CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–17
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Performance Characterization of Global Address Space Applications: A Case Study with NWChem

Jeff R. Hammond[*], Sriram Krishnamoorthy[+], Sameer Shende[‡]
Nichols A. Romero[*], Allen D. Malony[‡]

[*] *Argonne National Laboratory* [+] *Pacific Northwest National Laboratory* [‡] *University of Oregon*

SUMMARY

The use of global address space languages and one-sided communication for complex applications is gaining attention in the parallel computing community. However, lack of good evaluative methods to observe multiple levels of performance makes it difficult to isolate the cause of performance deficiencies and to understand the fundamental limitations of system and application design for future improvement. NWChem is a popular computational chemistry package which depends on the Global Arrays / ARMCI suite for partitioned global address space functionality to deliver high-end molecular modeling capabilities. A workload characterization methodology was developed to support NWChem performance engineering on large-scale parallel platforms. The research involved both the integration of performance instrumentation and measurement in the NWChem software, as well as the analysis of one-sided communication performance in the context of NWChem workloads. Scaling studies were conducted for NWChem on Blue Gene/P and on two large-scale clusters using different generation Infiniband interconnects and x86 processors. The performance analysis and results show how subtle changes in the runtime parameters related to the communication subsystem could have significant impact on performance behavior. The tool has successfully identified several algorithmic bottlenecks which are already being tackled by computational chemists to improve NWChem performance. Copyright © 2010 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: performance characterization, global address space, computational chemistry, NWChem

1. INTRODUCTION

The design of supercomputers with ever greater degrees of parallelism, more complex interconnection networks and communications hardware, and deeper hierarchies and multiple levels of locality in memory systems has always had a symbiotic relationship with the design of parallel algorithms and applications. While the computational demands of high-end applications drive the requirements for next-generation parallel platforms, it is clear that the nature of the parallel machines actually available to scientists will shape how the applications are developed, used, and optimized. Crucial to this process is a thorough knowledge of the characteristics of the applications that are expected to both produce scientific results with the present HPC generation and form the candidate workloads for future supercomputer procurements.

A cornerstone of system design has been the quantitative workload-driven exploration of the design space. Benchmarks have been developed to characterize various classes of applications, such as sequential [11], multi-threaded [49], transaction-based [52], shared-memory-based [49], and MPI-based [4]. Concern for both computational aspects (e.g., floating point rate and work throughput) as well as communication characteristics (e.g., bandwidth, latency, and collective operations) are highly relevant to high-end performance. However, the co-evolution of HPC systems and application design forces workload benchmarks to reflect modern programming methods. A case in point is the recent interest in addressing the productivity challenge in programming current and future supercomputers through the use of global address space languages and one-sided communication. Languages such as UPC [54], Co-Array Fortran [43], and newer HPCS languages – X10 [9], Chapel [8], and Fortress [50] – are examples based on the concept of extending global view programming techniques to operate efficiently on large-scale distributed memory machines.

Despite recent interest in applications based on these parallel programming methods, characterization of important workloads has not been adequately pursued. Lack of good evaluative methods to observe multiple levels of performance, from programming interfaces to hardware counters, makes it difficult to isolate the cause of performance deficiencies on current systems, and to understand the fundamental limitations of system design for future improvement.

In this paper, we characterize a key module in NWChem [13], an exemplar application employing the one-sided programming model. NWChem is a computational chemistry suite supporting electronic structure calculations using a variety of chemistry models. The computational cost to perform these calculations increases as a steep polynomial of the problem size ($O(N^{3-7})$). The capability provided by future supercomputers should enable more accurate calculations on larger molecular systems and have the potential to provide better understanding of such diverse phenomena as catalysis, photosynthesis and drug-design.

NWChem employs Global Arrays (GA) [41] as the underlying one-sided programming model. GA provides a global address space view of the distributed address spaces of different processes. Aggregate Remote Memory Copy Interface (ARMCI) [40] is the communication substrate that provides the remote memory access functionality used by GA. Being able to observe the performance characteristics of ARMCI in support of Global Arrays and in the context of NWChem is a key requirement for future development and optimization. For this purpose, we have extended the ARMCI library to support profiling using the TAU Performance System (henceforth referred to simply as TAU). The developed capability was then used to profile the key NWChem modules. The most interesting results obtained were for the module implementing the CCSD(T) method [45, 26], which is the subject of intense interest due to its capability to achieve petaflop/s performance [2]. We employ a modest problem size to identify challenges in strong-scaling real calculations on future supercomputers.

The primary contributions of this paper are:

1. Development of a profiling interface for ARMCI and integration with TAU.
2. Detailed performance analysis of the CCSD(T) module of NWChem on three different supercomputers.
3. Comparison of ARMCI with different usage modes related to the heavy use of one-sided communication by NWChem.
4. Determination of optimal comunication parameters for running NWChem on two state-of-the-art interconnects.

Section 2 describes ARMCI and its profiling interface. Section 3 discusses the organization of the NWChem software suite. Section 4 gives an overview of TAU. The systems configurations we used for performance testing are described in Section 5, followed by a detailed analysis in Section 6 of the workload characterization study.

