

# Chapter 1

## *Parallel Computational Chemistry: an Overview of NWChem*

David E. Bernholdt

### 1.1 Introduction

Computational chemistry has a long and venerable history, and with the help of improvements in computational methodology, and in computers themselves, is has been transformed into a virtually indispensable tool, used by a large cross-section of the discipline. The ability to model “real world” chemical systems with the necessary sophistication to obtain chemically meaningful results has helped produce a remarkable level of synergy between computational and experimental treatments of chemical problems. This, in turn, has fueled further interest in expanding the role of computational chemistry to even larger, more sophisticated, and more demanding simulations.

Vector supercomputers played a prominent role in the rise of computational chemistry, as chemists went beyond simple ports of existing codes, restructuring them and making important advances in algorithms. Today, few vector-based computers are still produced, but modern commodity CPUs make good use of the of the optimizations and algorithms originally designed for vector machines. The cutting edge of high-performance computing has shifted over to parallel computers, based on those same commodity CPUs, and computational chemistry is of course following. Numerous packages can make effective use of modestly sized shared memory parallel systems, but fewer are available for the high-end systems which use distributed memory architectures (including those in which each node is a shared memory multi-processor). The two inter-related issues primarily responsible for this situation are ease of programming and scalability of algorithms.

Computational chemistry methods tend to be computationally complex, and resource intensive (memory and disk as well as CPU), so parallelizing chemistry methods can be challenging, especially if scalability to large numbers of processors is required. In a shared-memory environment, programming is relatively straightforward, and reasonable parallel algorithms can provide adequate performance and scalability for many applications – sufficient for the modestly-sized shared resources typically available within a research group, department or university. However the largest and most complex problems require the largest massively parallel processors (MPPs), which are presently distributed memory systems. Chemistry algorithms scalable to hundreds or thousands of processors are far more challenging, and often too complex to be implemented within the message passing programming models widely used in distributed memory environments.

In this chapter, I present an overview of the NWChem software package[1, 2, 3, 4, 5, 6, 7, 8] as a representative of the current state of the art in highly-scalable fully-distributed parallel computational chemistry software. At it’s inception the goal for the NWChem project was to deliver molecular modeling software that provides 10 to 100 times the effective capability of what was currently available on conventional supercomputers. This necessitated the use of algorithms that exhibit parallel scalability; both in the size of the computational resource and in the molecular system being modeled. Scalable applications must not only effectively parallelize the requisite computations but must also utilize the aggregate subsystems of the MPP. Algorithms must distribute data across the total system memory not limiting the the functional problem

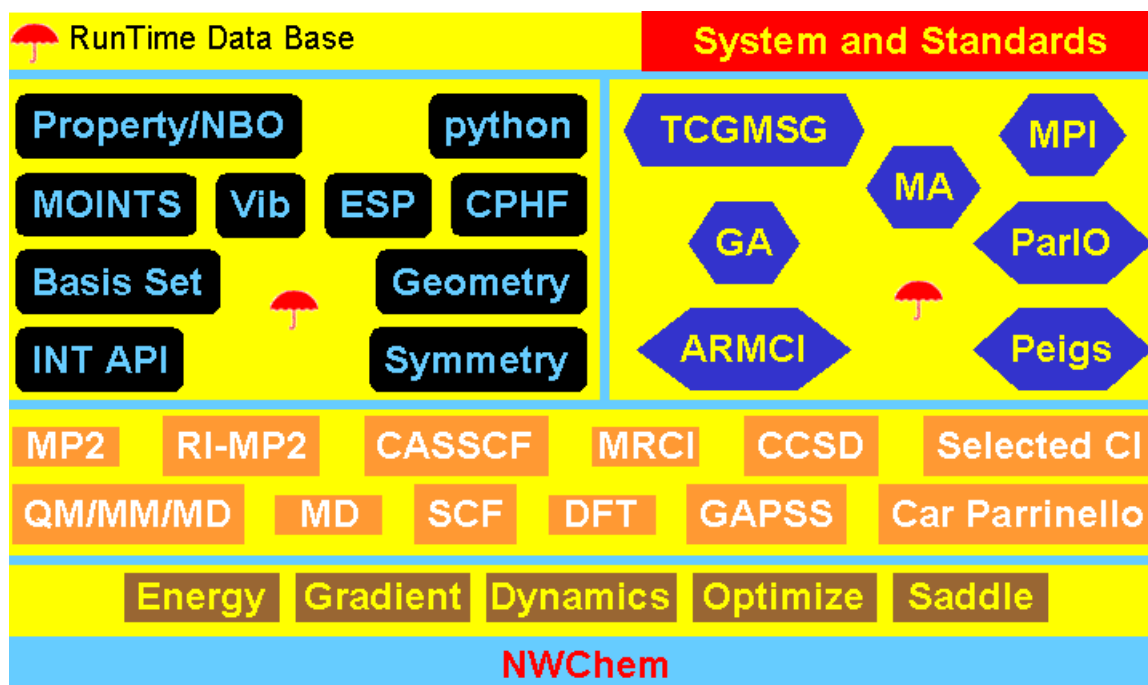


Figure 1.1: The NWChem architecture representing general functionality within NWChem which is built upon layers of other modules, tools, chemistry APIs, and computational and computer science standards. The link between NWChem and Ecce is a loosely coupled interface. The umbrella symbol identifies some of the software described in this section of the manuscript.

size by the effective memory of any single computational node. Furthermore, other MPP subsystems that algorithms exploit (i.e. communication and secondary storage) must be utilized in a scalable fashion.

The scalable modules in NWChem span the gamut of computational chemistry methods: Hartree-Fock or self consistent field (SCF), density functional theory, *ab initio* molecular dynamics, perturbation theory, coupled cluster, multiconfiguration self-consistent field (MCSCF), configuration interaction (CI), molecular mechanics, molecular dynamics, free energy simulations, Car-Parinello, etc. These modules have been implemented in the environment provided by a collection of supporting modules providing basic computational capabilities and fundamental services required for chemical computations. In the remainder of this chapter, I will describe the architecture of the NWChem package, and several critical supporting modules. I will conclude with descriptions of several of the chemistry methods in NWChem, focusing on their implementation in the NWChem environment and their performance.

