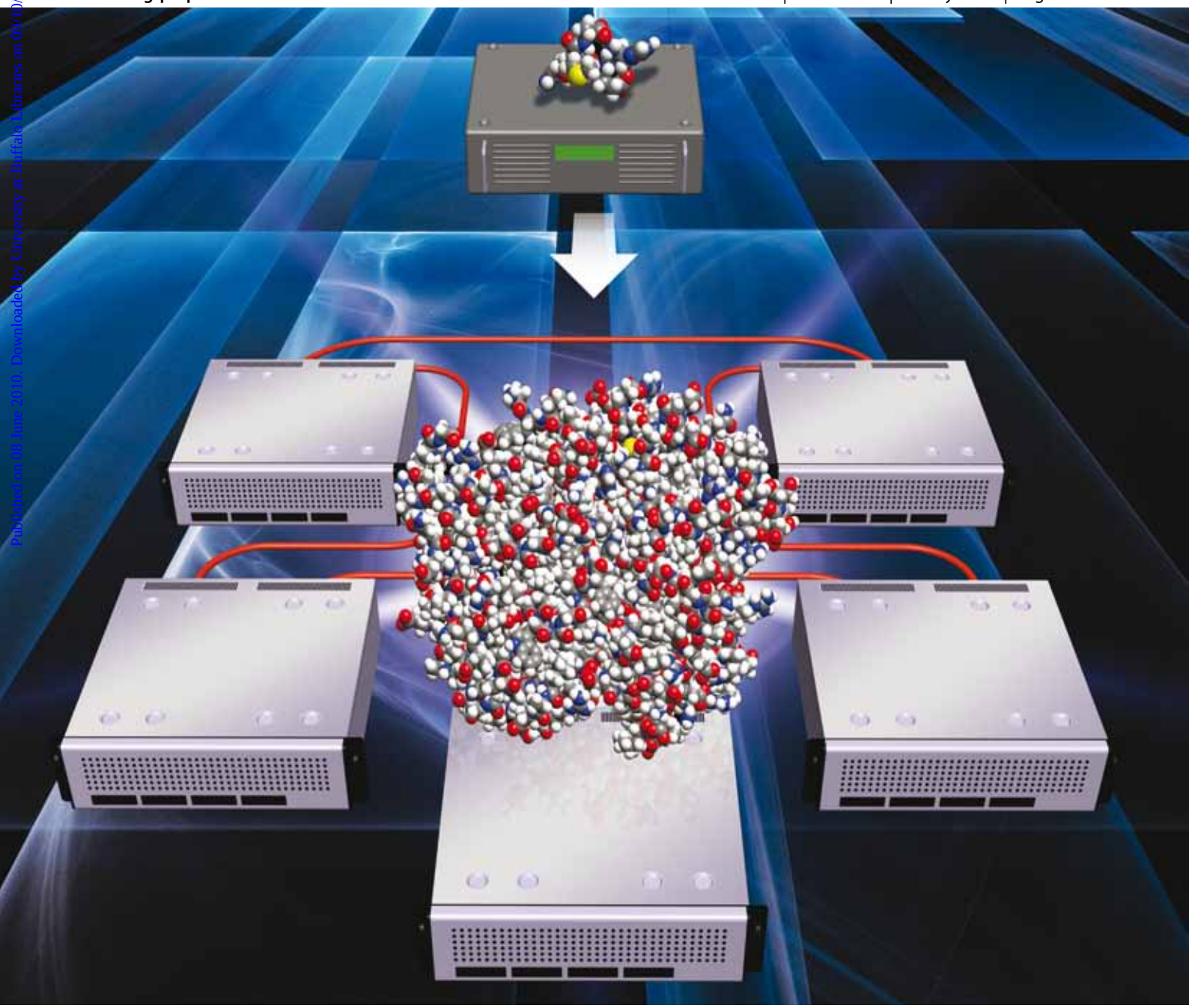
PCCP

Physical Chemistry Chemical Physics

www.rsc.org/pccp

Volume 12 | Number 26 | 14 July 2010 | Pages 6877–7296



ISSN 1463-9076

COVER ARTICLE
de Jong, Lindh *et al.*
Utilizing high performance computing for chemistry: parallel computational chemistry

HOT ARTICLE
Ohoyama and Kasai
Multi-dimensional steric effect for Xel^* (B) formation in the oriented Xe* ($^3P_2$, $M_J = 2$) + oriented CH_3I reaction

Utilizing high performance computing for chemistry: parallel computational chemistry

Wibe A. de Jong,*[a] Eric Bylaska,[a] Niranjan Govind,[a] Curtis L. Janssen,[c] Karol Kowalski,[a] Thomas Müller,[d] Ida M. B. Nielsen,[c] Hubertus J. J. van Dam,[a] Valera Veryazov[b] and Roland Lindh*[e]

Received 10th February 2010, Accepted 4th May 2010
First published as an Advance Article on the web 8th June 2010
DOI: 10.1039/c002859b

Parallel hardware has become readily available to the computational chemistry research community. This perspective will review the current state of parallel computational chemistry software utilizing high-performance parallel computing platforms. Hardware and software trends and their effect on quantum chemistry methodologies, algorithms, and software development will also be discussed.

1. Introduction

Scientific discovery utilizing quantum chemistry started off at a time when no digital computers existed. Early quantum chemists used sophisticated analog calculators and man power to perform their daily calculations. Hardly any scientific visionary at that time would have projected that 90 years later quantum chemistry would evolve into a mature science represented in all facets of chemistry. As a result, researchers of the 21st century have come to rely on parallel computers as the backbone for computational quantum chemistry (QC) studies to advance their field of science. The usage of computers for QC ranges from workstations to parallel architectures including teraflop departmental clusters and large petascale computing platforms. Even the current generation of workstations has multiple computational units that can operate in parallel.

The first computer was invented in 1937 by Stibitz, the father of modern day computers. Just a year earlier, Turing who is widely regarded as the father of modern computer science, developed the notion of algorithm and computation with the Turing machine. During the Second World War programmable computers such as the Harvard Mark 1 from IBM and the ENIAC were built, and the flow of new computer technologies hasn't stopped since. It is during the development of the ENIAC that the von Neumann architecture, or stored program architecture, was developed that is used on most modern day computers.

The start of computing opened many opportunities for quantum chemistry, and the first quantum chemistry software was in use early on. Over the years quantum chemists have strived to optimize the efficiency of their computer implementations. Some of the major advances that allow computational chemistry to address much larger systems are due to pure methodologic developments. It is worth to mention, for example, the so-called integral-direct methods by Almlöf and co-workers,[1] the introduction of the resolution-of-identity approach by Vahtras *et al.*,[2] and the introduction of localized methods to *ab initio* approaches as introduced by Pulay and Saebo[3] as developments which have resulted in quantum leaps in improvements with respect to the sizes of the molecular systems which today are accessible for *ab initio* investigations. However, additional improvements have been associated with that quantum chemical implementations over the years have been modified to take explicit account of the hardware advantages of particular computer platforms. Each time a shift occurred in hardware technology, quantum chemists have been there to modify theory or algorithms to ensure their software could utilize the full potential offered. For example, the development of fast direct access disk by IBM in the 1960s inspired Yoshimine[4] to develop a special bin-sort algorithm to be used in connection with the four one-quarter transformation steps involved in the transformation of the so-called two-electron integrals from the AO to MO basis. The use of large solid state devices (SSD) in the CRAY X-MP computers allowed computational chemistry scientists to use the SSD memory to replace the need for disk drives and by doing so avoiding the so-called I/O-bottleneck in QC simulations.[5]

The emergence of vector processing as expressed, for example, on the Floating Point System (FPS) attached array processor required Fortran code which parallelized on the second innermost loop. Quantum chemists were quick to rewrite their codes to gain performance enhancements as large as a factor of 25 compared with conventional technology.[6] A few years later the codes were changed back to vectorization on the innermost loops to fully utilize the new RISC technology and minimize core memory access and reduce cache-faults. This was of particular importance since workstation clusters were to a large extent based on this technology.[7]

One of the areas in which quantum chemists have had the most profound impact is in the field of parallel computing.

[a] *Pacific Northwest National Laboratory, P.O. Box 999, Richland, WA 99352, USA. E-mail: bert.dejong@pnl.gov*
[b] *Lund University, P.O. Box 124, 221 00 Lund, Sweden*
[c] *Sandia National Laboratories, P.O. Box 969, Livermore, CA 94550, USA*
[d] *Jülich Supercomputing Centre, Institute of Advanced Simulation, Research Centre Jülich, D-52425 Jülich, Germany*
[e] *Uppsala University, P.O. Box 518, SE-751 20 Uppsala, Sweden. E-mail: roland.lindh@kvac.uu.se*

Here quantum chemists have acted as "premature" users of hidden technology, visionaries, and as those setting the standards. The concept of parallel computing started to circulate as a topic in the late 1970s. In 1980 Rolf Seeger published the pathbreaking paper entitled "Parallel Processing on Minicomputers: A Powerful Tool for Quantum Chemistry".[8] In this paper, Seeger demonstrated that in utilizing micro-programming techniques he could get memory access, basic arithmetic operations and two floating-point arithmetic units to work in parallel. Seeger recorded a performance enhancement of a factor of 2.5–2.8 compared to conventional use of a Perkin-Elmer computer.

One of the true visionaries, developer and promotor of the emerging parallel technology was Enrico Clementi. During the 1980s his group at IBM Kingston developed the concept of the loosely coupled array of processors (lCAP), a predecessor of the so-called Beowulf clusters.[9] During this time their experiments taught them hard lessons concerning problems in parallel program implementation and the need for tools for the same purpose. In spite of the poorly developed hardware and software for inter-processor communication, the large-grain-size structure of quantum chemical algorithms makes them suitable for parallelization and the early lCAP systems were instrumental in demonstrating the power of the emerging technology. Software for message-passing and shared-memory communications was eventually implemented, tested and improved. For example, versions of Parallel Fortran[10] were developed and tested, but later abandoned by the community.

At present computational chemistry researchers around the world utilize teraflop and petaflop parallel hardware architectures ranging from tens to hundreds of thousands of processors to perform their simulations. In the United States the largest open-science supercomputer, over two petaflops of peak performance, is housed at the Oak Ridge National Laboratory, and two more multi-petaflop systems will become available at Argonne National Laboratory and the National Center for Supercomputing Applications at the University of Illinois. Outside the United States petaflop computers are becoming available to the science community, at the Forschungszentrum Jülich in Germany and in China. The European Union has funded the development of several petascale leadership computing facilities through the Partnership for Advanced Computing in Europe (PRACE).

In concert with the evolution of parallel computers, developers of computational chemistry software have been required to adjust their algorithms and codes to take full advantage of the hardware. In contrast to the relatively easy transition from scalar to vector architectures, the transition to parallel architectures has been much more complex and demanding on computational chemistry software. Vector processors only required longer loops that are naturally available in computational chemistry algorithms. Parallel computing platforms require significant rewites of software to handle communication of data and tasks between processors. Some computational chemistry codes, NWChem[11] (and it's underlying Global Array (GA) Toolkit) and MPQC,[12] were developed from the ground up with a massively parallel framework in mind when the parallel hardware started to emerge.

In the last few years, hardware developments such as multi-core processors have changed the notion of fine-grain parallelization, yet again triggering quantum chemists to rethink how to use the inherent power of some hardware configurations. The very recent emergence of so-called graphical processor units (GPU) used in demonstration QC codes has changed the picture once more. These two new developments only illustrate the insatiable appetite of quantum chemists for more computer power, which will not stop at today's standard parallel solutions.

Today, almost 30 years after Rolf Seeger's first explicit paper on parallel computing in QC, it is time to sit back and reflect on the achievements so far. In this paper we will present a perspective on the current state of parallel computing in quantum chemistry with respect to most of the standard models used by computational chemists. We will do so by discussing the current state and directions of the parallel hardware architectures in section 2, and software technologies and paradigms in section 3. This will be followed by detailed discussions of parallelization and performance of standard QC methods. Finally we will look into the future and provide a brief perspective on quantum chemistry on extreme scale computing architectures.

2. Parallel hardware architectures

Parallel computers can be classified as distributed or shared memory computers on the basis of the means by which data is exchanged between compute units. Distributed memory computers typically consist of a number of compute nodes that are interconnected *via* a communication network through which data is exchanged *via* message-passing, whereas all processors in shared memory computers have direct access to the same address space. Today, most massively parallel computers employ a combination of shared- and distributed-memory architectures, typically using smaller shared-memory compute units that are connected *via* a communication network.

In the following we will first take a look at the compute nodes in parallel computers, illustrating the trends in computational hardware. We will then discuss the memory architecture, in particular the memory hierarchy present within a parallel computer. Finally, we will give a brief overview of interconnection networks with a focus on factors pertinent to parallel performance.

2.1 Computational units

Today, parallel computers are often commodity computers that are built from widely available components, and the compute nodes in such computers tend to be standard personal computers (PCs), albeit outfitted with some special hardware to connect them to the communication network. While some parallel computers are custom computers built from special-purpose components, even custom computers frequently use off-the-shelf processors, which are then connected using custom-designed interconnects.

The trends in compute node technology therefore largely parallel those of PC technology. In Fig. 1 we illustrate the development of the integer and floating performance as well as

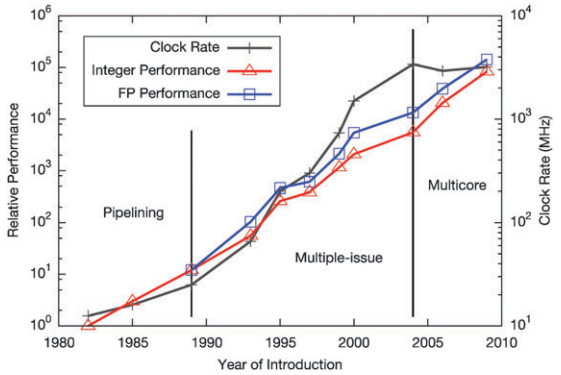


Fig. 1 Clock rate (MHz) and relative integer and floating point performance for selected Intel chips from 1982 until 2009. Integer performance was normalized relative to the 1982 value (yielding a relative performance of 1 for that year), and floating point performance was normalized to make the first data point (1989) match the integer performance for the same year. For details, see ref. 13.

the clock speed for selected Intel chips over the past nearly three decades. This time period can be divided into three parts on the basis of the development of the clock rate relative to the instruction rate: Pipelining (performing instructions in several, overlapping stages); multiple-issue (allowing multiple instructions to begin execution on each clock cycle); and multicore development (putting multiple processors on a single integrated circuit). While pipelining made the instruction rate grow significantly faster than the clockrate, performance per clock cycle decreased somewhat in the multiple-issue period mainly due to slower improvements in memory latency and bandwidth. Finally, in later years, the development of multicore chips enabled substantial improvements in performance despite the decrease in clockrate. Since the addition of more cores to a chip is expected to be a continuing trend, it will become increasingly important for parallel programmers to take advantage of this parallelism, for instance by means of multi-threaded programming.

The multicore performance gains illustrated in Fig. 1 have been obtained by adding multiple full-featured, general-purpose processors to a chip, and such processors are quite complex and require a large area of silicon to implement. However, by using simpler cores, such as those used in Graphics Processing Units (GPUs), which take up much less space, it is possible to fit many more cores onto a chip. Although still quite simple, GPUs have increased in sophistication to the point where they can be used to augment the system's general-purpose processor in non-graphics computing applications, and general-purpose computation on GPUs (GPGPU) is growing in popularity in a variety of scientific disciplines, including quantum chemistry.

2.2 Memory architecture

Modern computers have a hierarchical memory architecture with multiple data storage devices, each having distinguishing characteristics. Fig. 2 shows elements of the memory hierarchy in a parallel machine along with trends in the key characteristics of cost per byte, speed, and total capacity. At the top of the hierarchy are the memory components that are most expensive, fastest, and lowest capacity, namely the registers and caches. These layers provide data at a sufficiently high rate to support the rate at which the processor uses data. Because of the high cost, the capacity of the high speed memory components is low. At the top of the hierarchy are registers, containing a handful of data that is available for immediate use in computations. Next is the Level 1 (L1) cache, which provides a modest amount of memory (32 kbyte is typical) at very high speed. The L2 cache is larger than L1 cache (256 kbyte is common), but slightly slower than L1, and both the L1 and L2 caches are typically private to each core in the system. Some systems also have a L3 cache shared among all the cores on the chip, and this cache is of larger size, around 8 Mbyte. Below the caches is the system's main memory, which is substantially slower than the caches. Not all of the system's main memory is equivalent from the perspective of a given core. Some of the memory is attached directly to the processor chip that contains that core, and some is nonlocal and must be accessed through a high-speed network that is internal to the node. A still slower level in the memory hierarchy is the remote memory in a distributed memory machine, which must be accessed through the slower inter-node network. Finally, at the bottom of the hierarchy is the much slower disk system for holding data that does not fit into the main memory of the nodes. This can be provided as disks in each node in the system or through a parallel file system shared among all the nodes.

Each layer of this memory hierarchy must be carefully managed to achieve good performance from the machine, and different management techniques are needed for each layer in the hierarchy. For example, for high-level languages the compiler manages register allocation, although the details of how the application is written strongly affects the register allocation performed by the compiler. Caches are mostly managed automatically by the processor hardware; however, the effectiveness of cache utilization also strongly depends on how the application program is structured (see, for example ref. 14). Remote memory and disk storage are typically managed by the application directly or by libraries upon which the application is constructed. Applications must take into consideration the interplay between all of these hierarchies to obtain the best possible performance.

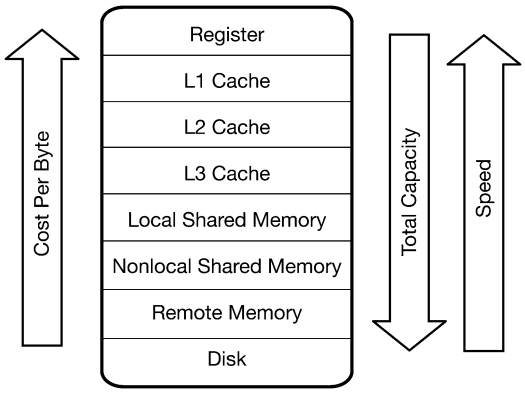


Fig. 2 The memory hierarchy in a parallel computer. Arrows indicate the direction of increasing memory cost, size and speed.

2.3 Interconnection networks

Interconnection networks consist of communication links and switching elements that connect the links. The communication network connecting the nodes plays a key role for the parallel performance of applications, and the network characteristics that are typically of greatest importance to the parallel programmer are the network topology as well as the network bandwidth and latency. The topology of a network refers to the way the nodes are physically connected. For instance, the nodes may be connected *via* a simple ring topology, or the communication channels may form a two-dimensional mesh (grid) with a compute unit on each grid point. Different network topologies may display very different parallel performance characteristics, and the choice of topology is usually a compromise between cost and performance. In Table 1, we show a number of commonly encountered network topologies along with their performance characteristics. The topologies are listed roughly in order of increasing performance and cost and are characterized by the network degree (n_{degree}), which represents the number of links supported by each switching element; the number of switching elements (n_{sw}); the maximum number of switching elements through which data must pass (n_{hop}); and the bisection width (B_{C}), which represents the minimum number of links that must be removed to separate the nodes into two groups of equal size. The cost of the network depends on n_{degree} and n_{sw}, and the network performance is largely determined by the parameters n_{hop} and B_{C}. While the communication network would ideally provide a direct connection between any two nodes (*e.g.*, by using a crossbar), such completely connected topologies are prohibitively expensive for massively parallel computers, and the largest parallel computers available today typically use a mesh topology. A thorough discussion of network topologies is beyond the scope of this article and can be found elsewhere.[15]

In addition to the network topology, the bandwidth and latency of the network are important factors determining parallel performance. Using an idealized machine model, which ignores potential network congestion as well as the relative positions of nodes, the time required to send a message of length l between two nodes can be expressed as $t = \tau_{\mathrm{lat}} + l\omega$ Here, τ_{lat} is the latency, *i.e.*, the startup time required for each message sent; ω denotes the inverse of the bandwidth, and the bandwidth itself, $1/\omega$, represents the data transfer rate, typically measured in Mbytes/s. Because fairly large differences in latencies and bandwidths are seen across modern communication networks, and because improvements in network performance have not kept up with improvements in processing power, developers must be cognizant of the cost of sending data between nodes when writing parallel applications and structure their algorithms accordingly.

