Computer Physics Communications 128 (2000) 260–283

Computer Physics Communications

www.elsevier.nl/locate/cpc

High performance computational chemistry: An overview of NWChem a distributed parallel application [☆]

Ricky A. Kendall [a,*], Edoardo Aprà [b,1], David E. Bernholdt [c], Eric J. Bylaska [b,1], Michel Dupuis [b,1], George I. Fann [b,1], Robert J. Harrison [b,1], Jialin Ju [b,2], Jeffrey A. Nichols [b,1], Jarek Nieplocha [b,2], T.P. Straatsma [b,1], Theresa L. Windus [b,1], Adrian T. Wong [d]

[a] *Scalable Computing Laboratory, Ames Laboratory, Ames, IA 50011, USA*

[b] *Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, PO BOX 999, Mail Stop K1-83, Richland, WA 99352-0999, USA*

[c] *Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100, USA*

[d] *National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA*

Abstract

NWChem is the software package for computational chemistry on massively parallel computing systems developed by the High Performance Computational Chemistry Group for the Environmental Molecular Sciences Laboratory. The software provides a variety of modules for quantum mechanical and classical mechanical simulation. This article describes the design and some implementation details of the overall NWChem architecture. The architecture facilitates rapid development and portability of fully distributed application modules. We also delineate some of the functionality within NWChem and show performance of a few of the modules within NWChem. © 2000 Elsevier Science B.V. All rights reserved.

PACS: 31.10; 31.15; 31.15.A; 31.15.Q

Keywords: Parallel computational chemistry; Electronic structure; Molecular dynamics; Distributed tools; Portability

1. Introduction

Computer science and the application of computational techniques in basic and applied science is a relatively new field beginning roughly 40 years ago when computing resources became available to the research community. As the hardware technology, operating systems, and the development of languages progressed many applications used on computers evolved to increase the performance on the computational resource at hand. During the transition to vector supercomputers from scalar systems, the restructuring of applications was accomplished over a number of years where the basic paradigm of vector or vector parallel algorithms essentially survived the changes in hardware technology. The overall gains in performance were not simply changes in the underlying computer technology but also in the algorithmic and theoretical developments in the basic and applied science fields.

[☆] This paper is published as part of a thematic issue on Parallel Computing in Chemical Physics.

[*] Corresponding author. E-mail: rickyk@scl.ameslab.gov.

[1] High Performance Computational Chemistry Group.

[2] Advanced Research Software Group.

The last two decades have encapsulated the transition to massively parallel processing (MPP) supercomputers. The synergism between technology and algorithmic changes is arguably greater now than ever before. The complicating factor is the time scale of the changes. Processors double in speed every 18 months; memory density and the performance of communication subsystems of generic MPPs are also increasing at a faster pace than ever before. Because of the man power effort required to develop scientific applications, they evolve at a significantly slower rate. It is not uncommon for an implementation of a specific scientific functionality to survive two or more generations of technology changes without tuning or an update of the algorithms used for that functionality.

This situation has made the computational science community in cooperation with their computer science colleagues fully aware of the need to cooperate and share their respective understanding of hardware and software. This requires a focused effort to build a mechanism for cooperation [1]. Applications are developed not only to utilize current technology but they must also be designed in such a way that expected hardware changes will not adversely effect the application's performance or usefulness. Applications are thus essentially required to be portable, modular, and easily tuned or restructured for adaptation to new computational resources. In this article we present an overview of an application, **N**orth**W**est **Chem**istry (NWChem), designed and implemented for use on MPPs [2–7].

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 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. The situation outlined above regarding the evolution of computational technology required the development of high-level data and control structures that facilitate writing, maintaining, and extending chemistry functionality as well as allowing for rapid prototyping of new functionality. These portable tools provide on demand memory allocation, message passing, distributed arrays in memory and on secondary storage, access to parallel linear algebra methods. Our goal here is to give a sampling of some of the modules within NWChem, defining the capabilities, and show the utility or performance of this sampling.

2. The NWChem umbrella

The NWChem umbrella software includes both the high performance software tools (known as ParSoft) and the chemistry based objects and application programmer interfaces (APIs). The ParSoft tools are used to provide both scalable and serialized services (e.g., memory allocation, global arrays, shared files, message passing, linear algebra, etc.) to the application developer. The NWChem objects and APIs provide computational chemistry specific information and functionality; these are often referred to "domain specific APIs" (DSAPIs). The overall NWChem architecture can be represented by Fig. 1. The application uses high level tasks that are interfaced to various chemistry modules. The NWChem chemistry modules have unique functionality that is also built upon DSAPIs and the ParSoft libraries. In this section we describe key elements of the ParSoft tools and DSAPI software.

2.1. High performance software tools

The effect of the software tools on the performance and development of the NWChem computational chemistry suite cannot be underestimated. Without some of the software tools described in this sec-

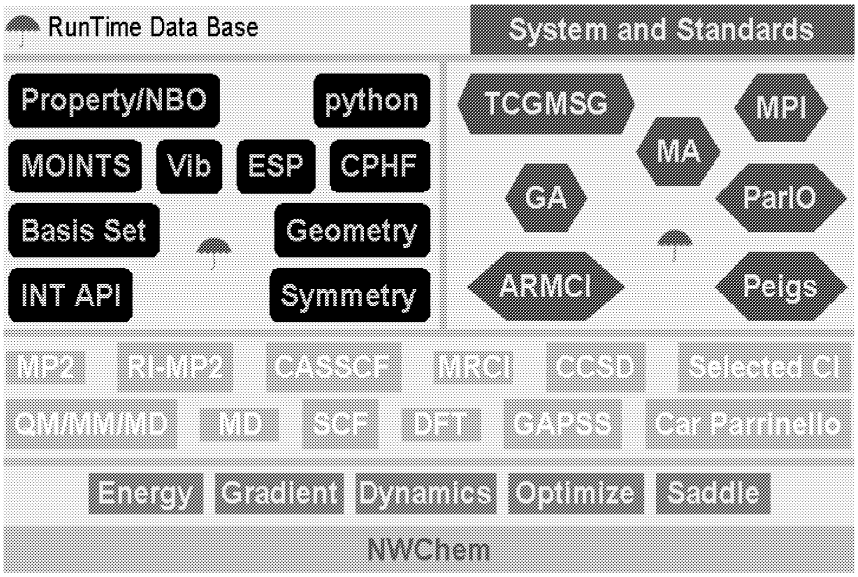


Fig. 1. The NWChem architecture representing general functionality within NWChem which is built upon layers of other modules, tools, chemistry APIs, and computational and computer science standards. The link between NWChem and Ecce is a loosely coupled interface. The umbrella symbol identifies some of the software described in this section of the manuscript.

tion the chemistry programs would be far more complex and difficult to support and maintain. Moreover, NWChem's chemistry modules would be far less flexible to adaptation and tuning for a specific underlying hardware platform. The design and implementation of these tools allow application porting to different platforms without modification of the APIs used in these tools. Here we delineate the important ParSoft tools used in NWChem.

2.1.1. *Global Arrays*

The Global Arrays (GA) programming model [8,9] is predicated on the fact that all computers from workstations to massively parallel supercomputers have multiple levels of memory and that the access is nonuniform. Non-uniform memory access (NUMA) implies that at all levels of the memory hierarchies (e.g., registers, cache, local memory, virtual memory, remote memory, etc.) have different bandwidth and latency characteristics. Workstation optimization techniques can be used to mitigate the single node performance of code (i.e. maximizing cache utilization). NUMA with respect to parallel algorithms and computers adds at least the remote memory layer to the programming architecture.

In general, there are many tradeoffs at multiple levels that must be made between portability, efficiency, and ease of application development. The message passing programming model is widely used because it completely portable using the MPI standard or any other message passing library that is widely available. There are some applications that have data access patterns that are not amenable to message passing because of the complexity involved (e.g., computational chemistry[3] and hydrodynamic Tokamak simulation). Tools like GA are one way of mitigating this complexity by providing distributed data mechanisms via a simple library interface. It is important to stress that the GA programming model is designed to be complimentary to message passing and not a total replacement. The two mechanisms can and do coexist in an application without any instability or incompatibility.

At the simplest level the programming model offered using GA assumes that all "global memory" ac-

[3] Some parts of computational chemistry can be coded using strictly message passing (e.g., SCF and DFT) and be somewhat efficient. However, the quadradicly convergent SCF is complex enough to benefit from the distributed data abstractions used in NWChem.

cess 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.

The parallelism using GAs is expressed and accessed using distributed data objects in "global" memory in conjunction with local memory and local operations. The distributed fragments of the global data object are stored "locally" on each of the working processors. GA has mechanisms that report the local portion of the distributed data and provide access to "remote" portions of data via "put" or "get" operations as well as "accumulate". By knowing the locality of data, programmers can explicitly manage the nature of the memory hierarchy for their parallel algorithm. GA operations differ in that they have different implied synchronicity and atomicity. Some are collective and others non-collective.

The basic GA operations include:
- initialize: with or without limits on the size of global memory,
- create: with a regular or irregular distribution,
- duplicate: where the new GA array inherits the distribution, data type, and dimensions of the existing array,
- destroy: remove a single GA,
- terminate: shutdown the GA interface.

The one-sided or asynchronous operations include:
- remote block-wise read or write (get or put),
- remote atomic update (accumulate or read and increment of integers),
- remote element read or write (gather or scatter).

Interprocess Synchronization operations include:
  – Lock with mutex:[4] one can create and lock a mutex to exclusively access a critical section of execution,
- Fence: guarantee that the GA operation(s) issued from the calling process are complete (the fence is "local" to the calling process),
- sync: the traditional collective barrier operation.