2. THE ARMCI COMMUNICATION SYSTEM

Aggregate Remote Memory Copy Interface (ARMCI) [40] is the one-sided communication library that underpins Global Arrays and NWChem. ARMCI operates on a distributed memory view of a parallel system. Communication is performed through one-sided put and get operations on remote memory. Accumulate operations that atomically add to remote locations are also provided to support algorithms for which put is not sufficient. ARMCI provides a rich set of primitives including optimized blocking and non-blocking operations on contiguous, strided, and vector data. Atomic operations on remote memory locations have been used to design distributed algorithms and scalable dynamic load balancing schemes [10]. The ability to operate on remote data without synchronizing with another process extends the flexibility of shared memory programming to distributed memory machines. The support provided to query the location of a process with respect to an SMP node enables careful tuning of an application to maximize locality and minimize communication. ARMCI provides portable performance on a variety of high-performance interconnects and has been used as the communication substrate in implementing frameworks that provide higher-level abstractions, such as Global Arrays [41, 42] and GPSHMEM [44]. It is fully inter-operable with MPI, allowing applications to intermix the use of both programming models.

Designing and optimizing applications that rely on ARMCI, such as NWChem, and scaling to hundreds of thousands of cores is a challenging task that requires performance feedback at multiple stages of development, system deployment, and application use. This process can benefit significantly through the use of scalable performance engineering tools and practices that enable a full characterization of performance factors, analysis of their interactions, mining/learning of correlated features, and discovery of high-performing solutions in a multi-dimensional space of options.

Of specific interest to multicore parallelism in NWChem is that one-sided communication requires remote agency, leading communication runtime systems to often spawn a thread for processing incoming one-sided message requests. In ARMCI this thread is known as the *data-server*. In the results below, we demonstrate that the communication intensity of NWChem is such that the ARMCI data-server requires a dedicated core for optimal application performance, especially at scale. This disproves the naïve assumption that maximizing the utilization of cores for computation is the right way to maximize application performance. As the number of cores per socket continues to increase, dedicating a core per node or per socket to communication becomes less significant for the computational workload. Our results provide insight into how to design applications and one-sided communication runtimes for current and future multicore processors.

3. NWCHEM SOFTWARE SUITE

NWChem [13] is the premier massively-parallel quantum chemistry software package due to its portable scalability and performance on high-performance computing resources and breadth of features. The success of NWChem culminated in its demonstrated scaling to more than one petaflop/s on the JaguarPF supercomputer at Oak Ridge National Laboratory [1, 2], which was the result of significant effort into scaling the GA toolkit to more than 100,000 cores [51] and tuning of the application code for maximum parallel efficiency.

In addition to its use on leadership-class supercomputers, NWChem is a widely used computational chemistry code consuming large amounts of computing time on clusters. Due to the broad spectrum of scientific functionality in NWChem, it makes an excellent package to study to understanding quantum chemistry algorithms in general. The primary barrier to performance characterization of NWChem is that it employs a one-sided programming model as implemented with the Global Arrays (GA) runtime, as opposed to MPI. Until now, there have not been any tools available to measure the communication behavior of the ARMCI communication subsystem used by GA. Due to the ubiquitous use of NWChem for performing chemistry simulations on parallel computers, it is an ideal application using one-sided communication to characterize.We believe certain NWChem modules which use GA could form a candidate workload to drive such research for one-sided programming methods as part of a more general effort to understand applications which employ non-standard programming models (that is, those not relying on message-passing).

The first part of this section describes the kernels of quantum chemistry and their basic performance characteristics, while the second discusses high-level integration of kernels and communication patterns.

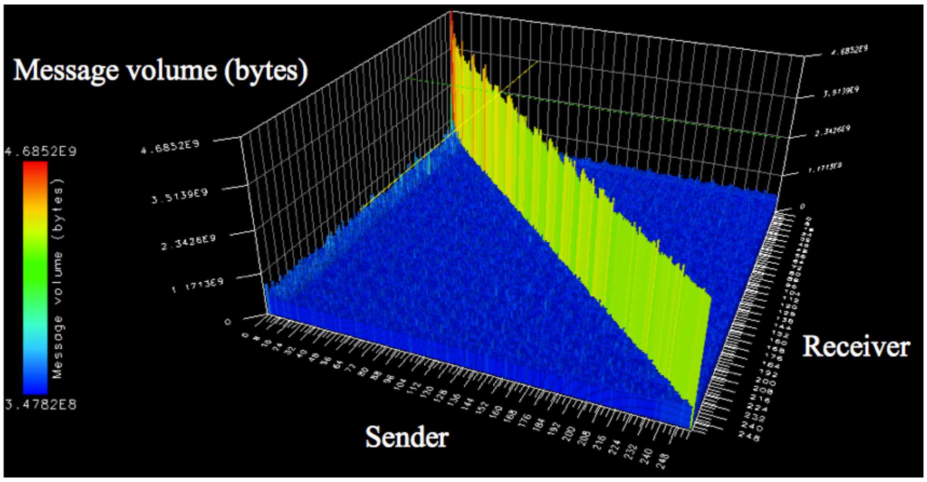


Figure 1. TAU's 3D communication matrix shows a flat communication profile. The yellow diagonal indicates local access via ARMCI, which does not utilize the network.