## 1.2 The NWChem Architecture

In order to meet the original goals of the project, the initial NWChem development team recognized that NWChem would be a fast-growing code, in which ease of development (a short learning curve) and the ability to rapidly prototype algorithms would be critical to its success. Consequently, we chose a highly structured approach to the design of the package, using object oriented (OO) design throughout[9]. In deference to the fact that relatively few chemists have experience with truly object oriented languages, we chose to implement the OO design of NWChem in a combination of Fortran77 and C. Since these languages do not provide the kind of enforcement mechanisms that are built into OO languages, such an approach relies on the developers themselves to enforce the OO design, but overall we have found it to be quite effective. Newcomers to the code who are unfamiliar with OO design concepts can easily pick up the basics required to work successfully in the NWChem environment, and are quickly productive since they can work in familiar languages.

Figure 1.1 provides a schematic representation of the overall architecture of NWChem. The bottom two layers depict some of the fundamental tasks that NWChem can do (compute an energy, a gradient, perform

Newtonian dynamics, etc.) and some of the chemistry methods with which these tasks can be carried out (i.e. MP2, SCF, DFT). These are the two layers most directly visible to the NWChem user; the remaining modules constitute the environment or “umbrella” which allows for (relatively) easy parallel implementation of the various chemistry methods and tasks. On the left are modules that “know something about” chemistry, in other words those providing basic objects needed for chemical calculations. On the right are modules that provide the computational infrastructure for the NWChem: the parallel programming environment, parallel I/O support, etc. While most of these modules were developed in conjunction with NWChem, they are not specific to chemistry applications. Most are freely available separately from NWChem and have been adopted by other software developers both inside and outside of chemistry.

At the heart of the NWChem programming environment is the Global Array parallel programming model, which provides the developer with the appearance of a global shared memory environment in a portable fashion. This important component of the NWChem umbrella will be described in greater detail below, along with the PeIGS parallel eigensolver. Many other components of the NWChem programming environment are relatively straightforward conveniences with the important function of facilitating general, portable, and rapid development of computational chemistry software. For example, MA is a portable memory allocator, implementing both stack and heap memory management models, which provides equal access to objects from both Fortran and C code. It also provides support for debugging and verification (especially detecting array overwriting, and memory leaks). The run-time database (RTDB) provides a simple mechanism to allow the storage of name/value pairs (values can be of the basic Fortran datatypes, including one-dimensional arrays; other modules may provide convenience routines to read/write more complex data structures to the RTDB in an opaque fashion) which NWChem uses to communicate information between high-level modules and also as persistent storage between related jobs. The ParIO module is an abstraction layer which provides the user with three types of files:

- Disk-Resident Arrays (DRA) are a simple means of providing secondary storage for global arrays, the distributed arrays provided by the global array toolkit. All operations are collective, and are therefore open to additional optimizations on some parallel file systems.
- Exclusive Access Files (EAF) are sets of process-private files which can be accessed independently. They are typically used for out-of-core computations which do not lend themselves to collective I/O operations and the use of DRAs.
- Shared Files (SF) are shared by all processes and can be read or written in non-collective fashion at any arbitrary location in the file.

The ParIO library is layered on top of a “device library”, ELIO (for elementary I/O), which provides a portable interface to the file system and allows NWChem to take advantage of special high-performance I/O libraries which might be available on various platforms.

The chemistry-specific portion of the NWChem umbrella is similarly designed to facilitate the rapid development of chemistry software. Consistent with the object oriented design philosophy used throughout NWChem, these modules typically expose well-defined “application program interfaces” (APIs) to provide the developer with access to all the information and functions of the object while hiding the specific data structures. This helps protect the underlying data structures against manipulation (accidental or intentional) which does not conform to their API – an all too common occurrence in older, less well structured chemistry software. Another distinction from older chemistry software is that where appropriate, multiple instances of objects are supported. This allows the developer to, for example, refer explicitly to three different basis sets to be used in different aspects of a calculation by simple “handles” rather than error-prone manipulations of a single monolithic basis set data structure. Two excellent examples in NWChem include the most fundamental chemical objects in quantum mechanical electronic structure calculations are the definition of the molecular system (the “geometry” object in NWChem) and the basis set. The geometry object is a well defined, extensible API that provides all the geometrical and atomic data for the molecular system under study (e.g., masses, atomic number, nuclear charges, applied external field, coordinates, etc.). The basis set object is also a well defined, extensible API that provides all the basis set functionality for all NWChem modules that utilize basis sets. The basis set object is interfaced to a library that contains a wide variety of published basis sets. The NWChem basis set library is periodically synchronized with the EMSL basis set

library which is available to the public via a WWW interface[10]. Currently the NWChem library has 2043 Gaussian basis sets and 462 effective core potentials conveniently specified for the user community.

Other modules encapsulate various chemistry-specific computations which used by the main chemistry methods rather than being invoked directly at the user level. Perhaps one of the most widely used within NWChem is the integral evaluation module (“int api”). This module computes integrals of the (usually Gaussian) basis functions, possibly belonging to different basis sets, with various operators, an operation which is central to all quantum mechanical electronic structure methods. The module provides a uniform interface to five separate integral evaluation codes with different capabilities and strengths. The choice of which method of integral evaluation to use is normally made within the module based on details of the requested computation, but it can also be explicitly controlled by the software developer, or even by the NWChem user if the need arises. Because these codes are hidden behind a uniform interface, all modules which use the integral package can benefit immediately from the introduction of new methods and optimizations.

The NWChem umbrella modules are not set in stone. Though we tried to design from the start with the necessary flexibility and generality, inevitably there have been occasions which require existing objects to be modified or extended. In general, the most substantial changes have been extensions of functionality, and rarely are significant changes required in existing application code. Implementation of new chemistry methods within NWChem will sometimes occasion the extension of the functionality of the existing umbrella or the development of new supporting modules. New modules are also sometimes created by abstracting the repeated use of the same or similar functionality in different places.

## 1.3 NWChem Parallel Computing Support

### 1.3.1 The Global Array Toolkit

The Global Array (GA) Toolkit[11, 12, 13] implements the primary parallel programming model used within NWChem, though traditional message passing is also available and is used as needed. GAs provide a portable shared memory programming environment, which is implemented using native one-sided communications on distributed memory systems, and the common System V interface on true shared memory systems. The shared memory programming environment is important for two reasons. In the first place, it is much easier for the software developer to deal with, thus shortening the learning curve and facilitating development. Second, and more fundamentally, many sophisticated, highly scalable chemistry algorithms (and those in other fields) are extremely complex when written in message-passing form; others may be impossible to implement in the message-passing model because of the coordination required among processors.

