculations. In both cases, the plateau in the plot corresponds to the true standard deviation value. Fluctuations associated with large block sizes result from dividing the data into a small number of blocks making the data very noisy.

Evaluating the standard deviation in the correlated VMC calculation without data blocking yields an estimate of the standard deviation that is much too small. This

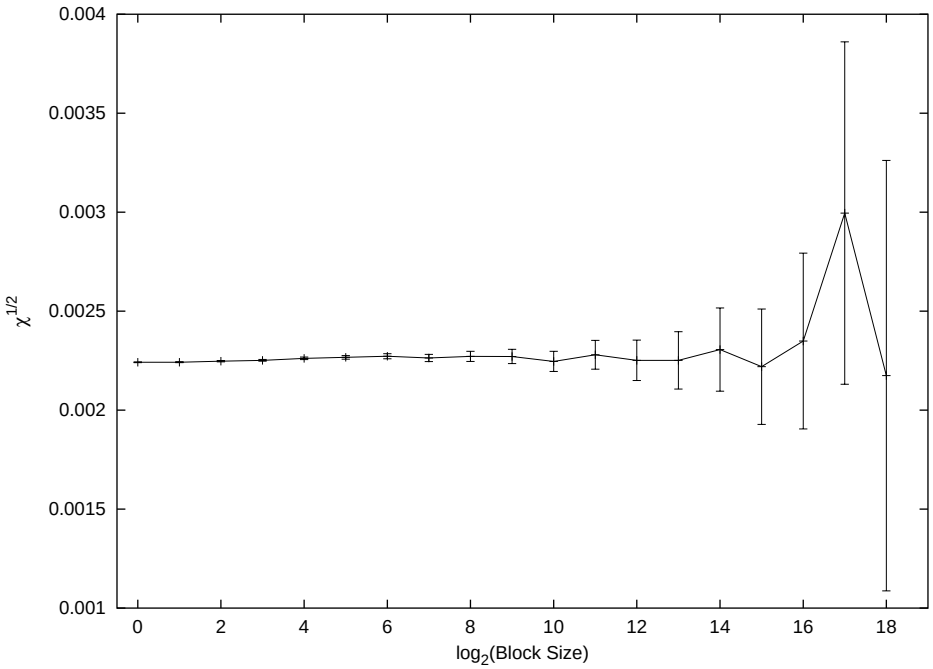


Figure 5.1: $\sqrt{\chi}$ (Equation 5.17) as a function of block size for a variational QMC "particle-in-a-box" calculation using uncorrelated data points (Method 1). The Flyvbjerg-Petersen algorithm and DDDA yield exactly the same results.

corresponds to $log_2(BlockSize) = 0$ in Figure 5.2 and illustrates the potential dangers in reporting error estimates without accounting for the serial correlation that may exist in the data.

The ability of DDDA to evaluate the standard deviation "on-the-fly" for a single processor calculation is demonstrated in Figure 5.3. During the VMC particle in a box calculations, the standard deviation was evaluated every 100 Monte Carlo steps.

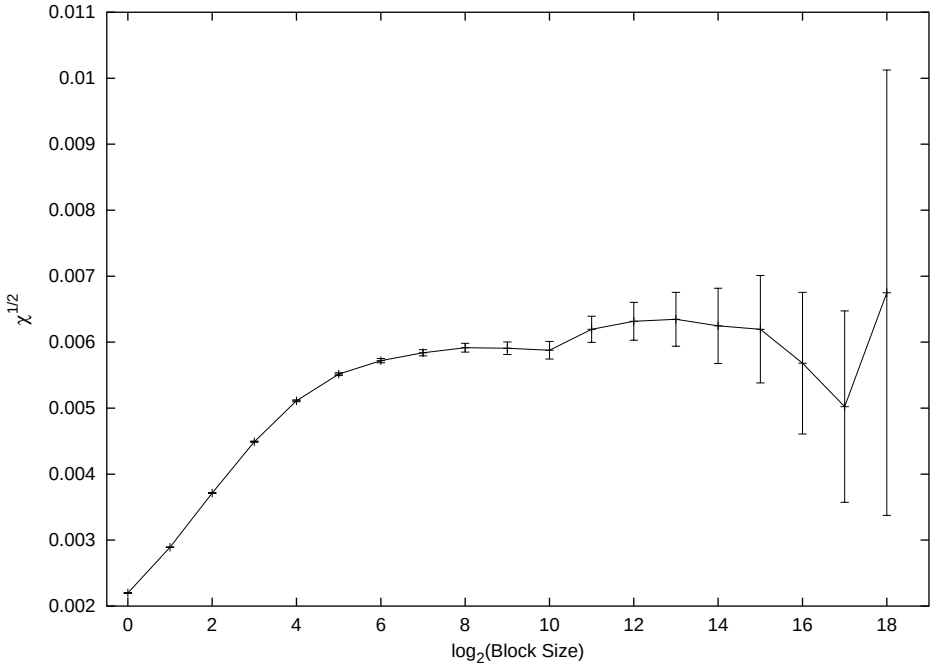


Figure 5.2: $\sqrt{\chi}$ (Equation 5.17) as a function of block size for a variational QMC "particle-in-a-box" calculation using serially correlated data points (Method 2). The Flyvbjerg-Petersen algorithm and DDDA yield exactly the same results.

5.4.2 "On-the-fly" Variance Determination for a Massively Parallel Variational QMC Calculation of RDX

To illustrate the ability of DDDA to evaluate the variance from a large parallel Monte Carlo calculation "on-the-fly," a series of 1024 processor massively parallel variational quantum Monte Carlo (VMC) calculations on the high explosive material RDX (Figure 5.4), cyclic $[CH_2 - N(NO_2)]_3$, were performed. Of the three calculations performed, one was the ground state structure, and the other two were unimolecular decomposition transition states for the concerted dissociation and $N - NO_2$ bond fission reactions. Geometries of the species were obtained from previous DFT calculations on the system [48].

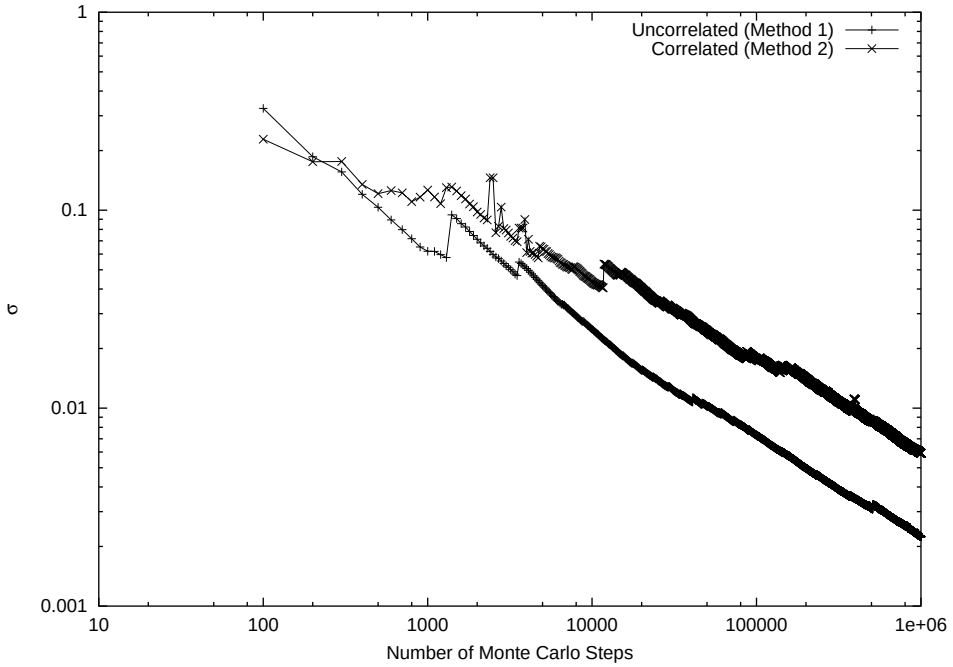


Figure 5.3: Standard deviation as a function of number of Monte Carlo steps for a variational QMC "particle-in-a-box" calculation. The standard deviation was evaluated "on-the-fly" using DDDA.

The VMC wave function, Ψ_{VMC}, used for the calculation is the product of a Hartree-Fock wave function, Ψ_{HF}, and a Padé-Jastrow correlation function, J_{Corr}.

$$\Psi_{VMC} = \Psi_{HF} J_{Corr} \tag{5.20}$$

$$J_{Corr} = \exp\left(\sum_i \sum_{j<i} u_{i,j}\right) \tag{5.21}$$

$$u_{i,j} = \frac{a_{i,j} r_{i,j}}{1 + b_{i,j} r_{i,j}} \tag{5.22}$$

Ψ_{HF} was calculated using Jaguar [49, 50] with a 6-31G** basis set [51]. The Padé-Jastrow parameters (Table 5.2) were chosen to remove singularities in the local energy. Furthermore, they maintain the structure of the Hartree-Fock wave function everywhere except where two particles closely approach one another. Though much work

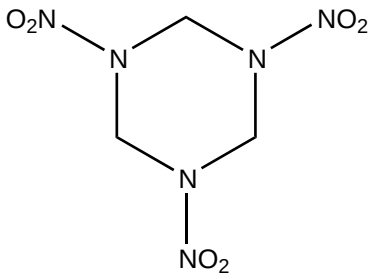


Figure 5.4: The RDX molecule, cyclic $[CH_2\text{-}N(NO_2)]_3$.

	a	b
$u_{\uparrow\downarrow}$	0.5	3.5
$u_{\uparrow\uparrow}$, $u_{\downarrow\downarrow}$	0.25	100
$u_{\uparrow H}$, $u_{\downarrow H}$	-1	100
$u_{\uparrow C}$, $u_{\downarrow C}$	-6	100
$u_{\uparrow N}$, $u_{\downarrow N}$	-7	100
$u_{\uparrow O}$, $u_{\downarrow O}$	-8	100

Table 5.2: Padé-Jastrow correlation function parameters for RDX.

has been done on wave function optimization techniques [32, 26, 52, 53, 30, 28, 54, 55, 56, 57], the Padé-Jastrow parameters are not optimized because this calculation is to demonstrate DDDA and not to obtain a high-accuracy VMC energy, which would require parameter optimization.

Calculations were performed on the ASCI Nirvana supercomputer at the Los Alamos National Laboratory using 1024 MIPS 10000 processors running at 250 MHz. Each calculation took approximately 8 hours to complete and was composed of roughly 3×10^7 Monte Carlo steps. Of the three calculations, two were run to completion while the third calculation was stopped a fraction of the way through the run and restarted from checkpoints to verify the ease and efficiency with which these new data structures allow for checkpointing of the program state variables. The RDX calculations successfully completed independently of whether they were run to

RDX Species	Hartree Fock	Variational Quantum Monte Carlo
Ground state	-892.491	-893.35(4)
Concerted dissociation	-892.369	-893.29(5)
$N - NO_2$ bond fission	-892.259	-893.20(4)

Table 5.3: Total energies (Hartree) for the various calculations on RDX. The HF results were obtained from Jaguar 4.1 with the 6-31G** basis set Variational Quantum Monte Carlo based on 3×10^7 points.

completion or checkpointed and restarted.

Energies for the Hartree Fock and variational quantum Monte Carlo [58] calculations are presented in Table 5.3. The VMC energies are presented for completeness and should not be taken to be highly accurate energies because the variational parameters have not been optimized.

Figures 5.5 and 5.6 show the evolution of the standard deviation of the total energy for three different RDX species as the Monte Carlo calculations progress. In Figure 5.5, notice that the plateau in the plot of standard deviation vs. block size, indicating the true variance, is reached for a block size of roughly 2^8. Results from the RDX transition state structures are similar and require a block size of 2^8 to 2^{13}, depending on the system. Figure 5.6 shows the standard deviations evaluated "on-the-fly" for the massively parallel calculations. These values are found to decrease roughly with the square root of the number of samples, as is expected.

5.5 Conclusions

The above analysis has shown that DDDA is significantly more efficient than standard blocking algorithms at evaluating the variance of a quantity multiple times, "on-the-

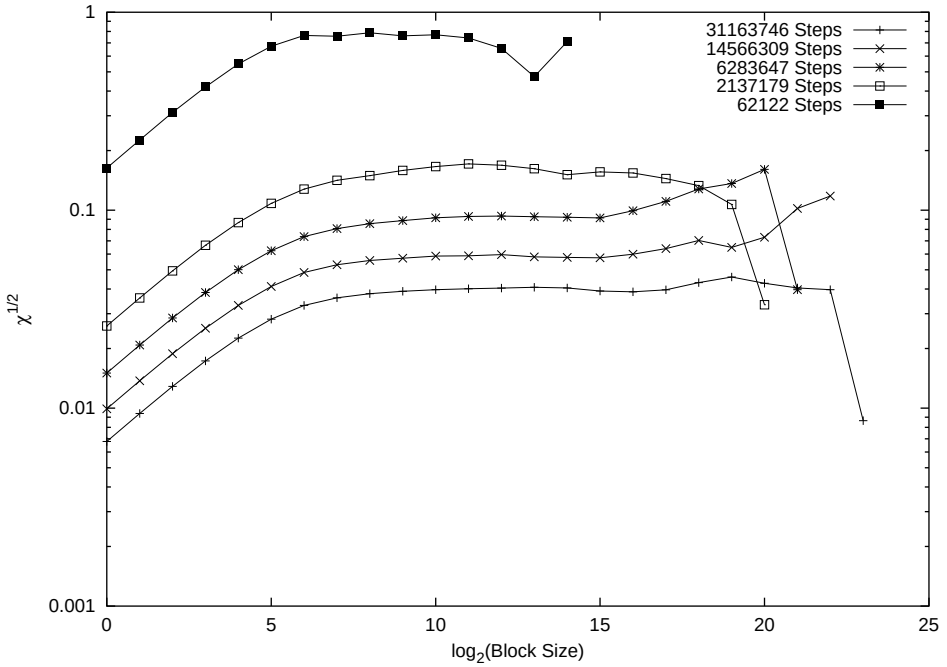


Figure 5.5: Evolution of $\sqrt{\chi}$ as a function of block size as a variational QMC calculation of the RDX ground state progresses. Shown here are the results for five cases with 62122, 2137179, 6283647, 14566309, and 31163746 total Monte Carlo steps.

fly", during a calculation ($O(N + m \log_2 N)$ vs. $O(mN)$). Additionally, the state needed to checkpoint the calculation or evaluate the variance in a parallel calculation is only $O(\log_2 N)$ for DDDA and $O(N)$ for current algorithms. This leads to smaller checkpoints and significantly less communication for parallel calculations. The small state size will facilitate calculations on computational grids where many processors are used but bandwidth is limited.