3.1. *Quantum Chemistry Kernels*

Almost all the computation associated with quantum chemistry calculations, at least inside of NWChem, are associated with two procedures: generation of atomic integrals and dense linear algebra.

Atomic integrals are the matrix elements of physical operations (e.g. Coulomb) in the atom-centered Gaussian basis set [24]. They can be summarized as being computationally intensive in both floating-point and integer computation, while generating significant memory traffic and branching [15, 20]. As such, they do not usually achieve more than 20% of the single-core peak performance on most architectures. A detailed description of the computation of these objects can be found elsewhere [17].

The asymptotically dominant cost of most quantum chemistry methods is the evaluation of complicated tensor-contractions. These operations are implemented as matrix-matrix multiplication (MMM) due to the immense effort devoted to optimized BLAS libraries [28], which provide vastly superior performance relative to implementations coded by non-experts [18], even if tensors must be reshaped in order to match MMM syntax [19].

3.2. *Coupled-Cluster Theory*

Coupled-cluster theory is a many-body method involving a complex set of multidimensional array computations. In this paper, we consider on the "gold-standard" CCSD(T) variant of coupled-cluster theory, which requires N^7 floating-point operations (almost all from `DGEMM`) and N^4 storage, where N is proportional to the molecular system and atomic basis set employed.

The first step in CCSD(T) is the four-index transformation, which requires a large number of `DGEMM` calls with a non-collective global transpose in the middle [57]. The iterative step, in which the CCSD equations are solved, requires `DGEMM` and atomic integral computations and is communication intensive due to the recursive intermediates which must be formed in the course of evaluating the residual for this complex set of nonlinear equations [26]. The final stage, in which the non-iterative (T) contribution is evaluated, requires first the processing of a large number of new atomic integrals and another four-index transformation, followed by a computationally-intensive stage which dominates the wall time and is dominated by large MMM operations. Nonetheless, it requires numerous one-sided `ARMCI_GetS` operations to build local intermediates. Overall, coupled-cluster is both computation- and communication-intensive. It is implemented without topology-awareness in NWChem, as demonstrated in the communication matrix shown in Figure 1. ARMCI optimizes local communication between processes on the same node, hence that communication does not contribute to network congestion. Note that while pictorially most of the communication appears to be along the diagonal, representing optimized intra-node shared memory copies, the range of the inter-processor communication volumes is shown by the heat map. In Figure 1 the inter-processor communication volumes vary from 3.4×10^8 to 4.6×10^9 bytes. Thus the non-local communication represents a significact fraction of the total data movement and is uniformly spread amongst all pairs of processes. Such flat communication matrices represent problematic usage on almost all networks, especially those with a torus topology, such as Blue Gene/P and, to a lesser extent, Cray XT (due to greater bandwidth).

3.3. Description of Test Input

The NWChem performance was studied using a mid-sized calculation which could be run across a large range of node counts with reasonable efficiency. Our benchmark test calculation is a small water cluster with a moderately large basis set, specifically, $(H_2O)_4$ with aug-cc-pVTZ (368 AO functions), henceforth referred to as w4. This test case is less than half the size of that used in recent performance demonstrations [2], but nonetheless large enough to strong-scale efficiently across a few hundred nodes of Blue Gene/P. The organization of the computation in CCSS(T) and its demonstration at scale imply that performance analysis of larger calculations will confirm findings at lower scale, and will not reveal any new information.

We performed our experiments on two large scale clusters, Fusion and Chinook, described in more detail in Section 5. For each performance experiment, most parameters of the NWChem job were kept exactly the same. The memory utilization was scaled accordingly for each machine's capacity and we disabled disk caching of integrals in CCSD on the Fusion system (`ccsd; nodisk; end`) as this decreased performance due to the disparity between the processor capability and the local disk performance. On Chinook, disk caching increases performance significantly due to excellent I/O capability, while on Blue Gene/P disk caching is effective due to the balance between I/O bandwidth and processor speed.

4. TAU PERFORMANCE SYSTEM

Our NWChem performance characterization studies targeted large-scale distributed memory platforms consisting of nodes with multicore processors. The tools we used for performance measurement and analysis needed to address challenges of observing intra-node thread performance and thread interactions via shared memory and global address space communication. We applied the TAU performance system for performance measurement and analysis. TAU (Tuning and Analysis Utilities) is research and development project at the University of Oregon creates state-of-the-art methods and technologies for parallel performance analysis and tuning with the primary objectives of portability, flexibility, and interoperability [48, 34, 47, 46]. The project produces and distributes the open source TAU parallel performance system, a robust, integrated suite of tools for instrumentation, measurement, analysis, and visualization of large-scale parallel applications. One of TAU's strengths is its support for instrumentation at multiple levels of program code transformation. Instrumentation calls can be directly inserted in the code using a manual API, or automatically using TAU's source code instrumentor for C, C++, and Fortran, which is based on the Program Database Toolkit (PDT) [31] package. TAU also supports compiler-based instrumentation to leverage the compiler's instrumentation capabilities for object code directly, rather than through source transformation. Profiling and tracing of performance data (e.g., execution time, hardware counters, communication statistics) are provided by TAU's measurement system.