Collective GA operations include:
- basic array operations on an entire array or patch of an array
  – zero, fill with a value, or scale,
  – print or copy;
- linear algebra operations
  – scale and add GAs and patches,
  – matrix multiplication of GAs and patches (dgemm style),
  – dot product of GAs and patches,
  – symmetrize a GA;
- interfaces to third party packages
  – Scalapack linear equation solvers,
  – PeIGS eigensystem solvers,
  – Interfaces to other libraries are straightforward to build; this requires the introduction of routines to map GAs to the requisite data structures of the library in question.

There are also utility routines that map global data (e.g., patches of a GA) to the processor(s) that "owns" the data, determines the portion of a GA stored locally, to provide access to and to release the GA data stored locally. Other utility routines report the lexical process index or "node id", the number of participating nodes, the virtual processor grid topology, the amount of GA memory available, and wrappers to normal broadcast and reduction operations that are inherently part of a message passing system. Finally, there are utility routines to determine the dimensionality, object name, and to print the accumulated statistics of GA operations performed in the application.

GA 3.0 is the current released version and when compared to the previous release it offers:
- multidimensional (1–7) array interface as supposed to 2-dim interface in earlier versions,[5]
- improved interoperability with MPI on clusters of workstations (communicator splitting not used),

[4] The term "mutex" implies mutual exclusion so that only a single process or thread executes a section of code at any given time.

[5] The limit of 7 is the same as the FORTRAN array dimension limit of the relevant standards.

- additional ports: Windows NT (WIN32 and Cygnus) and Cray J90,
- includes Disk Resident Array (DRA) parallel I/O library (developed in collaboration with ANL),
- this new version is implemented on top of aggregate remote memory copy interface (ARMCI) run-time library.[6]

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 (c.f., Section 2.1.2),
- 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.

2.1.2. *Parallel I/O*

The parallel I/O (input and output) library (ParIO), based on the original ChemIO library,[7] is a high-performance parallel I/O abstraction layer designed specifically for computational chemistry applications but is applicable to many parallel algorithms. The development of out-of-core methods for complex parallel applications requires an efficient and portable API for complex I/O patterns. The ParIO library addresses this problem by providing implementations that hide some of the complexity of the underlying computational I/O subsystem from the application developer via high-level functionality. The interface is tailored to the requirements large-scale computational chemistry problems and supports three distinct I/O models. These are:

(1) *Disk Resident Arrays* (DRA) – for explicit transfer between global memory (e.g., a global array) and secondary storage, allowing the programmer to manage the movement of array data structures between local memory, remote memory, and disk storage. This component supports collective I/O operations, in which multiple processors cooperate in a read or write operation and thereby enable certain useful optimizations.
(2) *Exclusive Access Files* (EAF) – for independent I/O (i.e. non-collective) to and from scratch files maintained on a per-processor basis. It is used for out-of-core computations in application modules that cannot easily be organized to perform collective I/O operations.
(3) *Shared Files* (SF) – for creation of a scratch file(s) that can be shared by all processors. Each processor can perform non-collective read or write operations to an arbitrary location in the file.

Each of these three ParIO library modules (DRA, EAF, and SF) are fully independent and can all be used in any given application. The ParIO library modules are layered on a "device library", The Elementary I/O library (ELIO), which provides a portable interface to different file systems. The ELIO can take advantage of any system specific performance libraries available.

2.1.3. *Memory Allocator*

The Memory Allocator (MA) is used to allocate data that will generally not be directly shared with other processes, such as workspace for a particular local calculation or for replication of very small sets of data. MA is designed to be portable across a large variety of platforms. The MA tool is a library of routines that comprises a dynamic memory allocator for

[6] The ARMCI library logically separates the remote memory copy "engine" from the rest of GA making both more portable.

[7] The ChemIO project was a joint effort of Argonne National Laboratory and Pacific Northwest National Laboratory, in affiliation with a DOE Grand Challenge project developing Massively Parallel Methods for Computational Chemistry, with the multi-agency Scalable I/O Project, and with the EMSL.

use by C, FORTRAN, or mixed-language applications.

It provides both heap and stack memory management disciplines, debugging and verification support (for detecting memory leaks, for example), usage statistics, and quantitative memory availability information. Applications written in FORTRAN require this sort of library because the language (i.e. the F77 standard) does not support dynamic memory allocation. Applications written in C can benefit from using MA instead of the ordinary `malloc()` and `free()` routines because of the above mentioned extra features that MA provides.

For consolidated memory access and to minimize the interaction with the operating system (OS), a segment of memory is obtained from the OS upon initialization. The size of the segment obtained is controllable by the NWChem user. The low end of the segment is managed as a heap. The heap region grows from low addresses to high addresses. The high end of the segment is managed as a stack. The stack region grows from high addresses to low addresses.

Each region consists of a series of contiguous blocks, one per allocation request, and possibly some unused space. Blocks in the heap region are either in use by the client (allocated and not yet deallocated) or not in use by the client (allocated and already deallocated). A block on the rightmost end of the heap region becomes part of the unused space upon deallocation. Blocks in the stack region are always in use by the client, because when a stack block is deallocated, it becomes part of the unused space.

A block consists of the client space, i.e. the range of memory available for use by the application. Guard words adjacent to each end of the client space to help detect improper memory access by the client. Bookkeeping information is stored in an "allocation descriptor" AD. Two gaps, each zero or more bytes long, are defined to satisfy alignment constraints (specifically, to ensure that AD and client space are aligned properly). Recent modifications to the library allow different alignment based on each request. This is not only important for computational performance but also when the allocated memory buffers are used for secondary storage access [10].

2.1.4. 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. λ 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" (cf. Refs. [11,12]) with the exception of the subspace inverse iteration and reorthogonalization scheme for finding basis vectors for degenerate eigen-subspaces [13,14] and the Dhillon–Fann–Parlett algorithm for computing eigenvectors of a real symmetric tridiagonal matrix [15].

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.

The performance of PeIGS in sequential mode is impressive. The data in Table 1 compares the current version of PeIGS with other standard solvers.

Table 1
Time for the solution of the tridiagonal matrix of rank 966 on a single IBM RS6000/590 processor [15]. 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 flourinated biphenyl

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

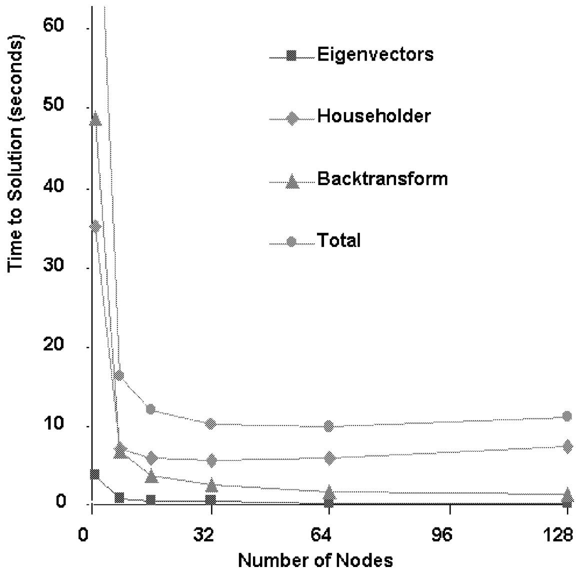


Fig. 2. The performance of PeIGS using a tridiagonal matrix (rank 966) which was generated via Householder reduction of the fitting basis set, overlap matrix from an RI-MP2 simulation of a flourinated biphenyl.

The parallel performance of the three major components and the total time to solution is shown in Fig. 2. The solution of the tridiagonal problem is scalable and fast; however at this point, the householder reduction and it's 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 eigen-solution operations.

2.1.5. Run time data base

The run time data base (RTDB) is the parameter and information repository (cf. Fig. 1) for the independent modules (e.g., SCF, RI-MP2) comprising NWChem. This approach is similar in spirit to the GAMESS dumpfile or the Gaussian checkpoint file. The only way modules can share data is via the RTDB or using files whose names are also stored in the database. Information is stored directly in the database as typed arrays, each of which is described by

(1) a name, which is a simple string of ASCII characters (e.g., `"geometry"`),
(2) the type of the data (real, integer, logical, or character),
(3) the number of data items, and
(4) the actual data (an array of items of the specified type).

A database is simply a file and is opened by name. Usually there is just one database per calculation, though multiple databases may be open at any instant. By default, access to all open databases occur in parallel, meaning that

- all processes must participate in any read/write of the database and any such operation has an implied synchronization,
- writes to the database write the data associated with process zero but the correct status of the operation is returned to all processes,
- reads from the database read the data named by process zero and broadcast the data to all processes, checking dimensions and types of provided arrays.

Alternatively, database operations can occur sequentially. This means that only process zero can read/write the database, and this happens with no communication or synchronization with other processes. Any read/write operations by any process other than process zero is not allowed. Usually, all processes will want the same data at the same time from the database, and all processes will want to know of the success or failure of operations. This is readily done in the default parallel mode. One major exception to this is during the input parsing by the NWChem input module. Usually, only process zero will read the input and needs to store the data directly into the database without involving the other processes. This is done using sequential mode.

This abstraction of the central data and information store offers a powerful mechanism for the construction of complex operations using a simple driver routine that sequentially sets the state of the RTDB for the appropriate calculation. For example, a numerical gradient driver need only set the appropriate geometry "states" in the RTDB, calculate and read the energy of each "state" and produce the gradient.