Another important feature of the global array model is the fact that it explicitly exposes the memory hierarchy to the programmer. Specifically, global arrays distinguish between “local” and “remote” memory with difference latency and bandwidth characteristics. This is different from most shared memory programming environments, in which all memory is presumed to have the same access characteristics, but we have found the distinction quite useful because it helps software developers create algorithms that work well on both distributed and shared memory systems. It is also easy to integrate this distinction into the non-uniform memory access (NUMA) hierarchy with which the most programmers are already familiar: registers, cache, local memory, remote memory, etc. (Note that the Disk Resident Array component of the ParIO module described above can be thought of as extending the hierarchy one more level, to disk storage.)

At the simplest level the programming model offered using GA assumes that all “remote” memory access is the rate limiting step and that local memory access is much faster. Memory access using GA provides one-sided or asynchronous access to global data elements. Using the GA programming model, algorithms can be designed with knowledge of data locality, that can be tuned for many different computational resources to essentially cover the worst case scenario. This may require multiple algorithmic implementations to cover different ranges of bandwidth and latency. For example, consider the situation where one has two algorithms for a specific kernel in an application. The first algorithm has low latency requirements and the second algorithm can tolerate latency but with a factor of four in computation. The second algorithm would likely be the mainstream choice to work on “all” machines. The first algorithm could be turned “on” after testing the viability on each system as the application is ported. This is obviously not limited to two algorithms.

Global arrays themselves are multidimensional arrays which are distributed among processors in blockwise fashion. The distribution can be completely specified by the programmer, and may be regular or irregular, or

a GA convenience routine can be used to quickly create a regular blocked distribution. Data may be accessed locally or remotely using block-oriented “put”, “get” and “accumulate” functions. It is also possible for the programmer to inquire as to boundaries of the local block a global array, and to obtain direct access to the appropriate region of memory. This makes it convenient to write data parallel operations using GAs. By knowing the locality of data, programmers can explicitly manage the nature of the memory hierarchy for their parallel algorithm. The operations mentioned above can be used in asynchronous or one-sided fashion by any processor. Other GA functions are collective, including creation and destruction of GAs, synchronization, and high-level linear algebra and convenience routines. The GA library also includes interfaces to a variety of external linear algebra libraries, including the PeIGS parallel eigensolver described below.

In conjunction with the normal message passing programming model, GA is a fully functional and portable parallel programming model that is suitable for a wide range of applications. It is not however suitable for all applications. General guidelines with respect to algorithmic design and usability imply that GA would be appropriate for applications:

- with dynamic and irregular communication patterns,
- with a need for 1-sided access to shared data structures,
- when data locality is important,
- when a message passing implementation is too complicated,
- with a need for high-level operations on distributed arrays for out-of-core array based algorithms,
- where simulations are driven by dynamic load balancing,
- when portable performance is important.

GA is not necessarily appropriate for algorithms that:

- have systolic or nearest neighbor communications,
- require synchronization and point-to-point message passing (e.g., Cholesky factorization),
- can be effectively parallelized using interprocedural analysis and compiler parallelization,
- can use existing parallel constructs of a programming language and robust compilers are available.

### Parallel Linear Algebra: PeIGS

PeIGS is a collection of commonly used linear algebra subroutines for computing the eigensystem of the real standard symmetric eigensystem problem  $Ax = \lambda x$  and the general symmetric eigensystem problem  $Ax = \lambda Bx$ .  $A$  and  $B$  are dense and real matrices with  $B$  being positive definite.  $\lambda$  is an eigenvalue corresponding to the eigenvector  $x$ . PeIGS can also handle associated computations such as the Cholesky factorization of a positive definite matrices in packed storage format and linear matrix equations involving lower and upper triangular matrices in distributed packed row or column storage.

The numerical algorithms implemented are “standard” (c.f., References [14] and [15]) with the exception of the subspace inverse iteration and reorthogonalization scheme for finding basis vectors for degenerate eigensubspaces[16, 17] and the Dhillon-Fann-Parlett algorithm for computing eigenvectors of a real symmetric tridiagonal matrix[18].

The current version of PeIGS has some unique features not found in any other eigensystem library:

- The Dhillon-Fann-Parlett inverse iteration algorithm.
- Guaranteed orthonormal eigenvectors in the presences of large clusters of degenerate eigenvalues.
- packed storage for matrices.
- small scratch space requirements.

Table 1.1: Time for the solution of the tridiagonal matrix of rank 966 on a single IBM RS6000/590 processor[18]. The tridiagonal matrix was generated via Householder reduction of the fitting basis set, overlap matrix from a resolution of the identity, second-order Møller-Plesset (RI-MP2) simulation of a fluorinated biphenyl.

Method	Time (s)
PeIGS 3.0	6
PeIGS 2.0	126
eispack	32
LAPACK: bisection + inverse iteration	112
LAPACK: QR	46
LAPACK: divide and conquer	20

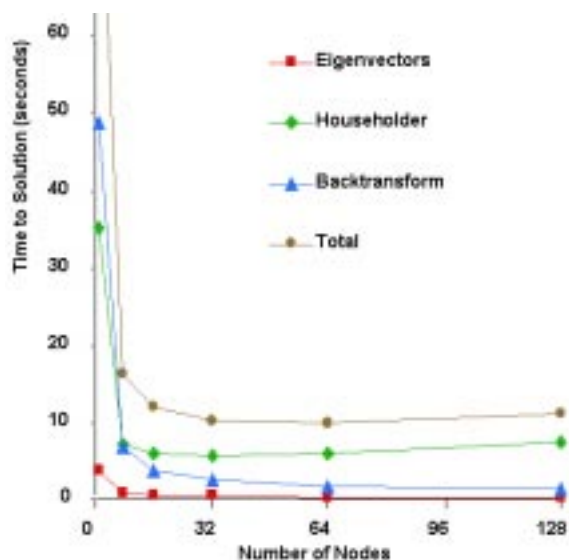


Figure 1.2: The performance of PeIGS using a tridiagonal matrix (rank 966) which was generated via Householder reduction of the fitting basis set, overlap matrix from an RI-MP2 simulation of a fluorinated biphenyl.

The second feature is particularly important in quantum chemistry applications, where degenerate eigenvalues are common and orthogonality is critical.

The performance of PeIGS in sequential mode is impressive. The data in Table 1.1 compares the current version of PeIGS with other standard solvers. The parallel performance of the three major components and the total time to solution is shown in Figure 1.2. The solution of the tridiagonal problem is scalable and fast; however at this point, the householder reduction and its back transform (i.e., producing the tridiagonal representation) is the identified bottle neck accounting for over 90% of the serial performance of the solver and up to 65% at 128 nodes.

Internally, PeIGS uses the traditional message-passing programming model and a column-wrapped distribution of the matrices. In NWChem the interface to PeIGS is hidden behind a GA based API, where the necessary data reorganization is conveniently hidden from the application programmer. The data transformation from the GA based global storage to that required for optimal PeIGS performance is very fast compared to the  $O(N^3/P)$  time required for the eigensolution operations.