Because DDDA efficiently evaluates the variance "on-the-fly" for both serial and parallel calculations, it is now possible to use a convergence-based termination scheme. Instead of prespecifying the number of data points a calculation will use, points are generated until the observed quantities are converged to the specified tolerance. This eliminates calculations terminating before they are completed or running too

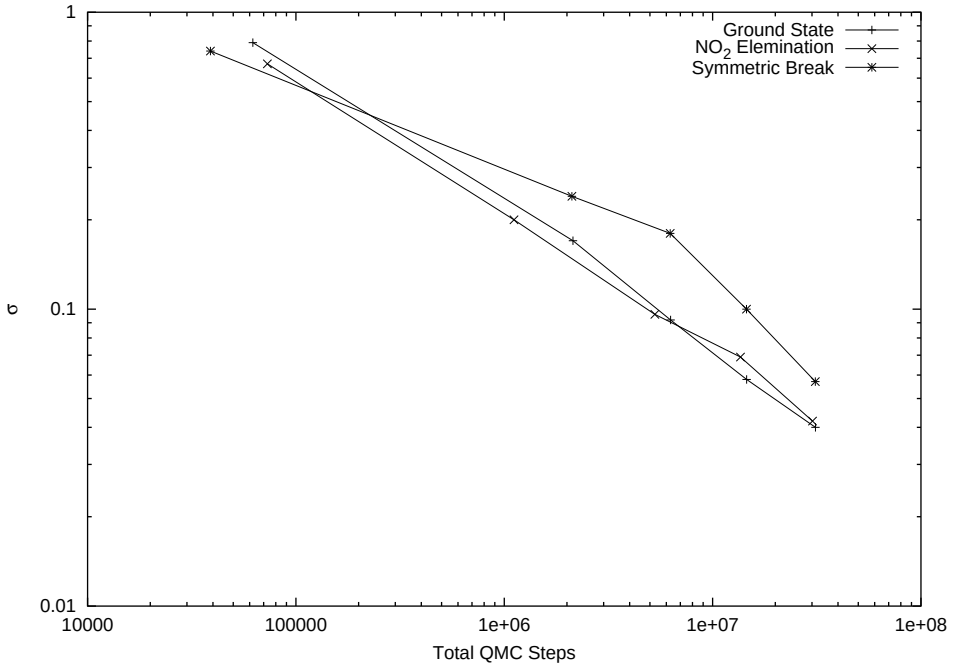


Figure 5.6: Standard deviation as a function of number of Monte Carlo steps for a 1024 processor variational QMC calculations of RDX. The standard deviation was evaluated "on-the-fly" using DDDA.

long and wasting computational resources. Additionally, specifying a desired level of convergence is much more natural than specifying the number of Monte Carlo steps for a non-expert user.

Often when blocking is used for error analysis, the data is preblocked before it is analyzed. This consists of blocking the data before any data analysis takes place. Because the correct block size is not known *a priori*, the Flyvbjerg-Petersen algorithm must then be used to analyze the preblocked data. Preblocking does reduce the amount of data that must be stored, analyzed, and communicated, but it does not change the complexity of the computational costs of the Flyvbjerg-Petersen algorithm (Table 5.1) making it inferior to DDDA. It is possible to preblock and then use DDDA, but this is not necessary. Because the storage and communication costs of DDDA are

$O(\log_2 N)$, reducing N by a constant factor makes only a small change in the state size negating the benefits of preblocking.

5.6 Statistic Class Pseudocode

5.6.1 Pseudocode for `Statistic.initialize()`

Initialize a new instance of the Statistic class

```
Statistic.initialize()

    NSamples = 0.0

    Sum = 0.0

    SumSq = 0.0
```

5.6.2 Pseudocode for `Statistic.addData(new_sample)`

Add a new data element to this Statistic object

```
Statistic.addData(new_sample)

    NSamples = NSamples + 1

    Sum = Sum + new_sample

    SumSq = SumSq + new_sample*new_sample
```

5.6.3 Pseudocode for `Statistic.addition(A,B)`

Add two Statistic objects and return the result

```
Statistic.addition(A,B)

    C=new Statistic()

    C.NSamples = A.NSamples + B.NSamples

    C.Sum = A.Sum + B.Sum

    C.SumSq = A.SumSq + B.SumSq

    return C
```

5.7 Decorrelation Class Pseudocode

5.7.1 Pseudocode for `Decorrelation.initialize()`

Initialize a new instance of the Decorrelation class

```
Decorrelation.initialize():

    Size = 0

    NSamples = 0

    BlockedDataStatistics = [new Statistic()]

    waiting_sample = [0]

    waiting_sample_exists = [false]
```

5.7.2 Pseudocode for `Decorrelation.addData(new_sample)`

```
# Add a new data element to this Decorrelation object
Decorrelation.addData(new_sample):
    NSamples = NSamples + 1

    # Lengthen the vectors, when necessary, to accommodate all entered data
    if NSamples >= 2^Size:
        Size = Size + 1
        BlockedDataStatistics =
            BlockedDataStatistics.append(new Statistic())
        waiting_sample = waiting_sample.append(0)
        waiting_sample_exists = waiting_sample_exists.append(false)

    BlockedDataStatistics[0].add_Data(new_sample)
    carry = new_sample
    i = 1
    done = false

    # Propagate the new sample up through the data structure
    while (not done):
        if waiting_sample_exists[i]:
            new_sample = (waiting_sample[i] + carry)/2
```

```
            carry = new_sample
            BlockedDataStatistics[i].addData(new_sample)
            waiting_sample_exists[i] = false
        else:
            waiting_sample_exists[i] = true
            waiting_sample[i] = carry
            done = true

        i = i+1

        if i > Size:
            done = true
```

5.7.3 Pseudocode for `Decorrelation.addition(A,B)`

Add two Decorrelation objects and return the result

```
Decorrelation.addition(A,B):
    C = new Decorrelation()
    C.NSamples = A.NSamples + B.NSamples

    # Make C big enough to hold all the data from A and B
    while C.NSamples >= 2^C.Size:
```

```
    C.Size = C.Size + 1
    C.BlockedDataStatistics =
        C.BlockedDataStatistics.append(new Statistic())
    C.waiting_sample = C.waiting_sample.append(0)
    C.waiting_sample_exists =
        C.waiting_sample_exists.append(false)

carry_exists = false
carry = 0

for i from 0 to C.Size-1:
    if i <= A.Size:
        StatA = A.BlockedDataStatistics[i]
        waiting_sampleA = A.waiting_sample[i]
        waiting_sample_existsA = A.waiting_sample_exists[i]
    else:
        StatA = new Statistic()
        waiting_sampleA = 0
        waiting_sample_existsA = false

    if i <= B.Size:
        StatB = B.BlockedDataStatistics[i]
```

```
    waiting_sampleB = B.waiting_sample[i]
    waiting_sample_existsB = B.waiting_sample_exists[i]
else:
    StatB = new Statistic()
    waiting_sampleA = 0
    waiting_sample_existsA = false

C.BlockedDataStatistics[i] =
     C.BlockedDataStatistics[i].addition(StatA,StatB)

if (carry_exists & waiting_sample_existsA & waiting_sample_existsB):
    # Three samples to handle
    C.BlockedDataStatistics[i].addData(
        (waiting_sampleA+waiting_sampleB)/2)
    C.waiting_sample[i] = carry
    C.waiting_sample_exists[i] = true
    carry_exists = true
    carry =(waiting_sampleA+waiting_sampleB)/2
else if (not carry_exists & waiting_sample_existsA &
    waiting_sample_existsB):
    # Two samples to handle
    C.BlockedDataStatistics[i].addData(
```

```
        (waiting_sampleA+waiting_sampleB)/2)
    C.waiting_sample[i] = 0
    C.waiting_sample_exists[i] = false
    carry_exists = true
    carry = (waiting_sampleA+waiting_sampleB)/2
else if (carry_exists & not waiting_sample_existsA &
    waiting_sample_existsB):
    # Two samples to handle
    C.BlockedDataStatistics[i].addData(
        (carry+waiting_sampleB)/2)
    C.waiting_sample[i] = 0
    C.waiting_sample_exists[i] = false
    carry_exists = true
    carry = (carry+waiting_sampleB)/2
else if (carry_exists & waiting_sample_existsA &
    not waiting_sample_existsB):
    # Two samples to handle
    C.BlockedDataStatistics[i].addData(
        (carry+waiting_sampleA)/2)
    C.waiting_sample[i] = 0
    C.waiting_sample_exists[i] = false
    carry_exists = true
```

```
            carry = (carry+waiting_sampleA)/2
        else if (carry_exists or waiting_sample_existsA or
            waiting_sample_existsB):
            # One sample to handle
            C.waiting_sample[i] = carry +
                waiting_sampleA + waiting_sampleB
            C.waiting_sample_exists[i] = true
            carry_exists = false
            carry = 0
        else:
            # No samples to handle
            C.waiting_sample[i] = 0
            C.waiting_sample_exists[i] = false
            carry_exists = false
            carry = 0
    return C
```

5.8 Simple Example Calculation Pseudocode

```
for all processors:
    # Initialize error analysis data structure for each processor
```

```
LocalErrorAnalysisDataStructure = new Decorrelation()


while generating new data points:
    # Generate new data and add it to the local error
    #   analysis data structure
    new_data = generateNewDataPoint()
    LocalErrorAnalysisDataStructure.addData(new_data)


if want global results:
    Obtain the global results for the calculation with a binary tree
        parallel reduction operation using Decorrelation.addition(.,.)
        to add LocalErrorAnalysisDataStructure from each processor
```

Chapter 6

Manager–Worker-Based Model for Parallelizing Quantum Monte Carlo on Heterogeneous and Homogeneous Networks

A manager–worker-based parallelization algorithm for Quantum Monte Carlo (QMC-MW) is presented and compared to the commonly used pure iterative parallelization algorithm [59]. The new manager–worker algorithm performs automatic load balancing, allowing it to perform near the theoretical maximum speed even on heterogeneous parallel computers. Furthermore, the new algorithm performs as well as the pure iterative algorithm on homogeneous parallel computers.

When combined with the Dynamic Distributable Decorrelation Algorithm (DDDA) [42], the new manager–worker algorithm permits the termination of QMC calculations upon obtaining a desired level of convergence, rather than when a given number of steps are performed (as is common practice). Additionally, a derivation and experimental verification are given to show that standard QMC implementations are not "perfectly parallel" as is often claimed.

6.1 Introduction

There is currently a great deal of interest in making Quantum Monte Carlo (QMC) methods practical for everyday use by chemists, physicists, and material scientists. Everyday application of QMC is very attractive since methods, such as variational QMC, diffusion QMC, and Green's function QMC, exist which can calculate an atomic or molecular system's energy to within chemical accuracy ($<$ 2 kcal/mol). High-accuracy quantum-mechanical methods generally scale very poorly with problem size, typically $O(N^6$ to $N!)$; however, QMC scales fairly well, $O(N^3)$, but with a large prefactor.

Current research efforts exist to improve QMC's scaling further [31]. Density Functional Theory (DFT) scales well, $O(N^3)$, and could potentially provide highly accurate solutions. Nevertheless, with the current generation of functionals, DFT typically has an accuracy of only 5 kcal/mol or more for typical systems. The results can not be systematically improved.

The primary issue facing the QMC community is that, although QMC scales well with problem size, the method's prefactor is generally very large, often requiring CPU months to calculate moderately sized systems. The Monte Carlo nature of QMC allows it to be easily parallelized, thus reducing the prefactor with respect to the wall clock.

Applying QMC to physically interesting systems almost always requires using supercomputers to enable calculations to complete in a reasonable amount of time. Currently, however, supercomputing resources are very expensive and can be difficult

to gain access to. To make QMC more useful for the average practitioner, algorithms must become more efficient, and/or large inexpensive supercomputers must be produced.

A current trend in large-scale supercomputing [60] is assembling "cheap supercomputers" with commodity components using a Beowulf-type framework. These clusters have proven to be very powerful for high-performance scientific computing applications [61]. Clusters can be constructed as homogeneous supercomputers if the hardware for each node is equivalent or as heterogeneous supercomputers if various generations of hardware are included.

Another interesting development is the use of loosely coupled, distributed grids of computational resources [62] with components that can even reside in different geographic locations across the globe. Such "grids" are upgraded by adding new compute nodes to the existing grid; this results in continuously upgradable supercomputers, which are inevitably heterogeneous. Ultimately, computational grids may provide computational resources on demand, just as electrical grids now provide electricity on demand.

To efficiently utilize the next generation of supercomputer, whether heterogeneous cluster or grid, a parallelization algorithm must first require little communication between processors and second must be able to efficiently use processors that are running at different speeds. We propose a manager–worker-parallelization algorithm for QMC (QMC-MW) designed for just such systems. This algorithm is compared against the pure iterative parallelization algorithm (QMC-PI), which is most commonly used in

QMC implementations [63, 64, 65].

6.2 Theory

Because QMC is a Monte Carlo method and thus stochastic in nature, it is one of the easiest algorithms to parallelize and can be scaled to large numbers of processors. In a parallel calculation, each processor performs an independent QMC calculation, and the resulting statistics from all the processors are combined to produce the global result.

QMC calculations can typically be broken into two major computationally expensive phases: initialization and statistics gathering. Points distributed with respect to a complicated probability distribution, in this case the square of the wave function amplitude, are required during a QMC calculation. In efficient implementations, this is almost always done using the Metropolis algorithm [24].

The first points generated by the Metropolis algorithm are not generated with respect to the desired probability distribution, so they must be discarded. Additionally, points generated for diffusion QMC and Green's function QMC must be discarded if there are significant excited state contributions which have not yet decayed. This represents the initialization phase.

Once the algorithm begins to generate points with respect to the desired distribution, the points are said to be "equilibrated" and can be used to generate valid statistical information for the QMC calculation. This represents the statistics gathering phase and is the phase where useful data is generated.

To obtain statistically independent data, each processor in a parallel calculation must perform its own initialization procedure, which is the same length as the initialization procedure on a single processor. When large numbers of processors are used, the fraction of the time devoted to initializing the calculation can be very large and will eventually limit the number of processors that can be used effectively in parallel (Section 6.2.3).

Sections 6.2.1 and 6.2.2 theoretically analyze the pure iterative (QMC-PI) and manager–worker (QMC-MW) parallelization algorithms for QMC. The analyses assume that an $O(\log_2(N_{processors}))$ method, where $N_{processors}$ is the total number of processors, is used to gather the statistical data from all processors and return it to the root processor [42]. To simplify analysis of the algorithms, the analysis is performed for variational QMC (VMC) with the same number of walkers on each processor; however, it is possible to extend the results to other QMC methods.