4.1. Multicore Performance Measurement

Multicore performance measurement in NWCHem is supported by TAU since it can capture performance data specific to each thread's execution on a processor core. TAU utilizes the PAPI [6] library to obtain counters about a thread's use of a specific core and its cache/memory interactions. TAU combines this performance data with per-core timing to maintain profile and trace information about each NWChem thread at runtime. NWChem can assign different execution roles to each thread and assign cores for specific tasks. By capturing core-level performance data in each thread, TAU can characterize the performance behavior of the NWChem application with respect to core usage.

4.2. Profiling interface for ARMCI (PARMCI)

An important challenge for profiling the NWChem software was to capture events associated with the use of Global Arrays and the ARMCI communication substrate. The ARMCI instrumentation approach we developed is similar to what is used in the MPI library whereby an alternate "name-shifted" interface to the standard routines is created (called *PMPI* [14] in the case of MPI, where 'P' stands for 'profiling') and a new library is provided to substitute for the original calls. In the spirit of PMPI, we call the profiling interface for ARMCI, *PARMCI*. The power of this approach is that the name-shifted interface allows any wrapper library to be developed that wants to intercept the original ARMCI calls. We use PARMCI to create a TAU-instrumented library for ARMCI that captures entry/exit events and make performance measurements of time as well as communication statistics (e.g., bytes transmitted) between sender and receivers. PARMCI is now included as part of the ARMCI distribution.

Around each wrapped ARMCI call are pairs of interval events with calls to start and stop timers, as well as events to trigger atomic events with the size of one-sided communication primitives. Calculating the size of one-sided remote memory access operations requires iterating through the indices of arrays and invoking ARMCI runtime system calls to determine the size of a given array. This cumulative size and the destination process id are used to trigger atomic events that resemble tracking of point-to-point communication primitives in MPI. This helps us generate a communication matrix that shows the extent of communication in one-sided operations between a pair of sender and receiver tasks. TAU's paraprof profile browser supports 2D and 3D displays of communication matrix to show the extent and type of communication. This helps highlight the gross pattern of communication in the application.

4.3. Alternative Library Wrappers

Another challenge facing us was how to create instrumentation of routines found in multiple object files which are linked together to generate an executable, but whose source is unavailable. For NWChem, such routines included the `DGEMM` call from the vendor-optimized BLAS library. The idea was to create a wrapper library that defines a new interface to replace `DGEMM`, call it `__wrap_DGEMM`, and then internally invokes an alternate `DGEMM` interface, call it `__real_DGEMM`, passing all parameters to it. The library is then instrumented with TAU. The problem became how to get the wrapper library linked in.

The solution is to utilize special linking options (such as `-Wl,--wrap` for the GNU compilers) that leverage linker support to substitute a given routine with an alternate instrumented version. Thus, while linking the NWChem application, we can provide an `-Wl,--wrap DGEMM` option telling the linker to resolve any reference to `DGEMM` by `__wrap_DGEMM`. It also resolves any undefined reference to `__real_DGEMM` with the actual `DGEMM` call provided by the BLAS library. In this way, all invocations of DGEMM are automatically instrumented. In fact, multiple routines across code modules can be instrumented in this way. It was our goal to use portable instrumentation techniques that would allow us to execute workload characterization experiments on different platforms and compare the results.

4.4. Runtime Preloading

Additionally, TAU supports runtime preloading of a measurement library in the address space of an executing application to intercept library calls using the *tau_exec* tool. For instance, when it is invoked with the ARMCI option (*tau_exec -armci ./application*) it replaces the ARMCI shared library with a wrapper interposition library that invokes the PARMCI interface at runtime. This may be used with an un-instrumented dynamic executable under Linux to assess the performance of the ARMCI library calls made by the application. TAU also supports runtime preloading of other libraries to track the performance of POSIX I/O, MPI, memory allocation and de-allocation routines, CUDA, and OpenCL libraries using *tau_exec*.

5. SYSTEM CONFIGURATIONS

Given the importance of runtime support for one-sided programming, we chose system configurations that highlight the different design decisions used to implement ARMCI.

5.1. Infiniband Clusters

The design of the ARMCI over the Infiniband (IB) network is typical of many high-performance networks. Here, contiguous `put` and `get` operations directly map to RDMA operations where possible while the remaining operations are processed by a dedicated data server thread running on each SMP node. The data server thread ensures progress of communication without requiring anything of the target process. The following two IB machines, both of which support the OFED stack, were used:
Fusion: A 320-node Linux cluster with dual-socket Intel Nehalem-series quad-core processors (Xeon X5550) connected by QDR IB (Mellanox Technologies MT26428). Each node has 36 GB of memory and is connected to a SATA local disk and a GPFS shared filesystem. This machine is operated by the Argonne Laboratory Computing Resource Center (LCRC). Hyperthreading is disabled on Fusion as it has not been shown to improve performance of the relevant HPC workloads.
Chinook: A 2310-node Linux supercomputer with dual-socket AMD Barcelona-series quad-core processors (AMD Opteron 2354) connected by DDR IB (Mellanox Technologies MT25418 NICs and Voltaire switches). Each node has 32 GB of memory and is connected to a four-disk RAID5 local disk array which can achieve nearly 1 TB/s bandwidth and 1.3 PB HP SFS (Lustre) shared filesystem. This machine is operated by Pacific Northwest National Laboratory's Molecular Science Computing Facility.