Table 1 Some common network topologies and parameters determining their cost and performance. n_{degree}: the degree of the network; n_{sw}: the number of simple switching elements; n_{hop}: the upper bound for the number of switches through which data must pass; B_{c}: the bisection width; p: the number of processors

Topology	n_{degree}	n_{sw}	n_{hop}	B_{c}
Ring	2	p	$p/2 + 1$	4
2D Mesh	4	p	$2\sqrt{p} - 1$	$2\sqrt{p}$
2D Torus	4	p	$\sqrt{p} + 1$	$4\sqrt{p}$
3D Torus	6	p	$\frac{3}{2}p^{1/3} + 1$	$4p^{2/3}$
k-ary Fat Tree	$2k$	$\frac{p}{k}\left(\log_k p - \frac{1}{2}\right)$	$2\log_k p - 1$	p
Crossbar	p	1	1	p

2.4 Parallel disk I/O

The slowest medium that is used to hold data in routine parallel computations is the disk. Although disk access is very slow compared to other components in the memory hierarchy the vast size of modern file systems occasionally make its usage desirable, as well as the fact that disk is the only medium available for permanent data storage. Obviously as the processing power increases with the number of compute units the aggregate data transfer rates for disk access needs to keep pace. The simplest way to achieve this is to provide individual nodes with local disks. While this ensures that the aggregate disk access rates grow linearly with the number of compute units, it has the disadvantage that the files are directly visible only within the node. As a result this approach is suitable only for scratch files.

The need for file systems with high bandwidth and visibility to all nodes has driven the development of a completely new class of file systems. These are essentially clusters of I/O nodes that are connected to the compute nodes with a switch. With this infrastructure it is possible to give every compute node access to all the disks and therefore provide an identical view of the file system to every compute node while also providing high aggregate bandwidth. In order for this high bandwidth to be realized it is important to prevent all compute nodes from accessing the same physical disk at the same time. A common strategy to achieve this is to partition a single file into a number of blocks, typically in the order of 1–64 MByte per block, and have consecutive blocks stored on different I/O nodes. This way as long as different compute nodes access different parts of the file the data traffic is spread across all physical disks. The details of the physical file storage exposed to the programmer are different depending on the particular storage solution. However, the performance crucially depends on the way by which access to the files is orchestrated. In practice this kind of solution may be implemented purely in software utilising the switching and storage hardware of the compute cluster, or it may involve dedicated hardware. Parallel file systems of this kind include Lustre,[16] GPFS,[17] and Panasas.[18]

3. Parallel programming paradigms

Parallelism offers great opportunities to extend the range of feasible applications but it also adds an extra layer of complexity to implementing any algorithm. In quantum chemistry where the basic algorithms tend to be complex in any case it becomes imperative to make clear choices to master these additional complexities. A coherent set of such choices, a paradigm, provides a framework within which parallelism can be expressed such that well defined results can be obtained. Parallel programming has seen a continued development of new paradigms and refinements of old paradigms as new hardware capabilities and new insights evolve. In the following

sections we discuss the now traditional message passing approach, virtual shared memory programming as supported for example by GA/ARMCI, as well as the increasingly ubiquitous multi-threaded approaches such as OpenMP and pthreads which thrive on the current directions of hardware development. Obviously to achieve optimal performance a modern paradigm will typically adopt components of all these approaches, but great care is required to combine choices to provide a manageable framework.

3.1 Internode communication

Fundamental to parallel programming is the need to communicate information between different threads of execution. In particular when communicating between nodes, where the shared memory mechanism cannot be used, special libraries are used to provide the transport mechanism. The first widely adopted paradigm for this was message passing. In this approach one process calls a send routine to transmit the data and the corresponding recipient process calls a receive routine to collect the data. As both the sending and receiving processes actively participate in the data transmission this is referred to as two-sided communication. This protocol is the basis for early parallel libraries such as TCGMSG,[19] PVM[20] and the current MPI.[21] The availability of standard libraries and the conceptual simplicity made this model attractive and has led to widespread adoption. Many quantum chemistry codes have adopted MPI.[22–35] Despite it's widespread use the message passing model has two severe limitations that result from the two-sidedness. Firstly, the requirement for a send call to be matched by a receive call introduces a synchronisation between the sending and receiving processes. This leads to load imbalance issues if the tasks executed by the processes differ significantly in the time required to complete them. Secondly, the fact that both processes need to participate explicitly to transfer a single message leads to significant design issues if complex communication patterns are to be implemented. Other quantum chemistry codes have adopted alternative communication paradigms.[11–12,35–41]

The limitations of the message passing paradigm can be overcome by providing facilities for a process to access data of another process without that process's direct involvement. These facilities have become known as remote memory access. Due to similarities with shared memory approaches it is also referred to as virtual shared memory. In this approach a process that wants to store data calls a put routine and to retrieve data a get routine. Accessing the remote node memory is handled without the remote processes's explicit involvement, hence this is also referred to as one-sided communication. A variety of libraries are available that provide support for this paradigm, such as GA/ARMCI,[42] DDI,[43,44] Linda,[45] PPIDD,[46] and MPI-2.[47] It should be noted that MPI-2, although it extends MPI among other things with one-sided communication primitives, leaves it up to the programmer to manage the exact data distribution and every single data transmission. Other libraries, *e.g.* GA/ARMCI, can hide much of these details but provide facilities for fine grained control where the programmer really needs it.

The development of ARMCI is driven by the need to support a global-address space programming model on distributed-memory computers in the context of regular or irregular distributed data structures, communication libraries, and compilers. Initially developed as the communication substrate for Global Arrays, a "shared-memory" programming interface that greatly simplifies the development of programs for parallel architectures, it has now been used to manage inter-processor communication in a variety of programming models such as GPSHMEM,[48] Co-Array Fortran, and others.[49]

ARMCI has been ported to several DOE leadership class machines such as BlueGene/P and Cray XT4 supercomputers, and ARMCI has been continuously developed to ensure its scalability and low overhead. ARMCI has been extended to support flexible process groups to enable a more dynamic task management system.

In order to ensure quick porting of ARMCI to new supercomputers, researchers have implemented ARMCI using MPI-2 Dynamic Process Management, and two-sided communication calls. This enables deployment of ARMCI on a new supercomputer once MPI, the most popular programming model, is deployed.

In addition to the ease of deployment, this work is related to the recently initiated MPI Forum activity on an improved MPI one-sided standard. Despite numerous implementations of the existing MPI one-sided standard over the last eleven years, it has been perceived as too restrictive for actual application use.[50] The current ARMCI implementation, though less efficient than an implementation closer to the hardware, demonstrated performance competitive with existing MPI one-sided calls, while supporting more flexible progress rules and a richer set of functionality than the original MPI one-sided model.

3.2 Threading and fine grain concurrency

The message-passing approach for distributed memory machines described in section 3.1 can be employed for communication between processes running on different nodes as well as between processes running on the same shared-memory node. In the latter case, the message-passing communication library can take advantage of the fact that memory is shared and can provide very high performance. An alternate approach providing even higher performance is to use concurrent execution contexts within a single process to exploit parallelism within the node. Each of these execution contexts is referred to as a thread, and all threads within a process typically have access to the full memory of the process. For large-scale parallelism discussed in this article, multi-threading parallelism is often employed in addition to message-passing parallelism in what is referred to as a hybrid approach. The use of multi-threading for node parallelism has the advantages that explicit data transfers and the associated performance costs can be avoided, thread synchronization can be accomplished with very low overhead, and the memory footprint is smaller as it is not necessary to replicate certain data structures among the threads. As a result of the performance advantages, much finer-grained parallelism can be obtained using multi-threading than message-passing.

However, a potential disadvantage of multi-threading is the increased programming complexity. Thus, programmers can accidentally introduce deadlocks and race conditions into a program in subtle ways. Also, when using a hybrid multi-threading/message-passing approach, it is necessary to write code explicitly for thread parallelism as well as for message-passing parallelism.

There are two primary ways to implement multi-threading in a program. One technique is to use a library-based approach, the most common of which is POSIX threads (Pthreads),[51] where the programmer explicitly specifies a function to be executed in a concurrent thread. Alternatively, the threading can be done semi-automatically by the compiler, for example by using OpenMP.[52] In this case the programmer annotates the program using pragmas that specify how the compiler is to parallelize the program.

The number of concurrent threads of execution possible on computation nodes is increasing at a tremendous rate. In addition to the growth in the number of cores on a chip discussed in section 2.1, it is becoming increasingly common for each of these cores to implement multiple hardware execution contexts that share a core's resources. In doing so, it is possible to keep the core busy when one execution context has stalled (for instance, because it is waiting for data to be loaded from memory) by executing an instruction from the other context. As a result, it is important for applications to expose as much parallelism as possible to be able to efficiently utilize large-scale parallel machines. Hybrid programming methodologies have already been employed for a number of years in parallel quantum chemistry programs. For example, an early application applied a hybrid approach to the two-electron integral transformation in an MP2 calculation.[53] Message-passing is limited to coarser-grained parallelism, and multi-threading can be used to expose parallelism at a fine-grained level. As a result, hybrid programming methodologies will become increasingly critical to efficient utilization of large-scale machines.

3.3 General purpose computing on graphics processing units

General Purpose computing on Graphics Processing Units (GPGPU), like multi-threading, provides for relatively fine-grained parallelism compared to message-passing; however, the GPUs do not share an address space with the processes running on the node. Because the memory transfers between the node's main memory and GPU memory can be accomplished with higher performance than network I/O, GPGPU programming supports parallelism granularities somewhere between the message-passing and multi-threading approaches. Programs utilizing GPGPU have elements that resemble both multi-threading approaches (a function that performs the computation kernel is specified) and message-passing (all of the data required by the GPU computation kernel must be explicitly specified by the programmer). An obstacle to programming GPUs has been that the programming methods are often hardware specific. Recently, however, the OpenCL[54] specification and implementations thereof have become available, providing a portable royalty-free environment for GPGPU programming.

3.4 Implications on quantum chemistry codes

The parallel efficiency ξ is an important measure of the successful parallel implementation and scalability of a code. ξ can be expressed in terms of the effective parallel fraction of execution time η and the number of CPUs (n_{cpu}).

$$\xi = \frac{t_{\mathrm{total}}}{n_{\mathrm{cpu}} t_{\mathrm{wall}}} = \frac{1}{n_{\mathrm{cpu}}(1-\eta)+\eta} \qquad (1)$$

To achieve a good efficiency of $\xi > 90\%$ on 1000 CPUs, η must exceed 99.999%-including I/O and communication. Another important measure is the scaling of I/O and communication volume as the number of CPUs increases. Ideally, accumulated network and I/O bandwidth increase linearly with the number of CPUs so that algorithms with an associated I/O and communication data volume increasing at most linearly with the number of CPUs display constant or decreasing data transfer overhead. While this assumption approximately holds, topology-dependent, for the network (*cf.* Table 1), this is rarely the case for I/O as the I/O channels are rapidly saturated. The current peak bandwith shared by the entire computer system on parallel filesystems is at $\sim$200 Gbyte/s.[55] On widely available computer installations one cannot expect more than ideally about a few Gbyte/s. With the advent of multi-core architectures even the bandwidth for local disk access is easily saturated. Finally, the compute density, *i.e.* the number of floating point operations executed per data item, is decisive, since both multi-core architectures and especially GPUs are increasingly memory bandwidth limited. The compute density is closely connected to the achieved percentage of peak performance. Further key issues are memory consumption, data access pattern, replicated *versus* distributed data structures, static *versus* dynamic load balancing. Hence, coarse grain parallelization (multiple instruction multiple data or MIMD) is popular as it allows to keep all the benefits of vectorization and retains most of the compute kernels. It is prudent to avoid disk storage altogether, except in small computer clusters. Clearly, the algorithms developed so far are not well-suited for graphic or accelerator cards which exploit tightly coupled vectorization and parallelization and are highly sensitive to the memory access patterns that are used with a single instruction multiple data (SIMD) programming paradigm. To take full advantage of these devices, it is desirable to have only a few small, simple kernels and a large ratio of work load to data transfer volume from/to the device. This requires a departure from the current complex compute and data movement intensive kernels and, consequently, a complete rewrite of the kernels to take into account specific hardware constraints.

4. Parallel computational chemistry algorithms

The field of quantum chemistry comprises a wide variety of methods due to different choices of energy expressions. The particular choice of energy expression has consequences for the choice of method to evaluate the energy of a chemical system and its properties. In practice the energy expressions separate broadly into two classes due to the physics they describe as well as strategies suitable for their implementation.

In the first category there are the effective one-electron models, such as Hartree–Fock (HF) and Density Functional Theory (DFT), which aim to calculate accurate one-electron functions and employ approximate energy expressions to provide total energies thereof. In the second category are the methods that are known as post Hartree–Fock methods which aim to accurately represent the two-electron interaction. Usually the latter are derived from extended choices for representing the wavefunction using many Slater determinants, for example coupled cluster (CC) and multi-reference configuration interaction (MRCI).

From the way these methods approach the physics of many electron systems important implementation considerations follow. Assume that the desired solutions are represented as linear combinations of some chosen set of functions, a basis set, of dimension N which grows linearly with the number of atoms. The effective one-electron models then provide solutions that require $O(N^2)$ variables to store and cost typically $O(N^3)$ operations to manipulate. By contrast the post Hartree–Fock methods require at least $O(N^4)$ variables assuming only single and double excitations are included in the wavefunction and typically $O(N^6)$ operations. These characteristics are central to algorithm choices, for example storing the $O(N^4)$ 2-electron integrals in Hartree–Fock leads to a major storage bottleneck as it would dominate the required space but in coupled cluster singles-doubles (CCSD) it only doubles the memory requirements.

It is also important to note the impact of O(N) methods in this context. These methods are based on the realization that the total energy is a size extensive property, *i.e.* increases linearly with system size for very large systems which implicitly requires the long range interactions in the system to decay rapidly.[56] Indeed linear scaling methods have been formulated for a variety of energy expressions, ranging from Hartree–Fock and DFT,[57–62] perturbation theory,[63–69] coupled cluster theory[70–72] in recent years, and these methods are available in codes such as CONQUEST,[73] ONETEP,[28] FreeON,[74] Q-Chem,[31] MOLPRO,[39] MPQC,[12] and others. Obviously, linear scaling methods have the advantage that they can obtain the same results on much smaller compute resources than other methods. However, for large scale parallel computations they do pose specific challenges of their own. For example, if linear scaling techniques are applied only to the work involved in a calculation, the applications will become data intensive causing major communication bottlenecks. This is in part aligned with common wisdom that parallelizing little work over many compute units is much harder than parallelizing a lot of work. Therefore linear scaling techniques have to be cast in a broader sense which includes linear scaling in work, data and communication, to be successful. Ultimately, if formulated in such a way they may actually help parallelization by eliminating data exchange to compute interactions between remote atoms and reducing data storage. This expectation is realistic given that the communication and memory capabilities for future compute platforms are projected to become worse relative to the compute capability. However, it is still premature at present to discuss in detail effective ways to implement large scale parallel linear scaling algorithms.

In the following sections different approaches are described for the different domains. We proceed by describing the effective 1-electron methods first as they are the oldest, the simplest, the most widely used methods, and their solution is typically a prerequisite for the post HF methods. The description of the local basis set HF and DFT methods are combined as their implementation is very similar. This is followed by a description of the plane wave basis set DFT method which differs due to the inherent periodicity of the basis set, the target applications, and hence the implementation. This is followed by post HF methods ranging from single reference based coupled cluster (CC) and many-body perturbation theory (MBPT) to multi-reference multi-configuration self-consistent-field (MCSCF) and MRCI.

4.1 Self-consistent field and density functional theory theories

An essential core functionality in any suite of quantum chemistry programs is the Hartree–Fock (HF) or self-consistent field (SCF) method. This basic functionality forms the basis for higher level electronic structure theories like Møller–Plesset perturbation theory to nth order (MPn), and CC, to name a few. The SCF or HF method assumes that the exact, N-body wave function of the system can be approximated by a single Slater determinant.[75,76] Using the variational principle, a set of coupled one-electron HF equations can be formally written as

$$(-\tfrac{1}{2}\nabla^2 + V_C(\mathbf{r}))\psi_i(\mathbf{r}) + \int V_X(\mathbf{r}, \mathbf{r}')\psi_i(\mathbf{r}')d\mathbf{r}' = \varepsilon_i\psi_i(\mathbf{r}) \quad (2)$$

where the first term on the right hand side is the kinetic contribution of the electrons, $V_C(\mathbf{r})$ represents the classical Coulomb potential consisting of nuclear and Hartree contributions, $V_X(\mathbf{r}, \mathbf{r}')$ is the non-local exchange contribution and $\psi_i(\mathbf{r})$, ε_i represent the orbitals and orbital energies, respectively. Due to the coupled nature of the HF approach, the above integro-differential equations have to be solved using an iterative procedure until self-consistency is achieved.

Density functional theory (DFT) provides an alternative approach to the many-electron problem and has been shown to be broadly applicable to a wide range of chemical and materials systems with an excellent mix of efficiency and accuracy. Analogous to the HF equations, DFT is firmly rooted in the Kohn–Sham (KS) equations.[77–81] The one-electron KS equations are essentially identical to the HF equations (eqn (2)) with the difference lying in the way the exchange term $V_X(\mathbf{r}, \mathbf{r}')$ is treated in the two approaches. In DFT, the complexities of the many-electron interactions are subsumed into the density dependent exchange–correlation term $V_{XC}(\mathbf{r}, \mathbf{r}')$, which is generally non-local. Because of the mathematical similarities of the HF and KS equations, the solutions to the KS equations are also obtained using a self-consistent field procedure.

Since the exact form of the exchange–correlation is unknown in terms of the density, a number of approximations of increasing complexity have been developed over the years. These range from pure density based forms like the LDA and GGAs (PW91, PBE96, BLYP, *etc.*) to more complicated popular hybrid forms (for example, B3LYP, PBE0, Minnesota functionals, CAM-B3LYP, *etc.*) which also contain a HF

exchange contribution. Because of this overlap, HF and DFT codes typically share a lot core capability. Since a detailed discussion of different XC functionals is beyond the scope of this perspective paper, we refer the interested reader to more comprehensive references on the subject.[78,81–86]

Both HF and KS equations can, in principle, be solved numerically. However, it is standard practice to expand the solutions in terms of a basis set expansion. Over the years a number of different basis set approaches have been developed that can be broadly divided into two categories: local basis set and plane-wave approaches. Both approaches have their advantages and disadvantages based on the system being treated. We will not discuss the hybrid basis approaches[30,87] in this perspective.