6.2.1 Pure Iterative Parallelization Algorithm

The pure iterative parallelization algorithm (QMC-PI) is the most commonly implemented parallelization algorithm for QMC (Algorithm 6.5) [63, 64, 65]. This algorithm has its origins on homogeneous parallel machines and simply allocates an equal fraction of the total work to each processor. The processors execute their required tasks and percolate the resultant statistics to the root node once every processor has finished its work.

In this algorithm, the number of QMC steps taken by each processor during the

statistics gathering phase, $Steps_{PI,i}$, is equal to the total number of QMC steps taken for the calculation, $Steps^{RequiredTotal}$, divided by the total number of processors, $N_{Processors}$.

$$Steps_{PI,i} = \frac{Steps^{RequiredTotal}}{N_{Processors}} \tag{6.1}$$

The number of QMC steps required to initialize each walker during the initialization, $Steps^{Initialize}$, is taken to be a constant. An optimally efficient initialization algorithm would determine how many QMC steps are required to equilibrate each walker, but in current practice, each walker is generally equilibrated for the same number of steps.

The wall clock time required for a QMC calculation using the QMC-PI algorithm, t_{PI}, can be expressed as

$$t_{PI} = t_{PI,i}^{Initialize} + t_{PI,i}^{Propagate} + t_{PI,i}^{Synchronize} + t_{PI}^{Communicate}, \tag{6.2}$$

where $t_{PI,i}^{Initialize}$ is the time required to initialize the calculation on processor i, $t_{PI,i}^{Propagate}$ is the time used in gathering useful statistics on processor i, $t_{PI,i}^{Synchronize}$ is the amount of time processor i has to wait for other processors to complete their tasks, and $t_{PI}^{Communicate}$ is the wall clock time required to communicate all results to the root node. These components can be expressed in terms of quantities that can be measured for

each processor and the network connecting them.

$$t_{PI,i}^{Initialize} = N_w(t_i^{GenerateWalker} + Steps^{Initialize} t_i^{QMC}) \tag{6.3}$$

$$t_{PI,i}^{Propagate} = \left(\frac{Steps^{RequiredTotal}}{N_{Processors}}\right) t_i^{QMC} \tag{6.4}$$

$$t_{PI}^{Communicate} = \log_2(N_{Processors})(t^{Latency} + \beta L) \tag{6.5}$$

Here N_w is the number of walkers per processor, $t_i^{GenerateWalker}$ is the time required to construct a walker on processor i, t_i^{QMC} is the time required for a QMC step on processor i, $t^{Latency}$ is the latency of the network, β is the inverse bandwidth of the network, and L is the amount of data being transmitted between pairs of processors when data is percolated to the root node.

The way this algorithm is constructed, all processors must wait for the slowest processor to complete all of its tasks before the program can terminate. Therefore, $t_{PI,slowest}^{Synchronize} = 0$, and the wall clock time to complete the QMC-PI calculation is

$$t_{PI} = t_{PI,slowest}^{Initialize} + t_{PI,slowest}^{Propagate} + t_{PI}^{Communicate}. \tag{6.6}$$

Furthermore,

$$t_{PI,i}^{Synchronize} = (t_{PI,slowest}^{Initialize} + t_{PI,slowest}^{Propagate}) - (t_{PI,i}^{Initialize} + t_{PI,i}^{Propagate}). \tag{6.7}$$

Similarly, the total CPU time required for a QMC calculation using the QMC-PI algorithm, T_{PI}, can be expressed as

$$T_{PI} \;=\; T_{PI}^{Initialize} + T_{PI}^{Propagate} + T_{PI}^{Synchronize} + T_{PI}^{Communicate}, \tag{6.8}$$

where $T_{PI}^{Initialize}$ is the total time required to initialize the calculation, $T_{PI}^{Propagate}$ is the total time used in gathering useful statistics, $T_{PI}^{Synchronize}$ is the total time used in synchronizing the processors, and $T_{PI}^{Communicate}$ is the total time used to communicate all results to the root node. These components can be expressed in terms of quantities that can be measured for each processor and the network connecting them.

$$T_{PI}^{Initialize} = \sum_{i}^{N_{Processors}} t_{PI,i}^{Initialize} \tag{6.9}$$

$$T_{PI}^{Propagate} = \sum_{i}^{N_{Processors}} t_{PI,i}^{Propagate} \tag{6.10}$$

$$T_{PI}^{Synchronize} = \sum_{i}^{N_{Processors}} t_{PI,i}^{Synchronize} \tag{6.11}$$

$$T_{PI}^{Communicate} = (N_{Processors} - 1)(t^{Latency} + \beta L) \tag{6.12}$$

6.2.2 Manager–Worker-Parallelization Algorithm

The manager–worker algorithm (QMC-MW) offers an entirely new method for performing parallel QMC calculations (Algorithm 6.6). This algorithm makes the root node a "manager" and all of the other nodes "workers." The worker nodes compute Monte Carlo steps until they receive a command from the manager node. The command either tells the worker to 1) percolate its results to the manager node and continue working or 2) percolate its results to the manager node and terminate. The manager periodically collects the statistics that have been calculated. If the statistics are sufficiently converged, the manager commands the workers to send all their data and terminate; otherwise, the manager will do some of its own work and repeat the process again later.

Unlike QMC-PI, QMC-MW dynamically determines how much work each processor performs. This allows faster processors to do more work, so the calculation is automatically load balanced.

The wall clock time required to perform a QMC-MW calculation can be broken into the same terms as were used for a QMC-PI calculation (Equation 6.3).

$$t_{MW} = t_{MW,i}^{Initialize} + t_{MW,i}^{Propagate} + t_{MW,i}^{Synchronize} + t_{MW,i}^{Communicate} \tag{6.13}$$

Because MW dynamically determines how many steps each processor performs,

each of the constituent terms has a more complicated form than in QMC-PI. Allowing $\hat{\tau}$ to be the minimum wall clock needed to achieve convergence on a given network and τ to be the approximate wall clock time minus communication time during the run, one can easily derive the following expressions. Once τ plus communication time exceeds $\hat{\tau}$, the QMC-MW algorithm will terminate.

$$t_{MW,i}^{Initialize} = N_w t_i^{GenerateWalker} + Steps_{MW,i}^{Initialize}(\hat{\tau}) t_i^{QMC} \tag{6.14}$$

$$t_{MW,i}^{Propagate} = Steps_{MW,i}^{Propagate}(\hat{\tau}) t_i^{QMC} \tag{6.15}$$

$$\begin{aligned} t_{MW,i}^{Communicate} = & \left\lceil \frac{Steps_{MW,0}^{Total}(\hat{\tau})}{N_w Steps^{Reduce}} \right\rceil \log_2(N_{Processors})(t^{Latency} + \beta L) + \\ & \left\lceil \frac{Steps_{MW,i}^{Total}(\hat{\tau})}{N_w Steps^{Poll}} \right\rceil t_i^{Poll} \end{aligned} \tag{6.16}$$

$$t_{MW,i}^{Synchronize} \leq N_w Steps^{Poll} t_{slowest}^{Poll} \left\lceil \frac{Steps_{MW,0}^{Total}(\hat{\tau})}{N_w Steps^{Reduce}} \right\rceil, \tag{6.17}$$

where

$$\begin{aligned} \tau &= t_{MW} - \left\lceil \frac{Steps_{MW,0}^{Total}(\tau)}{N_w Steps^{Reduce}} \right\rceil \log_2(N_{Processors})(t^{Latency} + \beta L) \\ &\approx t_{MW} - t_{MW,i}^{Synchronize} - t_{MW,i}^{Communicate} \\ &= t_{MW,i}^{Initialize} + t_{MW,i}^{Propagate} \end{aligned} \tag{6.18}$$

$$Steps_{MW,i}^{Total}(\tau) = \left\lceil \frac{\tau}{N_w t_i^{QMC}} \right\rceil, \tag{6.19}$$

$$Steps_{MW,i}^{Initialize}(\tau) = \min(Steps_{MW,i}^{Total}(\tau), N_w Steps_{Initialize}), \tag{6.20}$$

$$Steps_{MW,i}^{Propagate}(\tau) = Steps_{MW,i}^{Total}(\tau) - Steps_{MW,i}^{Initialize}(\tau), \tag{6.21}$$

and

$$\hat{\tau} = \min \tau \ni \left\{ \begin{array}{c} \sum_i^{N_{Processors}} Steps_{MW,i}^{Propagate}(\tau) \geq Steps^{RequiredTotal} \\ \tau/(Steps^{Reduce} t_0^{QMC}) \in \mathbf{Z}^+ \end{array} \right. . \tag{6.22}$$

$Steps^{RequiredTotal}$ is the minimum number of steps required to obtain the desired level of convergence, $Steps^{Poll}$ is the number of QMC steps that take place on a worker processor between checking for a message from the manager, and $Steps^{Reduce}$ is the number of QMC steps that take place on the manager processor between sending commands to the workers. Unlike t_{PI}, t_{MW} can not be simply expressed in terms of individual processor speeds.

The total time required for the MW algorithm, T_{MW}, can be expressed as

$$T_{MW} = T_{MW}^{Initialize} + T_{MW}^{Propagate} + T_{MW}^{Synchronize} + T_{MW}^{Communicate}, \tag{6.23}$$

which contains the same components as Equation 6.9.

$$T_{MW}^{Initialize} = \sum_{i}^{N_{Processors}} t_{MW,i}^{Initialize} \tag{6.24}$$

$$T_{MW}^{Propagate} = \sum_{i}^{N_{Processors}} t_{MW,i}^{Propagate} \tag{6.25}$$

$$T_{MW}^{Synchronize} = \sum_{i}^{N_{Processors}} t_{MW,i}^{Synchronize} \tag{6.26}$$

$$\begin{aligned} T_{MW}^{Communicate} = & \left\lceil \frac{Steps_{MW,0}^{Total}}{N_w Steps^{Reduce}} \right\rceil (N_{Processors} - 1)(t^{Latency} + \beta L) + \\ & \sum_{i}^{N_{Processors}} \left\lceil \frac{Steps_{MW,i}^{Total}(\hat{\tau})}{N_w Steps^{Poll}} \right\rceil t_i^{Poll} \end{aligned} \tag{6.27}$$

6.2.3 Initialization Catastrophe

QMC algorithms are described as being "embarrassingly parallel" and linearly scaling with respect to the number of processors used [66]. While these statements are true for a large fraction of Monte Carlo calculations, they are *not* true for QMC calculations which employ the Metropolis algorithm [24].

To obtain independent statistical data from each processor, at least one independent Markov chain must be initialized on each processor (Section 6.2). This gives an initialization cost, $T^{Initialize}$, which scales as $O(N_{Processors})$. The time devoted to generating useful statistical data during the calculation, $T^{Propagate}$, scales as $O(1)$

because a given number of independent Monte Carlo samples are required to obtain a desired statistical accuracy no matter how many processors are used. From this, the efficiency, or fraction of the total calculation time devoted to useful work, ϵ is

$$\epsilon = \frac{T^{Propagate}}{T^{Initialize} + T^{Propagate} + T^{Synchronize} + T^{Communicate}} \tag{6.28}$$

$$\approx \frac{O(1)}{O(N_{Processors}) + O(1)}. \tag{6.29}$$

This clearly demonstrates that QMC calculations using the Metropolis algorithm are *not* linearly scaling for large numbers of processors as is often claimed. This results from the initialization of the Metropolis algorithm and not the parallelization algorithm used.

For QMC calculations to efficiently use $> 10^4$ processors, new algorithms to efficiently generate equilibrated, statistically independent walkers are required. The effort to generate such walkers for the global calculation scales linearly with the number of processors, because $N_{Processors}N_w$ walkers are required. Thus, the initialization catastrophe can not be eliminated, but it can be minimized.

6.3 Experiments

Computational experiments comparing QMC-PI and QMC-MW parallelization algorithms were performed using QMcBeaver [58, 42], a finite all-electron QMC software package developed in conjunction with Michael Feldmann. Variational QMC was chosen as the particular QMC flavor to allow direct comparison with the theoretical

results in Section 6.2.

QMcBeaver percolates statistical results from all nodes to the root node using the Dynamic Distributable Decorrelation Algorithm (DDDA) [42] and the MPI_Reduce command from MPI [67]. This combination provides an $O(\log_2(N_{Processors}))$ method for gathering the statistical data from all processors, decorrelating the statistical data, and returning it to the root node.

The time spent initializing, propagating, synchronizing, and communicating during a calculation was obtained from timers inserted into the relevant sections of QMcBeaver. During a parallel calculation, each node has its own set of timers which provide information on how that particular processor is performing. At the completion of a calculation, the results from all processors are combined to yield the total CPU time devoted to each class of task.

6.3.1 Experiment: Varying Levels of Heterogeneity

For this experiment, a combination of Intel Pentium Pro 200 MHz and Intel Pentium III 866 MHz computers connected with a 100 Mb/sec network was used. The total number of processors was kept constant at 8, but the number of each type of processor was varied over the whole range. This setup provided a series of 8-processor parallel computers with a spectrum of heterogeneous configurations. For calculations with the current version of QMcBeaver, the Pentium III is roughly 4.4 times faster than the Pentium Pro at performing QMcBeaver on these test systems.

Variational QMC computational experiments were performed on a Ne atom using

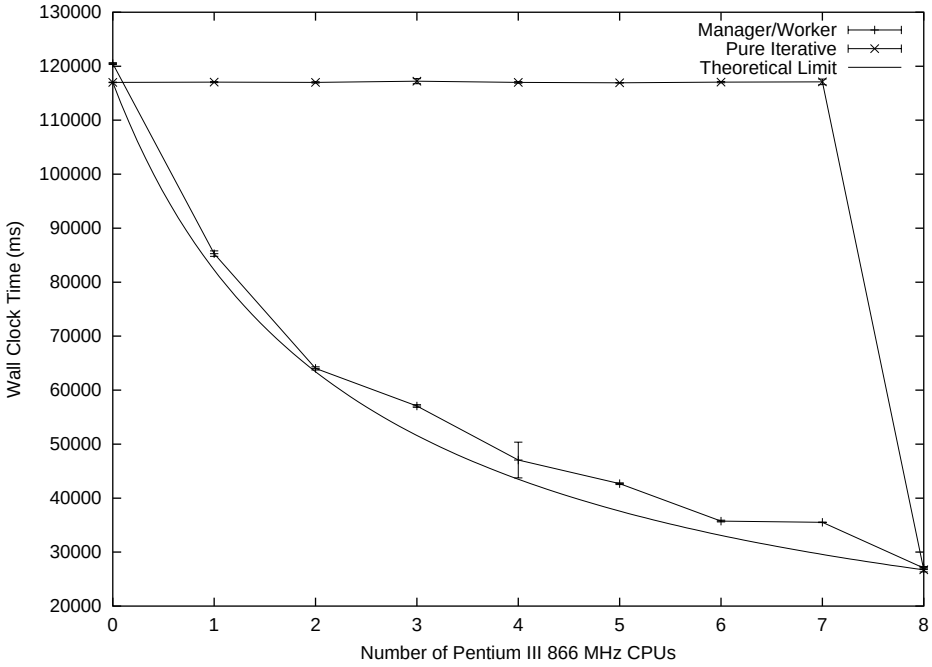


Figure 6.1: Time required to complete an 8-processor variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. The theoretical optimal performance for a given configuration of processors is provided by the curve.

a Hartree-Fock/TZV [68] trial wave function calculated using GAMESS [69, 70]. For the parallelization algorithms, the following values were used: $Steps^{RequiredTotal} = 2.5 \times 10^6$, $Steps^{Initialize} = 1 \times 10^3$, $Steps^{Poll} = 1$, $Steps^{Reduce} = 1 \times 10^3$, and $N_w = 2$.