2.2. NWChem objects and domain specific APIs

The NWChem objects and domain specific APIs provide generic chemistry related data and functionality to several or all modules. The abstractions of this functionality is at the heart of the modularity of NWChem and allows for an essential prototyping environment. Building upon other modules and the umbrella software infrastructure it is straightforward to construct almost any chemistry module. The caveat being if the infrastructure does not have the appropriate functionality for the module you want to develop (e.g., fourth derivative integrals).

2.2.1. Integral API

The integral (INT) Application Programmer's Interface (API) is the interface to the base integral technology available in the NWChem application software. The INT-API interfaces currently five integral codes

- the sp rotated axis code,
- the PNNL implementation of the McMurchie–Davidson code,
- the effective core potential and spin orbit potential integrals,
- the Texas 93/95 Integral code, and
- the HONDO integral code.

The API is currently limited to the requisite functionality of NWChem. Further functionality and new integral technology are straightforward to add as as requirements are determined, prioritized and implemented.

One component of the API that provides flexibility in developing applications, is that it is structured so that integrals over different basis sets can be computed. For example, the three center overlap integral used in the resolution of identity methods and the density functional code:

$$(\mu\nu\lambda) = \int_{-\infty}^{\infty} g_\mu(X_\mu, r_1) g_\nu(X_\nu, r_1) g_\lambda(X_\lambda, r_1)\,\mathrm{d}r_1$$

is rarely computed using the same basis set. Typically one function is from the "fitting basis" the other functions are from the traditional atomic orbital basis. This exists for all integral and integral derivative API functionality.

The extremely important facet of the INT-API, a property of most NWChem DSAPIs, is that integration of new integral technology does not interfere with the performance of existing NWChem modules or those currently being developed. For example, if new algorithms are developed by the computational quantum chemistry community which leads to integral code that is 10 times faster than current technology, incorporation of this new technology under the INT-API allows **all** NWChem modules that use the INT-API to immediately benefit from the increased performance, cf. Fig. 3.

The integral code operates as a single threaded suite and all parallelization is achieved at the level of the modules that use the API, i.e. the integral codes are **not** parallel. The API requires a collective initialization phase to determine operating parameters for the particular simulation based on both user input and the basis set(s) specification (this includes effective core potential and spin orbit potential specification). The API will select the appropriate base integral code for the requested integrals at the time of each request. This means that any given simulation might use a few or all of the base integral codes. The API is responsible for structuring the interface to the base integral codes so that the accuracy and precision of the resultant integrals is uniform. Once all integral computations have completed for the module the termination or shutdown of the instance of integral computation is done also in a collective fashion.

The INT-API has the following kinds of routines:

- initialization, integral accuracy and termination,
- memory requirements,
- integral routines (both shell quartet based and blocked[8]),
- derivative integral routines,
- property integral routines,
- periodic integral routines,
- Internal API Routines.

2.2.2. Geometry and basis set objects

The geometry object is a well defined, extensible API that provides all the geometrical and atomic data (e.g., masses, atomic number, nuclear charges, applied external field, coordinates, etc.) functionality for NWChem. All quantum chemistry application interoperate with the geometry object via it's API. Geometries can be specified in Cartesian or zmatrix format with or

[8] Blocks are groups of shell quartets.

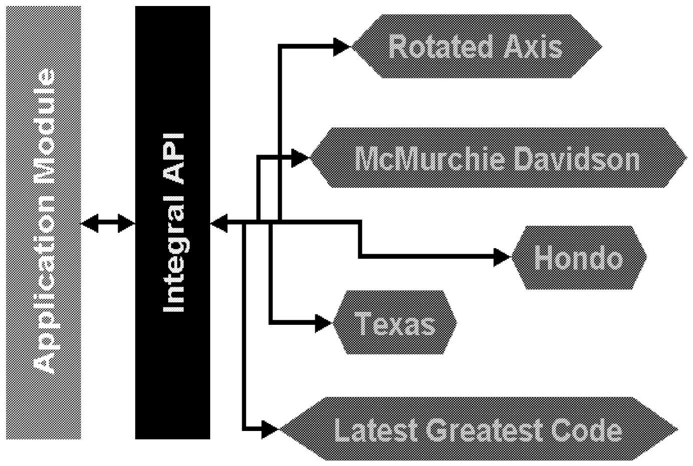


Fig. 3. The logical structure of the INT-API that centralizes the integral functionality within NWChem.

without symmetry. The geometry object will also automatically detect symmetry and generate a set of internal redundant coordinates. The user has control over some of these options and the tolerances for determining symmetry, etc. However, the default mechanisms should work for most molecular systems of interest.

The basis set object is also a well defined, extensible API that provides all the basis set functionality for NWChem. All applications that use basis sets interoperate with this object. 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 synched[9] with the EMSL basis set library [16] which is available to the public via a WWW interface [17]. Currently the NWChem library has 2043 Gaussian basis sets and 462 effective core potentials conveniently specified for the user community.

The basis set object stores only unique information for a given basis set so that the data for any given center is stored only once. The data structures and interface was designed to represent a basis set via a pointer or handle mechanism allowing for any number of basis sets to be used by an application module (cf. Section 2.2.1). The basis set object requires a geometry specification to determine the mappings between all centers and functions for the application. The data structures employed are completely dynamic (i.e. obtained from heap MA memory) and thus scale with molecular system size. The combination of the basis set and geometry objects contain all the non-state-specific information for an *ab initio* simulation.

The basis set object code was reworked with the addition of about one hundred lines of code to also function for effective core potentials (ECP) and spin orbit (SO) potentials. The reuse of the code in this fashion allowed developers the use of the same, familiar API when developing applications for ECP and SO functionality.

2.2.3. Symmetry

The symmetry API is a specialized tool that is used by many applications and other DSAPIs in NWChem. It generates character tables and operation matrices for many molecular point groups and for periodic space groups. The latter is only utilized in the periodic density function theory code [18]. The Symmetry API is augmented to symmetrize general gradient and Hessian matrices from skeleton representations and this is done at machine precision. There are also routines that report the symmetry scaling factors (e.g., q2 and q4) and routines to compute which atom pairs or atom quartets are the correct lexical set for computing the skeleton representation of matrices at both the atomic and shell levels. There are routines to force the geometric coordinates to be within the given point group symmetry. Other routines determine the number of functions per irreducible representation of the symmetry group for a given basis set, symmetry

[9] This happens usually whenever there is a major NWChem release, e.g., a new $X.Y$ release.

adapt a given set of molecular orbitals to a given threshold, and apply a symmetry operation to a given molecular orbital. Overall, this provides a robust symmetry environment that can be applied at multiple levels of an application.

2.2.4. *Property module*

Calculation of properties is accomplished by invoking the property module after the completion of an energy (or MP2 gradient) calculation. Properties can be computed for all wave functions that produce orbitals or natural orbitals, including Hartree–Fock (closed-shell RHF, open-shell ROHF, and open-shell UHF), DFT (closed-shell and open-shell spin unrestricted), MCSCF (complete active space), and MP2 (closed-shell RHF and open-shell UHF). The properties that can be computed are:

- dipole moment,
- quadrupole moment,
- octupole moment,
- Mulliken population analysis and bond order analysis,
- electrostatic potential (diamagnetic shielding) at nuclei,
- electric field at nuclei,
- electric field gradient at nuclei,
- electron and spin density at nuclei,
- NMR chemical shifts (GIAO method) for closed-shell RHF,
- Natural Bond Orbital Analysis.

There are two ways to carry out the Natural Bond Orbital (NBO) analysis. The properties module can generate an input deck for NBO4.0 [19]. A separate task is available for version 5.0 of the Natural Bond Orbital (NBO) code of Weinhold and coworkers [20, 21] has been integrated with NWChem. Currently, the module does not ship with the release code until a user confirms a license for NBO. [10] The traditional NBO input is preserved in normal NWChem block format.

2.2.5. *Python*

Python is an interactive, interpreted, object-oriented programming language [22]. It is often compared to other scripting languages such as Tcl, Perl, Scheme or Java. It is a powerful language with a clear syntax. It has modules, classes, exceptions, and high level dynamic data types. Like Perl and other scripting languages there are interfaces to many system calls and libraries, as well as to various windowing systems. Python is can be used as an extension language providing a programmable interface for applications; this is the goal of the incorporation of Python into NWChem. The Python implementation is portable: it runs on many brands of UNIX, on Windows, DOS, OS/2, Mac, etc. One significant reason for our choice of Python is that it is **copyrighted** but *freely usable and distributable*.

Python [11] programs may be embedded into the NWChem input and used to control the execution of NWChem. Python is a very powerful and widely used scripting language that provides useful things such as variables, conditional branches and loops, and is also readily extended. Example applications include scanning potential energy surfaces, computing properties in a variety of basis sets, optimizing the energy with respect to parameters in the basis set, computing polarizabilities with finite field, simple molecular dynamics, and direct computation of basis set superposition error corrected energies and potential energy surfaces.

The current NWChem implementation includes a python module "NWChem" which is automatically imported and contains NWChem specific functionality. This includes access to high level task processing routines (task_energy, task_gradient, task_optimize) which are exactly analogous to traditional NWChem tasks. The input parser is exported to the Python interface so the Python programs can generate any set of NWChem input directives. The NWChem–Python interface also has complete access to the run time data base which allows python programs to read and write RTDB entries directly. Future enhancements will include Python wrappers to the GA functionality. This provides a rather robust input parsing and task delegation mechanism to NWChem allowing NWChem users complete flexibility in constructing molecular simulations.

Since we have little experience using Python, the NWChem–Python interface might change in a non-

[10] Future releases will include this module after finalization of an agreement between PNNL and the Theoretical Chemistry Institute at the University of Wisconsin, Madison.