4.1.1 Local basis set based approaches. Within the context of a local basis set expansion, the HF and KS equations can be re-expressed as a set of matrix equations,[76,82,84]

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon} \tag{3}$$

where **F**, **C**, **S**, $\boldsymbol{\varepsilon}$ represent the Fock, coefficient, overlap and diagonal orbital energy matrices, respectively. The problem of solving the HF or KS equations boils down to solving a generalized eigenvalue problem. One can write the Fock matrix in a general way so as to incorporate both HF and KS formalisms within a single framework as,

$$F_{\mu\nu} = H_{\mu\nu}^{\text{core}} + G_{\mu\nu}^{J} + \alpha G_{\mu\nu}^{K} + \beta G_{\mu\nu}^{\text{X-DFT}} + \gamma G_{\mu\nu}^{\text{C-DFT}} \tag{4}$$

where $H_{\mu\nu}^{\text{core}}$ is the one-electron core part composed of the kinetic and ion-electron pieces, $G_{\mu\nu}^{J}$ represents two-electron Coulomb interaction between the electrons, $G_{\mu\nu}^{K}$ represents the two-electron explicit exchange contribution, $G_{\mu\nu}^{\text{X-DFT}}$ is the DFT exchange contribution and $G_{\mu\nu}^{\text{C-DFT}}$ is the DFT correlation contribution, respectively. The mixing coefficients α, β, γ help span the pure HF and DFT limits. With $\alpha = 1$, $\beta = 0$, $\gamma = 0$ one gets the pure HF limit, whereas the pure DFT limit is achieved with $\alpha = 0$, $\beta = 1$, $\gamma = 1$. The phase space of non-local hybrid-DFT forms discussed earlier is covered with $\alpha < 1$, $\beta < 1$, $\gamma = 1$. The one-electron core part is calculated once for a given basis whereas the two-electron terms depend on the density matrix as well as a set of two-electron integrals and can be written in a unified way as,

$$G_{\mu\nu}^{JK} = \sum_{\lambda\sigma} D_{\lambda\sigma}\left[(\mu\nu|\sigma\lambda) - \frac{\alpha}{2}(\mu\lambda|\sigma\nu)\right] \tag{5}$$

where $D_{\lambda\sigma}$ is the density matrix. As we shall discuss later, the DFT exchange and correlation pieces are conveniently calculated numerically on a grid. Since the Fock matrix itself depends on the density matrix (or expansion coefficients), the set of matrix equations has to be solved iteratively.

Computationally the time consuming steps in any standard implementation can be summarized as follows:

(a) Computation of the two-electron integrals

(b) Construction of the two-electron part of the Fock matrix which in turn depends on the two-electron integrals and the density matrix

(c) Computation of the exchange–correlation (XC) contribution to the Fock matrix in the case of KS-DFT

(d) Diagonalization of the Fock matrix

(e) Computation of the density matrix using the molecular orbital coefficients

Of all the pieces that constitute the Fock matrix, the computation of the two-electron integrals represent the most expensive step and considerable amounts of effort have been invested in finding more efficient ways of evaluating it.[88] Two computationally significant steps are involved: evaluation of the two-electron integrals $(\mu\nu|\sigma\lambda)$, contraction of the integrals with the one-electron density matrix $D_{\lambda\sigma}$. Because the two-electron integrals are functions only of the atom positions and the basis set of choice, they can be considered as constants throughout the solution of the HF or KS equations. Traditionally this fact has been exploited by evaluating the integrals only once and storing them on disk. This technique is also known as the "conventional Fock matrix construction" approach. Disk space requirements and I/O bandwidth limitations can be reduced by exploiting simple screening techniques such as not storing integrals with values below a given threshold. In real applications, however, the size of these integral files can be very large (~hundreds of Gbytes) and pose a calculation bottleneck.

In the light of this, Almlöf *et al.*[1] proposed an on-the-fly strategy where the two-electron integrals are computed on demand. This technique is known as the "direct Fock matrix construction" approach. Critical to the performance of this approach is the efficient evaluation of all required integrals including suitable screening techniques. Although the work involved in this approach is substantial, the actual time of the calculation can be smaller because of the CPU-I/O speed ratio. In fact, this is the approach of choice for large systems.

The semi-direct approach[89] combines the conventional and direct approaches and is based on the realization that certain groups of integrals and density matrix elements tend to be more significant. This small set of integrals is calculated once as in the conventional approach while the rest are evaluated on demand as in the direct approach.

In all three approaches described above most of the savings are achieved by exploiting the Schwarz-inequality. Integrals of the kind $(\mu\nu|\sigma\lambda)$ decay exponentially with the distance between the atoms on which basis functions μ and ν are centered and likewise for basis functions σ and λ. It has been realized that using an auxiliary basis to represent the density can lead to an $O(M^2)$ algorithm, where M is the number of auxiliary basis functions. These resolution of identity $(\text{RI})^2$ or Cholesky decomposition (CD)[90] or charge density fitting approaches[91] are the most efficient of all the schemes discussed so far. They can be made even more efficient as they are also amenable to large scale integral prescreening.[58,59] Efficient Fock matrix construction approaches based on the multipole approximations should also be noted in this context.[57,60]

Selection of the most suitable algorithm for computing integrals depends on the system. For example, the variation in the number of iterations for SCF convergence can affect the balance between direct and conventional techniques. The semi-direct approach is more efficient for medium sized molecules, however for relatively large molecules it has no advantages compared with the direct approach. For very large systems, with thousands of basis functions a conventional

calculation is prohibitive due to disk space requirements to store the integrals. In these systems, by virtue of explicit elimination of four-center integrals and efficient integral pre-screening, both RI and CD approaches are faster than the direct approach. The exact balance between the cost of the steps (a,b,c,d,e) depends on the size of the calculation, the effect that screening techniques can have for a given calculation and the ratio between data reuse and data recomputation. Nevertheless, to achieve good scalability all these components need to be parallelized. The following paragraphs will discuss ways to parallelize these steps. Since this is a perspective paper, the emphasis will be on approaches where we have first hand experience.[11,12,36,92–95] These concepts and strategies are also exploited in many other codes[96–100] with slight variations in the details.

Parallelization of the two-electron contribution. Both the evaluation of the one-electron integrals and the two-electron integrals scale ultimately as $O(N^2)$ for large molecules. However, with practical calculations on finite size systems the scaling is $a \cdot O(N^{2+\delta})$ where both the prefactor a and the exponent δ depend on the formal scaling. Since the formal scaling of the one-electron integrals is $O(N^3)$, for the nuclear attraction integrals, and the formal scaling of the two-electron integrals is $O(N^4)$ the latter are far more expensive to evaluate and therefore we discuss their parallelization. In most parallel implementations, the relevant matrices are either handled in a replicated or distributed fashion or some combination of the two.[13,101–104] We note that these strategies can be used independent of the integral generation algorithms discussed earlier.

The replicated data approach offers a straightforward way to achieve task parallelism and low communication. In this approach each processor maintains a complete copy of the necessary data. For the two-electron contribution this means replicating the density (**D**) and Fock (**F**) matrices over all the processors and the integral quartets are constructed in blocks which are assigned to each processor. In other words, the do loops only access a unique integral list of quartets. After a integral quartet has been looped over, the results are accumulated in the partial **F** matrix associated with the local processor. These partial results are then consolidated into a complete **F** matrix by using a global sum operation. Assuming a reasonable work load distribution and an efficiently implemented global summation operation, the replicated approach is perfectly parallelizable. It is also efficient for large systems because of the amount of parallel work available (especially if the integrals are evaluated using the direct approach). It, however, runs into a potential $O(N^2)$ memory bottleneck because two symmetric matrices (and four in the case of unrestricted calculations) of dimension N, where N is the number of basis functions, have to be stored on each of the processors *i.e.* $2*8*N^2$ bytes of memory for restricted and $4*8*N^2$ bytes of memory for unrestricted calculations. With the available memory on present day systems, this algorithm can be used for very large systems but the available local memory per processor is a limiting factor.

The memory bottleneck of the replicated data algorithm may be avoided by distributing the density and Fock matrices. This way each process only handles a portion of the data and consequently puts a smaller constraint on the available local memory per processor. Over the years a number of distributed data algorithms have been developed.[102,103] Although slightly different they share a common feature in that all offset the cost of accessing a density or Fock matrix element by smart data reuse. The essence lies in grouping the basis functions into atom blocks and distributing the **D** and **F** matrices, based on this partitioning, over all the processors. For example, for N basis functions partitioned into M blocks of N_b functions each, the density and Fock matrices are each divided into blocks of $\sim {N_b}^2$ elements and the two-electron integrals are also partitioned into blocks of $\sim {N_b}^4$ integrals each. The partitioned matrices and the blocks of integral quartets are each assigned to each processor. Since the communication cost is only quadratic compared with the quartic computation cost, small block sizes are good enough to keep the computation cost dominant.

By virtue of their design, distributed data algorithms require careful handling of communications as the Fock matrix is constructed. This makes it challenging to implement. To evaluate the contribution to the full **F** matrix, the assigned processor must have the necessary **D** matrix blocks available in its local memory. Requests have to be sent out for blocks that are not resident to the processor. This is optimally done by pre-fetching the necessary blocks to avoid processor idle time. Once the partial **F** blocks have been constructed, they are sent to the appropriate processor. Since computation and communication are overlapped on each processor, this requires care to avoid a dead-lock situation between processors. This approach has been efficiently implemented using one-sided access.[102,103] The Global Array (GA) Toolkit[105] provides a simple and seamless way to achieve this.

The load balancing can also be tricky in this algorithm because a simple model of distributing the integrals equally amongst the processors leads to poor load balancing problems. This is because the computational effort required for the two-electron integrals depends on the angular momentum and the contraction of the basis functions. For example, highly contracted basis functions require a larger number of floating point operations. Different schemes have been devised to mitigate this.[102,103]

Using a performance model, it can be shown that the computation to communication ratio for the distributed approach can be written as,[13,104]

$$\frac{T_{\mathrm{comp}}}{T_{\mathrm{comm}}} \approx \frac{\beta\left(\frac{N_{\mathrm{basis}}}{N_{\mathrm{atom}}}\right)^4}{8\left(t_0 + t_1\left(\frac{N_{\mathrm{basis}}}{N_{\mathrm{atom}}}\right)^2\right)} \qquad (6)$$

where β is the time required to compute an integral (assuming perfect load balancing), N_{atom}, N_{basis} represent the number of atoms and basis functions and t_0, t_1 represent the latency and transmission cost for a floating point number. The important point is that the above ratio is independent of the number of processors indicating that the algorithm is very scalable. Distributed data algorithms have also been successfully used in CPHF implementations.[106]

Parallelization of the DFT XC contribution. The other important component of the Fock matrix is the XC contribution in a DFT calculation which involves a numerical integration over a grid to calculate the matrix elements and the energy. Even though this term scales as $O(N^3)$ with the number of basis functions, it becomes a computationally significant step for large-scale DFT calculations. The quadrature is carried out on a superposition of atom-centered grids which are independent of each other and since the size of the grid grows with the size of the molecule, there is ample parallelizable work available. In typical parallel implementations, the atomic grid points are distributed as independent batches (albeit with proper load balancing) and the density matrix is broadcast over the processors.[97,107,108] The accumulated partial results on each processor are then consolidated using a global sum operation. This approach can also be generalized in a straightforward fashion to deal with subgroups of independent grid points for better load balancing.

Parallelization of the Fock matrix diagonalization. The diagonalization of the Fock matrix is a crucial step in the self-consistent solution procedure. Even though the parallel efficiency of the diagonalization step increases with matrix size, the $O(N^3)$ scaling dominates for large N. As we have already seen, the Fock matrix construction can be parallelized efficiently by either distributing or replicating the matrix elements over the processors. However, for large processor counts, it is the diagonalization of the Fock matrix that dominates because it is not easily parallelizable, and because its $O(N^3)$ scaling is worse than the $O(N^2)$ of the Fock matrix construction. As a result, a lot of work has been focused over the last several years to develop efficient diagonalization-free algorithms with better scaling characteristics.[59,61,109–112] Traditional HF/DFT implementations typically utilize parallel linear algebra libraries like Global Arrays[105] or ScaLAPACK[113] for matrix operations like diagonalization, addition, multiplication and scalar products.

In Fig. 3, an example (C_{240}) of timings *versus* the number of CPUs is shown for the Gaussian basis set HF/DFT implementation in NWChem. The calculations were performed using the PBE0 hybrid exchange–correlation functional and the 6-31G* basis with a total of 3600 basis functions for the whole system. The integral evaluations were performed using the direct approach and Fock matrix replication. As has been discussed the two-electron part of the Fock matrix lends itself to efficient parallelization with almost perfect scaling through 4096 CPUs. The local exchange–correlation component also exhibits good scaling, however the increase in communication costs limits the efficiency to approximately 1000 CPUs. As the cost of Fock matrix construction drops for large processor counts the performance is impacted by the diagonalization step which is not as easily parallelizable.

4.1.2 Plane wave based approaches. In the plane-wave approach the electronic orbitals are represented in terms of a plane-wave basis set. Some key features of this approach is that periodic boundary conditions (PBC) can be incorporated in a seamless fashion and Fast Fourier Transform (FFT) algorithms can be used for fast calculations of total energies

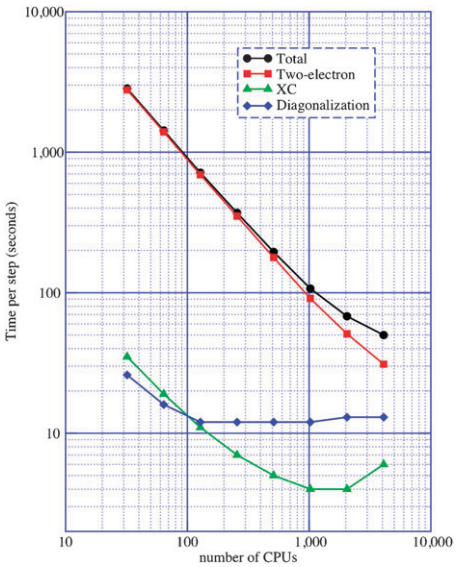


Fig. 3 Overall and component timings from Gaussian basis set based calculations of C_{240}. The calculations were performed on the Chinook computer system at the MSCF at EMSL.

and forces. However, the plane wave basis sets do have an important shortcoming: their inefficient description of the electronic wavefunction in the vicinity of the atomic nucleus or core region. Valence wave functions vary rapidly in this region, and much more slowly in the interstitial regions (or bonding regions). The accurate description of the rapid variation of the wavefunction inside the atomic or core region would require very large plane wave basis sets.

The pseudopotential plane wave (PSPW) method can be used to resolve this problem.[114–117] In this approach the fast varying core regions of the atomic potentials and the core electrons are removed or pseudized and replaced by smoothly varying pseudopotentials. The pseudopotentials are constructed such that the scattering properties of the resulting pseudoatoms are the same as the original atoms.[118,119] The rationale behind the pseudopotential approach is that the changes in the electronic wavefunctions during bond formation occur only in the valence region and therefore, proper removal of the core from the problem should not affect the prediction of bonding properties of the system. The projected augmented plane-wave (PAW) developed by Blöchl is a further enhancement to the pseudopotential in that it addresses some of the shortcomings encountered in traditional PSPW approach. Since the main computational algorithms are essentially the same in the two approaches, we will not specifically discuss the PAW approach and refer the reader to comprehensive reviews.[120–125]

Plane-waves are natural for solid-state applications, since crystals are readily represented using periodic boundary conditions. In terms of plane-waves, the molecular orbitals $\psi_i(\mathbf{r})$ are represented as

$$\psi_i(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} \sum_{\{\mathbf{G}\}} \psi_i(\mathbf{G}) e^{i\mathbf{G}\cdot\mathbf{r}} \qquad (7)$$

where Ω is the volume of the unit cell. Since the system is periodic, the plane-wave expansion must consist of only the

plane-waves $e^{i\mathbf{G}\cdot\mathbf{r}}$ that have the periodicity of the lattice, which can be determined using the following constraint

$$e^{i\mathbf{G}\cdot(\mathbf{r}+\mathbf{L})} = e^{i\mathbf{G}\cdot\mathbf{r}} \tag{8}$$

where **L** is the Bravais lattice vector and **G** represents the reciprocal lattice vectors. In typical plane-wave calculations, the plane-wave expansion is truncated in that only the reciprocal lattice vectors whose kinetic energy is lower than a predefined maximum cutoff energy,

$$\tfrac{1}{2}|\mathbf{G}|^2 < E_{\text{cut}} \tag{9}$$

are kept. Transformations between the real and reciprocal space representations are performed using efficient Fast Fourier Transform (FFT) algorithms.

In the plane-wave framework, the HF/KS equations are typically solved using a conjugate gradient algorithm or, for dynamics, a Car–Parrinello molecular dynamics (CPMD) algorithm that requires many evaluations of $H\psi_i(\mathbf{r})$ along with maintaining orthogonality

$$\int_{\Omega} \psi_i(\mathbf{r})\psi_j(\mathbf{r})\mathrm{d}\mathbf{r} = \delta_{ij} \tag{10}$$

Similar FFT-based solution methods are implemented in a number of widely distributed first principles simulation software packages such as NWChem.[11]

For hybrid-DFT, the Hamiltonian operator H may be written as[81]

$$\begin{aligned} H\psi_i(\mathbf{r}) = &\begin{pmatrix} -\tfrac{1}{2}\nabla^2 + V_L(\mathbf{r}) + V_{\text{NL}} + V_H[\rho](\mathbf{r}) \\ +(1-\alpha)V_x[\rho](\mathbf{r}) + V_c[\rho](\mathbf{r}) \end{pmatrix}\psi_i(\mathbf{r}) \\ &- \alpha\sum_j K_{ij}(\mathbf{r})\psi_i(\mathbf{r}) \end{aligned} \tag{11}$$

where the one electron density is given by

$$\rho(\mathbf{r}) = \sum_{i=1}^{N_e} |\psi_i(\mathbf{r})|^2 \tag{12}$$

The local and non-local pseudopotentials, V_L and V_{NL}, represent the electron-ion interaction. The Hartree potential V_H is given by

$$\nabla^2 V_H(\mathbf{r}) = -4\pi\rho(\mathbf{r}) \tag{13}$$

The local exchange and correlation potentials are V_x and V_c, and exact exchange kernels K_{ij} are given by

$$\nabla^2 K_{ij}(\mathbf{r}) = -4\pi\psi_i(\mathbf{r})\psi_j(\mathbf{r}) \tag{14}$$

During the course of a total energy minimization or Car–Parrinello simulation, the electron gradient $H\psi_i$, (eqn (11)) and orthogonalization (eqn (10)) are evaluated many times (*i.e.* >10 000 for Car–Parrinello), and hence need to be calculated as efficiently as possible. For a pseudopotential plane-wave calculation the main parameters that determine the cost of an electronic gradient are N_g, N_e, N_a, and N_{proj}, where N_g is the size of the three-dimensional FFT grid, N_e is the number of occupied orbitals, N_a is the number of atoms, and N_{proj} is the number of projectors per atom. Summaries of the computational costs for each of the constituent parts of electron gradient (and orthogonality) are given in Fig. 4. The major parts of the electronic gradient in order of increasing asymptotic cost are (note that conventional DFT does not compute the exact exchange term; $\alpha = 0$ in eqn (11):

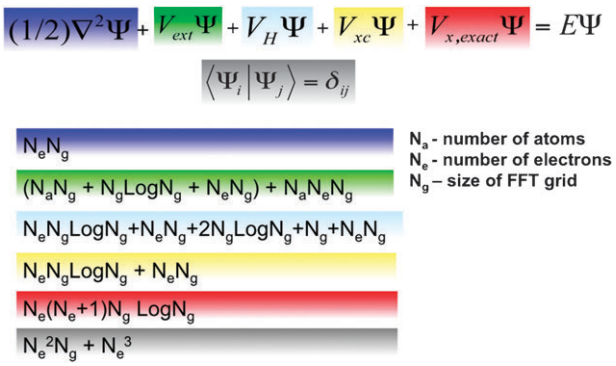


Fig. 4 Operation count in a plane-wave DFT simulation.

(a) The Hartree potential V_H, including the local exchange and correlation potentials $V_x + V_c$. The main computational kernel in these computations is the calculation of N_e three-dimensional FFTs.

(b) The non-local pseudopotential, V_{NL}. The major computational kernel in this computation can be expressed by the following matrix multiplications: $\mathbf{W} = \mathbf{P}^{\mathbf{t}}*\mathbf{Y}$, and $\mathbf{Y_2} = \mathbf{P}*\mathbf{W}$, where **P** is an $N_g \times (N_{\text{proj}}*N_a)$ matrix, **Y** and $\mathbf{Y_2}$ are $N_g \times N_e$ matrices, and **W** is an $(N_{\text{proj}}*N_a) \times N_e$ matrix. We note that for most pseudopotential plane-wave calculations $N_{\text{proj}}*N_a \approx N_e$.

(c) Enforcing orthogonality. The major computational kernels in this computation are following matrix multiplications: $\mathbf{S} = \mathbf{Y}^{\mathbf{t}}*\mathbf{Y}$ and $\mathbf{Y_2} = \mathbf{Y}*\mathbf{S}$, where **Y** and $\mathbf{Y_2}$ are $N_g \times N_e$ matrices, and **S** is an $N_e \times N_e$ matrix.