## 1.4 NWChem Chemistry Modules

NWChem implements a broad range of computational chemistry methods, emphasizing quantum mechanically-based methods. There is insufficient space to describe all of them in detail, but I will provide a list of NWChem's current capabilities here, and focus on a more detailed discussion of two methods: Hartree-Fock self-consistent field (SCF), and the resolution of the identity approximation to second-order many-body perturbation theory (RI-MP2).

### 1. Molecular electronic structure

The following quantum mechanical methods are available to calculate energies, and analytic first derivatives with respect to atomic coordinates. Second derivatives are computed by finite difference of the first derivatives.

- Self Consistent Field (SCF) or Hartree Fock (RHF, UHF, high-spin ROHF). Code to compute analytic second derivatives is being tested.
- Gaussian orbital based Density Functional Theory (DFT), using many local and non-local exchange-correlation potentials (RHF and UHF) with formal  $O(N^3)$  and  $O(N^4)$  scaling.
- MP2 including semi-direct using frozen core and RHF or UHF reference.
- Complete active space SCF (CASSCF).

The following methods are available to compute energies only. First and second derivatives are computed by finite difference of the energies.

- CCSD(T), with RHF reference.
- Selected-CI with second-order perturbation correction.
- MP2 fully-direct with RHF reference.
- Resolution of the identity integral approximation MP2 (RI-MP2), with RHF and UHF reference (analytic first derivatives are being implemented).

For all methods, the following operations may be performed:

- Single point energy
- Geometry optimization (minimization and transition state)
- Molecular dynamics on the fully *ab initio* potential energy surface
- Numerical first and second derivatives automatically computed if analytic derivatives are not available.
- Normal mode vibrational analysis in Cartesian coordinates.
- Generation of an electron density file for graphical display.
- Evaluation of static, one-electron properties.
- Electrostatic potential fit of atomic partial charges (CHELPG method with optional RESP restraints or charge constraints)

In addition, interfaces are provided to:

- The COLUMBUS multireference CI package
- The natural bond orbital (NBO) package
- Python scripting language
- The POLYRATE package for the computation of chemical reaction rates

### 2. Pseudopotential plane-wave electronic structure

The following modules are available to compute the energy, minimize the geometry and perform *ab initio* molecular dynamics using pseudopotential plane-wave DFT with local exchange-correlation potentials.

- Fixed step length steepest descent
- Car-Parinello (extended Lagrangian dynamics)

With

- LDA and LSDA exchange-correlation potentials (Vosko et. al.)
- ( $G$  point) Periodic orthorhombic simulation cells
- Hamann and Troullier-Martins norm-conserving pseudopotentials
- Modules to convert between small and large plane-wave expansions

3. Periodic system electronic structure. A module (Gaussian Approach to Polymers, Surfaces and Solids, GAPSS) is available to compute energies by periodic Gaussian based DFT with many local and non-local exchange-correlation potentials.

4. Molecular dynamics The following classical molecular simulation functionality is available:

- Single configuration energy evaluation
- Energy minimization
- Molecular dynamics simulation
- Free energy simulation (multistep thermodynamic perturbation (MSTP) or multiconfiguration thermodynamic integration (MCTI) methods with options of single and/or dual topologies, double wide sampling, and separation-shifted scaling)

NWChem also has the capability to combine classical and quantum descriptions in order to perform:

- Mixed quantum-mechanics and molecular-mechanics (QM/MM) energy minimization and molecular dynamics simulation
- Quantum molecular dynamics simulation by using any of the quantum mechanical methods capable of returning gradients.

The classical force field includes:

- Effective pair potentials (functional form used in AMBER, GROMOS, CHARMM, etc.)
- First order polarization
- Self consistent polarization
- Smooth particle mesh Ewald (SPME)
- Twin range energy and force evaluation
- Periodic boundary conditions
- SHAKE constraints
- Consistent temperature and/or pressure ensembles

#### 1.4.1 Hartree-Fock Self-Consistent Field

The Hartree-Fock self-consistent field module is an essential functionality for NWChem or any quantum chemistry package. The NWChem SCF module and associated gradient module computes energies, wave functions, and gradients for closed-shell restricted Hartree-Fock (RHF), restricted high-spin open-shell Hartree-Fock (ROHF), and spin-unrestricted Hartree-Fock (UHF). The algorithms are designed around using the aggregate memory available on the parallel supercomputer or cluster.

The construction of the Fock matrix. is the most time-consuming part of any SCF calculation[19, 20], and is iterated until the wavefunction reaches self-consistency. The “Fock build” provides an interesting illustration of the form which parallelism often takes in computational chemistry. The most computationally demanding part of the Fock matrix is defined by

$$F_{\mu\nu} \leftarrow D_{\lambda\sigma} \{2(\mu\nu|\lambda\sigma) - (\mu\lambda|\nu\sigma)\} \quad (1.1)$$



where  $D$  is the density matrix, and the  $(\mu\nu|\lambda\sigma)$  are the so-called two-electron integrals. The integrals are defined by

$$(\mu\nu|\lambda\sigma) = \int \phi_\mu(r_1)\phi_\nu(r_1)\frac{1}{|r_1 - r_2|}\phi_\lambda(r_2)\phi_\sigma(r_2)d^3r_1d^3r_2 \quad (1.2)$$

where  $r_1$  and  $r_2$  are the positions of the two electrons, and the  $\{\phi_\mu(r)\}$  are the basis functions, usually a linear combination of Gaussian functions.

The cost of the Fock build scales with the number of integrals, which is formally  $O(N^4)$  for  $N$  basis functions. The NWChem SCF module was designed with a goal of 10,000 basis functions, so that the Fock and density matrices would be  $10,000 \times 10,000$  and the number of two-electron integrals is formally  $10^{16}$  (neglecting permutational symmetries of the indices and other factors).