5.2. Blue Gene/P

The implementation of ARMCI on Blue Gene/P is substantially different from other platforms. In particular, the data server is not used because it is neither necessary due to the existence of active-message functionality within DCMF [30], nor optimal due to the limited memory bandwidth within the node. True passive progress is achieved either with a communication helper thread continuously polling for incoming active-messages or by operating system interrupts, which are extremely lightweight in the BGP compute node kernel (CNK) relative to Linux. The Argonne Leadership Computing Facility operates Intrepid and Surveyor, which are 40- and 1-rack Blue Gene/P systems, respectively. The specifications for the Blue Gene/P architecture are described in Ref. [25].

6. NWCHEM WORKLOAD CHARACTERIZATION

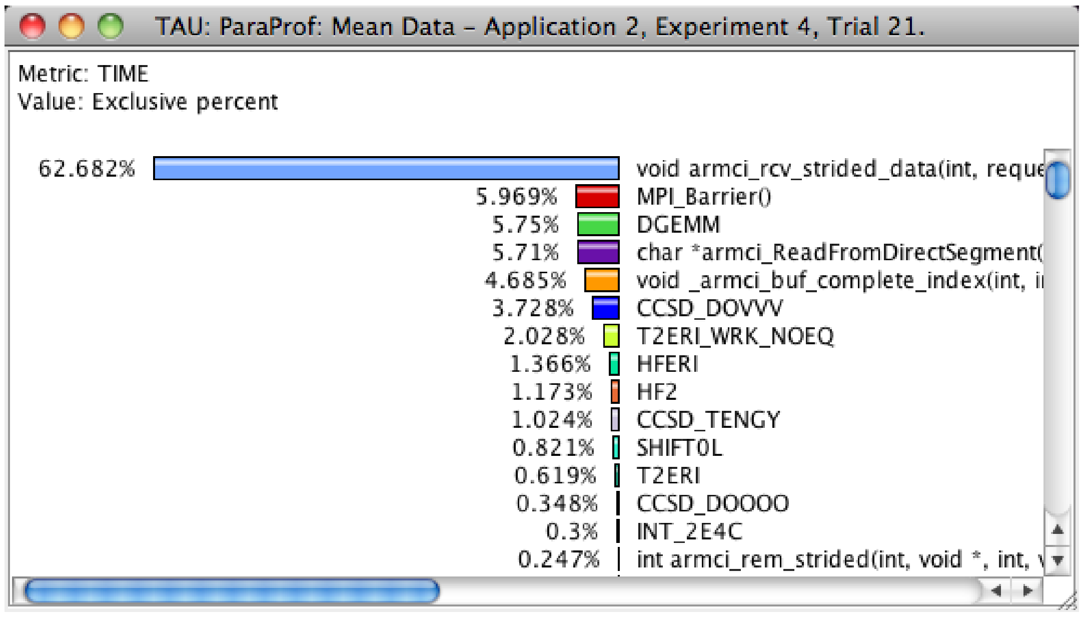
As described in Section 4, we chose automatic instrumentation of the NWChem source code, ARMCI, and MPI layers using TAU. Support for profiling of Pthreads was also critical since both the IB and BG/P implementations of ARMCI use a Pthread (optional on BG/P) to enable asynchronous progress.

An important goal in analyzing one-sided communication in NWChem was to understand the interplay between the data-server and compute processes as a function of scale. However, as the job is strong-scaled to larger numbers of nodes, not only does the computational work per node decrease, but the fragmentation of data across the system leads to an increase in the total number of messages. Our analysis investigated this in detail by varying the number of nodes, cores-per-node, and whether or not memory buffers were "pinned", that is, registered with the NIC and ineligible for paging. On BG/P, paging is disabled in the kernel and memory-registration of the entire address space is trivial, hence there is no comparison to be made with respect to buffer pinning.

There is an important trade-off between using all available processing power for numerical computation and dedicating some fraction of the cores to communication. Understanding these trade-offs as a function of scale is critical for adapting software for new platforms which may have widely varying capability for hardware offloading of message-processing, such as support for contiguous and/or non-contiguous RDMA operations. As these more complex interconnects may require more power, it will be even more important for designing future systems to understand what interconnect features are absolutely necessary.

6.1. Fusion Results

The first times that NWChem was run on the Fusion system, the default settings were used, meaning that NWChem buffers did not use pinned memory. It was natural to use all available cores for computation, so eight processes per node were utilized. Figure 2 was the first indication of a significant performance defect when running NWChem using these naïve settings. Clearly, more than 60% of the total time spent in a strided communication operation is not optimal.