(d) And when exact exchange is included, the exact exchange operator $\Sigma K_{ij}\psi_j$. The major computational kernel in this computation involves the calculation of $(N_e + 1)*N_e$ three-dimensional FFTs. The computation of the exact exchange operator, in which $O(N_e^2)$ independent Poisson equations must be solved, is by far the most demanding term in a pseudopotential plane-wave hybrid-DFT calculation.

Parallelization of plane-wave HF/DFT. There are several ways to parallelize a plane-wave HF/DFT program.[125–130] For many solid-state calculations, the computation can be distributed over the Brillouin zone sampling space. This approach is very simple to implement, however, it cannot be used for Γ-point ($k = 0$) calculations with large unit cells. Another approach is to distribute the one-electron orbitals across CPUs. The drawback of this method is that orthogonality will involve significant message passing. Furthermore this method will not work for simulations with very large cutoff energy requirements (*i.e.*, using large numbers of plane-waves to describe the one-electron orbitals) on parallel computers that have nodes with a small amount of memory, because a complete one-electron wavefunction must be stored on each node. Hence this approach is not practical for CPMD simulations with large unit cells, however, this approach can work well for simulations with modest size unit cells and with small cutoff energies, when used in combination with

minimization algorithms that perform orthogonalization sparingly, *e.g.* RMM-DIIS.[32,131]

Another straightforward way to parallelize CPMD is to spatially decompose the one-electron orbitals.[125,127,129] This approach is versatile, easily implemented, and well suited for performing CPMD simulations with large unit cells and cutoff energies. Moreover the parallel implementation of the non-local pseudopotential and orthogonality is trivial to implement, since it can be expressed using the simple global operation reduce. The drawback of this approach is that a parallel three-dimensional fast Fourier transform (FFT) must be used, which is known not to scale beyond $\sim N_g^{1/3}$ CPUs (or CPU groups), where N_g is the number of FFT grid points.

In Fig. 5, an example of timings *versus* the number of CPUs for this type of parallelization is shown. These simulations were taken from a CPMD simulation of UO_2^{2+} + $122H_2O$ with an FFT grid of $N_g = 96^3$ ($N_e = 1000$) using the plane-wave DFT module (PSPW) in NWChem.[11] These calculations were performed on all four cores on the quad-core Cray-XT4 system (NERSC Franklin) composed of a 2.3 GHz single socket quad-core AMD Opteron CPUs. The performance of the program is reasonable with an overall parallel efficiency of 84% on 128 CPUs dropping to 26% by 1024 CPUs. However, not every part of the program scales in exactly the same way. For illustrative purposes, the timings of the FFTs, non-local pseudopotential, and orthogonality are also shown. The parallel efficiency of the FFTs is by far the worst of the three major parts of the computation. Beyond 100 CPUs no gainful work was found in the FFT computation. However, at smaller CPU sizes the inefficiency of the FFTs are damped out due to the fact that these parts of the code make up less than 5% of the overall computation, and the largest part of the calculation is the non-local pseudopotential evaluation. Ultimately, however, the lack of scalability of the 3D FFT algorithm beyond $\sim N_g^{1/3}$ CPUs prevails and the simulation ceases to speedup. By 1000 CPUs, the non-local pseudopotential has also stalled. Interestingly, at this number of CPUs the costs of the non-local pseudopotential and the FFTs are roughly the same. Only orthogonality continues to scale beyond 1000 CPUs.

These results demonstrate an important guiding principle that is needed in the design of a parallel plane-wave DFT program: The number of CPUs that can be gainfully used in each of the major parts of the calculation is limited because they rely on global operations or all to all operations that use all CPUs in the calculation. Hence the overall parallel algorithm for plane-wave DFT should be designed to avoid global communications that span all CPUs in the calculation. For example, Gygi *et al.*[130] distributes across orbitals as well as over space,[130] resulting in a in a two-dimensional CPU distribution grids shown in Fig. 6 (where the total number of CPUs, N_p, can be written as $N_p = N_{pi}{*}N_{pj}$). This decomposition reduces the cost of the global operations in the major parts of the electron gradient computation, which only need $O(\log N_{pi})$ or $O(\log N_{pj})$ communications per CPU, instead of $O(\log N_p)$. For example, the FFT and non-local pseudopotential tasks only need to use global operations that span over N_{pi}, while the orthogonality step can be broken down into a series of alternating global operations that span

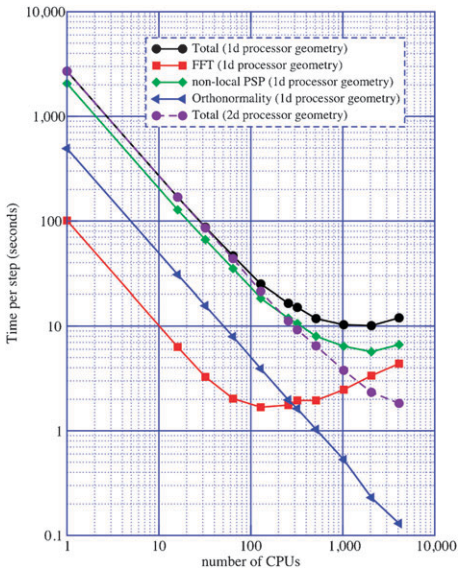


Fig. 5 Overall and component timings from AIMD simulations of UO_2^{2+} + 122 H_2O using one-dimensional CPU distribution geometry. Overall best timings also shown for the two-dimensional CPU geometry. Timings from calculations on the Franklin Cray-XT4 computer system at NERSC.

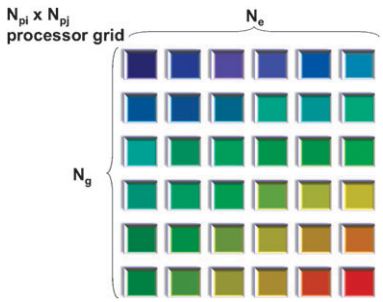


Fig. 6 The 2d parallel distribution over orbitals and space as suggested by Gygi *et al.*[130]

over either N_{pi} or N_{pj}, *e.g.* like the SUMMA algorithm.[132] Several codes use this approach.[11,33,34]

The overall performance of the plane-wave DFT simulations improve considerably using this new approach. In NWChem, for example, with the optimal CPU distribution grid, the running time per step took 2,699 s (45 min) for 1 CPU down to 3.7 s with a 70% parallel efficiency on 1024 CPUs. The fastest running time found was 1.8 s with 36% parallel efficiency on 4096 CPUs. As shown in Fig. 7, these timings were found to be very sensitive to the layout of the two-dimensional CPU distribution. For 256, 512, 1024, and 2048 CPUs, the optimal distribution geometries were 64 × 4, 64 × 8, 128 × 8 and 128 × 16 CPU grids, respectively. The timings of the FFTs, non-local pseudopotential, and orthogonality are also shown in Fig. 7. Not every part the program scaled perfectly. The parallel efficiency of several other key operations depends strongly on the shape of the CPU distribution. It was found that distributing the CPUs over the orbitals significantly improved the efficiency of the FFTs and the non-local pseudopotential, while distributing the CPUs over the spatial dimensions favored the orthogonality computations.

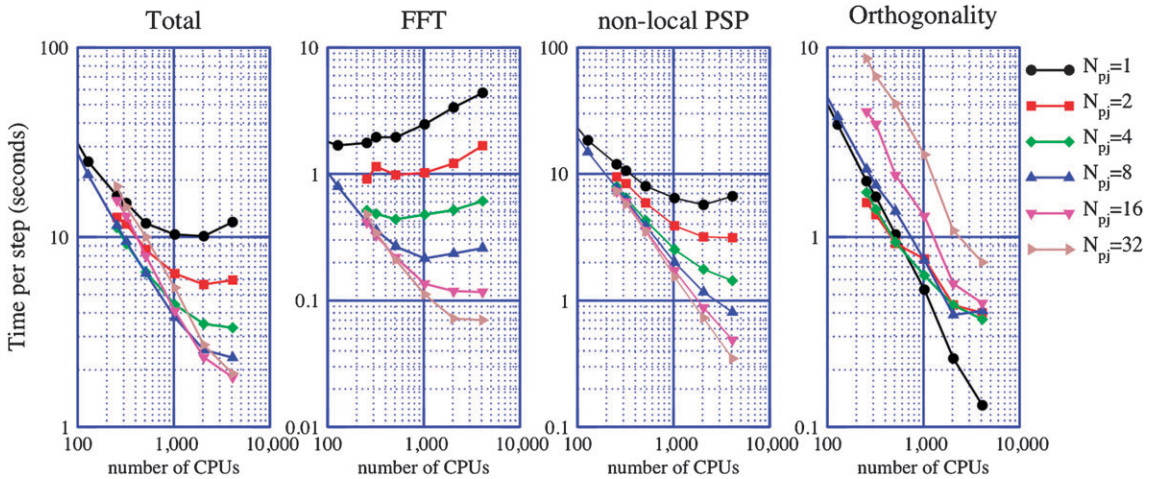


Fig. 7 Overall and component timings in seconds for UO_2^{2+} + $122H_2O$ plane-wave DFT simulations at various CPU sizes (N_p) and CPU distribution grids (N_{pj}, $N_{pi} = N_p/N_{pj}$). Timings were determined from calculations on the Franklin Cray-XT4 computer system at NERSC.

4.2 Correlation methodologies

Although the correlation energy, measured as a difference between exact electronic energy and electronic energy obtained within independent particle model (usually the HF method), contributes to less than 1% of the total energy, its inclusion is a prerequiste for the correct description of the most chemical processes. The development of chemically accurate methodologies for the description of correlation effects, or effects caused by instanteneous interactions between electrons, is a subject of significant effort aimed at developing reliable and robust algorithms for solving the electronic Schrödinger equation. Due to their accuracy, single- and multi-reference formulations of configuration interaction methodologies, many-body perturbation theory perturbative approaches and closely related (through the linked cluster theorems (LCT), see ref. 133 and references therein for details) coupled cluster (CC) methods have assumed a special position in computational chemistry. The widespread use of these methods should be also attributed to efficient parallel implementations developed over the last decade.[11,39,134–151] Several implementations and basic tools used in coding correlated methodologies are reviewed and discussed in forthcoming sections.

4.2.1 Single-reference methods. Single-reference many-body perturbation theory (MBPT) continues to be one of the driving engines of many computational calculations. The various-order contributions (MBPT(n)) to the energy ($E^{(n)}$) and wavefunction ($|\Psi^{(n)}\rangle$) are obtained order-by-order by solving the perturbed problem

$$(H_0 + \lambda V)(|\Psi^{(0)}\rangle + \lambda|\Psi^{(1)}\rangle + \cdots) = (E^{(0)} + \lambda E^{(1)} + \cdots)(|\Psi^{(0)}\rangle + \lambda|\Psi^{(1)}\rangle + \cdots) \quad (15)$$

where the electronic Hamiltonian H is partitioned as $H = H_0 + V$. There is some flexibility in defining the zeroth-order Hamiltonian H_0. Most formulations are based on the Møller–Plesset partitioning[152] where Hartree–Fock operator is chosen as the unperturbed Hamiltonian. This choice also assures proper factorization of denominators required in deriving the linked cluster theorem (see Ref. 153 for the so-called factorization theorem). In practice, due to the rapidly growing numerical cost, the application of MBPT is limited only to the lowest order contributions. For example, the numerical cost of calculating MBPT(2) and MBPT(3) energies scales as N^5 and N^6 respectively, where N refers to system size. The cost of the third order contribution is comparable to that of the approximate coupled cluster singles-doubles (CCSD) method. For this reason MBPT(2) or MP2 method is the most widely used in routine calculations.

Over the last twenty years tremendous progress has been made to make the MP2 applicable to large molecular systems. Among the major milestones one should mention: (a) elimination of energy denominator by the Laplace transformation approach;[154,155] (b) direct implementations through the decomposition of two-electron integrals, with the most efficient formulations being the resolution-of-identity (RI) for two-electron repulsion integrals (RI-MP2)[156–161] and the Cholesky decomposition;[162] (c) formulation of localized MBPT2 approaches.[3,65,68,163–166]

Parallel implementations of the MBPT(2) approaches were developed to meet various requirements concerning communication costs and memory consumption. A review of early, mostly disk based, MP2 implementations is provided in ref. 167 and 168. The first distributed-data MP2 algorithm was developed by Márquez and Dupuis.[169] However, this algorithm suffers from high memory demands and communication cost which are proportional to $n_o n^3$ and n^4, respectively (n_o, n_u and n refer to the occupied-, virtual-, and total-number of orbitals). The MP2 and RI-MP2 implementations by Bernholdt and Harrison in NWChem[156,170,171] were based on the distributed data model and the Global Array environment. Although computationally intensive, the unitary-invariant iterative version of MP2[170] allows, to a great extent, the utilization of spatial locality of the virtual orbital space. The performance numbers presented in ref. 170 are very promising for large processor counts. Using a similar distributed-data paradigm Schütz and Lindh developed an integral-direct distributed-data parallel MBPT(2) algorithm capable of achieving superlinear speedups.[172] In this algorithm

half-transformed integrals are stored locally (see ref. 172 for details). The communication cost of the most expensive step is proportional to $n_{\mathrm{usc}}n_o^2n_un$, where n_{usc} corresponds to the number of unique symmetry cells, while the local/global memory requirements are proportional to $n^2/n_{\mathrm{o}}n_{\mathrm{u}}n$. The three-quarter transformed integrals (as well as fully transformed integrals) are distributed evenly over all nodes using global arrays. In building three-quarter transformed integrals each processor owes its private task list corresponding to symmetry-unique shell doublets, while the last step is completely local. Other communication-intensive[173] and I/O driven MP2 algorithms[174–176] can be routinely used in calculations for chemical systems described by a few thousand orbitals. In the recent parallel implementation of the RI-MP2 approach Katouda *et al.*[177] reported test calculations performed for two-layer nanographene sheets $(C_{96}H_{24})_2$ using nearly 4000 basis functions. Test calculations showed good scalability accross 32 CPUs. Significant progress has also been acheived in local MP2 (LMP2) methodology, where scalability across hundreds of CPUs was recently obtained with the MP2 code employing MPI collective communication.[67] Recently reported linear-scaling atomic-orbital-based MP2 approach[178] can be used in calculations for systems composed of 1000 atoms with 10 000 basis set functions even using one processor. As pointed by the authors, the development of parallel implementation of this approach can significantly reduce time to solution.

The other very important and widely used correlation methodology rooted in many-body perturbation theory is the coupled cluster (CC) method. Using the LCT it can be shown that the cluster operator corresponds to infinite resummation of connected MBPT diagrams. Since its inception[153,179–185] coupled-cluster theory has evolved into one of the most reliable tools for describing correlation effects in atoms and molecules. The CC formalism enables the accurate description of correlation effects in a wide spectrum of many-electron systems ranging from weakly correlated to strongly correlated ones. Due to its simplicity, the single reference formulation of the CC method (SR-CC) plays a dominant role in routine high-level calculations. The SR-CC Ansatz for the ground-state wavefunction $|\Psi_0\rangle$ takes the following form

$$|\Psi_0\rangle = e^T|\Phi\rangle, \tag{16}$$

where the cluster operator T can be decomposed into different-rank components T_i, where each T_i creates i-tuply excited configurations when acting onto the reference function $|\Phi\rangle$ (usually represented by the HF determinant). The equations for cluster amplitudes are obtained by substituting the wavefunction in the CC parametrization (16) into the Schrödinger equation and subsequently pre-multiplying both sides of Schrödinger equation by e^{-T}. This procedure leads to an energy-independent form of equations for the cluster amplitudes

$$Qe^{-T}He^T|\Phi\rangle = 0, \tag{17}$$

where Q is the projection operator onto the subspace of excited determinants generated by the T operator when acting on the reference function. It can be shown that the above form of the CC equations can be expressed in terms of connected diagrams only, which assures the additive separability of the cluster operator in the non-interacting system limit (assuming that the corresponding reference function properly dissociates in this limit). Solving the CC equation (eqn (17)), the energy is obtained using the formula

$$E = \langle\Phi|\mathrm{e}^{-T}He^T|\Phi\rangle \tag{18}$$

The inclusion of various-rank excitations in the cluster operator results in a chain of more accurate approximations. In the basic CCSD approach (CC with singles and doubles)[186,187] the cluster operator is approximated as $T \approx T_1 + T_2$. In the more accurate CCSDT approach (CC with singles, doubles, and triples) the cluster operator is represented as $T \approx T_1 + T_2 + T_3$.[188,189] The high quality of the CCSDT energies comes at a high price because the numerical complexity of this method is proportional to N^8, where N refers to system size. For this reason many non-iterative approaches which mediate the cost between the CCSD method (characterized by the N^6 complexity) and CCSDT approaches, have dominated the area of high-level ground-state calculations. Among them, the ubiquitous CCSD(T) method[190] (N^7 complexity) is the most frequently used in routine calculations.

Single-reference CC theory can be extended to the excited-state problem using two theoretical platforms: (1) linear response CC (LR-CC)[191–193] and (2) equation-of-motion CC formalisms (EOMCC).[194–197] In the EOMCC approach the excited-state energies E_K (or poles in the LR-CC approach) are obtained by solving the eigenvalue problem

$$(P + Q)(e^{-T}He^T)R_K|\Phi\rangle = E_K(P + Q)R_K|\Phi\rangle, \tag{19}$$

where the R_K operator is used to parametrize the Kth state wavefunction

$$|\Psi_K\rangle = R_Ke^T|\Phi\rangle \tag{20}$$

and where P is the projection operator onto the reference function.

Again, in analogy with the ground-state case, various iterative (EOMCCSD[194–197] and EOMCCSDT[198]) and non-iterative approaches (EOMCCSD(T),[199,200] CCSDR(3), CCSDR(T),[201,202] CR-EOMCCSD(T),[203] N-EOMCCSD(T),[204] EOM-CCSD(2)$_{\mathrm{T}}$,[205,206] EOM-SF-CCSD(fT) and EOM-SF-CCSD(dT)[207]) have been developed in order to calculate excitation energies.

Since the CC equations involve contractions between tensors corresponding to the one- and two-electron integrals and cluster amplitudes, the task of writing efficient parallel CC code is extremely challenging. The main issues that define the overall parallel efficiency are related to (a) data storage and data transfer, (b) efficient load balancing, and (c) data flow during the execution of the code. Another problem is associated with the most efficient reduction of the numerical complexity of a given CC method. For example, approaches based on the use of recursive intermediates can be invoked to minimize the number of operations associated with calculating the simple tensor expression

$$R_j^i = A_k^iB_l^kC_j^l \tag{21}$$

where the Einstein summation convention over repeated indices is assumed. If we assume that each index runs from 1 to L, then the cost of calculating all components of the R-tensor by direct summation of all products amounts to L^4. However, if we calculate the recursive intermediate (or auxiliary tensor) defined as

$$I_j^k = B_l^k C_j^l \tag{22}$$

first, and then

$$R_j^i = A_k^i I_j^k \tag{23}$$

the overall cost is reduced to L^3. This is achieved at the expense of storing the I-tensor in memory (in the context of CC theory, see Kucharski and Bartlett's recursive factorization of ref. 208). An important role in designing the optimum memory *vs.* cost strategies is played by automatic code generators such as TCE[138,139] (tensor contraction engine) and SIAL[142,143] (super assembly instruction language) used to automatically generate the CC codes in NWChem and ACES III quantum packages (for review of various automatic code generators or symbolic algebra systems used for manipulating second-quantized expressions see ref. 209–214). These codes provide two examples where the organization of data in blocks determined by tiles or super numbers (NWChem, ACES III) is exploited in order to define the granular structure of the code and proper load balancing. Over the last few years this technique emerged as a prevalent approach.

Since the early days of CC parallel computing[141,147–150] the CC codes have undergone significant changes leading to substantial progress in scaling and system-size tractable by current CC implementations. Below we give a short review of the most essential developments in this area.