[11] The current tested Python versions in NWChem are 1.5.1 and 1.5.2.

backwardly compatible fashion as we discover better ways of providing useful functionality. We would appreciate suggestions about useful things that can be added to the NWChem–Python interface. In principle, nearly any Fortran or C routine within NWChem can be extended to Python, but we are also interested in ideas that will enable users to build completely new things. One example could be the ability to define your own energy functions that can be used with the existing optimizers or dynamics package.

3. NWChem modules

NWChem provides many methods to compute the properties of molecular and periodic systems by using standard quantum mechanical descriptions of the electronic wave function or density. In addition, NWChem has the capability to perform classical molecular dynamics and free energy simulations. These approaches may be combined to perform mixed quantum-mechanics and molecular-mechanics simulations.

NWChem is available on almost all high performance computing platforms, workstations, personal computers running LINUX, as well as clusters of desktop platforms or work-group servers. NWChem development has been devoted to providing maximum efficiency on massively parallel processors. It achieves this performance on the 512 node IBM SP system in the EMSL's MSCF and on the 512 node CRAY T3E-900 system in the National Energy Research Scientific Computing Center. It has not been optimized for high performance on single processor desktop systems.

In this section we start with a summary of the capabilities of NWCHEM and then highlight some of the specific modules available.

(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.

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, automatic interfaces are provided to:
- The COLUMBUS multireference CI package.
- The natural bond orbital (NBO) package.
- Python.
- PolyRate.

(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–Parrinello (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.

(5) Parallel tools and libraries (ParSoft) as outlined above.

3.1. NWChem task APIs

The task API is a high level "air traffic control" system for NWChem. Task API is responsible for starting generalized tasks and special pre-defined tasks. The generalized tasks include:

- task_energy: compute an energy,
- task_gradient: compute a gradient,
- task_hessian: compute a Hessian,
- task_optimize: optimize to a geometrical minimum,
- task_saddle: optimize to a geometrical transition state,
- task_dynamics: perform a molecular dynamics simulation.

Some of the special predefined tasks include:

- task_property: compute static one electron properties,
- task_frequencies: compute frequencies from a nuclear Hessian,
- task_python: start the python interface,
- task_drdy: compute gradients and Hessians along a reaction path and generate one of the PolyRate input files.

The task API is somewhat generic in the sense that the "state" of the RTDB determines exactly what is done in most of the tasks outlined above. To follow the air traffic control analogy the controller doesn't need to know where planes are going and coming from he/she simply has to coordinate their use of the limited runway and gate resources. The general tasks do the same thing. The pilot is responsible for pointing the plane in the right direction and the general and specific tasks have a similar role. For example, task_gradient is responsible for knowing if a particular theory has analytical gradient capabilities and calls that functionality for that theory. Alternatively the numerical gradient code is called. The generic numerical gradient code knows nothing of the specifics of the theory being employed it assumes that task_energy will compute the appropriate values and store them on the RTDB after completion.

This is a very powerful mechanism for building up complex simulations especially in conjunction with a user definable interface such as python (cf. Section 2.2.5). It also allows other standard modules to hide the complexity of computing energies, gradients, and Hessians from say an optimization code. If one were to develop a new walking algorithm it would be straight forward to test the viability of this within the NWChem development framework. Fig. 4 gives a pictorial representation of what is needed to do just this. The interoperability of each of the task modules is displayed as well.

3.2. Self consistent field

The self consistent field (SCF) 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 specifics of the algorithm are given elsewhere [23,24].

The module employs a parallel semi-direct algorithm allowing flexible application execution on a variety of computational resources. The convergence algorithm is the quadratic SCF [25] with both preconditioning and line search mechanisms built in. The user has some control over both the parallel allocation of resources and the kind of convergence needed. A significant amount of work has gone into setting default tolerances and default algorithmic choices for many different kinds of molecular systems. [12] The semi-direct algorithm also caches integrals not only on disk but in available memory. The default mechanism uses any extra memory first and then fills local disk storage (or the limit set by the user). Any remaining integrals are computed in a direct fashion as needed.

In addition to the semi-direct algorithm which can be made to compute the SCF energy and wave function in a complete conventional or direct fashion there is also a replicated data algorithm for the Fock build. When the molecular system is small enough to fit a replicated Fock matrix into local node memory and the latency and bandwidth characteristics of the parallel supercomputer or cluster are appropriate the replicated data algorithm can be employed.

The NWChem SCF user has control over the normal properties of any SCF calculation such as convergence criteria, utilization of symmetry, state symmetry of the molecule, the initial guess for starting orbitals, swapping of specific input orbitals, etc. Molecular orbitals are saved every iteration if more than 600 seconds have elapsed allowing a restart mechanism. At completion these orbitals are always canonically transformed.

The initial guess algorithms are rather robust allowing the user to easily determine the required state of the molecule. The initial guess choices are:
- superposition of atomic SCF densities (the default),
- previously converged orbitals from a different geometry,
- projection of orbitals from a converged result using a different basis set,
- superposition of fragment SCF densities,
- eigenvectors of the one-electron Hamiltonian (if all others fail).

Fig. 5 represents 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 with 1342 basis functions was completed in 5.7 hours on 240 nodes (160 MHz).

3.3. Multiconfiguration self consistent field

The NWChem multiconfiguration SCF (MCSCF) module is a complete active space SCF (CASSCF). The module is capable of doing simulations with at most 20 active orbitals and $\sim$500 basis functions. It will be extended at some point to handle over 1000 basis functions. Only Abelian point group symmetries are currently supported and energies and analytic gradients are available. The integral transformation used is either fully direct or disk resident; there is currently no semi-direct algorithm for the entire CASSCF. The module does reuse the SCF Fock build with all of it's available options (direct, semi-direct, and conventional). The output orbitals are cononicalized as follows:
- Doubly occupied and unoccupied orbitals diagonalize the corresponding blocks of an effective Fock operator.
- Active-space orbitals are chosen as natural orbitals by diagonalization of the active space 1-particle density matrix.

Fig. 6 shows the speed up and efficiency of the CASSCF module for the di-allyl (C_6H_{10}) molecular system in the cc-pVTZ basis set with 360 basis functions. The calculation computes the doublet state with 7 electrons in 12 active orbitals. The module

[12] It is imperative that users report convergence problems to nwchem-support@emsl.pnl.gov so that any inconsistencies can be repaired.

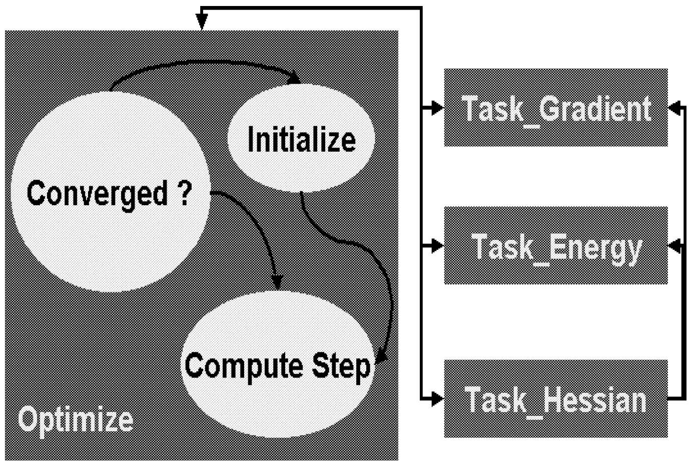
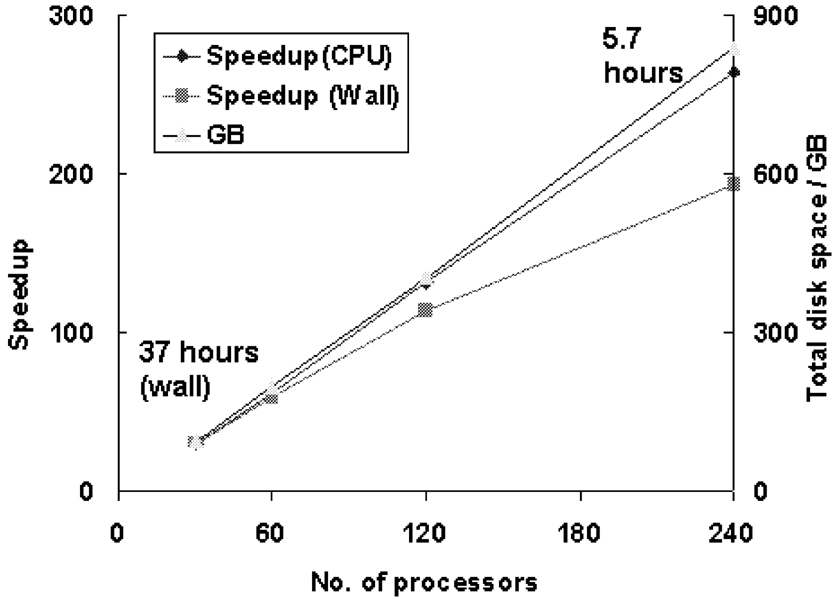


Fig. 4. The interoperability of the task modules for doing a geometry optimization.