The time required to complete the QMC calculation for the QMC-PI and QMC-MW parallelization algorithms is shown in Figure 6.1. Each data point was calculated five times and averaged to provide statistically relevant data.

The time required for the QMC-PI algorithm to complete is determined by the slowest processor. When between 1 and 8 Pentium Pro processors are used, the calculation takes the same time as when 8 Pentium Pro processors are used; yet, when 8 Pentium III processors are used (homogeneous network), the calculation completes

much faster. This matches the behavior predicted by Equation 6.6. This figure also shows that MW performs near the theoretical speed limit for each of the heterogeneous configurations. This is a result of the dynamic load balancing inherent in QMC-MW.

The total number of QMC steps performed during a calculation is shown in Figure 6.2. The QMC-PI method executes the same number of steps regardless of the particular network because the number of steps performed by each processor is determined *a priori*. On the other hand, QMC-MW executes a different number of steps for each network configuration. This results from the dynamic determination of the number of steps performed by each processor. The total number of steps is always greater than or equal to the number of steps needed to obtain a desired precision, $Steps^{RequiredTotal}$.

Figures 6.3 and 6.4 break the total calculation time down into its constituent components (Equations 6.8 and 6.23). QMC-MW spends essentially all of its time initializing walkers or generating useful QMC data. Synchronization and communication costs are minimal. On the other hand, QMC-PI devotes a huge portion of the total calculation time to synchronizing processors on heterogeneous networks. This is very inefficient and wasteful.

One should note that the value of $Steps^{RequiredTotal}$ required to obtain a desired precision in the calculated quantities is unknown before a calculation begins. QMC-PI requires this value to be estimated *a priori*. If the guess is too large, the calculation is converged beyond what is required, and if too small, the calculation must be restarted from a checkpoint. Both situations are inefficient. Because QMC-MW does not

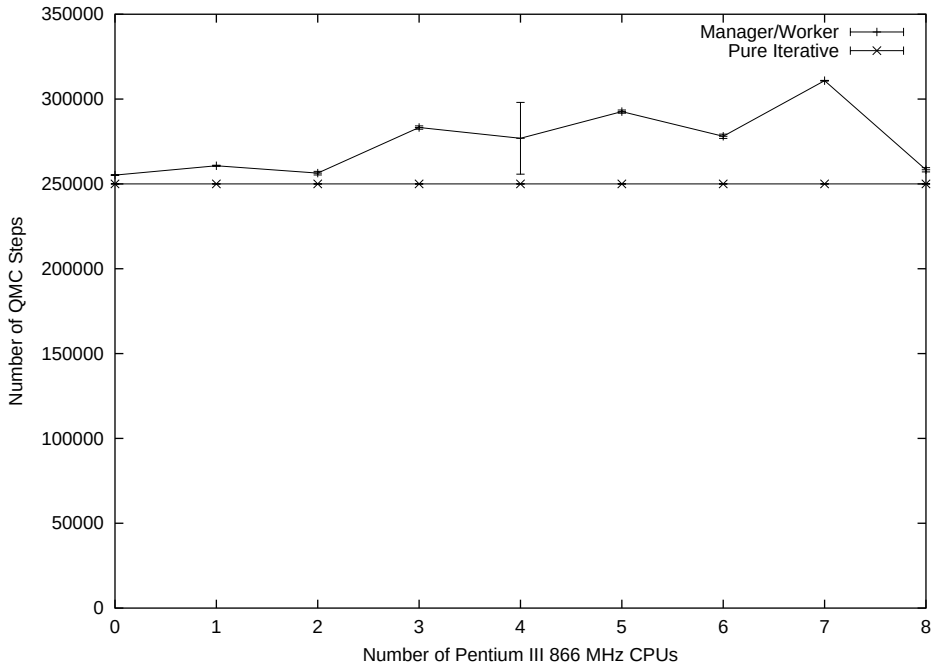


Figure 6.2: Number of variational QMC steps completed during an 8-processor calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) parallelization algorithms. The pure iterative algorithm always calculates the same number of steps, but the manager–worker algorithm dynamically determines how many steps to take. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

determine $Steps^{RequiredTotal}$ *a priori*, an optimal value can be determined "on-the-fly" by examining the convergence of the calculation. This provides the outstanding performance of QMC-MW on any architecture.

6.3.2 Experiment: Heterogeneous Network Size

Variational QMC computational experiments were performed on a Ne atom using a Hartree-Fock/TZV [68] trial wave function calculated using GAMESS [69, 70]. The network of machines used was a heterogeneous cluster of Linux boxes. The 5 processor data point was generated using an Intel Pentium Pro 200 MHz, Intel Pentium II 450

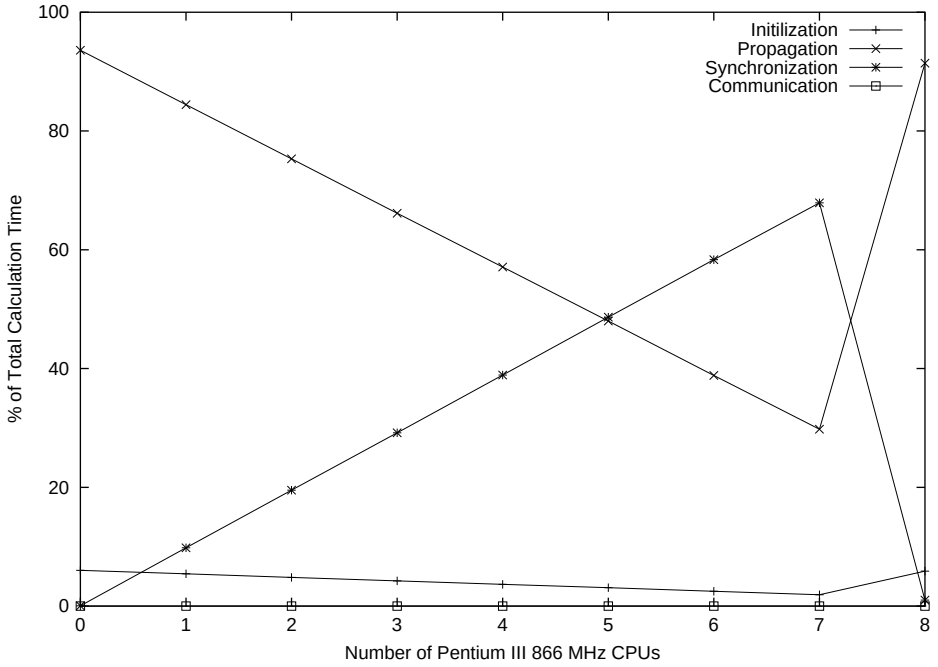


Figure 6.3: Percentage of total calculation time devoted to each component in the pure iterative parallelization algorithm (QMC-PI) during an 8-processor variational QMC calculation of Ne. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

MHz, Intel Pentium III Xeon 550 MHz, Intel Pentium III 600 MHz, and Intel Pentium III 866 MHz. The 10 and 20 processor data points represent 2 and 4 times as many processors, respectively, with the same distribution of processor types as the 5 processor data point. All computers are connected by 100 Mb/sec networking. For the parallelization algorithms, the following values were used: $Steps^{RequiredTotal} = 2.5 \times 10^6$, $Steps^{Initialize} = 1 \times 10^3$, $Steps^{Poll} = 1$, $Steps^{Reduce} = 1 \times 10^3$, and $N_w = 2$.

The time required to complete the QMC calculation for the QMC-PI and QMC-MW parallelization algorithms is shown in Figure 6.5. Each data point was calculated five times and averaged to provide statistically relevant data.

The results illustrate that QMC-MW performs near the theoretical performance

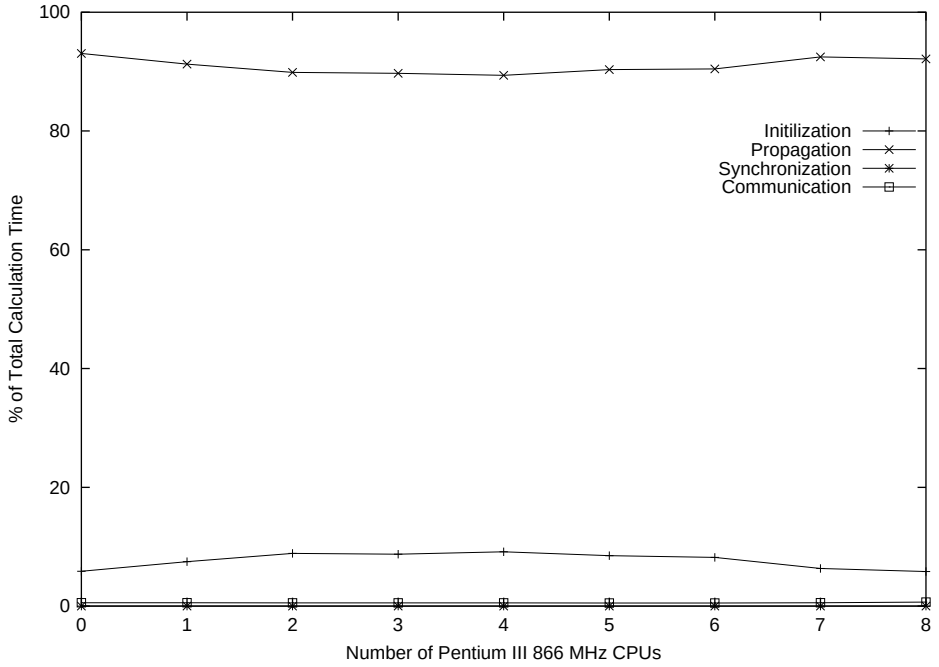


Figure 6.4: Percentage of total calculation time devoted to each component in the manager–worker-parallelization algorithm (QMC-MW) during an 8-processor variational QMC calculation of Ne. The 8 processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

limit as the size of a heterogeneous calculation increases. Because all three data points were calculated with an Intel Pentium Pro 200 MHz as the slowest processor, the QMC-PI calculations perform like 5, 10, and 20 processor Pentium Pro 200 MHz calculations. The scaling is essentially linear, but there is a huge inefficiency illustrated by the separation between the theoretical performance limit and the QMC-PI results.

6.3.3 Experiment: Large Heterogeneous Network

Variational QMC computational experiments were performed on NH_2CH_2OH using a B3LYP(DFT)/cc-pVTZ [71] trial wave function calculated using Jaguar 4.0 [72].

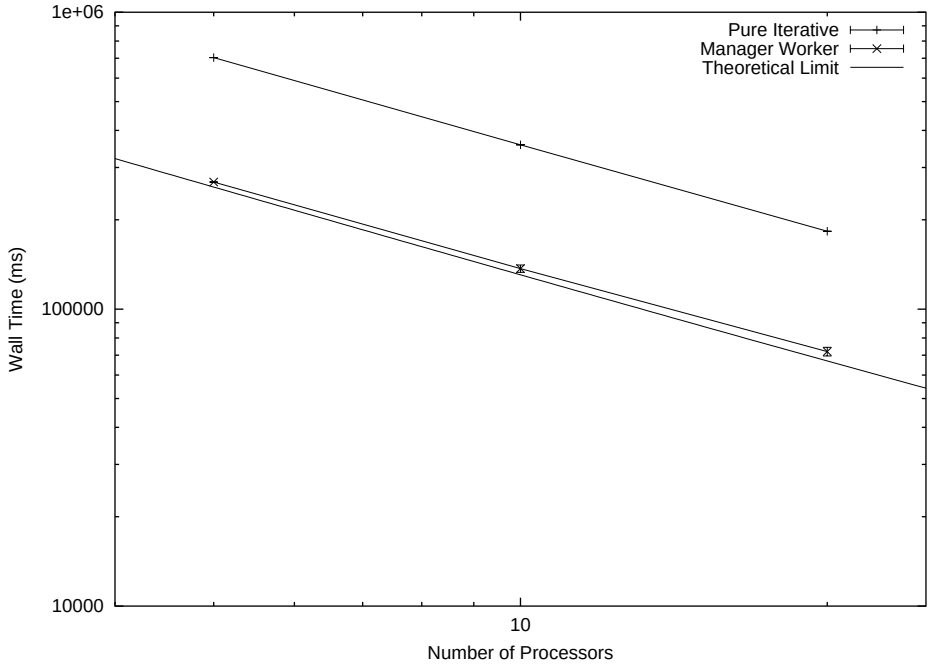


Figure 6.5: Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on a heterogeneous Linux cluster. The theoretical optimal performance for a given configuration of processors is provided by the line. The specific processor configuration is discussed in Section 6.3.2.

The calculations were run on the Parallel Distributed Systems Facility (PDSF) at the National Energy Research Scientific Computing Center (NERSC). This machine is a heterogeneous cluster of Linux boxes with Intel processors ranging from Pentium II 400 MHz to Pentium III 1 GHz.

Two calculations were performed on the cluster. The first used 128 processors with an average processor clock speed of 812 MHz, and the second used the whole cluster, 355 processors, with an average processor clock speed of 729 MHz. Both calculations were done using only the QMC-MW algorithm. For the parallelization algorithm, the following values were used: $Steps^{RequiredTotal} = 1 \times 10^7$, $Steps^{Initialize} = 2 \times 10^3$, $Steps^{Poll} = 1$, $Steps^{Reduce} = 1 \times 10^4$, and $N_w = 1$.