NWChem[11] has two classes of parallel CC codes, a spin-free code for closed-shell systems and a TCE implementation which can employ more general types of reference functions. The closed-shell CC implementations for the CCSD and CCSD(T) methods were the first parallel based on the GA library[140,215] and requires fast interconnects for communication between processors to provide the best performance. Even though these codes were developed over ten years ago, they have been very successful in taking advantage of present day massively parallel architectures.[216] It was recently demonstrated that the non-iterative (T) part can scale across 250 000 processors.[216]

The TCE generated CC codes are essentially spinorbital codes, which means that they can be universally used for a broad range of reference functions associated with restricted HF (RHF), restricted open-shell HF (ROHF), unrestricted HF (UHF), or DFT models. The only discrimination between these references takes place for the RHF reference where certain classes of cluster amplitudes are assumed to be equal. The block (tile) structure of all tensors is an ideal tool for dynamic load balancing, which is extensively used throughout the whole code. Global Arrays, which are an integral part of the code, are used to store all tensors. In alternative approaches shared or exclusion access files can be used to store large object accessible by all processors. The performance of TCE codes has been demonstrated with EOMCC

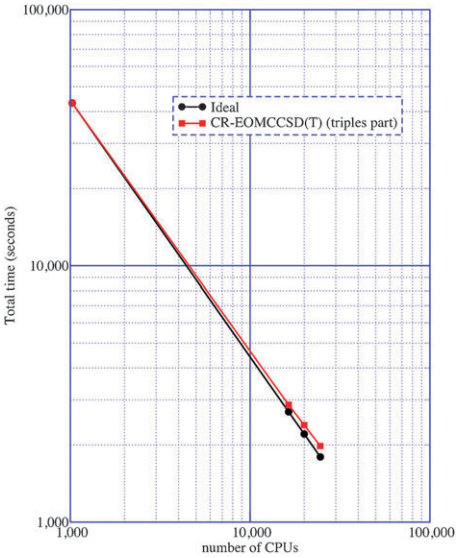


Fig. 8 An example of the scalability of the triples part of the CR-EOMCCSD(T) approach for Green Fluorescent Protein described by cc-pVTZ basis set (648 basis functions) as obtained from NWChem. Timings were determined from calculations on the Franklin Cray-XT4 computer system at NERSC.

calculations for zinc-porphyrin,[217] porphyrine dimer (942 basis functions and 270 correlated electrons),[219] free-base-porphyrine (702 basis functions, 114 correlated electrons),[219] and open-shell excited-state calculations for spiro molecule (646 basis functions, 75 correlated electrons). The linear response CC module was used in the dipole polarizabilities calculations for the C_{60} system using a basis set composed of 1080 functions.[218] All mentioned calculations were performed with several thousands of processors. The excited-state CR-EOMCCSD(T) corrections have also been shown (see Fig. 8) to scale across tens of thousands of CPUs (24 572).

Other CC codes use tools to manipulate data on distributed memory computer architectures are those of MOLPRO[39] and GAMESS-US.[151] While the CC code in MOLPRO uses GA, the GAMESS implementation is exploiting the distributed data interface (DDI).[43] The MOLPRO CC implementation exploits broadly used in RHF and ROHF based CCSD and CCSD(T) calculations. The GAMESS-US CC implementation evolved from the serial implementations of the CCSD, CCSD(T), and CR-CCSD(T) methods in GAMESS[145] and is deeply rooted in intra- and inter-node parallelisms. The former involves nodes connected *via* a dedicated communication network while the later is related to multiple processors within a single node. The third generation of DDI (DDI/3) significantly enhanced the shared-memory capabilities of DDI by supporting semaphores for interprocessor communication and three types of memory storage: replicated, shared, and distributed. As shown in ref. 135 and 144 the CCSD(T) code scales satisfactorily across 24 CPUs for systems described using several hundred basis functions.

The parallelism of ACES III[142,143] is also supported by the blocking structure of all tensors (super numbers) and basic operations performed on these blocks (super instructions).

Several methods including CCSD/CCSD(T) (energy and gradient), MBPT2 (energy, gradient, and Hessian), and EOMCCSD (energy) have been implemented using the SIAL language. So far only CC formulations based on the RHF to UHF reference functions have been developed. The novelty of the ACES III design is in the separation of two domains of expertise. The first domain uses SIAL to control the theoretical aspects of the underlying electronic structure method without controlling computational (user specified) details. The second domain, embodied by SIP (super instruction processor), executes and interprets the SIAL code providing the necessary level of optimization and tuning for a given computer architecture. As a result, ACES III can be seamlessly run on parallel architectures with shared and distributed memory. The Quantum Theory Project[220] at the University of Florida maintains a large number of benchmark calculations showing respectable scalability across several thousand of CPUs.

The PQS[134] implementation of the closed-shell CCSD and CCSD(T) methods mainly target the hardware related to moderately sized PC and workstation clusters. The PQS CC codes are implemented using the generator state spin adaptation of ref. 146, which significantly reduces the flop count compared to orthogonal spin adaptation. The CC PQS implementation draws heavily on the use of Array Files middleware[136] enabling the access to disk records distributed across all nodes in a cluster. This enables the effective re-use of large amounts of data needed in the CC calculations, which in conjunction with the efficient utilization of the sparsity of the atomic four-electron integrals used in the direct approach leads to significant savings in time per CCSD iteration. For example, for a linear decapeptide of the glycine molecule 90% of the integrals are numerically equal to zero, which drastically reduces the time required to calculate the contribution stemming from the most expensive CCSD diagram associated with the use of four-particle two-electron integrals. Two alternative implementations of the non-iterative (T) corrections are based on two algorithms, which are characterized by n_o^3 and n_u^3 (with n_o and n_u refering to the number of occupied and unoccupied orbitals respectively) local memory requirements. While the later can be faster for small molecules, the former one is better suited for large molecules due to smaller local memory requirements. The test cases discussed in ref. 136, 221, and 222 provide examples of the CCSD and CCSD(T) calculations of systems composed of 50–228 correlated electrons and described using basis sets as large as the cc-pVTZ basis set for Calix molecule. From data published in ref. 221 it follows that the iterative methods scale reasonably well across 16 CPUs. Beyond that limit the scalability is system-dependent but in most cases the scalability deteriorates.

The presented overview of various parallel implementations indicate that there is a broad range of implementations that target various parallel computer architectures and various user needs. For the users interested in large-scale closed-shell calculations the PQS, GAMESS, and MOLPRO CC implementations are a viable choice for machines with 10–60 CPUs. For those having access to massively parallel machines NWChem and ACES III should provide the most efficient utilization of many thousands of CPUs. It is also worth noting that each component of the CC calculation is characterized by different numerical needs associated with task pools of different sizes and diversified numerical complexity of each component. This should be a necessary indicator in the design of large scale calculations. The most prudent strategy is associated with performing each step using its optimum number of CPUs. For example, the four-index transformation (in the case when molecular integrals are used) and CCSD calculations can be performed together using smaller numbers of CPUs. The CCSD(T)-type calculations, in turn, due to inherent scalability can be performed on larger numbers of processors using stored integrals and cluster amplitudes. The improvement of data flow in the iterative approaches (CCSD/EOMCCSD), achieved by creating independent task pools is currently a subject of intensive work striving to improve the parallel performance and numerical efficiency.[223]

4.2.2 Multireference methods. The detailed description of multi-configurational electronic states has greatly benefited from the development of multi-reference methods. Prominent examples are the ground and excited states of open-shell transition metal complexes or molecules at strained geometries. Both multi-configurational (MC) SCF and configuration interaction (CI) methods are of great importance due to their general applicability and the availability of geometrical analytical derivatives[224–226] including the characterization of conical intersections[227] and the evaluation of non-adiabatic coupling vectors.[228,229] Both relativistic and electron correlation effects may be treated on the same footing within the framework of two- and four-component relativistic CI.[230–233]

In the following paragraphs, elements of the MCSCF and CI methods including derivatives are presented as far as significant for parallel implementations. Due to the rather wide scope of this perspective article the discussion is limited to selected representative implementations.

MCSCF. The MCSCF wave function is a linear expansion of spin-adapted configuration state functions (CSFs). The CSF expansion coefficients and the molecular orbital coefficients are optimized simultaneously. The CSF space is frequently determined through a partitioning of the initial N orbitals into orbital subspaces with occupation restrictions. The simplest form involves inactive, active and virtual orbitals, only. Inactive and virtual orbitals are kept doubly occupied and unoccupied, respectively, throughout, while distributing m active electrons in all possible ways within N_{act} active orbitals subject to symmetry and spin constraints defines the complete active space CAS(m, N_{act}). The CAS is equivalent to a Full CI expansion within the active orbitals.

While this simplifies the identification of redundant parameters, the number of CSFs for a state of a given spin multiplicity S grows factorially

$$N_{\mathrm{CSF}}^{\mathrm{FCI}}(m,N,S)=\frac{2S+1}{N+1}\binom{N+1}{\frac{1}{2}m-S}\binom{N+1}{\frac{1}{2}m+S+1} \quad (24)$$

Expanding a singlet state in a CAS(12,12) amounts to 226 512 CSFs, while a CAS(20,20) (5 924 217 936 CSFs) is well out of scope.

In a basis of determinants

$$N_{\mathrm{Det}}^{\mathrm{FCI}}(m_\alpha, m_\beta, N) = \binom{N}{m_\alpha}\binom{N}{m_\beta} \qquad (25)$$

the expansions are typically longer by a factor of two to eight since they cannot specifically describe a single spin multiplicity. Replacing a single active space by multiple subspaces with individual occupation and spin-coupling restrictions (*e.g.* the restricted active space (RAS) SCF method[234]) reduces the CSF space substantially. Alternatively, direct product spaces[235,236] yield very compact wave function expansions.

The total energy of the MCSCF wavefunction may be written as traces over one **h** and two-electron integrals **g** with symmetrized one- and two-electron density matrices **D** and **d**

$$E^{(0)} = Tr(\mathbf{hD}) + \tfrac{1}{2}Tr(\mathbf{gd}) \qquad (26)$$

The MCSCF energy is expanded up to second-order in the $(N_{\mathrm{CSF}} - 1)$ state rotation (**p**) and the $O(N^2)$ orbital rotation (κ) parameters around the current point

$$E^{(2)}(\kappa, \mathbf{p}) = E^{(0)} + \kappa^t\mathbf{w} + \mathbf{p}^t\mathbf{v} + \kappa^t\mathbf{Cp} + \tfrac{1}{2}\kappa^t\mathbf{B}\kappa + \tfrac{1}{2}\mathbf{p}^t\mathbf{Mp} \qquad (27)$$

where **w** and **v** represent the orbital and state gradient vectors while **B**, **M** and **C** denote the orbital, state and orbital-state coupling blocks of the electronic Hessian, respectively. E and **M** are evaluated from MO integrals with occupied (active plus inactive) orbital indices, only. To **w** and **C** additionally contribute integrals with one virtual orbital index, and **B** further requires integrals with two virtual orbital indices. **B**, **M** and **w** are formed by contraction with **d** and **D** while **C** and **v** require the transition densities between the optimized state and its orthogonal complement. Subsequently, these quantities are used to iteratively solve the Newton-Raphson (NR) equations in some variation (*cf.* ref. 237 and 238 and references cited therein)

$$\begin{pmatrix} \mathbf{B} & \mathbf{C} \\ \mathbf{C}^t & \mathbf{M} \end{pmatrix}\begin{pmatrix} \kappa \\ \mathbf{p} \end{pmatrix} + \begin{pmatrix} \mathbf{w} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad (28)$$

Some schemes do not explicitly include the **M** and **C** blocks of the electronic hessian. State-averaged (SA-)MCSCF calculations ensure a balanced, unbiased set of MOs for multiple electronic states by optimizing the average energy for several states of like and/or unlike symmetry and spin multiplicity. Here, the NR equations take a similar form[224,239,240] with **B** and **w** depending on averaged densities while the remaining terms contain individual contributions from all averaged states.

The three most CPU-time consuming steps executed per MCSCF iteration are (I) the partial AO-MO integral transformation, (II) the CI eigenvalue problem including density matrices and (III) solving the NR equations. The partial four-index transformation of step I is the time-consuming component, scaling as $O(N_{\mathrm{occ}}N^4)$ with N_{occ} denoting the number of occupied orbitals, of any parallel MP2 algorithm.[172,241] Complications arise only from the transposition of the integral distributions after the first half-transformation with $O({N_{\mathrm{occ}}}^2N^2)$ data transfer volume. Replicated data models are easy to implement but memory consumption or I/O overhead may be high ($O(N^4)$). With fast parallel integral codes at hand, the AO integrals can be computed on the fly trading $O(N^4)$ I/O against CPU time while the resulting $O({N_{occ}}^2N^2)$ MOs and intermediates are kept in distributed memory. The overall computational effort of the AO-MO transformation can be substantially reduced by exploiting auxiliary basis set expansions.[159,242] Step II is analogous to the MRCI method below. Implementations supporting a general choice of the MCSCF wave function[237] are conceptually less efficient than those restricted to CASSCF. Expanding the wavefunction in determinants is favorable for CAS and RAS type expansions[243] at a computational cost of $O(N_{\mathrm{det}})$. The increased size of the eigenvalue problem is of less importance here because typically the configuration space does not exceed a few million CSFs. Thus, a variety of coarse-graining operation modi are possible. Due to the size of the NR equations, they are usually solved iteratively in terms of subspace expansions which solely require the evaluation of matrix vector products of **B**, **C** and **M** with a trial vector. Hence, there is no need to store all of these large matrices explicitly at additional computational cost per NR iteration[237,238] so that Step III may contribute significantly to the execution time.

There are a few parallel implementations of the MCSCF method reported.[244,245] The parallel GAMESS CASSCF calculation was reported to scale moderately with an efficiency of 58% on 128 CPUs (~60 000 CSFs, 451 basis functions). Since Step I consumes 90% of the CPU time while Step II is negligible, the data mainly reflect the scaling properties of the AO-MO transformation step.

The AO-MO integral transformation step dominates for small CSF expansions, large basis sets and a small number of occupied orbitals. For the other extreme with large CAS-type CSF expansions and small basis sets a more evenly spread work load is expected for implementations discarding explicit consideration of **C** and **M**. Combining large basis sets and CSF expansions and number of occupied orbitals, however, is rather demanding both in terms of resource consumption and scaling behaviour. Yet, even with the most powerful computer systems-owing to the factorial growth of the CSF space-CASSCF approaches are close to their limits and restricted CI expansions are expected to be more favourable. In terms of fast and reliable optimization reducing the number of expensive four-index transformations while retaining second-order convergence a parallel implementation of the model hamiltonian is promising.[238]

Configuration interaction. The CI method has been implemented in several conceptually different ways both affecting parallelization schemes and imposing restrictions on applications. The principal form of the wave function is a linear expansion in CSFs or determinants Ψ_i. Optimization of the CSF coefficients as to minimize the energy corresponds to an eigenvalue problem $\mathbf{Hv} = \mathrm{E}\mathbf{v}$. Since only the few lowest eigenvalues are required, direct CI[246,247] schemes project the problem upon a small iteratively improved subspace. Several different update schemes are in use.[248,249] Thereby the storage of a sparse CI matrix **H** can be avoided and-assuming dense linear algebra-the computational effort is reduced from

$O((N_{\mathrm{CSF}})^3)$ to $O((N_{\mathrm{CSF}})^2)$. In fact, exploiting the sparsity of $\mathbf{H}$ the total effort observed scales only as $O(N_{\mathrm{CSF}})$. Realistic applications easily encounter matrix dimensions of 10^7–10^{10}.

The computational procedure proceeds in three steps repeated until convergence: (I) formation of the σ vector $\sigma = \mathbf{Hv}$; (II) computing the subspace representation of $\mathbf{H}$ and the overlap matrix, *i.e.* $\tilde{\mathbf{H}}_{ij} = \mathbf{v}_i^T \sigma_j$, $\tilde{\mathbf{S}}_{ij} = \mathbf{v}_i^t \mathbf{v}_j^t$; (III) solving the non-orthogonal subspace eigenvalue problem $\tilde{\mathbf{H}}\tilde{\mathbf{v}} = \mathrm{E}\tilde{\mathbf{S}}\tilde{\mathbf{v}}$, followed by trial vector update and optionally subspace reduction.

The σ vector construction (Step I) is the most time-consuming step. The coupling coefficients are the matrix elements of the unitary group generators[250] (ε_{ij}), which form an extremely sparse quantity and, hence, are best computed and processed on the fly.

$$H_{IJ} = \sum_{ij} h_{ij}\langle\Psi_I|\varepsilon_{ij}|\Psi_J\rangle + \tfrac{1}{2}\sum_{ijkl}(ij|kl)\langle\Psi_I|\varepsilon_{ij}\varepsilon_{kl} - \delta_{jk}\varepsilon_{il}|\Psi_J\rangle \tag{29a}$$

$$= \sum_{ij} h_{ij}\langle\Psi_I|\varepsilon_{ij}|\Psi_J\rangle + \tfrac{1}{2}\sum_{ijkl}(ij|kl)\left(\sum_K \langle\Psi_I|\varepsilon_{ij}|\Psi_K\rangle\langle\Psi_K|\varepsilon_{kl}|\Psi_J\rangle\right) - \delta_{jk}\langle\Psi_I\varepsilon_{il}|\Psi_J\rangle \tag{29b}$$

The σ vector is evaluated by a single pass over all integrals such that the majority of work occurs in terms of vectorizable dense matrix-matrix and matrix-vector multiplies or vector operations. In other words, we must identify the coupling coefficients to be combined with a block of integrals and a group of $\mathbf{v}$ coefficients and accumulate the result in a (different) group of σ vector elements with uniform stride. To achieve this, a suitable continuous enumeration of the N-electron basis, an efficient I/O free scheme to evaluate the coupling coefficients and a suitable ordering of the integrals is necessary.

There are two major approaches to this problem: the Graphical Unitary Group Approach (GUGA)[250] expands the wave function in CSFs. The non-vanishing coupling coefficients are computed in terms of products of a small number of pretabulated factors. The indices of the associated integrals, σ and $\mathbf{v}$ vector elements are readily identified (eqn (29a)). Alternatively, a basis of Slater determinants can be used following the work of ref. 251 and 252. Inserting the resolution of identity (RI) (eqn (29b)) only one-electron matrix elements occur which have simple eigenvalues $\{-1,0,1\}$ in a basis of determinants. Combined with a suitable enumeration of the determinants and ordering of the integrals the computation of the σ vector is highly vectorizable. Apart from the substantially larger size of a determinant basis as compared to a CSF basis, the trial vector update scheme must ensure that the eigenvector is an eigenfunction of S^2.

While the determinant approach is particularly easy to implement for the FCI case, the restriction to general truncated configuration spaces is more involved because the intermediate states Ψ_K extend beyond the specified configuration space. An efficient enumeration scheme for determinants can be derived from the the Symmetric Graphical Group Approach (SGGA).[253,254] Moreover, in the non-FCI case the usable vector lengths may be substantially reduced and overhead increases deteriorating performance.

To set up the subspace representation matrices $\tilde{\mathbf{H}}$ and $\tilde{\mathbf{S}}$ of dimension n_v (Step II) a series of n_v dot products over the current σ and the available n_v trial vectors is required, causing negligible CPU time consumption and an I/O volume of $8n_vN_{\mathrm{CSF}}$ bytes if the subspace basis vectors are stored on disk. The I/O volume for the trial vector update (Step III) amounts to $16n_vN_{\mathrm{CSF}}$ bytes, and the reduction of the subspace dimension to $\mathrm{n_{vmin}}$ causes additional $16n_{\mathrm{vmin}}N_{\mathrm{CSF}}$ bytes I/O.

The favorable I/O volume specified above applies to a non-orthogonal subspace basis only (linear algorithm[255]). Given a minimum and maximum subspace dimension $n_{\mathrm{vmin}} = 2$, $n_{\mathrm{vmax}} = 6$ and 10^9 CSFs or determinants, the I/O volume per iteration ranges between 72 Gbyte and 200 Gbyte-clearly not a negligible amount of data. Retaining these data in distributed memory almost completely avoids I/O and communication for Step II and III almost completely at the expense of distributed memory consumption.