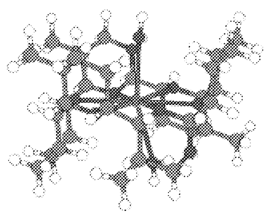
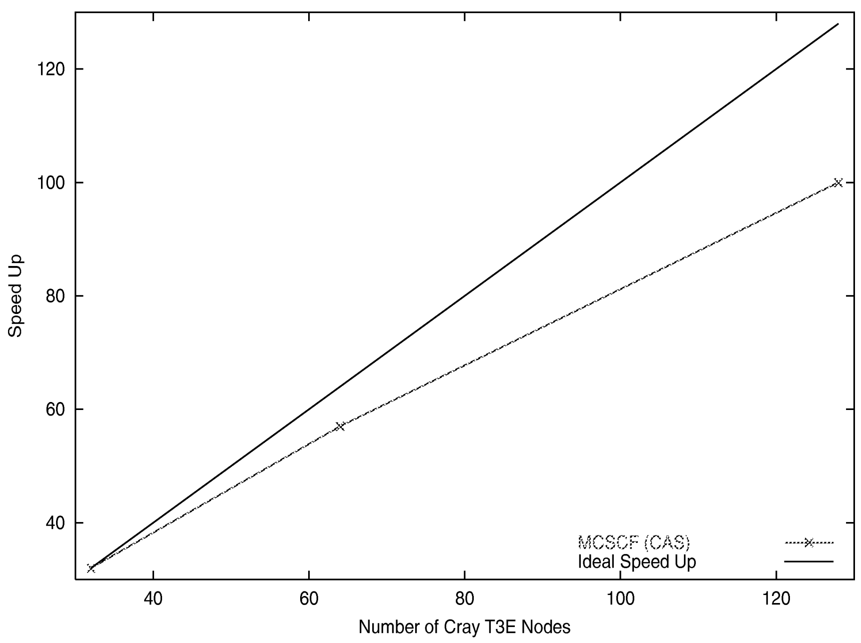


Fig. 5. The scaling of the semi-direct SCF module on an IBM SP, 160 MHz nodes, 512 MB memory per node, 3 GB of disk per node. 15 MB/sec/node sustained read bandwidth was achieved. The modified crown ether system has 105 atoms, 1343 basis functions using the Dunning augmented cc-pVDZ basis set, and 362 electrons.



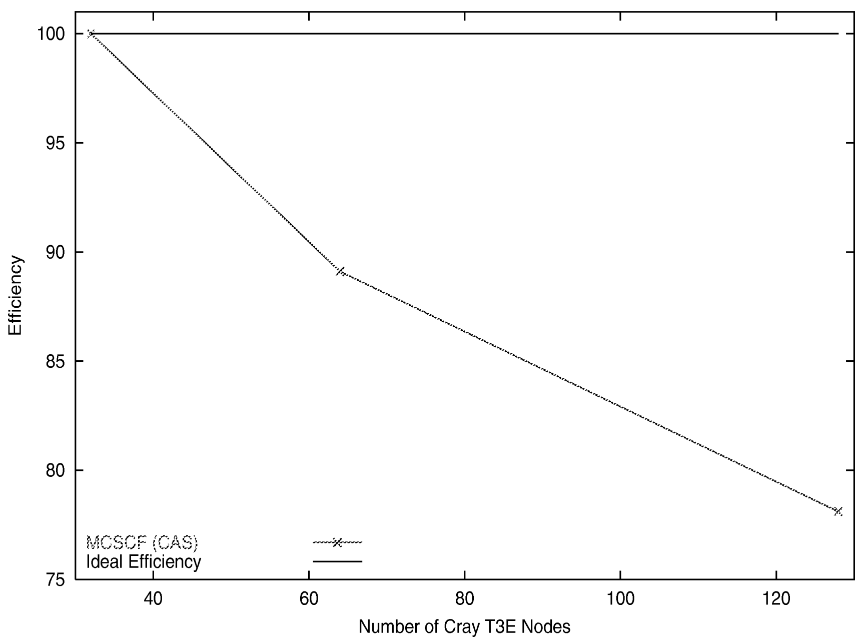


Fig. 6. The scalability and efficiency of the direct MCSCF (CASSCF) module for di-allyl (C_6H_{10}) with 360 basis functions and 7 electrons in 12 active orbitals.

scales well with a better than 75% efficiency on 130 Cray T3E nodes.

3.4. Density-functional theory

The NWChem density functional theory (DFT) module uses the Gaussian basis set approach to compute closed shell and open shell densities and Kohn-Sham (KS) [26,27] orbitals in the:

- local density approximation (LDA),
- non-local density approximation (NLDA),
- local spin-density approximation (LSD),
- non-local spin-density approximation (NLSD), and
- any empirical mixture of local and non-local approximations (including exact exchange).

The formal scaling of the DFT computation can be reduced by choosing to use auxiliary Gaussian basis sets to fit the charge density (CD) in the evaluation of the Coulomb potential [28]; this formally diminishes the number of integrals to be computed from $O(N^4)$ to $O(N^3)$. Several Exchange-Correlation functionals have been implemented in NWChem: LDA [29–33], NLDA [34–40] and the so-called “hybrid” functionals that mix exact exchange with DFT exchange [41–43]. The Exchange-Correlation potential and energy are evaluated on a grid by standard numerical techniques [44–46]. Several techniques have been incorporated to improve convergence in the iterative SCF procedure: damping, level shifting and DIIS.

The overall scaling of the code with respect to system size in shown in Fig. 7. In this test an LDA Exchange-Correlation functional was chosen together with the adoption of a fitting basis set for the evaluation of the Coulomb potential. The resulting scaling is very close to $O(N^2)$ from the formal $O(N^3)$ because of use of screening techniques. All steps involved in the evaluation of the DFT total energy and analytic gradients have been implemented to exploit MPP technology with maximum efficiencies; every step involved in the construction of the KS Hamiltonian uses the aggregate memory of the computing nodes involved in the calculation and use is made of parallel linear algebra libraries. In Fig. 8 we show the parallel speed up of the DFT module: almost linear speed up is observed in all the curves and in some regions super-linear speed up is caused by cache effects.

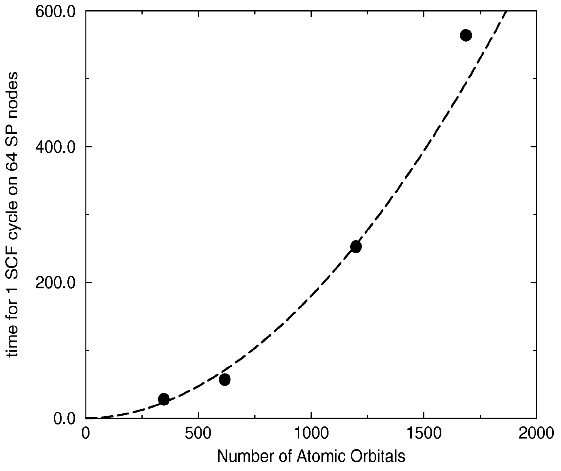


Fig. 7. The scaling of the SCF component of the DFT code (wall clock time versus number of atomic orbitals) on various zeolite fragments, using 64 IBM SP2 nodes. A curve of form $t = k \cdot n^B$ was fitted to the data with a resulting coefficient $B = 1.93$ (dashed line in the figure).

3.5. Coupled cluster module

The Coupled Cluster singles and doubles (CCSD) module with linearized triples (CCSD(T)) is essentially the Titan module of Rendell and Lee [47] that has been tightly integrated with the NWChem integral and transformed integral modules. The module is functional for only closed shell references and thus has somewhat limited functionality. There is no gradients module and the code does not take advantage of molecular symmetry. While the code scales very well only the triples component has a high level of performance. Since current development efforts have targeted a fully functional CCSD(T) for open and closed shell energies and gradients, we report here only that the functionality exists and is usable within the NWChem framework. Fig. 9 shows the scaling for the current implementation with the propanol ($CH_2OHCH_2CH_3$) molecule using 160 basis functions (C and O cc-pVTZ basis; H cc-pVDZ basis) on the 12 atoms, 34 electron system. Fig. 9 also shows that the triples part of the code has better scaling and is more efficient than the CCSD component.

3.6. Second-order Møller–Plesset module

There are multiple implementations of the second-order Møller–Plesset (MP2) theory in NWChem. There

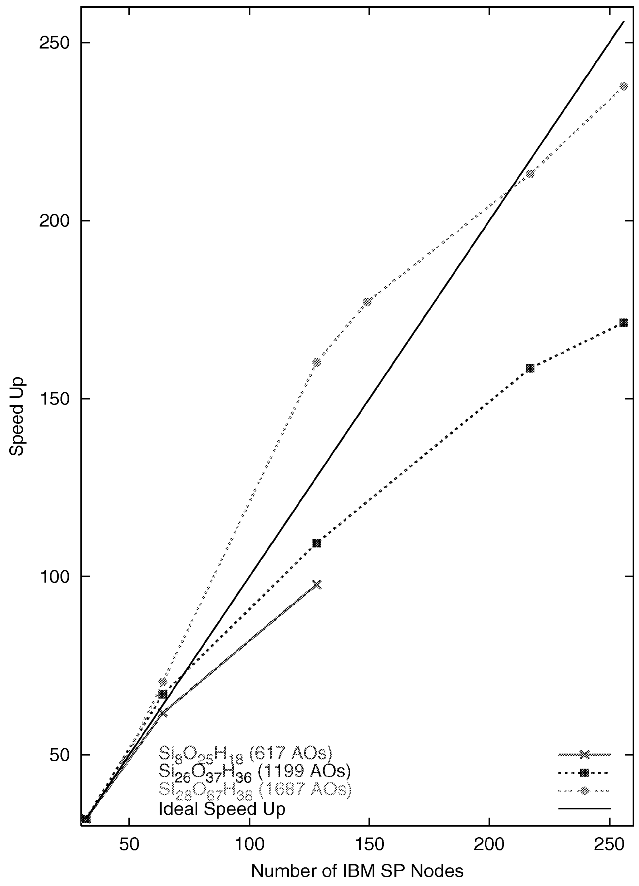
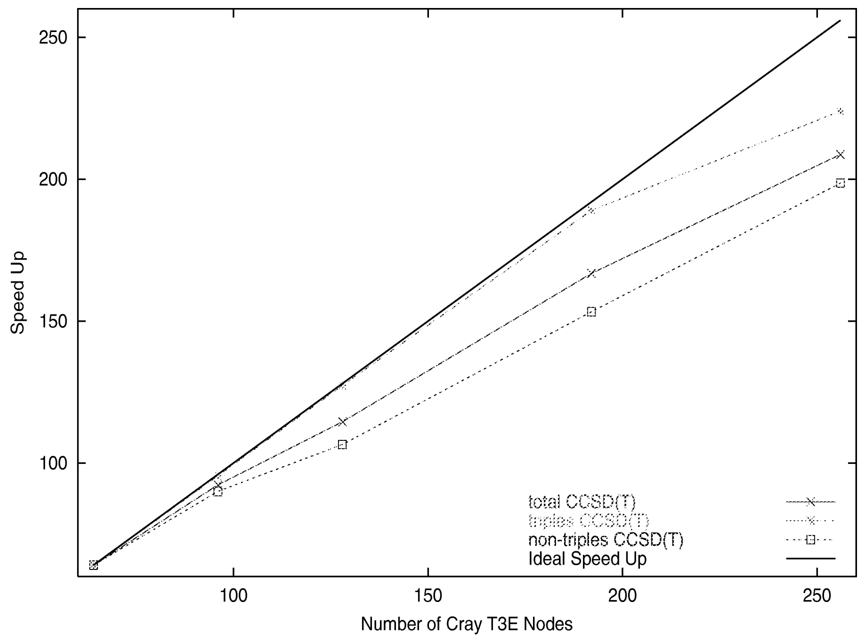