The 128 and 355 processor calculations completed in 6426656 ms and 2644823 ms, respectively. These results can be crudely compared by assuming that all processors used in the calculation perform the same amount of work per clock cycle. Using this assumption, the calculation is 98% efficient in scaling up from 128 to 355 processors. This demonstrates the ability of QMC-MW to efficiently deal with very large heterogeneous computers.

6.3.4 Experiment: Homogeneous Network

The QMC-PI algorithm was originally designed to work on homogeneous supercomputers with fast communication while the QMC-MW algorithm was designed to work on heterogeneous supercomputers with slow communication. To test the QMC-MW algorithm on the QMC-PI algorithm's native architecture, a QMC scaling calculation (Figure 6.6) was performed on the ASCI-Blue Pacific supercomputer at Lawrence Livermore National Laboratory. This machine is a homogeneous supercomputer composed of 332 MHz PowerPC 604e processors connected by HIPPI networking.

Variational QMC computational experiments were performed on a Ne atom using a Hartree-Fock/TZV [68] trial wave function calculated using GAMESS [69, 70]. For the parallelization algorithms, the following values were used: $Steps^{RequiredTotal} = 1 \times 10^6$, $Steps^{Initialize} = 2 \times 10^3$, $Steps^{Poll} = 1$, $Steps^{Reduce} = 1 \times 10^3$, and $N_w = 2$.

Figure 6.6 shows that the QMC-MW and QMC-PI algorithms perform nearly identically on Blue Pacific. The QMC-MW calculation is consistently slightly slower than the QMC-PI algorithm because the QMC-MW calculation performed more QMC

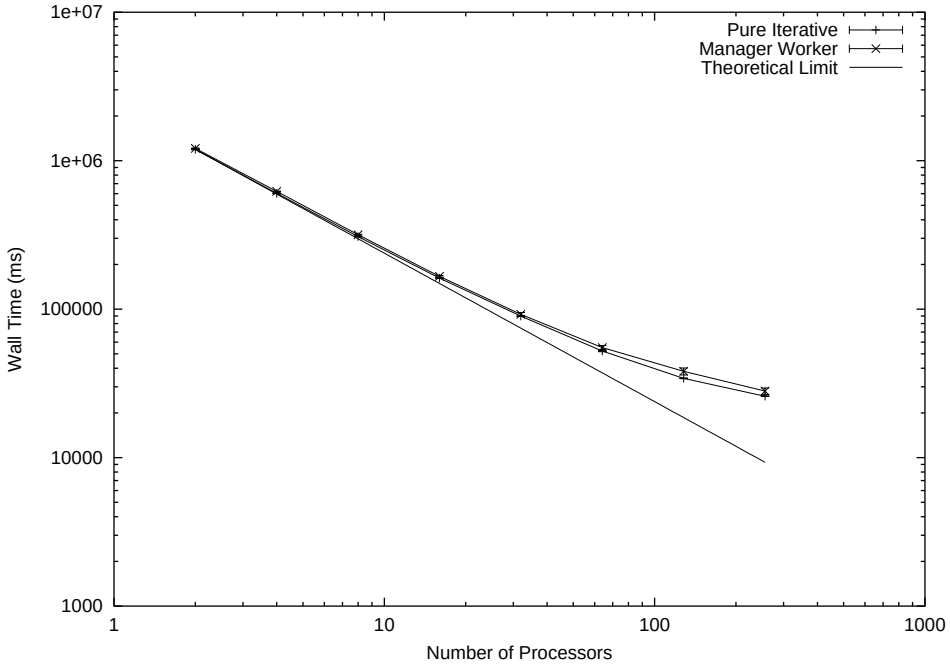


Figure 6.6: Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on the ASCI Blue Pacific homogeneous supercomputer. The theoretical optimal performance for a given configuration of processors is provided by the line.

steps. This results because the QMC-PI calculation performs a predetermined number of steps while the QMC-MW calculation performs *at least* this same predetermined number of steps for this experiment. Again, this assumes the user knows *a priori* exactly how many steps to complete for QMC-PI, in order to obtain the desired convergence, while QMC-MW requires no *a priori* knowledge of $Steps^{RequiredTotal}$. This experiment is an absolute best-case situation for the QMC-PI algorithm. This discrepancy can be reduced by decreasing $Steps^{Reduce}$.

Figure 6.7 plots the ratio of the total computational resources used by each algorithm. This shows that both algorithms perform within 2% of each other; therefore, they can be considered to take roughly the same time and expense on homogeneous

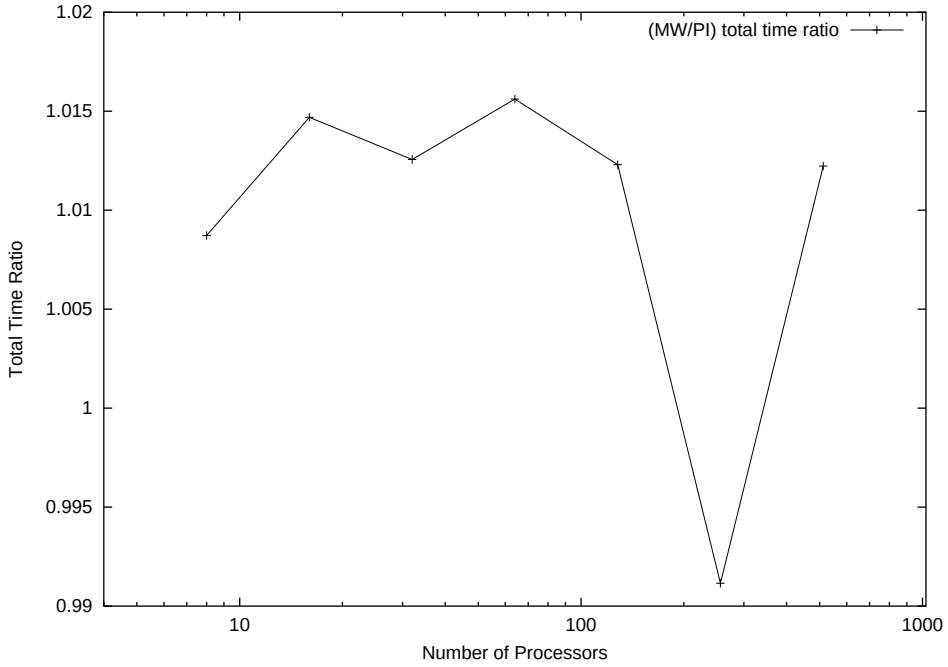


Figure 6.7: Ratio of wall time for QMC-MW/QMC-PI on ASCI Blue Pacific.

machines.

Both algorithms do not perform near the linear scaling limit for large numbers of processors. This is a result of the initialization catastrophe discussed in Sections 6.2.3 and 6.3.5.

6.3.5 Experiment: Initialization Catastrophe

To demonstrate the "initialization catastrophe" described in Section 6.2.3, a scaling experiment was performed on the ASCI-Blue Mountain supercomputer at Los Alamos National Laboratory (Figure 6.8). This machine is a homogeneous supercomputer composed of MIPS 10000 processors running at 250 MHz connected by HIPPI networking. Variational QMC calculations of RDX, cyclic-$[CH_2NNO_2]_3$, using the QMC-MW algorithm with $Steps^{RequiredTotal} = 1 \times 10^5$, $Steps^{Initialize} = 1 \times 10^3$,

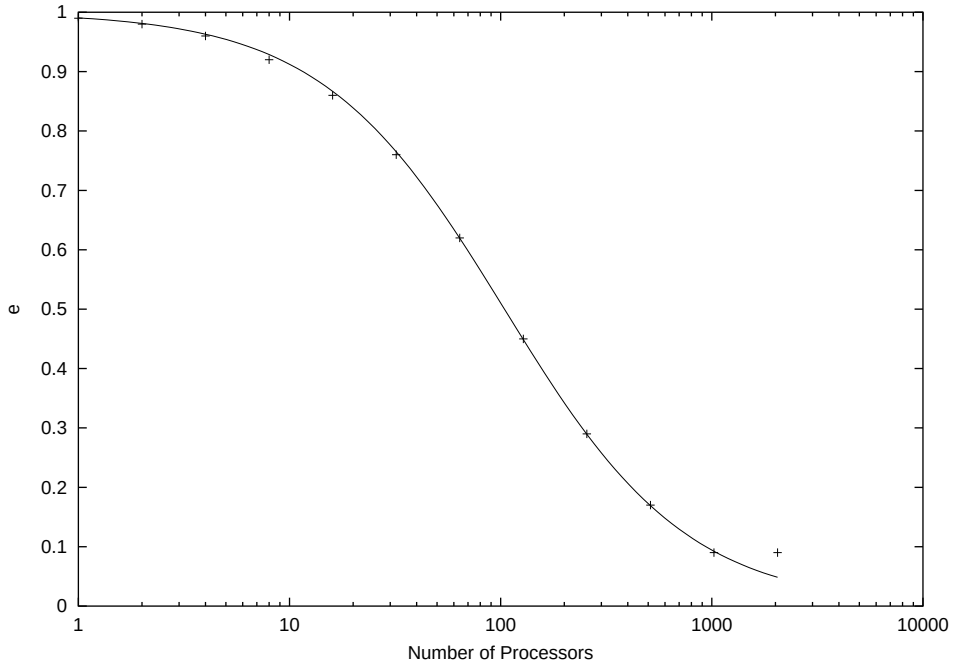


Figure 6.8: Efficiency of a variational QMC calculation of RDX as a function of the number of processors used. The calculations were performed using the manager–worker-parallelization algorithm (QMC-MW) on the ASCI-Blue Mountain supercomputer, which has 250 MHz MIPS 10000 processors connected by HIPPI networking. A similar result is produced by the Pure Iterative parallelization algorithm. The data is fit to $\epsilon(N_{Processors}) = a/(a + N_{Processors})$ with $a = 104.203$.

$Steps^{Poll} = 1$, $Steps^{Reduce} = 1 \times 10^2$, and $N_w = 1$ were performed. Jaguar 4.0 [72] was used to generate a HF/6-31G** trial wave function.

The efficiency of the scaling experiments was calculated using Equation 6.28, and the results were fit to

$$\epsilon = \frac{a}{a + N_{Processors}} \tag{6.30}$$

with $a = 104.203$. The efficiency at 2048 processors is better than the value predicted from the fit equation. This is an artifact of the QMC-MW algorithm which resulted from this calculation taking significantly more steps than $Steps^{RequiredTotal}$. Decreasing the value of $Steps^{Reduce}$ would reduce this problem.

The excellent fit of the data to Equation 6.30 clearly shows that QMC calculations using the Metropolis algorithm are *not* linearly scaling for large numbers of processors. This result holds true for both QMC-MW and QMC-PI because it results from the initialization of the Metropolis algorithm and not the parallelization of the statistics gathering propagation phase. Furthermore, longer statistics gathering calculations have better efficiencies and thus better scaling than short statistics gathering calculations. This can be seen by examining Equation 6.28.

6.4 Conclusion

The new QMC manager–worker-parallelization algorithm clearly outperforms the commonly used Pure Iterative parallelization algorithm on heterogeneous parallel computers and performs near the theoretical speed limit. Furthermore, both algorithms perform essentially equally well on a homogeneous supercomputer with high-speed networking.

When combined with DDDA, QMC-MW is able to determine, "on-the-fly," how well a calculation is converging, allowing convergence-based termination. This is opposed to the standard practice of having QMC calculations run for a predefined number of steps. If the predefined number of steps is too great, computer time is wasted, and if too short, the job will not have the required convergence and must be resubmitted to the queue, lengthening the total time for the calculation to complete. Additionally, specifying a calculation precision (2 kcal/mol for example) is more natural for the application user than specifying a number of QMC steps.

QMC-MW allows *very* low cost QMC-specific parallel computers to be built. These machines can use commodity processors, commodity networking, and no hard disks. Because the algorithm efficiently handles loosely coupled heterogeneous machines, such a computer is continuously upgradable and can have new nodes added as resources become available. This greatly reduces the cost of the resources the average practitioner needs access to, bringing QMC closer to becoming a mainstream method.

It is possible to use QMC-PI on a heterogeneous computer with good efficiency if the QMC performance on each processor is known. Determining and effectively using this information can be a great deal of work. If the user has little or inaccurate information about the computer, this approach will fail. QMC-MW overcomes these shortfalls with no work or input on the user's part. Also, when new nodes are added to the computer, QMC-MW can immediately take advantage of them, while the modified QMC-PI must have benchmark information recorded before they can be efficiently used. The benefits and displayed ease of implementation of QMC-MW clearly outweigh those of QMC-PI supporting its adoption as the method of choice for making QMC parallel.

For calculations with $> 10^4$ processors, a modification to the presented QMC-MW algorithm could yield a large performance increase. This modification involves two threads of execution per processor. A lightweight "listener" thread would manage all of the communication between the manager and worker nodes, while a heavyweight thread would perform the actual QMC calculation.

The prediction and verification of the initialization catastrophe clearly highlights

the need for efficient initialization schemes if QMC is to be scaled to tens of thousands or more processors. Producing such algorithms must be a focus of future work.

6.5 Pure Iterative Algorithm (QMC-PI)

for $Processor_i$; $i = 0$ to $N_{Processors} - 1$

$Steps_{PI,i} = Steps^{RequiredTotal}/N_{Processors}$

Generate N_w walkers

for $Steps^{Initialize}$ steps

Equilibrate walkers

for $Steps_{PI,i}$ steps

Generate QMC statistics

Percolate statistics to $Processor_0$

6.6 Manager–Worker Algorithm (QMC-MW)

for $Processor_i$; $i = 0$ to $N_{Processors} - 1$

$done = false$

$counter = 0$

Generate N_w walkers

while not *done*:

if $counter < Steps^{Initialize}$:

Equilibrate all local walkers 1 step

else:

Propagate all local walkers 1 step and collect QMC statistics

if $i = 0$:

if statistics are converged:

$done = true$

Tell workers to percolate statistics to $Processor_0$ and

set $done = true$

else if $counter$ mod $Steps^{Reduce} = 0$:

Tell workers to percolate statistics to $Processor_0$

else:

if $counter$ mod $Steps^{Poll} = 0$:

Check for commands from the manager and

execute the commands.

$counter = counter + 1$

Chapter 7

Robust Jastrow-Function Optimization Algorithm for Variational Quantum Monte Carlo

The functional forms for some Jastrow functions used in variational Quantum Monte Carlo (VMC) allow unwanted singularities for some parameter values. These functional forms are used because, with a small number of parameters, an unknown function can be closely approximated. Unfortunately, the possible singularities can make numerical optimization of the wave function difficult. Presented here is a numerically-stable, robust algorithm to numerically optimize wave functions during a VMC calculation, which avoids parameters which produce singularities in the Jastrow function.