Endorsing a complete N-electron basis FCI yields exact results within the given one-electron basis. Due to the factorial growth of the configuration space FCI implementations[249,256–259] are preferentially an option for small systems with small basis sets for benchmarks, only. The first group of MRCI implementations derived from FCI codes[230,243] with a structural selection scheme such as the generalized active space model (GAS) that divides the orbitals into groups with group specfic occupation and spin coupling constraints imposed. There is no limit on the number of electrons per orbital subspace except for resource requirements. The computational costs scale as $O(N_{\mathrm{Det}})$ with $N_{\mathrm{Det}} > 10^9$ for realistic applications. The second group of MRCI codes divides the orbital space into internal and external orbitals with the latter constrained to be occupied by at most two electrons.[39,41,260–262] The configuration space is obtained by applying all single and double excitations to a set of reference configurations (MR-CISD). CSFs sharing the same internal orbital occupation and spin-coupling pattern are enumerated sequentially and due to the simple form of the coupling coefficients the contributions can be written in terms of dense BLAS operations. Similarily, orbitals doubly occupied in all reference configurations could take advantage of a similar procedure. The COLUMBUS package[41,261,262] is designed to work well with arbitrary reference configuration spaces and, hence, exploits vectorization for the external orbitals, only.

The third group of MRCI implementations drops the dense enumeration of CSFs entirely, allowing arbitrary selection of CSFs out of some underlying configuration space for the variational treatment while the contributions of the remaining CSFs are estimated *e.g.* by perturbation theory.[263–265] The selection may be based on a perturbation estimate of the correlation energy. While the size of the variational problem is reduced by factor of 10–100, the efficiency of the CI code is much lower and the variational portion is limited to a few million CSFs. The theme to reduce the number of variational parameters resurfaces in variations. Internal contraction schemes[238] consider the reference wave function as a single

entity so that the number of parameters scales no longer proportional to the number of reference CSFs but instead depends on the number of internal orbitals squared and yet vectorizes well. This scheme is particularly beneficial for CAS reference spaces. On the downside, analytical gradients are more complicated and relaxation of the internal contraction coefficients is expensive. A promising variation of this theme is the graphically contracted function method,[266] which is closely connected to the GUGA representation of the FCI space. A matrix element of $\tilde{\mathbf{H}}$ can be evaluated serially in a few seconds for an underlying FCI expansion beyond 10^{20} while analytical gradients are available.

In terms of parallelization Step I is the most difficult part as it is subject to load-balancing, bandwidth and memory constraints. With coupling coefficients readily available, σ can be computed from a single pass through the integrals and random access to the entire σ and $\mathbf{v}$ vector, *i.e.* σ, $\mathbf{v}$ and integrals are kept replicated in local memory followed by a global sum of the partial σ vectors. This approach is seriously limited by the CI dimension and local memory resources but load-balancing is easy to achieve dynamically. Alternatively, the CI matrix can be partitioned into n_s times n_s blocks and σ and $\mathbf{v}$ vectors into n_s segments. The matrix-vector product now decomposes into $(n_s)^2$ independent partial matrix vector products (tasks) each requiring random access to one σ and $\mathbf{v}$ segment along with a full pass through the integrals. Taking advantage of the index symmetry of $\mathbf{H}$, the number of tasks can be reduced to $n(n + 1)/2$ at the expense of doubling the number of local σ and $\mathbf{v}$ segments to be kept in local memory. To feed n_{cpu} processors $O(n_{\mathrm{cpu}})$ independent tasks are required. This corresponds to $O((n_{\mathrm{cpu}})^{1/2})$ segments leading to a linear increase of the total data transfer volume with respect to σ and $\mathbf{v}$ vectors with increasing number of processes. Hence, trial and sigma vector can be placed in distributed memory while the integrals are kept either replicated in local or non-redundant in distributed memory. Each independent task requires the $\mathbf{v}$ vector to be copied to the workers local memory, computing the partial sigma vector with a single pass through a subset of integrals followed by an update of the distributed σ vector. In practice, the tasks are a little more fine-grained.[40]

The relative computational effort to compute CI matrix blocks within the GUGA approach may vary by many orders of magnitude. The spatial distribution of the work load largely depends on choice and enumeration of the CSF space and reflects the sparsity of the CI matrix. Hence, it is difficult to balance memory consumption, communication overhead and to prevent monopolization by a few costly tasks for a large number of processors without resorting to some performance model. Some performance characteristics of the COLUMBUS code[40,267] on an 32-way symmetric multiprocessing (SMP) cluster for a CI dimension of $\sim$1.7 billion CSFs and 316 basis functions: the scaling is close to ideal up to 1024 CPUs with the total communication overhead of 2 to 5% of the wall clock time depending on the task-generation scheme. No more than 4 to 8 tasks per CPU are necessary. At small CPU counts the communication load increases due to memory constraints while for a large number of CPUs load-balancing constraints force an increasing communication volume. The average transfer rate per CPU is with up to 20 Mbyte/s way below the actual inter-node peak bandwidth. To this end a sufficiently flexible task generation and a reasonably accurate performance model seem to be the right track for the usage of today's parallel hardware (*cf.* Fig. 9).

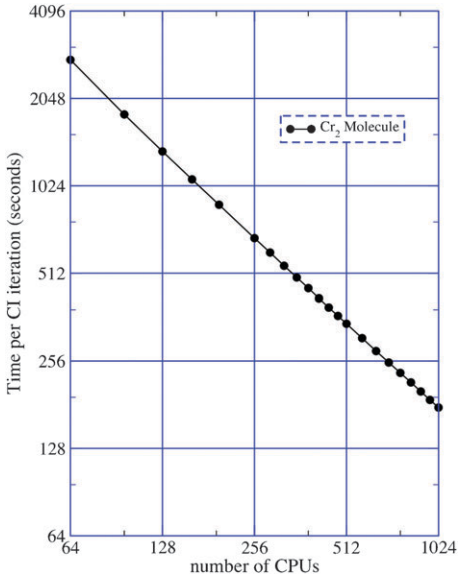


Fig. 9 Scaling of wall-clock time per CI iteration for the Cr_2 molecule (1.713 billion CSFs, basis set is [10s10p8d6f4g]) obtained running the COLUMBUS MR-CISD code. Timings were determined from calculations at the Research Centre, Jülich.

The effective bandwidth may be increased by using an error-controlled, bit-oriented data compression scheme[268] at the expense of non-local data-access in Step II and III. For an alternative GUGA-MRCI implementation[260] with local disk storage of the subspace expansion vectors, modest speedups on up to 12 processors and a low-cost Gigabit ethernet have been observed. A parallel implementation of the internally contracted MRCI method[269] has displayed reasonable but molecular system dependent scaling on up to 256 CPUs with an efficiency of about 50%.

Recently, performance data for a determinant based MRCI code have been reported incorporating 1.6 billion determinants[270] on quadcore Linux-Cluster equipped with local disks. Good speedups up to 16 nodes using one core each with an efficiency of above 0.88 have been shown. Increasing the number of cores used the efficiency drops to 0.30. The overhead of step II and III relative to Step I is up to 100% for quadcore and 30% for dualcore operation which indicates that I/O is a bottleneck despite of the use of local disks and is worse on a parallel file system. The speedup for Step I drops significantly beyond 8 nodes possibly due to the static load balancing scheme.

The apparently largest FCI calculations performed so far use 10 billion determinants and disk based storage of the subspace basis vectors. Considerable effort has been spent on I/O optimization and data representation yet I/O overhead is high and no speedup data have been reported.[257] Performance data for FCI codes from other groups are available for less than 20 million determinants, only.[258,259]

Currently the COLUMBUS code is the only code capable of using thousand(s) of processors with high efficiency. Since the lack of scaling of various CI implementations appears to arise from I/O and load balancing issues, given appropriate modifications of the implementation, similar scalability may be expected in future for most alternative implementations of the CI method.

Analytic geometric derivatives. The first analytical derivatives with respect to a geometrical distortion λ for variationally optimized MCSCF and CI wavefunctions is given by

$$E^{\lambda} = \langle \Psi | H^{\lambda} | \Psi \rangle \tag{30}$$

The energy derivative can be written in terms of traces over products of displacement dependent (integral derivatives $\mathbf{S}^{\lambda}$, $\mathbf{g}^{\lambda}$, $\mathbf{h}^{\lambda}$) and displacement independent quantities ($\hat{\mathbf{D}}$, $\hat{\mathbf{d}}$, $\hat{\mathbf{F}}$).

$$E^{\lambda} = \mathrm{Tr}(\mathbf{h}^{\lambda}\hat{\mathbf{D}}) + \tfrac{1}{2}Tr(\mathbf{g}^{\lambda}\hat{\mathbf{d}}) - Tr(\mathbf{S}^{\lambda}\hat{\mathbf{F}}) \tag{31}$$

where the last term accounts for the renormalization of the atom-centered basis set. The effective Fock matrix $\hat{\mathbf{F}}$ is defined in MO basis as

$$\hat{\mathbf{F}}_{pq} = \sum_{t} \mathbf{h}_{pt}\hat{\mathbf{D}}_{tq} + \sum_{rst} \mathbf{g}_{ptrs}\hat{\mathbf{d}}_{qtrs} \tag{32}$$

For state-specific fully variatonal MCSCF, we have $\hat{\mathbf{D}} = \mathbf{D}$ and $\hat{\mathbf{d}} = \mathbf{d}$. In the case of CI, only the CSF coefficients are variationally optimized while the MO coefficients are taken from an MCSCF calculation. Since it is highly desirable to fully decouple MCSCF and CI wavefunction constraints (*e.g.* state-averaged MCSCF MOs, different reference spaces, *etc.*), the separation into redundant and essential orbital rotations is important. Redundant orbital rotations are those that can be compensated by a corresponding transformation of the CSF coefficients so that only the wavefunction representation changes but the wavefunction itself remains unaffected. To fully decouple MCSCF and CI calculations, redundant orbital rotations must be resolved at the MCSCF level in order to have well-defined orbitals. The respective contributions derived from the orbital rotation gradient (obtained from CI) and the orbital response (obtained from MCSCF) can be folded into modified, geometry-independent (effective) density matrices and require solely the solution of a single coupled perturbed MCSCF equation.[225] In the case of state-averaged (SA) MCSCF calculations, the orbitals are optimized for an energy-weighted average of several electronic states at the MCSCF level. This leads to a single modified set of coupled perturbed MCSCF equations which must be solved for both MCSCF and CI gradients based on these state-averaged orbitals.[224] As before, this results in modified effective density matrices. It is convenient to evaluate $\hat{\mathbf{D}}$, $\hat{\mathbf{d}}$, $\hat{\mathbf{F}}$, in the MO basis followed by a transformation into the AO basis. This avoids both storage and expensive $O(N_{\lambda})$ transformation of the derivative integrals into the MO basis. Sorting the effective two-electron density matrix into an appropriate order, the contraction of the derivative integrals is carried out on the fly without the need to store any integrals or to randomly access the two-particle density matrix elements.

Non-adiabatic coupling coefficients[228,229] can be divided into two components $f_{JI}^{\mathrm{CI}\lambda}$ and $f_{JI}^{\mathrm{CSF}\lambda}$

$$f_{JI}^{\lambda} = \left\langle \Psi_J \left| \frac{\partial}{\partial R_{\lambda}} \Psi_I \right. \right\rangle = f_{JI}^{\mathrm{CI}\lambda} + f_{JI}^{\mathrm{CSF}\lambda} = \frac{1}{E_I - E_J} \langle c^J | H^{\lambda} | c^I_{CSF} \rangle + \sum_{ij} c_i^I c_j^J \left\langle \Psi_j \left| \frac{\partial}{\partial R_{\lambda}} \Psi_i \right. \right\rangle_r \tag{33}$$

Using MOs from a state-averaged MCSCF calculations, the procedure to evaluate the matrix elements follows the formalism described in ref. 225. The final equations are similar to those for the gradients except that the symmetrized one and two-electron densities are replaced by symmetrized one and two-electron transition densities plus an additional term involving an antisymmetrized one-particle transition density arising from the $f_{JI}^{\mathrm{CSF}\lambda}$ term[229]

$$f_{JI}^{\lambda} = \frac{1}{E_I - E_J}\left[Tr(\hat{\mathbf{D}}_{JI}\mathbf{h}^{\lambda}) + \frac{1}{2} Tr(\hat{\mathbf{d}}_{IJ}\mathbf{g}^{\lambda}) - Tr(\mathbf{S}^{\lambda}\hat{\mathbf{F}}_{JI}) \right] + \mathrm{Tr}(\hat{\mathbf{D}}^{a}_{JI}\mathbf{h}^{\lambda}) \tag{34}$$

where $\hat{\mathbf{d}}_{IJ}$ and $\hat{\mathbf{D}}_{JI}$ refer to effective symmetric one- and two particle transition densities, $\hat{\mathbf{F}}_{JI}$ a correspondingly defined effective Fock matrix. The last term involves a contraction of antisymmetric one-particle transition density $\hat{\mathbf{D}}^{a}_{JI}$ with the symmetrized half-differentiated overlap matrix $\mathbf{f}^{\lambda}$

Computationally there are four major steps: (I) the construction of CI density matrices at the expense and the same degree of parallelization as a single CI iteration (two for transition densities) (II) construction of effective Fock and density matrices in the MO basis, (III) transformation to the AO basis and (IV) evaluation of the derivative integrals on the fly and contraction with their counterparts (eqn (31)). All steps require I/O of order $O(N^4)$ which is favourably replaced by distributed memory. These steps are either trivially parallel or can use the parallel algorithms mentioned before. While analytical gradients are independent of the number of degrees of freedom N_d of a molecule, the computational effort for numerical gradients scales as $O(N_d)$ albeit is trivially parallel over the independent wavefunction optimizations at slightly displaced geometries.

5. Looking into the future: computational chemistry at the extreme scale

As this perspective is written, the first computational chemistry science is delivered on the first petaflop computers consisting of hundreds of thousands of processing units. At the same time teraflop computers are emerging at the desktop with GPGPU technology as the next opportunity to deliver one to two orders of magnitude of performance improvement for quantum chemistry algorithms. Computational chemistry is continuing its trend of being an early adopter of new technologies with the first implementations of traditional quantum chemistry algorithms on a GPGPU appearing in the literature.[271–274]

High-performance computing has entered an era where steady advances in chip technology are superseeded by new

disruptive technologies, such as the GPGPU, that have the potential to fundamentally affect computational chemistry research because researchers can utilize more accurate yet computationally expensive approximations and study systems using more realistic simulations and equations. Teraflop parallel supercomputers from a couple of years ago are now easily accessible by research groups around the world. Looking forward, the technological advances needed to achieve exaflop computing could bring the petaflop computer to individual research groups. This means that the quantum chemistry software developed today for these high-end architectures become the research tools for groups at home in the future.

While the architecture of a potential exaflop supercomputer is still sketchy, one can be assured that it will consist of millions of computational units (CPUs, GPGPUs, or more likely a combination of these technologies), each performing hundreds to thousands of simultaneous threads, connected by fast hierarchical interconnect networks. Hybrid and massively-threaded programming models will be key to exploiting the increased complexity of billions of threads and a hierarchy network connections combined with mixed computational units with different performance characteristics, and to attain the highest possible performance. New algorithms or new methods for solving the quantum chemistry equations may need to be developed to take full advantage of massively-threaded programming models.

With extreme scale computing platforms, the high performance computing community is entering an era of combining components in the orders of hundreds of thousands to millions. One of the major challenges for quantum chemistry algorithms is how to handle the reduced Mean Time Between Failure (MTBF) of various components in the system, and to recover from hardware failure. At this scale of hardware and stored simulation data checkpoint-restart is not an option. The development of fault-resilient or -tolerant software is still in its infancy, and is being pursued both in the computer science arena and at the application level. Quantum chemistry application programmers may need to adjust their software architecture as traditional implementations of quantum chemistry algorithms cannot always be easily adapted to become fault-resilient.

6. Summary and conclusions

In this perspective we have presented a brief overview of the present status of parallel quantum chemistry methods and applications and put forward our best estimate of what lies in the future. We discussed the hardware and software technologies that enable program developers and quantum chemists to take advantange of the full potential of parallel computing platforms. In particular, we note that in addition to the MPI protocol, other parallel communication protocols have been developed specifically to enable parallelization and scalability of quantum chemical algorithms. The combination of advanced hardware and sophisticated software tools in the context of different quantum chemical methods has been presented. This part of the review covered the major set of standard methods available to the computational chemistry of today.

To conclude, this perspective presented something which must be evident to any reader by now. The present and, even more so, the future quantum chemist has and will have an interesting occupation. However, that position comes with a constraint that will get more and more difficult to fulfill or acquire. The true computational chemist has to be an expert in chemistry, quantum chemistry, quantum chemical methods, numerical methods, mathematics, software techniques and management, and finally computer hardware technology and development. Educating students so they can face the challenges posed by the present and future trends in parallel computing might turn out to be more difficult than generating new computer technologies that outperform those of the past.

Acknowledgements

This work was done in part using EMSL, a national scientific user facility sponsored by the Department of Energy's Office of Biological and Environmental Research and located at Pacific Northwest National Laboratory, operated for the U.S. Department of Energy by Battelle under contract DE-AC05-76RL01830. Part of the funding for this work was provided by the Department of Energy Office of Basic Energy Science. VV and RL thank the Swedish Research Council directly and through the Linnaeus Center of Excellence on Organizing Molecular Matter at Lund University, Sweden, for financial support. CJ and IN acknowledge support from Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000. KK and EJB acknowledge support from the Extreme Scale Computing Initiative, a Laboratory Directed Research and Development Program at Pacific Northwest National Laboratory. EJB acknowledges support from DOE ASCR petascale tools program and he would like to thank Scott Baden for his help in developing parallel algorithms. TM acknowledges support by the John-von-Neumann Institute for Computing at the Research Centre Jülich.

References

1 J. Almlöf, K. Faegri and K. Korsell, *J. Comput. Chem.*, 1982, **3**, 385.
2 O. Vahtras, J. Almlöf and M. Feyereisen, *Chem. Phys. Lett.*, 1993, **213**, 514.
3 P. Pulay and S. Saebo, *Theor. Chim. Acta*, 1986, **69**, 357.
4 M. Yoshimine, *The Use of Direct Access Devices in Problems Requiring the Reordering of Long Lists of Data*, Report RJ-555, IBM Research Laboratory, San Jose, CA, 1969.
5 P. R. Taylor and C. W. Bauschlicher Jr., *Theor. Chim. Acta*, 1987, **71**, 105.
6 R. A. Bair and T. H. Dunning Jr., *J. Comput. Chem.*, 1984, **5**, 44.
7 A. Nanayakkra, D. Moncrieff and S. Wilson, *Parallel Comput.*, 1993, **19**, 1053.
8 R. Seeger, *J. Comput. Chem.*, 1981, **2**, 168.
9 E. Clementi, D. Logan and J. Saarinen, *IBM Systems Journal*, 1988, **27**, 475.
10 L. J. Toomey, E. C. Plachy, R. G. Scarborough, R. J. Sahulka, J. F. Shaw and A. W. Shannon, *IBM Systems Journal*, 1988, **27**, 416.
11 E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, D. Wang, E. Apra, T. L. Windus, J. Hammond, P. Nichols, S. Hirata, M. T. Hackler, Y. Zhao,

P.-D. Fan, R. J. Harrison, M. Dupuis, D. M. A. Smith, J. Nieplocha, V. Tipparaju, M. Krishnan, Q. Wu, T. Van Voorhis, A. A. Auer, M. Nooijen, E. Brown, G. Cisneros, G. I. Fann, H. Fruchtl, J. Garza, K. Hirao, R. A. Kendall, J. A. Nichols, K. Tsemekhman, K. Wolinski, J. Anchell, D. Bernholdt, P. Borowski, T. Clark, D. Clerc, H. Dachsel, M. Deegan, K. Dyall, D. Elwood, E. Glendening, M. Gutowski, A. Hess, J. Jaffe, B. Johnson, J. Ju, R. Kobayashi, R. Kutteh, Z. Lin, R. Littlefield, X. Long, B. Meng, T. Nakajima, S. Niu, L. Pollack, M. Rosing, G. Sandrone, M. Stave, H. Taylor, G. Thomas, J. van Lenthe, A. Wong and Z. Zhang, *NWChem, A Computational Chemistry Package for Parallel Computers, Version 5.1.1*, Pacific Northwest National Laboratory, Richland, Washington 99352-0999, USA, 2008.