Evaluation of the integrals occurs in irregular blocks, according to details of the basis set structure, so that a block may contain anything from a single integral to 10,000 integrals or more. The cost of each block is also highly variable and can only be crudely estimated in advance; it averages 500 FLOPs per integral value. Their cost, combined with permutational symmetries among indices makes it most efficient to drive the Fock build with a loop over the unique integrals, making the four different contributions dictated by those symmetries at one time rather than duplicating integral evaluation. In NWChem, integral evaluation is dynamically distributed across the processors (controlled by an atomic read-and-increment counter) without regard to the distribution of the global arrays containing the density and Fock matrices. Each processor fetches into a local buffer the four patches of the density matrix it needs to contract with the integral block it has been assigned, and puts the results into another set of local buffers which are accumulated into the proper places in the Fock matrix global array when the integral block is completed. To minimize communications, multiple integral blocks are aggregated into parallel tasks (maintaining a roughly 100 tasks per processor to insure load balance), and intelligent caching is used to avoid unnecessary communications for density and Fock matrix patches. Because of the irregular distribution, dimensions and timings of the parallel tasks, programming the Fock build using message passing, this algorithm would be extremely challenging to implement in a message passing environment, requiring synchronization between sender and receiver[21]. However using the one-sided communications of the GA model, it is straightforward; and the fact that the NUMA nature of the parallel processor is exposed to the programmer leads to the aggregation of integral blocks, and the use of intelligent caching, both of which provide significant performance gains.

The integrals do not change from one iteration of the SCF algorithm to the next, and may be stored or recomputed. Many SCF codes offer either “conventional” or “direct” modes, in which the integrals are either stored on disk and reused or are recomputed every iteration (the relative efficiency of these two approaches depends on both hardware performance factors, and on the particular molecule and basis set). NWChem provides a more flexible “semi-direct” algorithm, which includes memory as well as disk storage, and can span the entire range from fully disk- (or memory-) based to full recomputation according to available disk and memory space, or directly under user control. In addition to the fully distributed Fock build, a replicated data algorithm (Fock and density matrices replicated; integral evaluation distributed across the machine) is also implemented to take advantage of those situations where available memory and the molecule under study allow this approach. The convergence algorithm is the quadratic SCF[19] with both preconditioning and line search mechanisms built in.

Figure 1.3 shows the speed-up obtained for a modified crown-ether complex running on an IBM SP system using the semi-direct algorithm and taking advantage of the local secondary storage on the system. The 105 atom system, shown in Figure 1.4, has 1342 basis functions, and the calculation was completed in 5.7 hours on 240 nodes (160 MHz).

## 1.5 Resolution of the Identity Second-Order Many-Body Perturbation Theory (RI-MP2)

The RI-MP2 method is the result of applying the so-called “resolution of the identity” (RI) integral approximation [22, 23, 24] to the traditional second-order many-body perturbation theory method [25], often abbreviated MP2. MP2 is the simplest method to include the effects of dynamic electron correlation, which are important to the proper description of many chemical phenomena, and it is also the most widely used correlated method. MP2 calculations can be systematically improved upon by going to higher orders of perturbation theory or to coupled cluster methods [25].

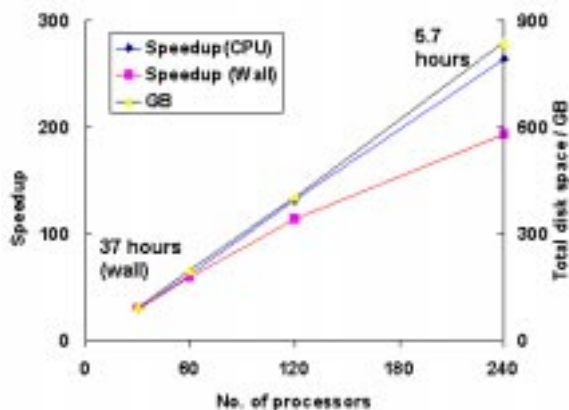


Figure 1.3: The scaling of the semi-direct SCF module for a modified crown ether system on an IBM SP, 160 MHz nodes, 512 MB memory per node, 3 GB of disk per node. 15 MB/sec/node sustained read bandwidth was achieved.

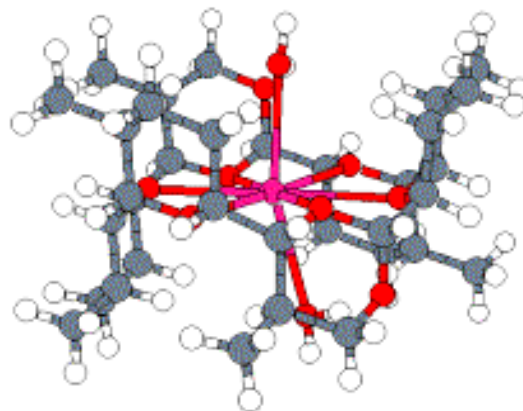


Figure 1.4: The modified crown ether system, with 105 atoms, 1343 basis functions using the Dunning augmented cc-pVDZ basis set, and 362 electrons

The MP2 energy can be simply expressed (in spin orbital form), as

$$E^{(2)} = \frac{1}{2} \sum_{i,j,a,b} \frac{(ia|jb)[(ia|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}, \quad (1.3)$$

with the  $\{\epsilon_p\}$  being the SCF orbital energies. The integrals are the same as in the SCF method, but transformed from the original “atomic orbital” (AO) basis to the “molecular orbital” (MO) basis which is one of the products of the SCF calculation. Given the MO basis integrals, the energy expression above costs  $O(N^4)$  to evaluate, but the transformation of the integrals from the AO to MO basis has a cost of  $O(N^5)$ , which dominates the calculation.

The RI approximation represents the two-electron integrals in the form [24]

$$(pq|rs) = \sum_{\Delta, \Phi} (pq|\Delta) V_{\Delta\Phi}^{-1}(\Phi|rs) \quad (1.4)$$

involving three-center two-electron integrals

$$(pq|\Delta) = \int \phi_p(r_1) \phi_q(r_1) \frac{1}{|r_1 - r_2|} \alpha_\Phi(r_2) d^3r_1 d^3r_2 \quad (1.5)$$

and two-center two-electron integrals

$$V_{\Delta\Phi} = \int \alpha_\Delta(r_1) \frac{1}{|r_1 - r_2|} \alpha_\Phi(r_2) d^3r_1 d^3r_2, \quad (1.6)$$

where upper case greek indices denote functions from a “fitting basis” introduced by this approximation. Essentially, the fitting basis  $\{\alpha_\Delta(r)\}$  is used to approximate the product space of the AO basis ( $\{\phi_i(r)\phi_j(r)\}$ ). To obtain the RI-MP2 energy[22, 26], Eq. 1.4 is simply substituted into the MP2 energy expression (Eq. 1.3)

$$E^{(2)} = \frac{1}{2} \sum_{i,j,a,b,\Delta,\Phi} \frac{(ia|\Delta) V_{\Delta\Phi}^{-1}(\Phi|jb)[(ia|\Delta) V_{\Delta\Phi}^{-1}(\Phi|jb) - (ib|\Delta) V_{\Delta\Phi}^{-1}(\Phi|ja)]}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}. \quad (1.7)$$

The RI approximation has several important strengths. Most obviously, it replaces a fourth-rank tensor (two-electron integrals) with a combination of third- and second-rank quantities, dramatically reducing the volume of data which must be computed, stored, and manipulated. Second, as the AO basis set gets larger

(for a fixed molecule) the product space will be increasingly redundant, making it possible to (nearly) span the space with a fitting set that is smaller, in relative terms. In a sense, the RI approximation could be said to “take advantage of” the use of large basis sets.