Fig. 8. Parallel speed up as a function of the number of processors on an IBM-SP2 for a DFT calculation of zeolite fragments of various size ($Si_8O_{25}H_{18}$, $Si_{26}O_{37}H_{36}$ and $Si_{28}O_{67}H_{30}$).

is a fully direct (D-MP2) module, the default semi-direct module (SD-MP2), and the Resolution of the identity approximate MP2 (RI-MP2). All three modules allow for frozen core orbitals and the D-MP2 and RI-MP2 allow for frozen virtual orbitals. The SD-MP2 algorithm is explained in another article in this issue.

The RI-MP2 code is exact in the limit of a complete fitting basis set. A fitting basis set is required to approximate the four center integrals with three center integral summations. RI-MP2 can obtain high accuracy with a relatively modest fitting basis set. The RI-MP2 can be as much as a factor 40 times faster than the exact MP2. Care must be taken in choosing the fitting basis set and the proper choice of this basis set is still under investigation by the computational chemistry research community.

The D-MP2 and RI-MP2 only compute energies where the SD-MP2 module computes both energies and analytical gradients. The general synopsis of the algorithm is that it follows conventional semi-direct algorithms but is implemented in parallel using the GA technology to minimize the complexity of the implementation when summing components of the matrices contracted against integrals or integral derivatives. One computational savings introduced is the understanding that many of the MP2 contributions can be derived from half-transformed integrals. Thus the module is designed to trade computation for storage and communication of the molecular integrals by computing a block of half-transformed integrals on demand. This leads to computing the integrals twice but avoiding global synchronization of the system.



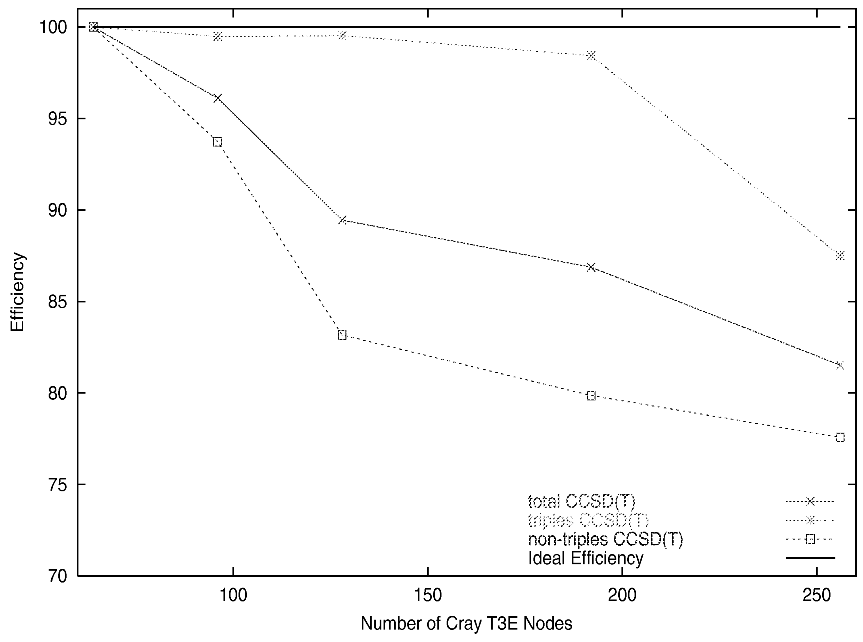
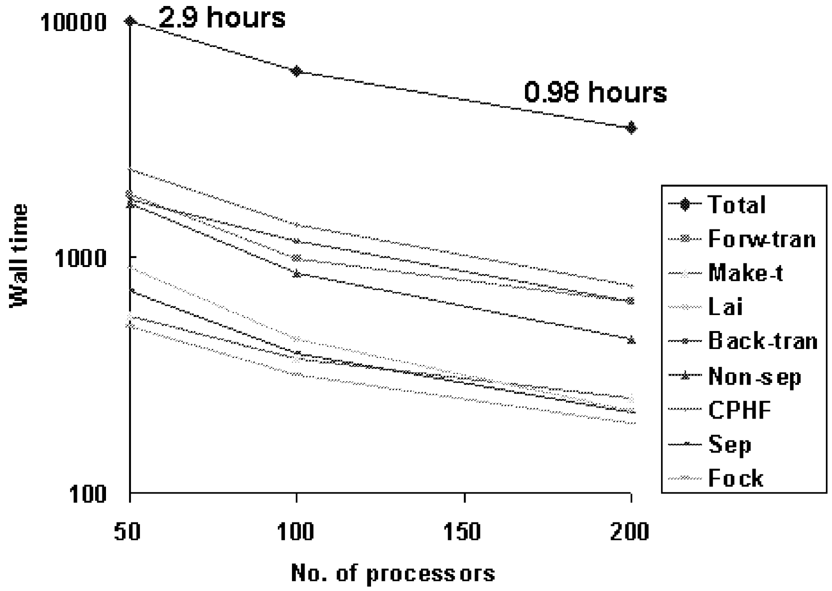


Fig. 9. The scalability and efficiency of the CCSD(T) module. The triples computation is separated out from the non-triples components (initial guess and CCSD solution). The total scalability and efficiency is also shown. The molecular system is propanol ($CH_2OHCH_2CH_3$) with 160 basis functions, 12 atoms, and 34 electrons.



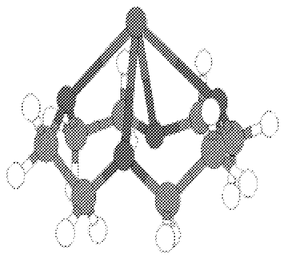


Fig. 10. Parallel speed up as a function of the number of processors on the EMSL IBM-SP2 for a MP2 Gradient simulation of $KC_8O_4H_{16}$ with 458 basis functions and 114 electrons.

The parallel scaling of MP2 gradient code and many component phases of the algorithm is shown in Fig. 10 for a 29 atom system with 458 basis functions. The plot shows a 74% efficiency for this system.

3.7. Direct dynamics interface to POLYRATE

The Direct Dynamics (DRDY) module generates one of the input files (file30) necessary to calculate reaction rates using POLYRATE [48]. This module does the necessary electronic structure calculations of nuclear gradient and Hessian at the saddle point, reactants, products, and along the minimum energy path for the reaction. The three potential energy critical points are input to the module as well as criteria for following the minimum energy path. The code was adapted from the DIRDYGAUSS code with hooks to the appropriate NWChem task modules. DRDY is fully integrated with NWChem having access to results from the run time data base and can exploit the full parallel functionality of NWChem. The integration schemes used to follow the reaction path include the Euler and three Page–McIver (PM) methods. The latter methods are the local quadratic approximation, the corrected local quadratic approximation, and the cubic algorithms.