7.1 Introduction

Variational Quantum Monte Carlo (VMC) is becoming a popular method for accurately calculating the properties of atomic and molecular systems (Section 4.1). In VMC, a parameterized wave function is constructed; then, the parameters are optimized to yield an approximation to a wave function of the many-body quantum-

mechanical system.

The optimization problem for VMC can be formulated as

$$\min_{\mathbf{p}} f(\mathbf{p}) \tag{7.1}$$

where $f(\mathbf{p})$ is the objective function for the problem. The most commonly used objective function for VMC optimization is

$$f(\mathbf{p}) = \int \rho_{VMC}(\mathbf{x};\mathbf{p}) \left(E_{local}(\mathbf{x};\mathbf{p}) - E_G\right)^2 d\mathbf{x}^{3N} \tag{7.2}$$

where $\rho_{VMC}(\mathbf{x};\mathbf{p})$ and $E_{local}(\mathbf{x};\mathbf{p})$ are defined in Section 4.1 and E_G is a guess for the energy of the state we are interested in. The integral is evaluated using Monte Carlo integration (Section 4.1) with correlated sampling [32] to reduce the statistical error in $f(\mathbf{p}_1) - f(\mathbf{p}_2)$.

The most common wave function used for VMC calculations of atomic and molecular systems is (Sections 4.1.1 and 4.1.2)

$$\Psi_{VMC} = \sum_i c_i \psi_i J \tag{7.3}$$

where c_i are constants, ψ_i is a determinantal wave function which is the product of a Slater determinant for the up-spin electrons and a Slater determinant for the down-spin electrons, and J is a symmetric function, called the Jastrow function, of the electron-electron and electron-nuclear distances, which introduces explicit particle-

particle correlations into the wave function.

The Jastrow function can be expanded as a sum of 1-body, 2-body, etc., terms. It has been shown that the most important terms are the electron-nuclear and electron-electron terms [26, 33]; therefore, the majority of calculations employ only these terms. Such a Jastrow function can be expressed as

$$J = e^{\sum u_{ij}(r_{ij};\mathbf{p})} \tag{7.4}$$

where the sum is over all electron-electron and electron-nuclear pairs, r_{ij} is the distance between particles i and j, $u_{ij}(r;\mathbf{p})$ is a function describing the correlations of particles i and j in the wave function, and $\mathbf{p}$ is the set of parameters which can be adjusted to modify $u_{ij}(r;\mathbf{p})$ and, therefore, optimize the wave function.

Numerous functional forms for $u_{ij}(r;\mathbf{p})$ have been used in practice [32, 73, 74, 75]. Many of these, including the Padé-Jastrow function, have singularities for certain values of $\mathbf{p}$. For VMC calculations, only non-negative singularities are important because r is the distance between two particles, and $r \geq 0$.

The above singularities make the VMC optimization difficult. Because Monte Carlo integration is used to evaluate $f(\mathbf{p})$, there is only a very small probability that a singularity will be sampled during the integration; therefore, the value of $f(\mathbf{p})$, evaluated using Monte Carlo integration, is significantly less than the true value of $f(\mathbf{p})$ if there is a singularity in the Jastrow function. This can lead to very unstable numerical optimization calculations.

Presented here is a numerically stable, robust algorithm to numerically optimize

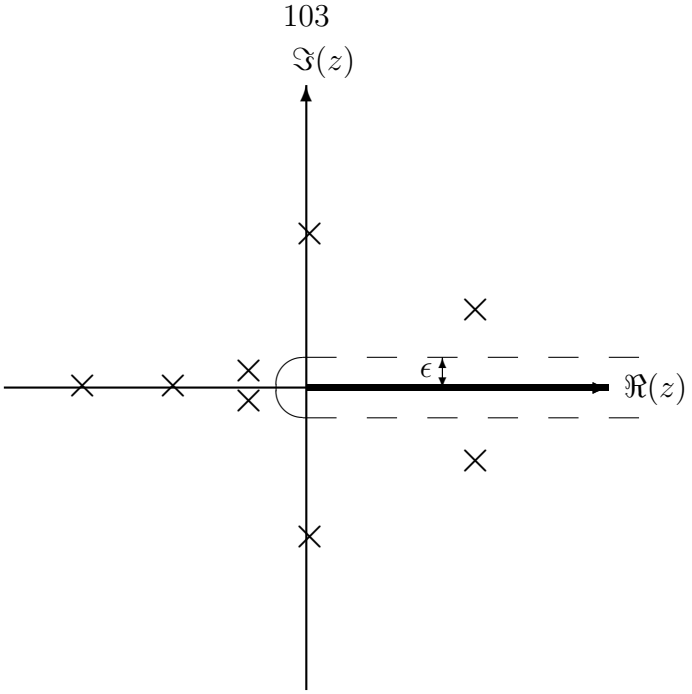


Figure 7.1: Poles of the Jastrow function. If any poles fall within the region of size ϵ, the Jastrow function is considered to be singular.

VMC calculations using Jastrow functions which have unwanted singularities for some values of $\mathbf{p}$.

7.2 Algorithm

To construct a numerically stable, robust VMC optimization algorithm, it must be possible to determine for which values of the parameters, $\mathbf{p}$, $u_{ij}(r;\mathbf{p})$ has a singularity in the range $r \in [0, \infty)$. To accomplish this, the poles of $u_{ij}(z;\mathbf{p})$, where z is a complex number, are determined. A parameter set is then said to be singular if $u_{ij}(z;\mathbf{p})$ has a pole within a small distance, ϵ, of the positive real axis (Figure 7.1).

The distance from any point in the complex plane, z, to the positive real axis,

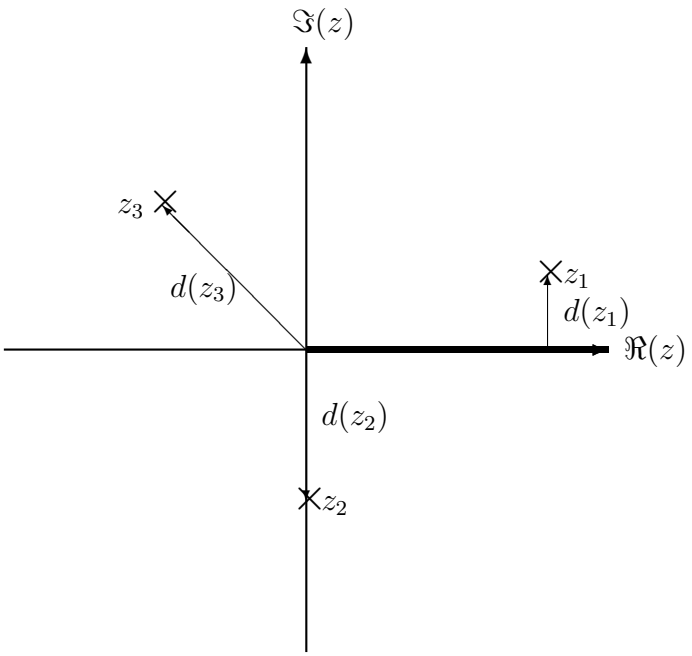


Figure 7.2: Distance of poles from the positive real axis.

$d(z)$, is

$$d(z) = \begin{cases} |\Im(z)| & \text{if } \Re(z) \geq 0, \\ |z| & \text{otherwise.} \end{cases} \tag{7.5}$$

where $\Re(z)$ is the real component of z and $\Im(z)$ is the imaginary component of z. This is pictorially represented in Figure 7.2.

Using the above results, the VMC optimization problem (Equation 7.1) can be reformulated as a constrained optimization problem.

$$\min_{\mathbf{p}} f(\mathbf{p}) \quad \text{subject to } d(z_i(\mathbf{p})) > \epsilon \text{ for all poles, } z_i(\mathbf{p}), \text{ of the Jastrow function} \tag{7.6}$$

There are a number of methods to solve constrained optimization problems of the same form as Equation 7.6. These include the augmented Lagrangian method,

logarithmic-barrier method, and the sequentially linearly constrained methods [76]. Of these methods, the logarithmic-barrier method is most natural and simple to implement in this case.

The logarithmic-barrier method can be used to solve Equation 7.6 by transforming the constrained optimization problem into the unconstrained optimization problem

$$\min_{\mathbf{p}} F(\mathbf{p};\mu) \tag{7.7}$$

where

$$F(\mathbf{p};\mu) = f(\mathbf{p}) - \mu \sum_{z_i \in \mathcal{P}(\mathbf{p})} \log\left(d(z_i)\right) \tag{7.8}$$

is the new objective function, μ is a real constant known as the barrier parameter, and $\mathcal{P}(\mathbf{p})$ is the set of all poles of the Jastrow function for parameter set $\mathbf{p}$. It can be shown that the solution of Equation 7.7 as $\mu \to 0$ is equal to the solution of Equation 7.6 [76].

By solving Equation 7.7 instead of Equation 7.1, the numerical optimization will be much more stable and robust, since no parameter sets will be chosen which produce a singular Jastrow function. Furthermore, this algorithm is very easy to implement in existing software by selecting a small value for μ and adding a logarithmic barrier (Equation 7.8) to the objective function being optimized.

7.3 Example: Padé-Jastrow Function

The Padé-Jastrow function (Section 4.1.2) is the most commonly used Jastrow function in QMC. It is a rational polynomial

$$u_{ij}(r) = \frac{\sum_{k=1}^{N} a_{ij,k} r^k}{1 + \sum_{k=1}^{M} b_{ij,k} r^k} \tag{7.9}$$

where $a_{ij,k}$ and $b_{ij,k}$ are parameters.

The poles of $u_{ij}(z)$ are equal to the zeroes of the denominator

$$1 + \sum_{k=1}^{M} b_{ij,k} z^k = 0 \tag{7.10}$$

and can be evaluated efficiently using Laguerre's method [23].

It is possible for a zero of the numerator to cancel a zero of the denominator. This case will be ignored because the calculation takes place on a finite precision computer. When the rational polynomial is evaluated, the numerator and denominator are calculated separately, and then division is performed [23]. Evaluating the Jastrow function at this zero will yield a division by zero. This can be avoided by treating all zeroes of the denominator as poles.

Using this machinery, the robust Jastrow-function optimization algorithm can be applied to wave functions using Padé-Jastrow functions.

7.4 Conclusion

This new algorithm provides a robust and stable means of optimizing wave functions in VMC if the Jastrow function has possible singularities for some parameter values. The new algorithm can be easily added to current software simply by appending a logarithmic barrier to the objective function.

Implementing this algorithm for Padé-Jastrow functions is very easy. Details are provided in Section 7.3.

Chapter 8

Generic Jastrow Functions for Quantum Monte Carlo Calculations on Hydrocarbons

A Jastrow function with parameters which are transferable to large classes of hydrocarbons is demonstrated [77]. Such a Generic Jastrow function can lead to improved initial guesses for variational Quantum Monte Carlo (VMC) wave function optimizations. Furthermore, with more development, it may be possible to construct wave functions for diffusion Quantum Monte Carlo (DMC) calculations without first performing a VMC wave function optimization. Both possibilities will significantly reduce the time necessary to perform a Quantum Monte Carlo (QMC) calculation of a molecular system.

8.1 Introduction

QMC methods are gaining popularity for high-accuracy quantum-mechanical calculations of molecular systems. Methods, such as DMC [55, 35, 78, 79], can potentially provide accuracies better than coupled-cluster methods while scaling significantly

better [31].

The standard approach to accurately calculate molecular properties using QMC involves performing DMC on a high-quality wave function obtained from a VMC calculation [32, 26, 52, 53, 30, 28, 54, 55, 56, 57]. The better the VMC wave function, the faster the DMC calculation will converge and the smaller the error in observables which do not commute with the Hamiltonian. During typical QMC calculations, obtaining a high-quality VMC wave function often requires 50% of the computing resources devoted to the problem [80].

Presented here is the first indication that it may be possible to construct wave functions of high-enough quality for efficient DMC calculations without performing a VMC calculation. This is done, first, by choosing a parameterized form for the wave function and then by finding parameter sets which are transferable between different molecular systems. This is analogous to the contraction coefficients used in contracted Gaussian basis sets such as 6-31G [51] or cc-pVTZ [71].

For a system of N electrons, VMC evaluates the $3N$-dimensional energy expectation integral for the system

$$\langle E \rangle = \int \Psi^*_{VMC}(\mathbf{x};\mathbf{p})\hat{H}\Psi_{VMC}(\mathbf{x};\mathbf{p})d\mathbf{x}^{3N} \tag{8.1}$$

using Monte Carlo integration. $\hat{H}$ is the Hamiltonian operator for the system, $\mathbf{x}$ is the position of all electrons, $\Psi_{VMC}(\mathbf{x};\mathbf{p})$ is a parameterized wave function, and $\mathbf{p}$ is a set of parameters. The parameters are variationally optimized to yield a high-quality wave function [32]. These high-quality parameters minimize $\langle E \rangle$ and the variance in

$\langle E \rangle$ in some sense.

In practical applications of VMC, $\Psi_{VMC}(\mathbf{x};\mathbf{p})$ is often chosen to be

$$\Psi_{VMC}(\mathbf{x};\mathbf{p}) = \Psi_{Trial}(\mathbf{x}) J(\mathbf{x};\mathbf{p}) \tag{8.2}$$

where $\Psi_{Trial}(\mathbf{x})$ is a wave function obtained from a standard quantum-mechanical calculation (Hartree-Fock, Density Functional Theory, etc.) and $J(\mathbf{x},\mathbf{p})$ is a Jastrow function, which introduces particle-particle correlations.

Electron-nuclear (one-body) and electron-electron (two-body) interactions in the Jastrow function are the most important terms. Although the addition of three-body terms results in a large reduction in the fluctuations of the local energy and a lowering of the energy [32], only a simple one-body and two-body Jastrow function is considered here.

The Jastrow function containing one- and two-body interactions can be expressed as

$$J(\mathbf{x};\mathbf{p}) = \exp\left(\sum u_{ij}(r_{ij};\mathbf{p})\right) \tag{8.3}$$

where the sum is over all pairs of particles (electron-nuclear and electron-electron), r_{ij} is the distance between particles i and j, and $u_{ij}(r;\mathbf{p})$ is a parameterized function describing the correlation between particles i and j.

For the work presented here, we have chosen to use a Hartree-Fock wave function for $\Psi_{Trial}(\mathbf{x})$ and the popular, for finite systems, Padé-Jastrow function [26]. Because both Hartree-Fock and VMC energies are variational, this combination (HF-

GJ) provides an objective method for examining the quality of different transferable parameter sets.