12 C. L. Janssen, I. M. B. Nielsen, M. L. Leininger, E. F. Valeev, J. P. Kenny and E. T. Seidl, *The Massively Parallel Quantum Chemistry Program (MPQC), Version 3.0*, Sandia National Laboratories, Livermore, CA, USA, 2008.

13 C. L. Janssen and I. M. B. Nielsen, *Parallel Computing in Quantum Chemistry*, CRC Press, Taylor & Francis Group, Boca Raton, FL, 2008.

14 U. Drepper, What Every Programmer Should Know About Memory, http://people.redhat.com/drepper/cpumemory.pdf, 2009.

15 W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc, San Francisco, CA, 2004.

16 Lustre File System, Sun Microsystems: https://www.sun.com/offers/details/LustreFileSystem.xml, 2008.

17 F. Smuck and R. Haskin, in *Proceedings of theUSENIX FAST 2002 Conference on File and Storage Technology*, Monterey, 2002.

18 B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka and B. Zhou, in *Proceedings of the USENIX FAST 2008 Conference on File and Storage Technology*, 2008.

19 R. J. Harrison, *Int. J. Quantum Chem.*, 1991, **40**, 847.

20 V. S. Sunderam, *Concurr. Comput. Pract. Exper.*, 1990, **2**, 315.

21 MPI: A Message-Passing Interface standard. Message Passing Interface Forum, http://www.mpi-forum.org, 2008.

22 G. te Velde, F. M. Bickelhaupt, S. J. A. van Gisbergen, C. Fonseca Guerra, E. J. Baerends, J. G. Snijders and T. Ziegler, *J. Comput. Chem.*, 2001, **22**, 931.

23 X. Gonze, B. Amadon, P.-M. Anglade, J.-M. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Cote, T. Deutsch, L. Genovese, P. Ghosez, M. Giantomassi, S. Goedecker, D. R. Hamann, P. Hermet, F. Jollet, G. Jomard, S. Leroux, M. Mancini, S. Mazevet, M. J. T. Oliveira, G. Onida, Y. Pouillon, T. Rangel, G.-M. Rignanese, D. Sangalli, R. Shaltaf, M. Torrent, M. J. Verstraete, G. Zerah and J. W. Zwanziger, *Comput. Phys. Commun.*, 2009, **180**, 2582.

24 W. Andreoni and A. Curioni, *Parallel Comput.*, 2000, **26**, 819.

25 J. M. Soler, E. Artacho, J. D. Gale, A. Garcia, J. Junquera, P. Ordejon and D. Sanchez-Portal, *J. Phys.: Condens. Matter*, 2002, **14**, 2745.

26 S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. J. Probert, K. Refson and M. C. Payne, *Z. Kristallogr.*, 2005, **220**, 567.

27 B. Delley, *J. Chem. Phys.*, 2000, **113**, 7756.

28 C.-K. Skylaris, P. D. Haynes, M. C. Mostofi and M. C. Payne, *J. Chem. Phys.*, 2005, **122**, 084119.

29 P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari and R. M. Wentzcovitch, *J. Phys.: Condens. Matter*, 2009, **21**, 395502.

30 J. VandeVondele, M. Krack, F. Mohamed, M. Parrinello, T. Chassaing and J. Hutter, *Comput. Phys. Commun.*, 2005, **167**, 103.

31 Y. Shao, L. F. Molnar, Y. Jung, J. Kussmann, C. Ochsenfeld, S. T. Brown, A. T. B. Gilbert, L. V. Slipchenko, S. V. Levchenko, D. P. O'Neill, R. A. DiStasio, R. C. Lochan, T. Wang, G. J. O. Beran, N. A. Besley, J. M. Herbert, C. Y. Lin, T. Van Voorhis, S. H. Chien, A. Sodt, R. P. Steele, V. A. Rassolov, P. E. Maslen, P. P. Korambath, R. D. Adamson, B. Austin, J. Baker, E. F. C. Byrd, H. Dachsel, R. J. Doerksen, A. Dreuw, B. D. Dunietz, A. D. Dutoi, T. R. Furlani, S. R. Gwaltney, A. Heyden, S. Hirata, C. P. Hsu, G. Kedziora, R. Z. Khalliulin, P. Klunzinger, A. M. Lee, M. S. Lee, W. Liang, I. Lotan, N. Nair, B. Peters, E. I. Proynov, P. A. Pieniazek, Y. M. Rhee, J. Ritchie, E. Rosta, C. D. Sherrill, A. C. Simmonett, J. E. Subotnik, H. L. Woodcock, W. Zhang, A. T. Bell, A. K. Chakraborty, D. M. Chipman, F. J. Keil, A. Warshel, W. J. Hehre, H. F. Schaefer, J. Kong, A. I. Krylov, P. M. W. Gill and M. Head-Gordon, *Phys. Chem. Chem. Phys.*, 2006, **8**, 3172.

32 G. Kresse and J. Furthmüller, *Comput. Mater. Sci.*, 1996, **6**, 15.

33 F. Gygi, *IBM J. Res. Dev.*, 2008, **52**, 137.

34 A. Canning and D. Raczkowski, *Comput. Phys. Commun.*, 2005, **169**, 449.

35 M. F. Guest, I. J. Bush, H. J. J. van Dam, P. Sherwood, J. M. H. Thomas, J. H. van Lenthe, R. W. A. Havenith and J. Kendrick, *Mol. Phys.*, 2005, **103**, 719.

36 R. A. Kendall, E. Apra, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus and A. T. Wong, *Comput. Phys. Commun.*, 2000, **128**, 260.

37 M. S. Gordon and M. W. Schmidt, in *Theory and Applications of Computational Chemistry: The First Forty Years*, ed. G. Frenking, K. S. Kim and G. E. Scuseria, Elsevier, Amsterdam, 2005, p. 1167.

38 M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, P. G. A., H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, M. Rega, N. J. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. E. Gomperts, O. Stratmann, A. J. Yazyev, R. Austin, C. Cammi, J. W. Pomelli, R. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski and D. J. Fox, *GAUSSIAN 09 (Revision A.1)*, Gaussian, Inc., Wallingford, CT, 2009.

39 H.-J. Werner, P. J. Knowles, R. Lindh, F. R. Manby, M. Schütz, P. Celani, T. Korona, A. Mitrushenkov, G. Rauhut, T. B. Adler, R. D. Amos, A. Bernhardsson, A. Berning, D. L. Cooper, M. J. O. Deegan, A. J. Dobbyn, F. Eckert, E. Goll, C. Hampel, G. Hetzer, T. Hrenar, G. Knizia, C. Köppl, Y. Liu, A. W. Lloyd, R. A. Mata, A. J. May, S. J. McNicholas, W. Meyer, M. E. Mura, A. Nicklaß, P. Palmieri, K. Pflüger, R. Pitzer, M. Reiher, U. Schumann, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, M. Wang and A. Wolf, MOLPRO is a package of *ab initio* programs.

40 T. Müller, *J. Phys. Chem. A*, 2009, **113**, 12729.

41 H. Lischka, R. Shepard, R. M. Pitzer, I. Shavitt, M. Dallos, T. Müller, P. G. Szalay, M. Seth, G. S. Kedziora, S. Yabushita and Z. Zhang, *Phys. Chem. Chem. Phys.*, 2001, **3**, 664.

42 J. Nieplocha, B. Palmer, V. Tipparaju, M. Kirshnan, H. Trease and E. Apra, *Int. J. High Perf. Comp. App.*, 2006, **20**, 203.

43 G. D. Fletcher, M. W. Schmidt, B. M. Bode and M. S. Gordon, *Comput. Phys. Commun.*, 2000, **128**, 190.

44 D. M. Fedorov, R. M. Olsen, K. Kitaura, M. S. Gordon and S. Koseki, *J. Comput. Chem.*, 2004, **25**, 872.

45 N. Carriero and D. Gelernter, *How to Write Parallel Programs: A First Course*, MIT Press, New Haven, 1990.

46 M. Wang, A. J. May and P. J. Knowles, *Comput. Phys. Commun.*, 2009, **180**, 2673.

47 W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir and M. Snir, *MPI The complete reference, Volume 2—The MPI-2 Extensions*, MIT Press, New Haven, 1998.

48 K. Parzyszek, J. Nieplocha and R. A. Kendall, in *Proceedings of the IASTED Parallel and Distributed Computing and Systems PDCS-2000*, Calgary, 2000.

49 R. W. Numrich and J. Reid, *ACM Fortran Forum*, 2005, **24**, 4.

50 H. J. J. van Dam, M. Wang, A. G. Sunderland, I. J. Bush, P. J. Knowles and M. F. Guest, *Is MPI-2 suitable for Quantum Chemistry? Performance of passive target one-sided communications, HPCxTR0807, STFC*, 2008.
51 B. Lewis and D. J. Berg, *Multithreaded Programming with Pthreads*, Prentice-Hall Inc., Upper Saddle River, NJ, 1998.
52 OpenMP specification: OpenMP Application Program Interface, Version 3.0. Available from http://www.openmp.org, May 2008.
53 I. M. B. Nielsen and C. L. Janssen, *Comput. Phys. Commun.*, 2000, **128**, 238.
54 OpenCL specification. Version 1.0.48. Available from http://www.khronos.org/opencl, October 2009.
55 G. M. Shipman, D. A. Dillow, S. Oral and F. Wang, *Cray User Group Proceedings*, Atlanta, 2009.
56 S. Goedecker, *Rev. Mod. Phys.*, 1999, **71**, 1085.
57 M. Challacombe and E. Schwegler, *J. Chem. Phys.*, 1997, **106**, 5526.
58 D. S. Lambrecht and C. Ochsenfeld, *J. Chem. Phys.*, 2005, **123**, 184102.
59 C. Ochsenfeld, J. Kussmann and D. S. Lambrecht, in *Rev. Comput. Chem.*, ed. K. B. Lipkowitz, T. R. Cundari and D. B. Boyd, John Wiley & Sons Inc., Hoboken, 2007, vol. 23.
60 C. A. White and M. Head-Gordon, *J. Chem. Phys.*, 1994, **101**, 6593.
61 C. K. Gan, P. D. Haynes and M. C. Payne, *Comput. Phys. Commun.*, 2001, **134**, 33.
62 N. Govind, J. Wang and H. Guo, *Phys. Rev. B*, 1994, **50**, 11175.
63 P. Y. Ayala and G. E. Scuseria, *J. Chem. Phys.*, 1999, **110**, 3660.
64 R. A. Friesner, R. B. Murphy, M. D. Beachy, M. N. Ringnalda, W. T. Pollard, B. D. Dunietz and Y. J. Cao, *J. Phys. Chem. A*, 1999, **103**, 1913.
65 G. Hetzer, M. Schütz, H. Stoll and H.-J. Werner, *J. Chem. Phys.*, 2000, **113**, 9443.
66 M. S. Lee, P. E. Maslen and M. Head-Gordon, *J. Chem. Phys.*, 2000, **112**, 3592.
67 I. M. B. Nielsen and C. L. Janssen, *J. Chem. Theory Comput.*, 2007, **3**, 71.
68 G. Rauhut, P. Pulay and H.-J. Werner, *J. Comput. Chem.*, 1998, **19**, 1241.
69 S. Saebø and P. Pulay, *J. Chem. Phys.*, 2001, **115**, 3975.
70 N. Flocke and R. J. Bartlett, *J. Chem. Phys.*, 2004, **121**, 10935.
71 M. Schütz and F. R. Manby, *Phys. Chem. Chem. Phys.*, 2003, **5**, 3349.
72 G. E. Scuseria and P. Y. Ayala, *J. Chem. Phys.*, 1999, **111**, 8330.
73 D. R. Bowler, R. Choudhury, M. J. Gillan and T. Miyazaki, *Phys. Status Solidi B*, 2006, **243**, 989.
74 M. Challacombe, *J. Chem. Phys.*, 2000, **113**, 10037.
75 F. Jensen, *Introduction to Computational Chemistry*, John Wiley & Sons Ltd., Chichester, 1999.
76 A. Szabo and N. S. Ostlund, *Modern Quantum Chemistry*, McGraw-Hill Inc., New York, 1996.
77 R. Dreizler and E. Gross, *Density Functional Theory*, Plenum Press, New York, 1995.
78 *A Primer in Density Functional Theory*, ed. C. Fiolhais, F. Nogueira and M. Marques, Springer Berlin, Heidelberg, 2003.
79 P. Hohenberg and W. Kohn, *Phys. Rev.*, 1964, **136**, B864.
80 W. Kohn and L. J. Sham, *Phys. Rev.*, 1965, **140**, A1133.
81 R. G. Parr and W. Yang, *Density-Functional Theory of Atoms and Molecules*, Oxford University Press, New York, 1989.
82 J. Kohanoff, *Electronic Structure Calculations for Solids and Molecules: Theory and Computational Methods*, Cambridge University Press, Cambridge, 2006.
83 S. Kummel and L. Kronik, *Rev. Mod. Phys.*, 2008, **80**, 3.
84 R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press, Cambridge, 2004.
85 J. P. Perdew and K. Schmidt, in *AIP Conference Proceedings: Density Functional Theory and Its Application to Materials*, ed. V. Van Doren, C. Van Alsenoy and P. Geerlings, American Institute of Physics, 2001, vol. 577.
86 G. E. Scuseria and V. N. Staroverov, in *Theory and Applications of Computational Chemistry The First Forty Years*, ed. C. E. Dykstra, G. Frenking, K. S. Kim and G. E. Scuseria, Elsevier, Amsterdam, 2005.
87 T. Shimazaki and Y. Asai, *J. Chem. Theory Comput.*, 2009, **5**, 136.
88 P. M. W. Gill, *Adv. Quantum Chem.*, 1994, **25**, 141.
89 M. Häser and R. Ahlrichs, *J. Comput. Chem.*, 1989, **10**, 104.
90 T. B. Pedersen, F. Aquilante and R. Lindh, *Theor. Chem. Acc.*, 2009, **124**, 1.
91 B. I. Dunlap, J. W. D. Connolly and J. R. Sabin, *J. Chem. Phys.*, 1979, **71**, 3396.
92 R. Lindh, J. Krogh, M. Schütz and K. Hirao, *Theor. Chim. Acta*, 2003, **110**, 156.
93 G. Karlström, R. Lindh, P.-Å. Malmqvist, B. O. Roos, U. Ryde, V. Veryazov, P.-O. Widmark, M. Cossi, B. Schimmelpfennig, P. Neogrady and L. Seijo, *Comput. Mater. Sci.*, 2003, **28**, 222.
94 V. Veryazov, P.-O. Widmark, L. Serrano-Andrés, R. Lindh and B. O. Roos, *Int. J. Quantum Chem.*, 2004, **100**, 2207.
95 F. Aquilante, L. De Vico, N. Ferré, G. Ghigo, P.-Å. Malmqvist, P. Neogrády, T. B. Pedersen, M. Pitoňak, M. Reiher, B. O. Roos, L. Serrano-Andrés, M. Urban, V. Veryazov and R. Lindh, *J. Comput. Chem.*, 2010, **31**, 224.
96 H. Fruchtl, R. A. Kendall, R. J. Harrison and K. Dyall, *Int. J. Quantum Chem.*, 1997, **64**, 63.
97 T. R. Furlani, J. Kong and P. M. W. Gill, *Comput. Phys. Commun.*, 2000, **128**, 170.
98 C. Hättig, A. Hellweg and A. Köhn, *Phys. Chem. Chem. Phys.*, 2006, **8**, 1159.
99 F. Neese, F. Wennmohs, A. Hansen and U. Becker, *Chem. Phys.*, 2009, **356**, 98.
100 M. von Arnim and R. Ahlrichs, *J. Comput. Chem.*, 1998, **19**, 1746.
101 Y. Alexeev, R. A. Kendall and M. S. Gordon, *Comput. Phys. Commun.*, 2002, **143**, 69.
102 I. T. Foster, J. L. Tilson, A. F. Wagner, R. L. Shepard, R. J. Harrison, R. A. Kendall and R. J. Littlefield, *J. Comput. Chem.*, 1996, **17**, 109.
103 T. R. Furlani and H. F. King, *J. Comput. Chem.*, 1995, **16**, 91.
104 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. L. Tilson, R. L. Shepard, A. F. Wagner, I. T. Foster, E. Lusk and R. Stevens, *J. Comput. Chem.*, 1996, **17**, 124.
105 J. Nieplocha, R. J. Harrison and R. J. Littlefield, in *Proceedings of the 1994 Conference of Supercomputing*, Los Alamos, 1994.
106 Y. Alexeev, M. W. Schmidt, T. L. Windus and M. S. Gordon, *J. Comput. Chem.*, 2007, **28**, 1685.
107 E. Apra, E. J. Bylaska, D. J. Dean, A. Fortunelli, F. Gao, P. S. Krstic, J. C. Wells and T. L. Windus, *Comput. Mater. Sci.*, 2003, **28**, 209.
108 R. E. Stratmann, G. E. Scuseria and M. J. Frish, *Chem. Phys. Lett.*, 1996, **257**, 213.
109 J. Hutter, M. Parrinello and S. Vogel, *J. Chem. Phys.*, 1994, **101**, 3862.
110 K. N. Kudin and G. E. Scuseria, *Math. Modell. Numer. Anal.*, 2007, **41**, 281.
111 A. P. Rendell, *Chem. Phys. Lett.*, 1994, **229**, 204.
112 A. T. Wong and R. J. Harrison, *J. Comput. Chem.*, 1995, **16**, 1291.
113 Scalapack specifications can be found at http://www.netlib.org/scalapack/scalapack_home.html.
114 J. C. Phillips, *Phys. Rev.*, 1958, **112**, 685.
115 J. C. Phillips and L. Kleinman, *Phys. Rev.*, 1959, **116**, 287.
116 B. J. Austin, V. Heine and L. J. Sham, *Phys. Rev.*, 1962, **127**, 276.
117 M. T. Yin and M. L. Cohen, *Phys. Rev. B*, 1982, **25**, 7403.
118 G. B. Bachelet, D. R. Hamann and M. Schluter, *Phys. Rev. B*, 1982, **26**, 4199.
119 D. R. Hamann, *Phys. Rev. B*, 1989, **40**, 2980.
120 P. E. Blochl, *Phys. Rev. B*, 1994, **50**, 17953.
121 N. A. W. Holzwarth, G. E. Mathews, A. R. Tackett and R. B. Dunning, *Phys. Rev. B*, 1998, **57**, 11827.
122 M. Valiev and J. H. Weare, *J. Phys. Chem. B*, 1999, **103**, 10588.
123 G. Kresse and D. Joubert, *Phys Rev B*, 1999, **59**, 1758.
124 M. Valiev, E. J. Bylaska, A. Gramada and J. H. Weare, in *Reviews In Modern Quantum Chemistry: A Celebration Of The Contributions Of R. G. Parr*, ed. K. D. Sen, World Scientific, Singapore, 2002.
125 E. J. Bylaska, M. Valiev, R. Kawai and J. H. Weare, *Comput. Phys. Commun.*, 2002, **143**, 11.
126 L. J. Clarke, I. Stich and M. C. Payne, *Comput. Phys. Commun.*, 1992, **72**, 14.
127 J. S. Nelson, S. J. Plimpton and M. P. Sears, *Phys. Rev. B*, 1993, **47**, 1765.