3.8. Pseudopotential plane-wave method

A recent enhancement to NWChem capabilities has been the development of a pseudopotential plane-wave (PSPW) module. This method, which is essen-

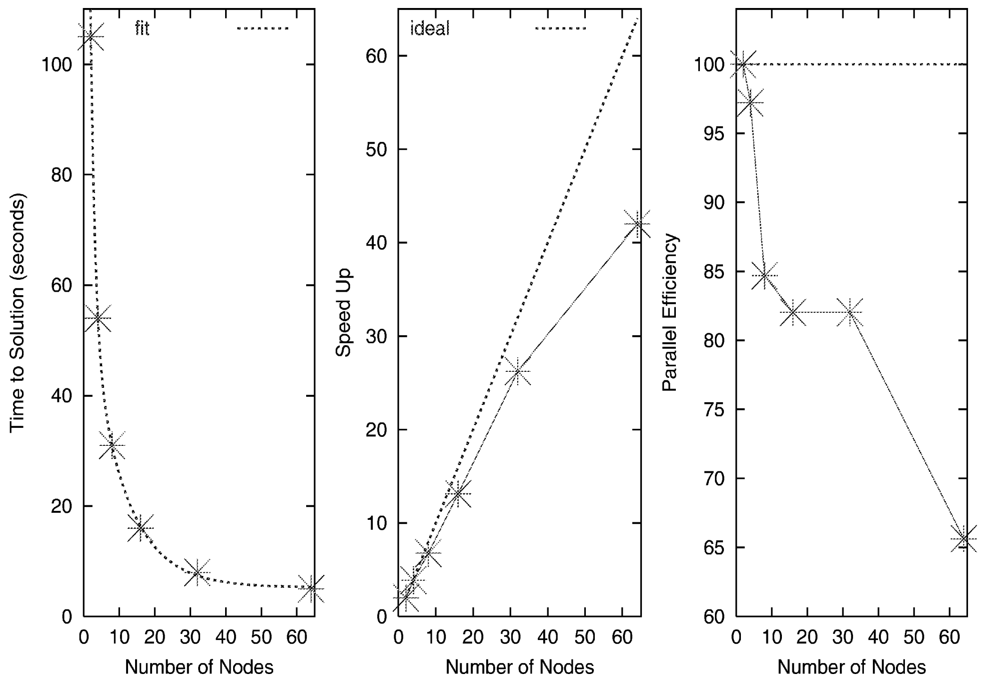


Fig. 11. These three plots represent the performance of the PSPW module by displaying the time to solution, speed up, and parallel efficiency of a 16 water simulation using a 64 × 64 × 64 dimension FFT Grid. The data was determined from work done on the EMSL IBM SP. The efficiency drop off at 64 nodes is due to the lack of work on each node; the time to solution is about 5 seconds.

tially a Density Functional Theory pseudopotential band-structure code [49,50], was first popularized by Car and Parrinello [51]. Based on this type of band-structure code, they developed a unified molecular dynamics and Density Functional Theory that can accurately and efficiently simulate atomic motion directly from forces calculated on the fly from a Density Functional Theory ground state energy surface. Over the last decade, pseudopotential plane-wave methods have become one of the more popular first principle methods, and are the methods of choice for studying complex molecular, liquid, and solid state systems at a first principles level.

The PSPW module is in a preliminary development stage. However, it is robust and efficient enough to treat systems equivalent to 32 water molecules on an IBM-SP or Cray-T3E. Currently, the PSPW module contains a steepest descent operation to minimize the total energy functional and optimize geometries, a Car–Parrinello operation to perform constant particle-energy-volume (NVE) molecular dynamics, a pseudopotential generator operation to generate Hamann [52] and Troullier–Martins [53] norm-conserving pseudopotentials. In addition, there are operations to initialize wave functions, put the pseudopotentials into a Kleinman–Bylander separable form [54], and convert the wave functions to other simulation cells. Currently, only LDA and LSDA exchange-correlation potentials are implemented [55].

Parallelization of the time-consuming PSPW operations (steepest descent, Car–Parrinello) are based upon a slab decomposition of the three-dimensional (plane-wave basis) grids. With this decomposition the majority of routines are trivially parallel. However, this decomposition requires a significant amount of communication between processors for performing three-dimensional fast Fourier transforms and dot products between two three-dimensional (plane-wave basis) grids. For most systems, the computationally dominant step is the computation of numerous dot products. Currently, the PSPW module is parallelized using the TCGMSG message passing library. Fig. 11 shows the time to solution, speed up, and parallel efficiency on 64 processors on the EMSL IBM SP.

We are currently working to improve the PSPW module with the implementation of new functionalities and parallel algorithms. An efficient Conjugate Gradient algorithm developed by Edeleman et al. [56] is being implemented into the PSPW module. This operation will significantly speed-up the minimization of the total energy functional. In addition, we are working on putting in the PSPW module the PBE96 exchange-correlation potential [57], Vanderbilt pseudopotentials [58], and an aperiodic convolution scheme [59–61] which allows for the calculation of charged systems. The parallelization of the PSPW module is being updated to use the Global Array toolkit and new algorithms are being developed to make three-dimensional fast Fourier transforms and dot products more efficient on the new SMP architectures as well as PC based Beowulf clusters.

3.9. Molecular dynamics simulations

The code includes a module for classical molecular dynamics simulations of macromolecular systems. The target for the module are large molecular systems with upwards of 10,000 atoms. Simulations of such systems require the use of parameterized effective pair models for the description of atomic interactions. The parallel implementation in NWChem is based on a domain decomposition [13] of the physical simulation volume. For large molecular systems, this implementation takes advantage of the distribution of the atomic data to reduce the memory requirements, and uses the locality of molecular interactions to reduce the interprocessor communication [63,64].

To improve computational efficiency, solvent and solute are treated separately. Solvent molecules are assigned to the domains according to their center of geometry. This avoids bonded interactions crossing node boundaries. Solute molecules are broken up into segments, and each of the segments is assigned based on its center of geometry. This limits the number of solute bonded interactions that cross node boundaries. The processor to which a domain is assigned is responsible for the calculation of interactions of atoms within the domain. For the calculation of forces and energies in which atoms are involved on neighboring domains assigned to different processors, information from these domains need to be exchanged between processors. The number of neighboring domains is determined by the size and shape of the domains and the range of interaction specified. It is this exchange of information that represents the main communication requirement, which is taking place every simulation time step. Consequently, one of the main efforts has been to design a structure that minimizes the cost of this communication. However, for very large molecular systems, memory requirements also need to be taken into account. A compromise between these requirements is found in performing the communication in successive point to point communications rather than using the collective shift algorithm which reduces the number of communication calls for the same amount of communicated data.

Domain decomposition is also the most complex approach to implement, since periodic redistribution and associated data communication is needed for atoms that move from one processor's domain to another. Moreover, for heterogeneous molecular systems the computational work is generally not evenly distributed over the processors. Moreover, the distribution of work will change during a simulation. This requires the use of dynamic load balancing in order for this parallelization approach to be efficient.

The scheme used in NWChem to load balance molecular dynamics simulations consists of two methods that can be used because of the point-to-point one-sided asynchronous communication that the GA library allows to access data non-local to the processor that requires the data. Based on the time that it takes to synchronize after all force contributions have been evaluated, the code can use a combination of collectively resizing the domain of each processor, such that the busiest processor's domain becomes smaller, and non-collectively redistributing the work to evaluate forces between atoms on different processors from the busiest to a lesser busy processor.

This domain decomposition is used in the three basic operations that the molecular dynamics module performs: energy minimization, molecular dynamics simulation and free energy evaluation. The force fields implemented in NWChem include AMBER and CHARMM, and can be extended with terms for

[13] For a general description of "domain decomposition" see Ian Foster's book [62].

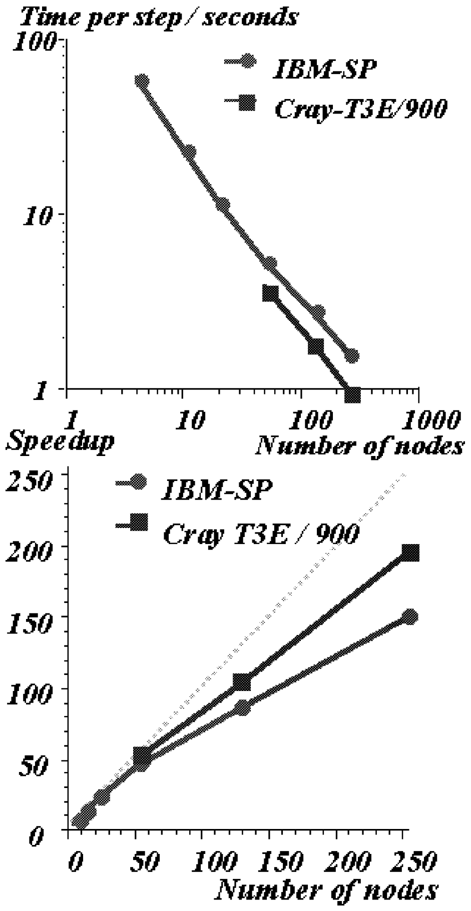


Fig. 12. Times per molecular dynamics step and parallel speed up as a function of the number of processors on an IBM-SP and a CRAY-T3E of a simulation of a drop of dichloroethane in water (a 100,000 atom system). The AMBER force field was used with a cutoff of 2.4 nm.

first order or self consistent electronic polarization and smooth particle-mesh Ewald corrections for long-range electrostatic interactions.

4. Concluding remarks

We have presented an overview of the computational chemistry suite NWChem. Complete details of the implementations of each of the modules is beyond the scope of any single paper. We have outlined the capabilities and performance of some of the modules within NWChem as well as the parallel software tools used in NWChem. You will find papers covering the specifics of some NWChem functionality within this issue of Computer Physics Communications and in the general chemistry and computational science literature. NWChem was designed from the beginning to be an MPP based code and like all computational chemistry applications it has particular strengths and weaknesses. To obtain a copy of NWChem for your research group you need only to request a User's Agreement, complete the agreement, and return it to Pacific Northwest National Laboratory. Once the agreement is confirmed you will be given a username and password to down-load the binary or source distribution from the EMSL web site. [14]

Acknowledgments

This work was performed under the auspices of the Department of Energy under contract DE-AC06-76RLO with Battelle Memorial Institute which operates the Pacific Northwest National Laboratory, and under contract W-7405-ENG-82 at Ames Laboratory. The funding sources for the development of NWChem and the parallel software tools over the lifetime of the project include the division of Chemical Sciences, the Office of Basic Energy Sciences, Mathematical, Information, and Computational Sciences division of the Office of Advanced Scientific Computing Research, the US DOE High Performance Computing and Communications Initiative, and the Environmental and Molecular Sciences Laboratory, the Construction Project, D-384 and Operations, from the Office of Biological and Environmental Research. We would also like to acknowledge the generous flexibility of the editors of this particular issue of Computer Physics Communications. This research was performed in part using the Molecular Science Computing Facility (MSCF) in the William R. Wiley Environmental Molecular Sciences Laboratory at the Pacific Northwest National Laboratory. The MSCF is funded by the Office of Biological and Environmental Research in the U.S. Department of Energy. This research also used resources of the National Energy Research Scientific Computing Center, which is supported by

[14] To obtain the NWChem User Agreement fill out the form at http://www.emsl.pnl.gov/pub/docs/nwchem/download.html.

the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

References

[1] P.R. Taylor, Building HPC software infrastructure: How do we best couple computer science and computational science? Speedup J. (1999) Submitted.