The Padé-Jastrow function uses

$$u_{ij}(r) = \frac{a_{ij}r}{1 + b_{ij}r} \tag{8.4}$$

where short range interactions are determined by a_{ij} and the range for the interactions is determined by b_{ij}. By using the cusp condition [18, 19] to determine the value of a_{ij}, singularities in the local energy are removed, greatly decreasing the variances of calculated quantities.

8.2 Computational Experiments

As a proof of concept, we examine the possibility of a Generic Jastrow for hydrocarbon systems using the Hartree-Fock/Padé-Jastrow wave function described above (Equations 8.2, 8.3, and 8.4). The dominant electron correlations not described by a Hartree-Fock wave function are expected to be from spatially similar opposite-spin electrons. These electrons do not actively avoid one another, leading to significant fluctuations in the local energy. On the other hand, antisymmetry forces same-spin electrons to avoid one another, leading to smaller local-energy fluctuations.

Using the cusp condition and the fact that opposite-spin electron correlations are the most important, a "generic" Jastrow function can be constructed (Equations 8.5-

8.8).

$$u_{\uparrow\downarrow}(r) = \frac{\frac{1}{2}r}{1 + b_{\uparrow\downarrow}r} \tag{8.5}$$

$$u_{\uparrow\uparrow}(r) = u_{\downarrow\downarrow}(r) = \frac{\frac{1}{4}r}{1 + 100r} \tag{8.6}$$

$$u_{\uparrow H}(r) = u_{\downarrow H}(r) = \frac{-r}{1 + 100r} \tag{8.7}$$

$$u_{\uparrow C}(r) = u_{\downarrow C}(r) = \frac{-6r}{1 + 100r} \tag{8.8}$$

Here, a_{ij} is chosen so that the cusp condition is satisfied, making the local energy non-singular. The range of the correlations between opposite-spin electrons is determined by an adjustable parameter, $b_{\uparrow\downarrow}$, and the range of all other correlation functions is limited by a large $b_{ij} = 100$ value. This Generic Jastrow function corrects the Hartree-Fock wave function as two particles approach one another but leaves the wave function undisturbed at longer range where the Hartree-Fock wave function performs well. The one free parameter, $b_{\uparrow\downarrow}$, can then be explored to find a value which is transferable to all hydrocarbon compounds.

8.2.1 Generation of Transferable Hydrocarbon Parameters

To examine the transferability of $b_{\uparrow\downarrow}$ values, a set of hydrocarbons with a variety of bonding was used (Table 8.1). This set has compounds with single, double, and triple bonds as well as delocalized π-systems. Optimal geometries and trial wave functions were obtained using Hartree-Fock calculations with the 6-31G** basis set [51, 81]. They were performed using the Jaguar program suite [72]. Quantum Monte Carlo

methane	ethane
ethylene	acetylene
allene	benzene
cis-butadiene	trans-butadiene

Table 8.1: Compounds used to determine transferability of hydrocarbon Generic Jastrow parameters.

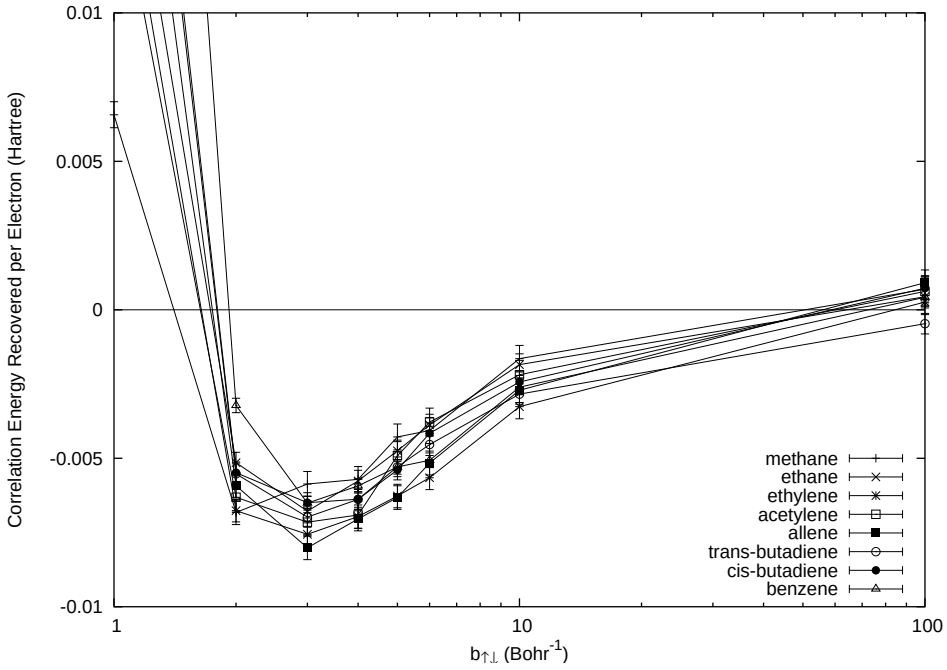


Figure 8.1: Correlation energy (Hartree) recovered divided by total nuclear charge.

calculations were performed using QMcBeaver [82].

The results of varying $b_{\uparrow\downarrow}$, for these simple hydrocarbons, are shown in Figures 8.1 and 8.2. Figure 8.1 shows the correlation energy recovered per electron for various $b_{\uparrow\downarrow}$ values. This shows a clear minimum with $b_{\uparrow\downarrow}$ between 2 and 4. The minimum for all compounds, except for methane, is 3; methane's minimum is 2. Figure 8.2 shows the ratio of the variance in energy of a HF-GJ wave function and a HF wave function evaluated using VMC. Again, there is a minimum for all compounds with $b_{\uparrow\downarrow}$ between 2 and 4. For $b_{\uparrow\downarrow} = 3$ the variance in the energy is reduced by a factor

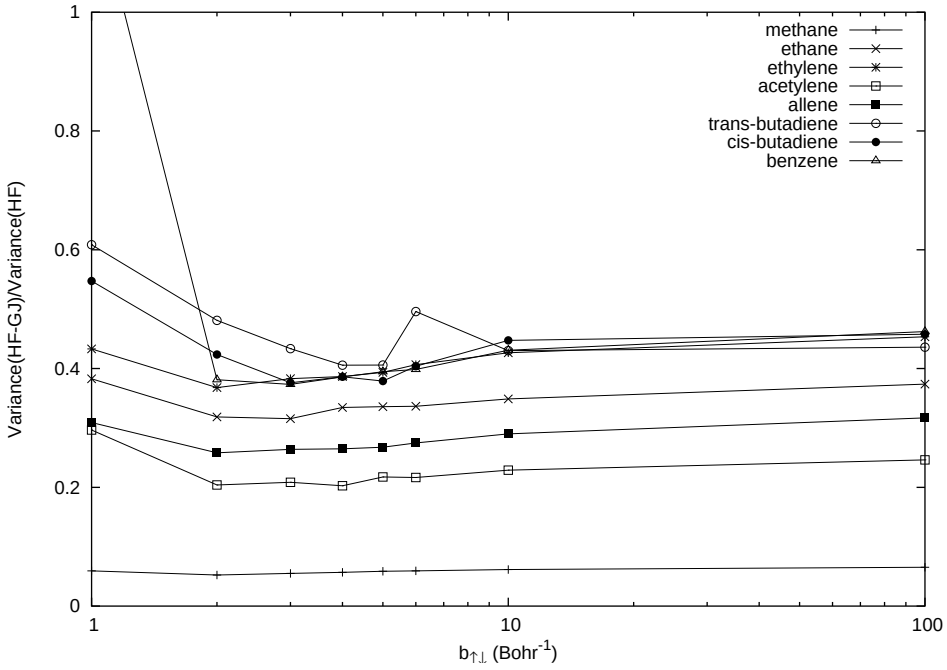


Figure 8.2: Reduction of the QMC variance for a wave function containing a Generic Jastrow compared to a Hartree-Fock wave function.

of 18 for methane, 3.2 for ethane, 2.6 for ethylene, 4.8 for acetylene, 3.8 for allene, 2.3 for trans-butadiene, 2.7 for cis-butadiene, and 2.7 for benzene. A wave function is generally considered to be of "good enough" quality to efficiently use DMC if the variance is reduced by a factor of 3 [80] over a Hartree-Fock wave function. With no optimizations, the Generic Jastrow with $b_{\uparrow\downarrow} = 3$ comes close to accomplishing this for all of the test compounds in Table 8.1.

To test the transferability of these generic Jastrow parameters to other hydrocarbons, two different conformations of [10]annulene ($C_{10}H_{10}$) were examined. Standard quantum-mechanical calculations have problems correctly evaluating the relative energies of conformers of [10]annulene [8] making it an interesting case for future QMC calculations.

For these validation calculations, Schaefer's geometries [8] were used with a HF/cc-pVTZ [71, 72] trial wave function and a Generic Jastrow function with $b_{\uparrow\downarrow} = 3$. VMC calculations recovered $(5.6 \pm 0.4) \times 10^{-3}$ Hartree per electron of correlation energy for the naphthalene-like conformer and $(6.0 \pm 0.4) \times 10^{-3}$ Hartree per electron for the twist conformer. These values fall within the same range as the test compounds (Figure 8.1), indicating that the Generic Jastrow could be extended to hydrocarbons not in the test set.

8.2.2 Generic Jastrow for DMC

To examine how well the Generic Jastrow parameters transfer to DMC, fixed-node DMC energy calculations were performed. The DMC algorithm used is the small time-step error algorithm of Umrigar, Nightingale, and Runge [27]. During a calculation, 10^5 time steps were performed after the walkers were equilibrated.

To assess how rapidly the DMC calculation converges, and hence the quality of the HF-GJ wave function, the standard deviation of the energy at the end of the calculations is analyzed (Figure 8.3). The standard deviations and the associated error bars are calculated using the DDDA algorithm [42].

Figure 8.3 shows that the standard deviation in the fixed-node DMC energy has a minimum when $b_{\uparrow\downarrow}$ is between 2 and 4. This is the same range as for VMC.

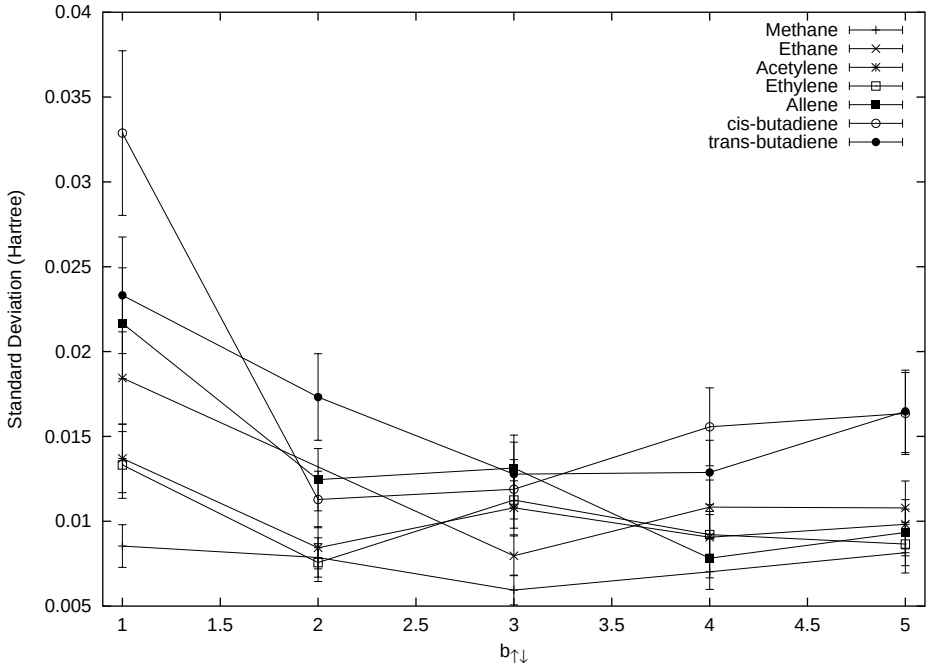


Figure 8.3: Variances of DMC calculations using Generic Jastrow functions.

8.3 Conclusion

The work presented here shows that it is possible to find parameter sets for Jastrow functions which are transferable between different compounds. For a simple 1-parameter Padé-Jastrow function (Equations 8.5-8.8), a parameter value of 2 to 4 is optimal when calculating the properties of hydrocarbons.

Transferable parameter sets can be used as initial guess parameters for VMC calculations. These parameters are, in some sense, near the optimal parameters for a system; therefore, fewer optimization steps would be required to optimize the VMC wave function. This would lead to significantly shorter VMC calculations.

Additionally, transferable parameter sets could be used to skip the VMC wave function optimization before a DMC calculation. For this approach to work in gen-

eral, more work must be done to find "generic" Jastrow parameters for better functional forms of the Jastrow function than were used here. The need for higher-quality Generic Jastrow functions, when skipping the VMC wave function optimization, results from DMC evaluating a mixed estimator of an observable. The error in the calculated value of a non-commuting observable is second order in the difference between the exact wave function and the trial wave function for the DMC calculation [41, 38, 35]. This requires trial wave functions to be of very high quality to minimize this error.

This work is a proof-of-concept showing that it is possible to construct Jastrow functions with parameters that are transferable to many compounds. A great deal of further study is necessary to fully explore the possibilities of Generic Jastrow functions.

Chapter 9

QMcBeaver

> *I can't think of a job I'd rather do than computer programming. All day, you create patterns and structure out of the formless void, and you solve dozens of smaller puzzles along the way. The wit and ingenuity of the human brain is pitted against the remorseless speed and accuracy of the electronic one [83].*
>
> Peter van der Linden, 1994

The QMC calculations presented in this work were performed using QMcBeaver. This software package was designed and implemented by myself and Michael T. Feldmann while graduate students at the California Institute of Technology (1999-2003).

Unlike existing QMC software, QMcBeaver uses object-oriented design principles. This allows the software to be easily modified, so that new ideas can rapidly be evaluated without extensive, time-consuming modifications to the source code. This has proven to be very beneficial in evaluating the ideas presented in the previous chapters.

As of 2003, QMcBeaver has become Open Source and has been released under the

GNU General Public License (GPL). The Open Source project is being hosted by *SourceForge.net*. To obtain the most current release go to:

http://qmcbeaver.sourceforge.net

or

http://sourceforge.net/projects/qmcbeaver

9.1 QMcBeaver Copyright Statement

To contact the authors, write to:

drkent@users.sourceforge.net

or

mtfeldmann@users.sourceforge.net

this software and its documentation, even if the California Institute of Technology has been advised of the possibility of such damage. Lastly, the California Institute of Technology has no obligations to provide maintenance, support, updates, enhancements or modifications.

To receive a copy of the GNU General Public License, go to

http://www.gnu.org/licenses/gpl.txt

or write to

The Free Software Foundation, Inc.