TAU: ParaProf: Mean Data - Application 2, Experiment 4, Trial 21.

```
Metric: TIME
Value: Exclusive percent

62.682%  void armci_rcv_strided_data(int, reque
 5.969%  MPI_Barrier()
  5.75%  DGEMM
  5.71%  char *armci_ReadFromDirectSegment(
 4.685%  void _armci_buf_complete_index(int, i
 3.728%  CCSD_DOVVV
 2.028%  T2ERI_WRK_NOEQ
 1.366%  HFERI
 1.173%  HF2
 1.024%  CCSD_TENGY
 0.821%  SHIFTOL
 0.619%  T2ERI
 0.348%  CCSD_DOOOO
   0.3%  INT_2E4C
 0.247%  int armci_rem_strided(int, void *, int, v
```

Figure 2. Mean exclusive time spent in different NWChem routines for the w4 testcase.

In response to the subroutine timing data, we used PerfExplorer to conduct a similar analysis across 24, 32, 48, 64, 96 and 128 nodes using both 7 and 8 compute cores per node, again with pinning disabled (Figures 3 and 4). We see clearly that the total time associated with functions doing computation — e.g. `DGEMM`, `HF2` (atomic integrals) — decrease with increasing node count, while functions associated with ARMCI communication increase dramatically, not just as a percentage of the overall time, which is natural due to decreasing computational work per process, but in total time, which is pathological for performance. There is a small difference between 7 and 8 compute cores per node for 48 and 64 nodes, but this is likely an artifact of context-switching and interrupt-handling, and does not affect the composite analysis showing that execution time does not scale when buffer pinning is disabled.

After having determined that the performance of ARMCI was nominally affected by having a dedicated core for the data server, we considered the role of using pinned buffers in NWChem. Again we ran tests on 24, 32, 48, 64, 96 and 128 nodes using 7 and 8 compute processes per node (see Figures 5 and 6). The change from the previous results are dramatic: we observe excellent total execution time and an absence of the anti-scaling of ARMCI operations. The time spent in

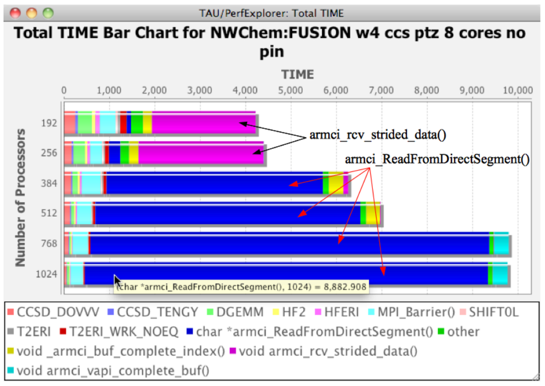


Figure 3. Stacked bar chart in PerfExplorer shows the growth of time spent in communication operations with increasing core counts. Note that the time axis is not the same in all figures.

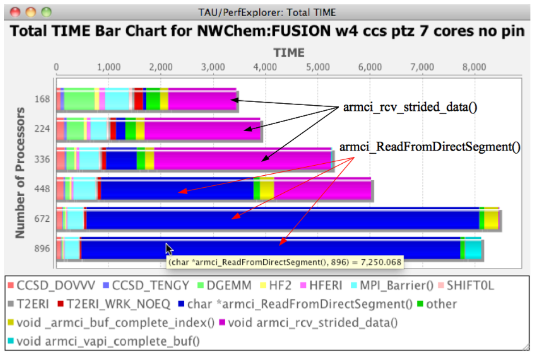


Figure 4. PerfExplorer shows the relative performance of each thread when we reduce the number of application threads to 7 per node without pinning Infiniband memory. Note that the time axis is not the same in all figures.

armci_vapi_complete_buf grows slightly with node count, but this is expected in a strong-scaling scenario since it is increasingly difficult to overlap computation with communication when the amount of computation decreases.

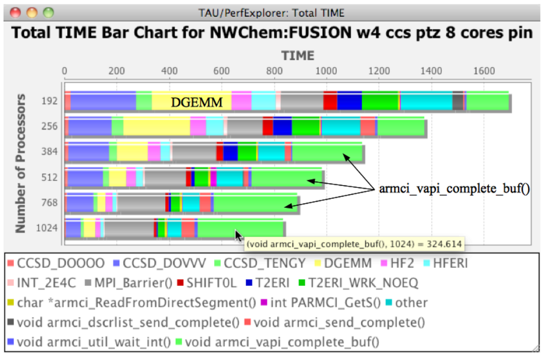


Figure 5. TAU's PerfExplorer shows the stacked bar charts when Infiniband memory is pinned by ARMCI for communication operations and all 8 cores are used. Note that the time axis is not the same in all figures.

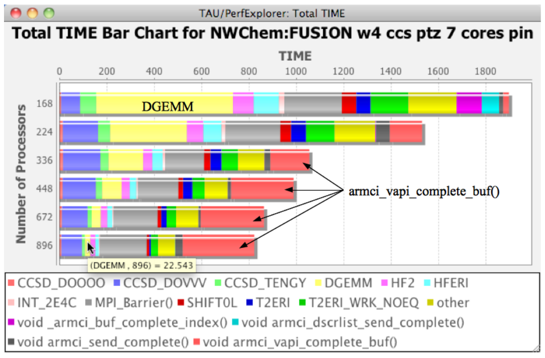


Figure 6. The time spent in different events when Infiniband memory is pinned and only 7 cores are used in each node. Note that the time axis is not the same in all figures.