[2] See http://www.emsl.pnl.gov/pub/docs/nwchem/index.html for the latest NWChem release information.

[3] 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, J. Nieplocha, Int. J. Quantum Chem. Symposium 29 (1995) 475.

[4] 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, J. Nieplocha, High Performance Computational Chemistry: NWChem and Fully Distributed Parallel Applications in Advances in Parallel Computing, Vol. 10, High Performance Computing: Technology, Methods, and Applications, J. Dongarra, L. Gradinetti, G. Joubert, J. Kowalik (Eds.) (Elsevier Science B.V., Amsterdam, 1995).

[5] 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, J. Nieplocha, High Performance Computational Chemistry: Towards Fully Distributed Parallel Algorithms in High Performance Computing: Symposium 1995, Grand Challenges in Computer Simulation, A.M. Tentner (Ed.), Proceedings of the 1995 Simulation Multiconference, April 9–13, 1995 (The Society for Computer Simulation, San Diego, CA, 1995).

[6] 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, J. Nieplocha, Advances in Parallel Distributed Data Software; Computational Chemistry and NWChem in Applied Parallel Computing, in: Computations in Physics, Chemistry, and Engineering Science, J. Wasnieski, J. Dongarra, K. Madsen (Eds.) (Springer, Berlin, 1996).

[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, J. Nieplocha, Future Gener. Comput. Systems 12 (1996) 273.

[8] J. Nieplocha, R.J. Harrison, R.J. Littlefield, in: Proc. Supercomputing '94, 1994, p. 340.

[9] J. Nieplocha, R.J. Harrison, R.J. Littlefield, J. Supercomput. 10 (1996) 197.

[10] R.J. Harrison, Private communication (1999).

[11] J.H. Wilkinson, The Algebraic Eigenvalue Problem (Oxford University Press, Oxford, 1965).

[12] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide (2nd edn.) (SIAM, Philadelphia, 1995).

[13] G. Fann, R. Littlefield, Parallel inverse iteration with reorthogonalization, in: Proc. 6th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, 1993).

[14] G.I. Fann, R.J. Littlefield, D.M. Elwood, Performance of a fully parallel dense real symmetric eigensolver in quantum chemistry applications, in: Proc. High Performance Computing '95, Simulation MultiConference (Society for Computer Simulation, San Diego, CA, 1995).

[15] I. Dhillon, G. Fann, B. Parlett, Application of a new algorithm for the symmetric eigenproblem to computational quantum chemistry, in: Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, 1997).

[16] D.F. Feller, K. Schuchardt, Extensible computational chemistry environment basis set database, Version 1.0, as developed and distributed by the Molecular Science Computing Facility of the Environmental and Molecular Sciences Laboratory which is part of the Pacific Northwest National Laboratory.

[17] http://www.emsl.pnl.gov/pub/forms/basisform.html.

[18] The periodic DFT code, General Approach to Polymers, Surfaces, and Solids (GAPSS) is currently **not** distributed with NWChem. You must contact Dr. Maciej Gutowski at maciej.gutowski@pnl.gov to obtain and use this NWChem module.

[19] E.D. Glendening, A.E. Reed, J.E. Carpenter, F. Weinhold, QCPE Bull. 10 (1990) 58.

[20] E.D. Glendening, J.K. Badenhoop, A.E. Reed, J.E. Carpenter, F. Weinhold, NBO 5.0 (Theoretical Chemistry Institute, University of Wisconsin, Madison, WI, 1998).

[21] E.D. Glendening, F. Weinhold, J. Comput. Chem. 19 (1998) 593.

[22] Your best resource for learning more about Python is the WWW site http://www.python.org.

[23] A.T. Wong, R.J. Harrison, J. Comput. Chem. 16 (1995) 1291.

[24] R.J. Harrison, M.F. Guest, R.A. Kendall, D.E. Bernholdt, A.T. Wong, M. Stave, J.L. Anchell, A.C. Hess, R.J. Littlefield, G.I. Fann, J. Nieplocha, G.S. Thomas, D. Elwood, J. Tilson, R.L. Shepard, A.F. Wagner, I.T. Foster, E. Lusk, R. Stevens, J. Comput. Chem. 17 (1996) 124.

[25] G.B. Bacskay, Chem. Phys. 61 (1982) 385.

[26] P. Hohenberg, W. Kohn, Phys. Rev. B 136 (1964) 864.

[27] W. Kohn, L.J. Sham, Phys. Rev. A 140 (1965) 1133.

[28] B.I. Dunlap, J.W.D. Connolly, J.R. Sabin, J. Chem. Phys. 71 (1979) 4993.

[29] J.C. Slater, Quantum Theory of Molecules and Solids, Vol. 4: The Self-Consistent Field for Molecules and Solids (McGraw-Hill, New York, 1974).

[30] D.M. Ceperley, B.J. Alder, Phys. Rev. Lett. 45 (1980) 566.

[31] S.J. Vosko, L. Wilk, M. Nusair, Can. J. Phys. 58 (1980) 1200.

[32] J.P. Perdew, A. Zunger, Phys. Rev. B 23 (1981) 5048.

[33] J.P. Perdew, Y. Wang, Phys. Rev. B 45 (1992) 13 244.

[34] J.P. Perdew, Phys. Rev. B 33 (1986) 8822.

[35] C. Lee, W. Yang, R.G. Parr, Phys. Rev. B 37 (1988) 785.

[36] A.D. Becke, J. Chem. Phys. 88 (1988) 3098.

[37] J.P. Perdew, J.A. Chevary, S.H. Vosko, K.A. Jackson, M.R. Pederson, D.J. Singh, C. Fiolhais, Phys. Rev. B 46 (1992) 6671.
[38] J.P. Perdew, K. Burke, M. Ernzerhof, Phys. Rev. Lett. 77 (1996) 3865.
[39] P.W. Gill, Mol. Phys. 89 (1996) 433.
[40] F.A. Hamprecht, A.J. Cohen, D.J. Tozer, N.C. Handy, J. Chem. Phys. 109 (1998) 6264.
[41] A.D. Becke, J. Chem. Phys. 98 (1992) 1372.
[42] A.D. Becke, J. Chem. Phys. 98 (1993) 5648.
[43] A.D. Becke, J. Chem. Phys. 107 (1997) 8554.
[44] A.D. Becke, J. Chem. Phys. 88 (1988) 2547.
[45] C.W. Murray, N.C. Handy, G.L. Laming, Mol. Phys. 78 (1993) 997.
[46] B. Delley, J. Comput. Chem. 17 (1996) 1152.
[47] A.P. Rendell, T.J. Lee, J. Chem. Phys. 94 (1991) 6219.
[48] A.D. Isaacson, D.G. Truhlar, S.N. Rai, R. Steckler, G.C. Hancock, B.C. Garrett, M.J. Redmon, Comput. Phys. Commun. 47 (1987) 91.
[49] J. Ihm, A. Zunger, M.L. Cohen, J. Phys. C: Solid State Phys. 12 (1979) 4409.
[50] M.C. Payne, M.P. Teter, D.C. Allen, T.A. Arias, J.D. Joannopoulos, Rev. Mod. Phys. 64 (1992) 1045.
[51] R. Car, M. Parrinello, Phys. Rev. Lett. 55 (1985) 2471.
[52] D.R. Hamann, Phys. Rev. B 40 (1989) 2980.
[53] N. Troullier, J.L. Martins, Phys. Rev. B 43 (1991) 1993.
[54] L. Kleinman, D.M. Bylander, Phys. Rev. Lett. 48 (1982) 1425.
[55] S.H. Vosko, L. Wilk, M. Nusair, Can. J. Phys. 58 (1980) 1200.
[56] A. Edelman, T.A. Arias, S.T. Smith, SIAM J. Matrix Anal. Appl. 20 (1998) 303.
[57] J.P. Perdew, K. Burke, Y. Wang, Phys. Rev. B 54 (1996) 16 533.
[58] D. Vanderbilt, Phys. Rev. B 41 (1990) 7892.
[59] R.N. Barnett, U. Landman, A. Nitzan, G. Rajagopal, J. Chem. Phys. 94 (1991) 608.
[60] E.J. Bylaska, P.R. Taylor, R. Kawai, J.H. Weare, J. Phys. Chem. 100 (1996) 6966.
[61] M.R. Jarvis, I.D. White, R.W. Godby, M.C. Payne, Phys. Rev. B 56 (1997) 14 972.
[62] I. Foster, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering (Addison-Wesley Publishing Company, Inc., Reading, MA, 1995).
[63] A.R.C. Raine, D. Fincham, W. Smith, Comput. Phys. Commun. 55 (1989) 13.
[64] S.J. Plimpton, B.A. Hendrickson, G.S. Heffelfinger, in: Proc. 6th SIAM Conference on Parallel Processing for Scientific Computing (SIAM, Philadelphia, 1993).