128 J. Wiggs and H. Jonsson, *Comput. Phys. Commun.*, 1994, **81**, 1.
129 D. Marx and J. Hutter, in *Modern Methods and Algorithms of Quantum Chemistry*, ed. J. Grotendorst, Forschungszentrum, Julich Germany, 2000, vol. 1, p. 301.
130 F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, F. Franchetti, S. Kral, C. W. Ueberhuber and J. Lorenz, in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.
131 T. P. Hamilton and P. Pulay, *J. Chem. Phys.*, 1986, **84**, 5728.
132 van de Geign, Robert and J. Watts, *Concurrency and Computation: Practice and Experience*, 1997, **9**, 255.
133 W. Kutzelnigg, *Int. J. Quantum Chem.*, 2009, **109**, 3858.
134 J. Baker, K. Wolinski, M. Malagoli, D. Kinghorn, P. Wolinski, G. Magyarfalvi, S. Saebo, T. Janowski and P. Pulay, *J. Comput. Chem.*, 2009, **30**, 317.
135 J. L. Bentz, R. M. Olsen, M. S. Gordon, M. W. Schmidt and R. A. Kendall, *Comput. Phys. Commun.*, 2007, **176**, 589.
136 A. R. Ford, T. Janowski and P. Pulay, *J. Comput. Chem.*, 2007, **28**, 1215.
137 M. E. Harding, T. Metzroth and J. Gauss, *J. Chem. Theory Comput.*, 2008, **4**, 64.
138 S. Hirata, *J. Phys. Chem. A*, 2003, **107**, 9887.
139 S. Hirata, *Theor. Chem. Acc.*, 2006, **116**, 2.
140 R. Kobayashi and A. P. Rendell, *Chem. Phys. Lett.*, 1997, **265**, 1.
141 H. Koch, A. S. de Meras, T. Helgaker and O. Christiansen, *J. Chem. Phys.*, 1996, **104**, 4157.
142 T. Kuś, V. F. Lotrich and R. J. Bartlett, *J. Chem. Phys.*, 2009, **130**, 124122.
143 V. F. Lotrich, N. Flocke, M. Ponton, A. D. Yau, A. Perera, E. Deumens and R. J. Bartlett, *J. Chem. Phys.*, 2008, **128**, 194104.
144 R. M. Olsen, J. L. Bentz, R. A. Kendall, M. W. Schmidt and M. S. Gordon, *J. Chem. Theory Comput.*, 2007, **3**, 1312.
145 P. Piecuch, S. A. Kucharski, K. Kowalski and M. Musial, *Comput. Phys. Commun.*, 2002, **149**, 71.
146 P. Pulay, S. Saebo and W. Meyer, *J. Chem. Phys.*, 1984, **81**, 1901.
147 A. P. Rendell, M. F. Guest and R. A. Kendall, *J. Comput. Chem.*, 1993, **14**, 1429.
148 A. P. Rendell, T. J. Lee and A. Komornicki, *Chem. Phys. Lett.*, 1991, **178**, 462.
149 A. P. Rendell, T. J. Lee, A. Komornicki and S. Wilson, *Theor. Chim. Acta*, 1993, **84**, 271.
150 A. P. Rendell, T. J. Lee and R. Lindh, *Chem. Phys. Lett.*, 1992, **194**, 84.
151 M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis and J. A. Montgomery, *J. Comput. Chem.*, 1993, **14**, 1347.
152 C. Møller and M. S. Plesset, *Phys. Rev.*, 1934, **46**, 618.
153 J. Paldus and X. Z. Li, in *Adv. Chem. Phys.*, ed. I. Prigogine and S. A. Rice, John Wiley & Sons Inc, New York, 1999, vol. 110, p. 1.
154 J. Almlöf, *Chem. Phys. Lett.*, 1991, **181**, 319.
155 M. Häser and J. Almlöf, *J. Chem. Phys.*, 1992, **96**, 489.
156 D. E. Bernholdt and R. J. Harrison, *J. Chem. Phys.*, 1998, **109**, 1593.
157 M. Feyereisen, G. Fitzgerald and A. Komornicki, *Chem. Phys. Lett.*, 1993, **208**, 359.
158 C. Hättig, *Phys. Chem. Chem. Phys.*, 2005, **7**, 59.
159 F. Weigend and M. Häser, *Theor. Chim. Acta*, 1997, **97**, 331.
160 F. Weigend, M. Häser, H. Patzelt and R. Ahlrichs, *Chem. Phys. Lett.*, 1998, **294**, 143.
161 F. Weigend, A. Köhn and C. Hättig, *J. Chem. Phys.*, 2002, **116**, 3175.
162 H. Koch, A. Sánchez de Merás and T. B. Pedersen, *J. Chem. Phys.*, 2003, **118**, 9481.
163 P. E. Maslen and M. Head-Gordon, *Chem. Phys. Lett.*, 1998, **283**, 102.
164 P. E. Maslen and M. Head-Gordon, *J. Chem. Phys.*, 1998, **109**, 7093.
165 S. Saebo and P. Pulay, *Annu. Rev. Phys. Chem.*, 1993, **44**, 213.
166 M. Schütz, G. Hetzer and H.-J. Werner, *J. Chem. Phys.*, 1999, **111**, 5691.
167 R. J. Harrison and R. Shepard, *Annu. Rev. Phys. Chem.*, 1994, **45**, 623.
168 J. D. Watts and M. Dupuis, *J. Comput. Chem.*, 1988, **9**, 158.
169 A. M. Márquez and M. Dupuis, *J. Comput. Chem.*, 1995, **16**, 395.
170 D. E. Bernholdt and R. J. Harrison, *J. Chem. Phys.*, 1995, **102**, 9582.
171 D. E. Bernholdt and R. J. Harrison, *Chem. Phys. Lett.*, 1996, **250**, 477.
172 M. Schütz and R. Lindh, *Theor. Chim. Acta*, 1997, **95**, 13.
173 I. M. B. Nielsen and E. T. Seidl, *J. Comput. Chem.*, 1995, **16**, 1301.
174 J. Baker and P. Pulay, *J. Comput. Chem.*, 2002, **23**, 1150.
175 G. D. Fletcher, M. W. Schmidt and M. S. Gordon, *Adv. Chem. Phys.*, 1999, **110**, 267.
176 K. Ishimura, P. Pulay and S. Nagase, *J. Comput. Chem.*, 2006, **27**, 407.
177 M. Katouda and S. Nagase, *Int. J. Quantum Chem.*, 2009, **109**, 2121.
178 B. Doser, D. S. Lambrecht, J. Kussmann and C. Ochsenfeld, *J. Chem. Phys.*, 2009, **130**, 064107.
179 R. J. Bartlett and M. Musial, *Rev. Mod. Phys.*, 2007, **79**, 291.
180 J. Cizek, *J. Chem. Phys.*, 1966, **45**, 4256.
181 F. Coester, *Nucl. Phys.*, 1958, **7**, 421.
182 F. Coester and H. Kümmel, *Nucl. Phys.*, 1960, **17**, 477.
183 T. D. Crawford and H. F. Schaefer III, in *Rev. Comput. Chem.*, ed. K. B. Lipkowitz and D. B. Boyd, Wiley-Vch Inc, New York, 2000, vol. 14, p. 33.
184 J. Paldus, I. Shavitt and J. Cizek, *Phys. Rev. A*, 1972, **5**, 50.
185 P. Piecuch, K. Kowalski, I. S. O. Pimienta and M. J. McGuire, *Int. Rev. Phys. Chem.*, 2002, **21**, 527.
186 J. M. Cullen and M. C. Zerner, *J. Chem. Phys.*, 1982, **77**, 4088.
187 G. D. Purvis and R. J. Bartlett, *J. Chem. Phys.*, 1982, **76**, 1910.
188 J. Noga and R. J. Bartlett, *J. Chem. Phys.*, 1987, **86**, 7041.
189 G. E. Scuseria and H. F. Schaefer, *Chem. Phys. Lett.*, 1988, **152**, 382.
190 K. Raghavachari, G. W. Trucks, J. A. Pople and M. Head-Gordon, *Chem. Phys. Lett.*, 1989, **157**, 479.
191 E. Dalgaard and H. J. Monkhorst, *Phys. Rev. A*, 1983, **28**, 1217.
192 H. Koch, H. J. A. Jensen, P. Jorgensen and T. Helgaker, *J. Chem. Phys.*, 1990, **93**, 3345.
193 H. J. Monkhorst, *Int. J. Quantum Chem*, 1977, **S11**, 421.
194 D. C. Comeau and R. J. Bartlett, *Chem. Phys. Lett.*, 1993, **207**, 414.
195 J. Geertsen, M. Rittby and R. J. Bartlett, *Chem. Phys. Lett.*, 1989, **164**, 57.
196 H. Sekino and R. J. Bartlett, *Int. J. Quantum Chem.*, 1984, **S18**, 255.
197 J. F. Stanton and R. J. Bartlett, *J. Chem. Phys.*, 1993, **98**, 7029.
198 K. Kowalski and P. Piecuch, *J. Chem. Phys.*, 2001, **115**, 643.
199 J. D. Watts and R. J. Bartlett, *Chem. Phys. Lett.*, 1995, **233**, 81.
200 J. D. Watts and R. J. Bartlett, *Chem. Phys. Lett.*, 1996, **258**, 581.
201 O. Christiansen, H. Koch and P. Jørgensen, *J. Chem. Phys.*, 1996, **105**, 1451.
202 O. Christiansen, H. Koch, P. Jørgensen and J. Olsen, *Chem. Phys. Lett.*, 1996, **256**, 185.
203 K. Kowalski and P. Piecuch, *J. Chem. Phys.*, 2004, **120**, 1715.
204 K. Kowalski, *J. Chem. Phys.*, 2009, **130**, 194110.
205 S. Hirata, M. Nooijen, I. Grabowski and R. J. Bartlett, *J. Chem. Phys.*, 2001, **114**, 3919.
206 T. Shiozaki, K. Hirao and S. Hirata, *J. Chem. Phys.*, 2007, **126**, 244106.
207 P. U. Manohar and A. I. Krylov, *J. Chem. Phys.*, 2008, **129**, 10.
208 S. A. Kucharski and R. J. Bartlett, *Theor. Chim. Acta*, 1991, **80**, 387.
209 T. D. Crawford, T. J. Lee and H. F. Schaefer III, *J. Chem. Phys.*, 1997, **107**, 7943.
210 P. Jankowski and B. Jeziorski, *J. Chem. Phys.*, 1999, **111**, 1857.
211 C. L. Janssen and H. F. Schaefer III, *Theor. Chim. Acta*, 1991, **79**, 1.
212 M. Kállay and P. R. Surján, *J. Chem. Phys.*, 2001, **115**, 2945.
213 X. Z. Li and J. Paldus, *J. Chem. Phys.*, 1994, **101**, 8812.
214 M. Nooijen and V. F. Lotrich, *THEOCHEM*, 2001, **547**, 253.
215 L. Pollack, T. L. Windus, W. A. de Jong and D. A. Dixon, *J. Phys. Chem. A*, 2005, **109**, 6934.
216 E. Apra, R. J. Harrison, W. A. de Jong, A. P. Rendell, V. Tipparaju, S. S. Xantheas and R. M. Olsen, in *Proceedings of the ACM/IEEE Supercomputing 2009 Conference*, 2009.
217 P. D. Fan, M. Valiev and K. Kowalski, *Chem. Phys. Lett.*, 2008, **458**, 205.

218 K. Kowalski, J. R. Hammond, W. A. de Jong and A. J. Sadlej, *J. Chem. Phys.*, 2008, **129**, 226101.
219 K. Kowalski, O. Villa, S. Krishnamoorthy, J. R. Hammond and N. Govind, *J. Chem. Phys.*, 2010, **132**, 154103.
220 QTP Coupled cluster benchmarks are listed on http://www.qtp.ufl.edu/PCCworkshop/PCCbenchmarks.html.
221 T. Janowski and P. Pulay, *J. Chem. Theory Comput.*, 2008, **4**, 1585.
222 M. Pitoňak, T. Janowski, P. Neogrády, P. Pulay and P. Hobza, *J. Chem. Theory Comput.*, 2009, **5**, 1761.
223 S. Krishnamoorthy and K. Kowalski, unpublished, 2010.
224 H. Lischka, M. Dallos and R. L. Shepard, *Mol. Phys.*, 2002, **100**, 1647.
225 R. L. Shepard, in *Modern Electronic Structure Theory*, World Scientific, Singapore, 1995.
226 R. L. Shepard, H. Lischka, P. G. Szalay, T. Kovar and M. Ernzerhof, *J. Chem. Phys.*, 1992, **96**, 2085.
227 G. J. Atchity, S. S. Xantheas and K. Ruedenberg, *J. Chem. Phys.*, 1991, **95**, 1862.
228 B. H. Lengsfield III, P. Saxe and D. R. Yarkony, *J. Chem. Phys.*, 1984, **81**, 4549.
229 H. Lischka, M. Dallos, P. G. Szalay, D. R. Yarkony and R. L. Shepard, *J. Chem. Phys.*, 2004, **120**, 7322.
230 T. Fleig, H. J. A. Jensen, J. Olsen and L. Visscher, *J. Chem. Phys.*, 2006, **124**, 104106.
231 S. Yabushita, Z. Zhang and R. M. Pitzer, *J. Phys. Chem. A*, 1999, **103**, 5791.
232 S. Knecht, H. J. A. Jensen and T. Fleig, *J. Chem. Phys.*, 2010, **132**, 014108.
233 L. Visscher, O. Visser, P. J. C. Aerts, H. Merenga and W. C. Nieuwpoort, *Comput. Phys. Commun.*, 1994, **81**, 120.
234 P.-Å. Malmqvist, A. Rendall and B. O. Roos, *J. Phys. Chem.*, 1990, **94**, 5477.
235 L. B. Harding and W. A. Goddard III, *J. Am. Chem. Soc.*, 1975, **97**, 6293.
236 C. C. J. Roothaan and J. H. Detrich, *Int. J. Quantum Chem*, 1979, **S13**, 79.
237 R. L. Shepard, in *Advances in Chemical Physics: Ab initio Methods in Quantum Chemistry Part 2, Volume 69*, ed. K. P. Lawley, John Wiley & Sons Ltd., Chichester, 1987.
238 H.-J. Werner, in *Advances in Chemical Physics: Ab initio Methods in Quantum Chemistry Part 2, Volume 69*, ed. K. P. Lawley, John Wiley & Sons Ltd., Chichester, 1987.
239 R. N. Diffendorfer and D. R. Yarkony, *J. Phys. Chem.*, 1982, **86**, 5098.
240 H.-J. Werner and W. Meyer, *J. Chem. Phys.*, 1981, **74**, 5794.
241 A. T. Wong, R. J. Harrison and A. P. Rendall, *Theor. Chim. Acta*, 1996, **93**, 317.
242 F. Aquilante, T. B. Pedersen, B. O. Roos, A. Sanchez de Meras and H. Koch, *J. Chem. Phys.*, 2008, **129**, 024113.
243 J. Olsen, B. O. Roos, P. Jørgensen and H. J. A. Jensen, *J. Chem. Phys.*, 1988, **89**, 2185.
244 G. D. Fletcher, *Mol. Phys.*, 2007, **105**, 2971.
245 T. L. Windus, M. W. Schmidt and M. S. Gordon, *Theor. Chim. Acta*, 1994, **89**, 77.
246 B. O. Roos and P. E. M. Seigbahn, in *Methods of Electronic Structure Theory, Volume 3*, ed. H. F. Shaefer III, Plenum Press, New York, 1977, p. 277.
247 I. Shavitt, in *Methods of Electronic Structure Theory, Volume 3*, ed. H. F. Shaefer III, Plenum Press, New York, 1977, p. 189.
248 E. R. Davidson, *J. Comput. Phys.*, 1975, **17**, 87.
249 J. Olsen, P. Jørgensen and J. Simons, *Chem. Phys. Lett.*, 1990, **169**, 463.
250 I. Shavitt, in *The Unitary Group for the Evaluation of Electronic Energy Matrix Elements (Lecture Notes in Chemistry 22)*, ed. J. Hinze, Springer-Verlag, Berlin, 1981, p. 51.
251 N. C. Handy, *Chem. Phys. Lett.*, 1980, **74**, 280.
252 P. J. Knowles and N. C. Handy, *Chem. Phys. Lett.*, 1984, **111**, 315.
253 W. Duch, *GRMS or Graphical Representation of Model Spaces*, Springer, Berlin, 1986.
254 W. Duch and J. Karwowski, in *The Unitary Group for the Evaluation of Electronic Energy Matrix Elements (Lecture Notes in Chemistry 22)*, ed. J. Hinze, Springer-Verlag, Berlin, 1981, p. 260.
255 R. L. Shepard, I. Shavitt and H. Lischka, *J. Comput. Chem.*, 2002, **23**, 1121.
256 R. Ansaloni, G. L. Bendazzoli, S. Evangelisti and E. Rossi, *Comput. Phys. Commun.*, 2000, **128**, 496.
257 N. Ben-Amor, S. Evangelisti, D. Maynau and E. Rossi, *Chem. Phys. Lett.*, 1998, **288**, 348.
258 Z. Gan, Y. Alexeev, M. S. Gordon and R. A. Kendall, *J. Chem. Phys.*, 2003, **119**, 47.
259 K. Tanaka, Y. Mochizuki, T. Ishikawa, H. Terashima and H. Tokiwa, *Theor. Chem. Acc.*, 2007, **117**, 397.
260 B. Suo, G. Zhai, Y. Wang, Z. Wen, X. Hu and L. Li, *J. Comput. Chem.*, 2005, **26**, 88.
261 H. Lischka, R. Shepard, F. B. Brown and I. Shavitt, *Int. J. Quantum Chem*, 1981, **S15**, 91.
262 R. Shepard, I. Shavitt, R. M. Pitzer, D. C. Comeau, M. Pepper, H. Lischka, P. G. Szalay, R. Ahlrichs, F. B. Brown and J. Zhao, *Int. J. Quantum Chem.*, 1988, **S22**, 149.
263 S. Krebs and R. J. Buenker, *J. Chem. Phys.*, 1995, **103**, 5613.
264 R. J. Buenker and S. D. Peyerimhoff, *Theor. Chim. Acta*, 1975, **39**, 217.
265 E. R. Davidson, in *Modern Techniques in Computational Chemistry: MOTECC-90*, ed. E. Clementi, ESCOM Science Publishers, Leiden, 1990, p. 553.
266 R. Shepard, M. Minkoff and S. R. Brozell, *Int. J. Quantum Chem.*, 2007, **107**, 3203.
267 H. Lischka, R. Shepard, I. Shavitt, R. M. Pitzer, M. Dallos, T. Müller, P. G. Szalay, F. B. Brown, R. Ahlrichs, H. J. Boehm, A. Chang, D. C. Comeau, R. Gdanitz, H. Dachsel, C. Ehrhardt, M. Ernzerhof, P. Höchtl, S. Irle, G. Kedziora, T. Kovar, V. Parasuk, M. J. M. Pepper, P. Scharf, H. Schiffer, M. Schindler, M. Schüler, M. Seth, E. A. Stahlberg, J.-G. Zhao, S. Yabushita, Z. Zhang, M. Barbatti, S. Matsika, M. Schuurmann, D. R. Yarkony, S. R. Brozell, E. V. Beck and J.-P. Bleaudeau, COLUMBUS, an *ab initio* electronic structure program, release 5.9.2.JSC, 2008.
268 H. Dachsel and H. Lischka, *Theor. Chim. Acta*, 1995, **92**, 339.
269 A. J. Dobbyn, P. J. Knowles and R. J. Harrison, *J. Comput. Chem.*, 1998, **19**, 1215.
270 S. Knecht, H. J. A. Jensen and T. Fleig, *J. Chem. Phys.*, 2008, **128**, 014108.
271 K. Yasuda, *J. Chem. Theory Comput.*, 2008, **4**, 1230.
272 L. Vogt, R. Olivares-Amaya, S. Kermes, Y. Shao, C. Amador-Bedolla and A. Aspuru-Guzik, *J. Phys. Chem. A*, 2008, **112**, 2049.
273 I. S. Ufimtsev and T. J. Martinez, *J. Chem. Theory Comput.*, 2009, **5**, 1004.
274 I. S. Ufimtsev and T. J. Martinez, *J. Chem. Theory Comput.*, 2008, **4**, 222.