RI-MP2 calculations occur in two steps: the integral transformation, followed by the energy evaluation[26, 27]. The general form of the integral transformation can be written as

$$(ai|\Delta') = (ai|\Delta)V_{\Delta\Phi}^{-\frac{1}{2}} = C_{\mu a}C_{\nu i}(\mu\nu|\Delta)V_{\Delta\Phi}^{-\frac{1}{2}} \quad (1.8)$$

where the indices  $\mu$  and  $\nu$  represent the AO basis and  $C$  is the SCF eigenvector matrix, which defines the transformation from AOs to MOs. The  $V^{-\frac{1}{2}}$  term comes from rewriting Eq. 1.4 in a symmetric form that further simplifies integral handling, as first suggested by Rendell and Lee [28]. This step requires  $O(N^4)$  operations as opposed to the  $O(N^5)$  for the exact MP2 transformation. The first two transformation steps ( $C_{\mu a}$  and  $C_{\nu i}$ ) are handled, in succession, locally to each processor. The fitting basis index is distributed across processors, so that each node generates AO integrals for all  $\mu$  and  $\nu$  and a subset of  $\Delta$ . In order to make the matrix multiplications more efficient, the integral blocks are aggregated in a local buffer sized according to the available memory before the two transformations are applied. The results are accumulated into a global array with  $ai$  as the combined row index and  $\Delta$  as the column index, distributed in the same fashion as the integral evaluation loop (making the accumulate a local operation). The third transformation step is carried out as a parallel matrix multiplication (*ga\_dgemm*) of the GA just produced with another GA holding  $V^{-\frac{1}{2}}$  (computed using GA and PeIGS routines). If there is insufficient total memory available to complete the entire transformation in a single pass, multiple passes are made based on the  $i$  index.

The primary data structure of the energy evaluation phase is a fourth-rank tensor representing quantities like the (approximate) four-center two-electron integral ( $ia|jab$ ). It is organized as a supermatrix with row and column indices  $i$  and  $j$ , each element of which is a complete matrix labeled by  $a$  and  $b$ . The calculation is performed as a loop over  $i$  and  $j$ , blocked according to available memory. All of the GAs of this type are distributed across the machine in regularly sized blocks. For given  $i$  and  $j$  blocks, the first step of the energy evaluation is to produce the approximate integrals ( $ia|jb$ ) according to Eq. 1.4. It is implemented straightforwardly by reading in blocks of transformed three-center integrals corresponding to the  $i$  and  $j$  ranges required and multiplying them in parallel with *ga\_dgemm* in a step costing  $O(N^5)$ . Given the approximate ( $ia|jb$ ), the remaining operations (formation of ( $ia|jb$ ) – ( $ib|ja$ ), application of denominators, and the evaluation of the actual energy contributions) are carried out almost entirely in data parallel fashion – each process working with the portion of the data it “owns”. As in the exact MP2, these remaining operations cost  $O(N^4)$ .

The RI-MP2 method illustrates a different use of the GA toolkit than the SCF algorithm described above. The RI-MP2 integral transformation uses many of the same concepts as the Fock build, but in this case constitutes a small portion of the computational effort. The dominant cost in the RI-MP2 calculation is a simple call to the GA matrix multiplication routine. And the remainder of the calculation involves mostly data parallel operations implemented variously with standard GA calls, as adaptations of standard GA routines specific to this application, or built from the lower-level utility routines provided by the GA toolkit.

Figure 1.5 shows the parallel speedup of a large RI-MP2 calculation on an IBM RS/6000 SP parallel computer (120 MHz Power2 Super CPU, 512 MB RAM, 5 GB local scratch disk per node)[27]. The calculations were part of a study of the relative energetics of the four conformations of tetramethoxycalix[4]arene (Fig. 1.6)[29], in which this 68 atom molecule was treated with a modified aug-cc-pVTZ AO basis (just cc-pVTZ on the hydrogens) and the corresponding aug-cc-pVTZ-fit2-1 (cc-pVTZ-fit2-1 on H) fitting basis (2460 AO basis functions, 8260 fitting functions) [30, 31]. The total wall clock time for the RI-MP2 calculation ranged from 55.6 hours on 16 nodes to 4.7 hours on 128 nodes. The overall scaling is quite good – the line is fairly straight, and at 128 nodes show no sign of saturation. The jumps in the curve are clearly associated with jumps in the integral transformation speedup. The overall speedup is uniformly at or above the “ideal” linear speedup line, primarily due to the fact that as the graph is presented, the 16-node calculation is implicitly assumed to be 100% efficient. If the actual efficiency (<100%) at 16 nodes were known, it would shift the entire curve downwards. The apparently extraordinary speedup of the transformation arises from the fact that 16 nodes (the reference point) the algorithm is forced to make five passes through the integrals to complete the transformation. As more nodes are added, the algorithm uses the additional memory as well as CPU, so that the number of passes required drops to one by 66 nodes.

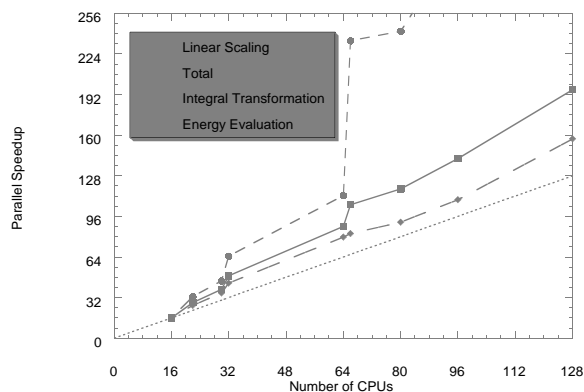


Figure 1.5: Parallel speedup of RI-MP2 calculations on tetramethoxycalix[4]arene on the IBM RS/6000 SP computer[27]. All speedups are referenced to the 16-node timings.