This dramatic shift in performance confirms our hypothesis that pinning the Infiniband memory can significantly impact both the execution time and scaling characteristics of codes that use one-sided communication operations.

Figure 7 summarizes the distribution of time in various operations in NWChem as a function of scale for the best combination of execution parameters (a dedicated communication core and pinned buffers). As one would expect, as the time spent on computation decreases directly proportional to the number of processors used, the percentage of time spent in communication grows. One also sees a slight growth in the time spent in MPI_Barrier, which is, of course, due to load-imbalance (implicit time) rather than the cost of this collective operation itself (explicit time).

Figure 8 shows the relative speedup of all four cases. The problems with using the default communication substrate are clearly apparent. By pinning bufferes used for communication and giving the communication thread its own core we see a marked improvement in performance. While the scaling is not perfect, pinning is the difference between scaling and anti-scaling (wall time increases as more processors are used) and the scaling is noticeably better using a dedicated core for communication.

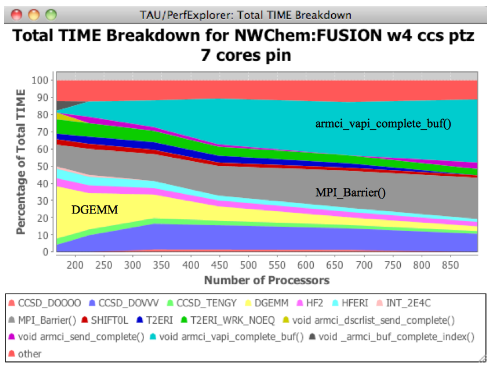


Figure 7. PerfExplorer's runtime breakdown chart shows the contribution and rate of growth of each event for the case with 7 cores and pinned Infiniband memory

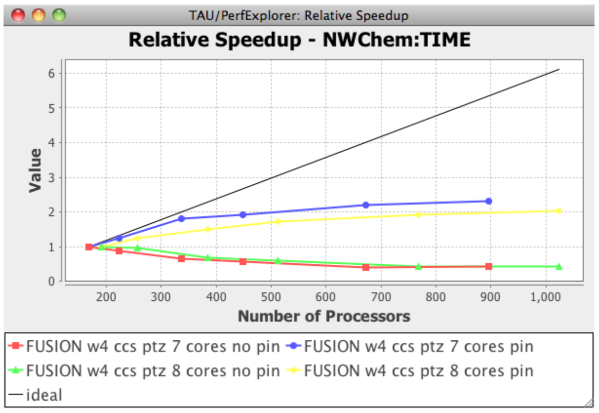


Figure 8. TAU's PerfExplorer relative speedup chart compares each case with an ideal speedup

6.2. *Chinook Results*

To test the generality of our experiments on Fusion, we performed similar tests on Chinook, which is superficially similar due to the use of Mellanox Infiniband and dual-socket quad-core nodes with more than 30 GB of memory. A key difference is that the newer Intel nehalem process has approximately twice the memory bandwidth per core and per node (as measure by the STREAM benchmark [36, 35]), and the system software is different (Chinook runs HP's HPC-oriented Linux and communication stack).

Table I reports a high-level performance analysis of NWChem based upon module timings. This information can be obtained by TAU or directly from the NWChem output file, albeit with more effort. Because efficient ARMCI communication requires both the data-server and interrupt handling by the operating system, we tested the use of 6, 7 and 8 compute processes per node on 32, 48, 64, 96 and 128 nodes with and without pinning.

Table I. Performance data on Chinook, with 48- and 96-node data omitted.

nodes	cores	pin	Total Wall	Tints	Triples
32	6	on	5538.9	684.7	1073.4
32	6	off	5052.5	665.6	1073.2
32	7	on	6302.5	793.1	929.3
32	7	off	6410.6	897.0	928.3
32	8	on	7857.9	2496.6	801.1
32	8	off	11813.5	6044.1	814.0
64	6	on	3364.3	482.2	545.0
64	6	off	3585.9	480.0	543.6
64	7	on	4290.6	615.6	473.7
64	7	off	4223.7	569.7	474.1
64	8	on	4737.0	1552.5	413.5
64	8	off	4832.1	1614.5	412.7
128	6	on	2433.3	328.8	291.3
128	6	off	2456.0	350.3	291.5
128	7	on	2939.4	382.9	258.1
128	7	off	2942.3	436.7	259.4
128	8	on	3948.5	1157.3	233.3
128	8	off	3664.5	1134.3	237.0