59 Temple Place, Suite 330

Boston, MA 02111–1307 USA

Bibliography

[1] J. L. Gay-Lussac. Memoir on the combination of gaseous substances with each other. *Mémoires de la Société d'Arcueil*, 2:207, 1809.

[2] P. A. M. Dirac. Perspective on quantum mechanics of many-electron systems. *Proc. Roy. Soc. (London) A*, 123:714, 1929.

[3] R. G. Parr and W. Yang. *Density Functional Theory of Atoms and Molecules*. Oxford University Press, New York, 1989.

[4] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Physical Review*, 136:B864, 1964.

[5] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140:A1133, 1965.

[6] K. Morokuma, D. G. Musaev, T. Vreven, H. Basch, M. Torrent, and D. V. Khoroshun. Model studies of the structures, reactivities, and reaction mechanisms of metalloenzymes. *IBM Journal of Research and Development*, 3-4:367, 2001.

[7] W. Koch and M. C. Holthausen. *A Chemist's Guide to Density Functional Theory*. Wiley, New York, 2001.

[8] R. A. King, T. D. Crawford, J. F. Stanton, and H. F. Schaefer III. Conformations of [10]annulene: More bad news for density functional theory and second-order perturbation theory. *Journal of the American Chemical Society*, 121:10788–10793, 1999.

[9] T. Torelli and L. Mitas. Electron correlation in C_{4N+2} carbon rings: Aromatic versus dimerized structures. *Physical Review Letters*, 85(8):1702–1705, 2000.

[10] A. E. Mattsson. In pursuit of the "divine" functional. *Science*, 298:759, 2002.

[11] R. P. Feynman, R. B. Leighton, and M. Sands. *The Feynman Lectures on Physics*. Addison-Wesley, 1965.

[12] C. Cohen-Tannoudji, B. Diu, and F. Laloe. *Quantum Mechanics*. Wiley-Interscience, New York, 1977.

[13] J. J. Sakurai. *Modern Quantum Mechanics*. Addison-Wesley, New York, 1994.

[14] P. A. M. Dirac. *The Principles of Quantum Mechanics*. Oxford Scientific Publications, 1957.

[15] J. M. Thijssen. *Computational Physics*. Cambridge, New York, 1999.

[16] R. P. Feynman and A. R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill, 1965.

[17] Atilla Szabo and Neil S. Ostlund. *Modern Quantum Chemistry*. McGraw-Hill, New York, 1982.

[18] T. Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Commun. Pure Appl. Math.*, 10:151, 1957.

[19] R. T. Pack and W. Byers-Brown. Cusp conditions for molecular wavefunctions. *Journal of Chemical Physics*, 45:556, 1966.

[20] M. MacCallum. The breakdown of physics? *Nature*, 257:362, 1975.

[21] R. Y. Rubinstein. *Simulation and the Monte Carlo Method.* Wiley, New York, 1981.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in FORTRAN: The Art of Scientific Programming.* Cambridge University Press, Cambridge, second edition, 1992.

[23] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, 1993.

[24] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087, 1953.

[25] H. Eyring, J. Walter, and G. E. Kimball. *Quantum Chemistry.* Wiley, New York, 1944.

[26] C. J. Umrigar. Variational Monte Carlo basics and applications to atoms and molecules. In M. P. Nightingale and C. J. Umrigar, editors, *Quantum Monte*

Carlo Methods in Physics and Chemistry, volume 525 of *Nato Science Series C: Mathematical and Physical Sciences*, pages 129–160, Dordrecht, The Netherlands, 1999. Kluwer Academic Publishers.

[27] C. J. Umrigar, M. P. Nightingale, and K. J. Runge. A diffusion Monte Carlo algorithm with very small time-step errors. *Journal of Chemical Physics*, 99(4):2865–2890, 1993.

[28] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum Monte Carlo simulations of solids. *Reviews of Modern Physics*, 73(1):33–83, 2001.

[29] P. R. C. Kent, R. Q. Hood, A. J. Williamson, R. J. Needs, W. M. C. Foulkes, and G. Rajagopal. Finite-size errors in quantum many-body simulations of extended systems. *Physical Review B*, 59(3):1917–1929, 1999.

[30] J. C. Grossman and L. Mitas. High accuracy molecular heats of formation and reaction barriers: Essential role of electron correlation. *Physical Review Letters*, 79(22):4353–4356, 1997.

[31] A. J. Williamson, R. Q. Hood, and J. C. Grossman. Linear-scaling Quantum Monte Carlo calculations. *Physical Review Letters*, 87(24):246406, 2001.

[32] C. J. Umrigar, K. G. Wilson, and J. W. Wilkins. Optimized trial wave-functions for Quantum Monte Carlo calculations. *Physical Review Letters*, 60(17):1719–1722, 1988.

[33] A. Mushinski and M. P. Nightingale. Many-body trial wave-functions for atomic systems and ground-states of small noble-gas clusters. *Journal of Chemical Physics*, 101(10):8831–8841, 1994.

[34] M.P. Nightingale and C.J. Umrigar. *Quantum Monte Carlo Methods in Physics and Chemistry*. Kluwer Academic Publishers, Dordrecht, 1999.

[35] L. Mitas. Diffusion Monte Carlo. In M. P. Nightingale and C. J. Umrigar, editors, *Quantum Monte Carlo Methods in Physics and Chemistry*, volume 525 of *Nato Science Series C: Mathematical and Physical Sciences*, pages 247–261, Dordrecht, The Netherlands, 1999. Kluwer Academic Publishers.

[36] J. W. Moskowitz, K.E. Schmidt, M.A. Lee, and M. H. Kalos. A new look at correlation-energy in atomic and molecular-systems .2. the application of the green- function Monte Carlo method to lih. *Journal of Chemical Physics*, 77(1):349–355, 1982.

[37] M. H. Kalos and F. Pederiva. Fermion Monte Carlo. In M. P. Nightingale and C. J. Umrigar, editors, *Quantum Monte Carlo Methods in Physics and Chemistry*, volume 525 of *Nato Science Series C: Mathematical and Physical Sciences*, pages 263–286, Dordrecht, The Netherlands, 1999. Kluwer Academic Publishers.

[38] B. L. Hammond, W. A. Lester, and P. J. Reynolds. *Monte Carlo Methods in Ab Initio Quantum Chemistry*. World Scientific, Singapore, 1994.

[39] S.W. Zhang and M. H. Kalos. Exact Monte Carlo calculation for few-electron systems. *Physical Review Letters*, 67(22):3074–3077, 1991.

[40] J. B. Anderson, C. A. Traynor, and B. M. Boghosian. Quantum-chemistry by random-walk-exact treatment of many-electron systems. *Journal of Chemical Physics*, 95(10):7418–7425, 1991.

[41] D. M. Ceperley and M. H. Kalos. *Monte Carlo Methods in Statistical Physics*. Springer-Verlag, 1979.

[42] D. R. Kent IV, R. P. Muller, W. A. Goddard III, and M. T. Feldmann. Efficient algorithm for "on-the-fly" error analysis of local or distributed serially-correlated data. *Journal of Chemical Physics*, submitted, 2002.

[43] H. Flyvberg and H. Peterson. Error estimates on averages of correlated data. *Journal of Chemical Physics*, 91:461–466, 1989.

[44] K. Wilson. Recent developments in guage theories, 1979.

[45] C. Whitmer. Over-relaxation methods for Monte-Carlo simulations of quadratic and multiquadratic actions. *Physical Review D*, 29:306–311, 1984.

[46] S. Gottlieb, P. Mackenzie, H. Thacker, and D. Weingarten. Hadronic coupling-constants in lattice gauge-theory. *Nuclear Physics B*, 263:704–730, 1986.

[47] S. M. Rothstein and J. Vrbik. Statistical error of diffusion Monte Carlo. *Journal of Computational Chemistry*, 74:127, 1988.

[48] D. Chakraborty, R. P. Muller, S. Dasgupta, and W.A. Goddard III. The mechanism for unimolecular decomposition of RDX (1,3,5-trinitro-1,3,5-triazine), an ab initio study. *Journal of Physcial Chemistry*, 104:2261–2272, 2000.

[49] M. N. Ringnalda, J.-M. Langlois, R. B. Murphy, B. H. Greeley, C. Cortis, T. V. Russo, B. Marten, R. E. Donnelly, Jr., W. T. Pollard, Y. Cao, R. P. Muller, D. T. Mainz, J. R. Wright, G. H. Miller, W. A. Goddard III, and R. A. Friesner. Jaguar v4.1, 2001.

[50] B. H. Greeley, T. V. Russo, D. T. Mainz, R. A. Friesner, W. A. Goddard III, R. E. Donnelly, Jr., and M. N. Ringnalda. New pseudospectral algorithms for electronic structure calculations: Length-scale separation and analytical two-electron integral calculations. *Journal of Chemical Physics*, 101:4028, 1994.

[51] W. J. Hehre, R. Ditchfield, and J. A. Pople. Self-consistent molecular orbital methods. XII. further extensions of gaussian-type basis sets for use in molecular orbital studies of organic molecules. *Journal of Chemical Physics*, 56:2257, 1972.

[52] L. Mitas and J. C. Grossman. Quantum Monte Carlo for electronic structure of clusters and solids. *Abstracts of Papers of the American Chemical Society*, 211(1):21–COMP, 1996.

[53] L. Mitas. Electronic structure calculations by quantum monte carlo methods. *Physica B*, 237:318–320, 1997.

[54] Y. Kwon, D. M. Ceperley, and R. M. Martin. Transient-estimate Monte Carlo in the two-dimensional electron gas. *Physical Review B*, 53(11):7376–7382, 1996.

[55] D. M. Ceperley and L. Mitas. Quantum Monte Carlo methods in chemistry. In I. Prigogine and Stuart A. Rice, editors, *Advances in Chemical Physics*, volume XCIII. John Wiley and Sons, Inc., 1996.

[56] S. Fahy, X. W. Wang, and S. G. Louie. Variational Quantum Monte Carlo nonlocal pseudopotential approach to solids-formulation and application to diamond, graphite, and silicon. *Physical Review B*, 42(6):3503–3522, 1990.

[57] S. Fahy, X. W. Wang, and S. G. Louie. Variational Quantum Monte Carlo nonlocal pseudopotential approach to solids-cohesive and structural-properties of diamond. *Physical Review Letters*, 61(14):1631–1634, 1988.

[58] M. T. Feldmann and D. R. Kent IV. QM^{c}Beaver v20020109 ©, 2001-2002.

[59] D. R. Kent IV, R. P. Muller, W. A. Goddard III, M. T. Feldmann, and J. C. Cummings. Manager–worker-based model for the parallelization of quantum Monte Carlo on heterogeneous and homogeneous networks. *Journal of Computational Chemistry*, submitted, 2003.

[60] O. Yaser. New trends in high performance computing. *Parallel Computing*, 27:1–2, 2001.

[61] Y. Deng and A. Korobka. The performance of a supercomputer built with commodity components. *Parallel Computing*, 27:91–108, 2001.

[62] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15:200–222, 2001.

[63] R. P. Muller, M. T. Feldmann, R. N. Barnett, B. L. Hammond, P. J. Reynold, L. Terray, and W. A. Lester Jr. California Institute of Technology Material Simulation Center parallel QMAGIC, version 1.1.0p, 2000.

[64] R. Needs, G. Rajagopal, M. D. Towler, P. R. C. Kent, and A. Williamson. CASINO, the Cambridge Quantum Monte Carlo code, version 1.1.0, 2000.

[65] L. Smith and P. Kent. Development and performance of mixed OpenMP/MPI Quantum Monte Carlo code. *Concurrency: Practice and Experience*, 12:1121–1129, 2000.

[66] P. R. C. Kent. *Techniques and Applications of Quantum Monte Carlo*. PhD thesis, Robinson College at Cambridge, 1999.

[67] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference Volume 1, The MPI Core*. The MIT Press, Cambridge, Massachusetts, second edition, 1998.

[68] T. H. Dunning, Jr. Gaussian basis functions for use in molecular calculations. iii. contraction of $(10s5p)$ atomic basis sets for first-row atoms. *Journal of Chemical Physics*, 55:716, 1971.

[69] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery. General atomic and molecular electronic structure system. *Journal of Computational Chemistry*, 14:1347–1363, 1993.

[70] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery. Gamess, a package of ab initio programs, version 2000, 2000.

[71] T. H. Dunning. Gaussian basis sets for use in correlated molecular calculations. 1. the atoms boron through neon and hydrogen. *Journal of Chemical Physics*, 90(2):1007–1023, 1989.

[72] M. N. Ringnalda, J.-M. Langlois, R. B. Murphy, B. H. Greeley, C. Cortis, T. V. Russo, B. Marten, R. E. Donnelly, Jr., W. T. Pollard, Y. Cao, R. P. Muller, D. T. Mainz, J. R. Wright, G. H. Miller, W. A. Goddard III, and R. A. Friesner. Jaguar v4.0, 2001.

[73] J. W. Moskowitz and K.E. Schmidt. Correlated Monte Carlo wave-functions for some cations and anions of the 1st row atoms. *Journal of Chemical Physics*, 97(5):3382–3385, 1992.

[74] S. Y. Huang, Z. W. Sun, and W. A. Lester. Optimized trial functions for Quantum Monte Carlo. *Journal of Chemical Physics*, 92(1):597–602, 1990.

[75] P. W. Kozlowski and L. Adamowicz. An effective method for generating nonadiabatic many-body wave functions using explicitly correlated Gaussian-type functions. *Journal of Chemical Physics*, 95:6681, 1991.

[76] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[77] D. R. Kent IV, R. P. Muller, W. A. Goddard III, and M. T. Feldmann. Generic Jastrow functions for quantum Monte Carlo calculations on hydrocarbons. *Journal of Chemical Physics*, submitted, 2003.

[78] J. B. Anderson. Fixed-node Quantum Monte Carlo. *International Reviews in Physical Chemistry*, 14(1):85–112, 1995.

[79] P. J. Reynolds, D. M. Ceperley, B. J. Alder, and W. A. Lester. Fixed-node Quantum Monte Carlo for molecules. *Journal of Chemical Physics*, 77(11):5593–5603, 1982.

[80] J. Grossman. Personal communication.

[81] P. C. Hariharan and J. A. Pople. Influence of polarization functions on molecular-orbital hydrogenation energies. *Theoretica Chimica Acta*, 28:213, 1973.

[82] M. T. Feldmann and D. R. Kent IV. QMcBeaver v20021112 ©, 2001-2003.

[83] P. van der Linden. *Expert C Programming: Deep C Secrets*. Prentice Hall, New Jersey, 1994.