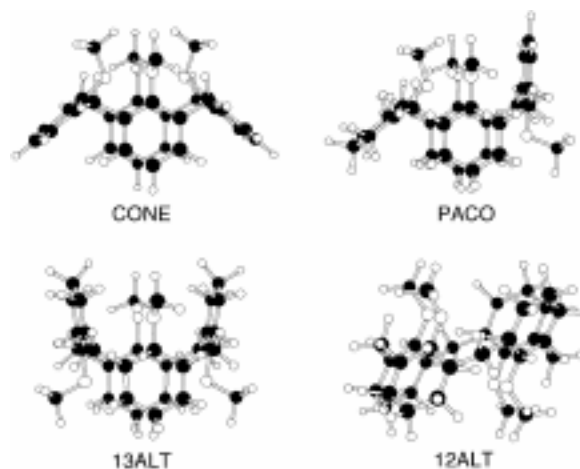


Figure 1.6: The four conformations of tetramethoxycalix[4]arene[29]. The molecule is composed of four anisoles linked at the meta position by methylene bridges and conformations differ in the relative orientation of the anisoles.

## 1.6 Summary and Futures

I have presented an overview of NWChem as an example of the state of the art in fully-distributed parallel computational chemistry software package. The Global Array programming model is at the heart of almost all of the parallel algorithms in NWChem, and parallel linear algebra libraries such as PeIGS have also proven extremely important both for ease of development and performance. I have sketched the parallel algorithms behind two chemistry methods in NWChem, SCF and RI-MP2, which illustrate the importance of the GA programming model as well as its flexibility. Both methods have been demonstrated to be scalable to hundreds of processors, and work efficiently on distributed memory parallel systems, as have the other methods implemented in NWChem.

The development of NWChem continues in conjunction with a variety of projects. Most of the work currently centers on extending and enhancing chemistry methods already in NWChem, and implementing new methods based on the needs of the user community. While the requirements of the chemistry have always been the primary driver for the development of NWChem's computational infrastructure, it is possible to suggest some of the ways that NWChem *might* change in the near future, from a computational viewpoint:

- Increasing use of scripting languages at the top levels of the package. The object oriented scripting language Python[32, 33] is already incorporated into NWChem, so that Python scripts can be used to drive some calculations. An interface to the GAs has been created, and interfaces to other NWChem modules are under development. The use of scripting languages as (part of) the high-level control structure of a package like NWChem makes it easier for users to perform more complex calculations that would otherwise require unmaintainable “one-off” modifications to the source of NWChem itself.
- With the recent release of version 3.0 of the GA Toolkit, general multidimensional arrays became available (previously, GA supported only two-dimensional arrays). Because they are new, they have not yet been used extensively in NWChem chemistry modules. However they promise to be particularly useful in high-level correlated methods (perturbation theory and coupled cluster methods especially) where the primary data structures are tensors of rank 4 and 6. Expressing these data structures in their natural multidimensional form offers opportunities for the introduction of block-structured sparsity and automatic rearrangement of data to make tensor contractions more efficient.
- The current trend in large MPPs is a distributed memory system composed of multiprocessor shared memory nodes. While GAs can already take advantage of this type of system, the parallel algorithms in NWChem are not currently designed with explicit consideration of this new layer in the NUMA

hierarchy – they assume that all memory not “local” is essentially equally “remote”. One can imagine several different ways in which algorithms in NWChem might be adapted to incorporate this deeper memory hierarchy. It will be interesting to see which are most effective in terms of both performance and ease of development.

## 1.7 Acknowledgments

NWChem has been the work of more than forty people since 1993[1], under the leadership of the High Performance Computational Chemistry Group at the Pacific Northwest National Laboratory. I gratefully acknowledge their contributions to the experience described in this paper. All opinions expressed in this paper are the opinions of the author alone, and do not necessarily represent those of other NWChem developers.

The Pacific Northwest National Laboratory is a multiprogram laboratory operated by the Battelle Memorial Institute for the U.S. Department of Energy (DOE) under Contract DE-AC06-76RLO-1830, and the development of NWChem has been supported by the DOE’s Office of Scientific Computing and Office of Health and Environmental Research. Work at Syracuse has also been supported by the Alex G. Nason Prize Fellowship.

Finally, I am grateful to Rick Kendall his assistance with parts of this presentation.

# Bibliography

- [1] J. Anchell, E. Apra, D. Bernholdt, P. Borowski, E. Bylaska, T. Clark, D. Clerc, H. Dachsel, W. de Jong, M. Deegan, M. Dupuis, K. Dyall, D. Elwood, G. Fann, H. Früchtl, E. Glendenning, M. Gutowski, R. Harrison, A. Hess, J. Jaffe, B. Johnson, R. Kendall, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, J. Nichols, J. Nieplocha, A. Rendell, M. Rosing, G. Sandrone, M. Stave, T. Straatsma, H. Taylor, G. Thomas, J. van Lenthe, T. Windus, K. Wolinski, A. Wong, and Z. Zhang. *NWChem, A Computational Chemistry Package for Parallel Computers, Version 3.3.1*. Pacific Northwest National Laboratory, Richland, Washington 99325-0999 USA, 1999.
- [2] Pacific Northwest National Laboratory Environmental Molecular Sciences Laboratory. NWChem homepage. <http://www.emsl.pnl.gov:2080/docs/nwchem/>.
- [3] Ricky A. Kendall, Edo Aprà, David E Bernholdt, Eric J. Bylaska, Michel Dupuis, George I. Fann, Robert J. Harrison, Jialin Ju, Jeffrey A. Nichols, Jarek Nieplocha, T. P. Straatsma, Theresa L. Windus, and Adrian T. Wong. High performance computational chemistry; overview of NWChem a distributed parallel application. *Computer Phys. Comm.*, in press.
- [4] M.F.Guest, E.Apra, D.E.Bernholdt, H.A.Frucht, R.J.Harrison, R.A.Kendall, R.A.Kutteh, X.Long, J.B.Nicholas, J.A.Nichols, H.L.Taylor, A.T.Wong, G.I.Fann, R.J.Littlefield, and J.Nieplocha. High-performance computing in chemistry; NWChem. *Future Generation Computer Systems*, 12(4):273–289, December 1996.
- [5] M. F. Guest, E. Aprà, D. E. Bernholdt, H. A. Früchtl, R. J. Harrison, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. Advances in parallel distributed data software; computational chemistry and NWChem. In *Applied Parallel Computing. Computations in Physics, Chemistry and Engineering Science*, volume 1041 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 1996.
- [6] D. E. Bernholdt, E. Aprà, H. A. Früchtl, M. F. Guest, R. J. Harrison, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. Parallel computational chemistry made easier: The development of NWChem. *Int. J. Quantum Chemistry: Quantum Chem. Symposium*, 29:475–483, 1995.
- [7] M. F. Guest, E. Aprà, D. E. Bernholdt, H. A. Früchtl, R. J. Harrison, R. A. Kendall, R. A. Kutteh, X. Long, J. B. Nicholas, J. A. Nichols, H. L. Taylor, A. T. Wong, G. I. Fann, R. J. Littlefield, and J. Nieplocha. High performance computational chemistry: NWChem and fully distributed parallel algorithms. In *High Performance Computing: Technology, Methods, and Applications*, volume 10 of *Advances in Parallel Computing*, pages 395–427. Elsevier, Amsterdam, 1995.
- [8] M. F. Guest, E. Aprà, D. E. Bernholdt, H. A. Früchtl, R. J. Harrison, R. A. Kendall, R. A. Kutteh, J. B. Nicholas, J. A. Nichols, M. S. Stave, A. T. Wong, R. J. Littlefield, and J. Nieplocha. High performance computational chemistry: Towards fully distributed parallel algorithms. In A. M. Tentner, editor, *High Performance Computing 1994: Grand Challenges in Computer Simulation*, pages 511–521, San Diego, 1994. Society for Computer Simulation.
- [9] David E. Bernholdt. Object oriented methods without object oriented languages. In Michael E. Henderson, Christopher R. Anderson, and Stephen L. Lyons, editors, *Object Oriented Methods for Inter-operable*

- Scientific and Engineering Computing*, pages 40–49. Society for Industrial and Applied Mathematics, 1999.
- [10] Extensible Computational Chemistry Environment Basis Set Database, Version 1.0, developed and distributed by the Molecular Science Computing Facility, Environmental Molecular Science Laboratory which is part of the Pacific Northwest Laboratory, P.O. Box 999, Richland, Washington 99352, USA. The database is accessible via the URL <http://www.emsl.pnl.gov:2080/forms/basisform.html>.
- [11] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global arrays: A non-uniform-memory-access programming model for high-performance computers. *J. Supercomputing*, 10(2):169, 1996.
- [12] Global Array Toolkit home page. <http://www.emsl.pnl.gov:2080/docs/global/ga.html>.
- [13] Jaroslaw Nieplocha, Robert J. Harrison, and Richard J. Littlefield. Global arrays: a portable “shared-memory” programming model for distributed memory computers. In *Supercomputing '94*, pages 340–349, Los Alamitos, California, USA, 1994. Institute of Electrical and Electronics Engineers and Association for Computing Machinery, IEEE Computer Society Press.
- [14] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, 1965.
- [15] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pennsylvania, 2nd edition, 1994.
- [16] G. Fann and R. Littlefield. Parallel inverse iteration with reorthogonalization. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 409–413, Philadelphia, 1993. Society for Industrial and Applied Mathematics.
- [17] G. I. Fann, R. J. Littlefield, and D. M. Elwood. Performance of a fully parallel dense real symmetric eigensolver in quantum chemistry applications. In *Proceedings of High Performance Computing '95, Simulation MultiConference*, San Diego, CA, 1995. The Society for Computer Simulation.
- [18] Inderjit Dhillon, George Fann, and Beresford Parlett. Application of a new algorithm for the symmetric eigenproblem to computational chemistry. In Michael Heath, Virginia Torczon, Greg Astfalk, Petter E. Bjørstad, Alan H. Karp, Charles H. Koebel, Vipin Kumar, Robert F. Lucas, Layne T. Watson, and David E. Womble, editors, *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. Society for Industrial and Applied Mathematics, 1997.
- [19] Adrian T. Wong and Robert J. Harrison. Approaches to large-scale parallel self-consistent field calculations. *J. Computat. Chem.*, 16(10):1291–1300, 1995.
- [20] Robert J. Harrison, Martyn F. Guest, Rick A. Kendall, David E. Bernholdt, Adrian T. Wong, Mark Stave, James Anchell, Anthony Hess, Rik Littlefield, George I. Fann, Jarek Nieplocha, Greg S. Thomas, David Elwood, Jeff Tilson, Ron L. Shepard, Albert F. Wagner, Ian T. Foster, Ewing Lusk, and Rick Stevens. High performance computational chemistry. II. A scalable SCF program. *J. Computat. Chem.*, 17:124, 1995.
- [21] Thomas R. Furlani and Harry F. King. Implementation of a parallel direct SCF algorithm on distributed memory computer. *J. Computat. Chem.*, 16(1):91, January 1 1995.
- [22] Martin Feyereisen, George Fitzgerald, and Andrew Komornicki. Use of approximate integrals in ab initio theory. An application in MP2 energy calculations. *Chem. Phys. Lett.*, 208(5,6):359–363, 1993.
- [23] Rick A. Kendall and Herbert A. Früchtl. The impact of the resolution of the identity approximate integral method on modern ab initio algorithm development. *Theoret. Chem. Acct.*, 97(1–4):158–163, 1997.
- [24] O. Vahtras, J. Almlöf, and M. W. Feyereisen. Integral approximations for LCAO-SCF calculations. *Chem. Phys. Lett.*, 213(5,6):514–518, 1993.

- [25] Frank E. Harris, Hendrik J. Monkhorst, and David L. Freeman. *Algebraic and Diagrammatic Methods in Many-Body Theory*. Oxford University Press, New York, 1992.
- [26] David E. Bernholdt and Robert J. Harrison. Large-scale correlated electronic structure calculations: The RI-MP2 method on parallel computers. *Chem. Phys. Lett.*, 250:477–484, 8 March 1996.
- [27] David E. Bernholdt. Scalability of correlated electronic structure calculations on parallel computers: A case study of the ri-mp2 method. *Parallel Comp.*, in press.
- [28] Alistair P. Rendell and Timothy J. Lee. Coupled-cluster theory employing approximate integrals: An approach to avoid the input/output and storage bottlenecks. *J. Chem. Phys.*, 101(1):400–408, 1 July 1994.
- [29] John B. Nicholas, David E. Bernholdt, and Benjamin P. Hay. On the conformational energetics of tetramethoxycalix[4]arene: RI-MP2 benchmark calculations. *J. Am. Chem. Soc.*, submitted.
- [30] Thom H. Dunning, Jr. Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *J. Chem. Phys.*, 90(2):1007–1023, 15 January 1989.
- [31] David E. Bernholdt and Robert J. Harrison. Fitting basis sets for the RI-MP2 approximate second-order many-body perturbation theory method. *J. Chem. Phys.*, 109(5):1593–1600, 1 August 1998.
- [32] Python language website. URL: <http://www.python.org>.
- [33] Mark Lutz. *Programming Python*. O’Reilly and Assoc., 1996.