In contrast to the data for Fusion, which showed a much stronger dependence on the use of pinning than how many compute cores were used, Chinook is much less sensitive to pinning. On the other hand, it is much more sensitive to how cores are allocated. In particular, using all 8 cores per node for computation increases the time for certain procedures markedly. With 32 nodes and 8 cores/node computing, the time spent generating atomic integrals required for the triples calculation (`Tints`) is 3 and 9 times greater than when only 6 cores per node are used for the pinned and non-pinned cases, respectively. Similarly, on 128 nodes, using 8 cores per node for computation increases the wall time by approximately 3 times the 6- and 7-compute core per node cases. The `DGEMM`-rich `Triples` procedure benefits from the use of more cores per node due to high computation-to-communication ratio. Figure 9 shows the relative efficiency and Figure 10 compares the total wallclock time for all cases comparing Chinook with Fusion. Obviously one way to maximum performance of `Tints` and `Triples` at the same time would be use to use threads within BLAS. However, not all matrices passed to BLAS by NWChem are sufficiently large to warrant the use of multiple threads and one would have to reduce the number of processes per node to four to use multiple threads in BLAS without oversubscription. Threading was recently introduced to the non-BLAS portions of the `Triples` kernel [38] but we have not yet analyzed that implementation.

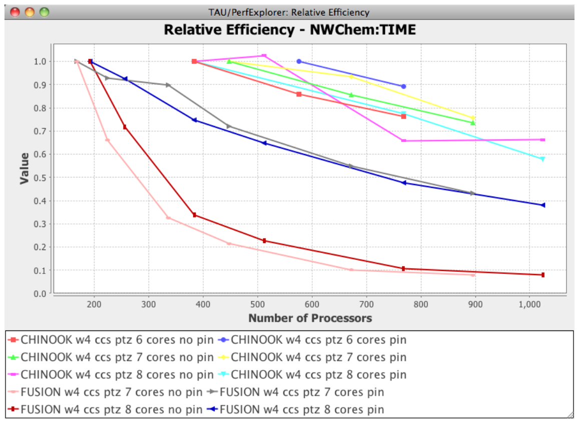


Figure 9. Relative efficiency comparison between Chinook and Fusion.

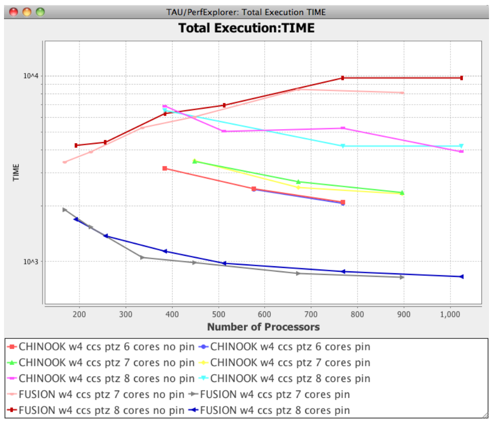


Figure 10. Total time comparison between Chinook and Fusion.

6.3. *Blue Gene/P Results*

We compare to the Blue Gene/P (BGP) system to provide an entirely different context for our scaling experiments. BGP not only has very different hardware — low-memory, slow-clock-rate processors, no local disk and an extremely low-latency but modest bandwidth network — but it uses a lightweight operating system which does not permit Linux-style oversubscription, nor does it support SysV shared-memory. As such, the implementation of ARMCI is quite different and relies heavily upon active-message capability within DCMF [30], which is enabled by lightweight operating system interrupts. Based upon preliminary investigations of ARMCI performance on BGP, a primitive communication helper thread (CHT) was added to the ARMCI implementation. The performance results reported herein demonstrate the utility of this approach to asynchronous progress relative to interrupts. Both the CHT and interrupt-mode provide a means to achieve passive-target progress in one-sided communication, although only the CHT requires a dedicated core. We compare these two context for running NWChem across a range of node counts (64, 128, 256 and 512) which is approximately comparable to the range used on Chinook and Fusion. Our NWChem calculations on BGP executed in SMP and DUAL mode – 1 and 2 processes per node respectively. VN mode does not provide enough memory per process to run CCSD(T), nor does it permit a comparison of interrupt and CHT mode due to the inability to oversubscribe in the latter case.

Figures 11 (CHT mode) and 12 (interrupt mode) summarize the total time spent in various procedures for the same test as before. Unlike the IB systems, ARMCI calls are barely noticeable, whereas a normally negligible BLAS2 operations, `DAXPY`, shows up in an unusually significant way. This result is not surprising due to the low clock-frequency of the BGP processor and relatively low bandwidth from L2 cache. As we see from the total times, the scaling on BGP is excellent since the absence of pathological communication behavior, as was observed for IB in some contexts, allows for straightforward halving of total execution time with a doubling of node count.

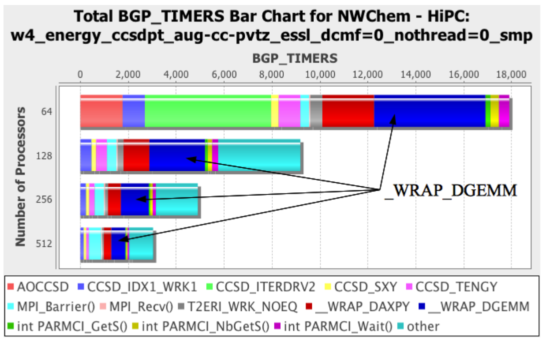


Figure 11. TAU's PerfExplorer shows relative efficiency plot for the strong scaling experiment. Note that the time axis is not the same in all figures.

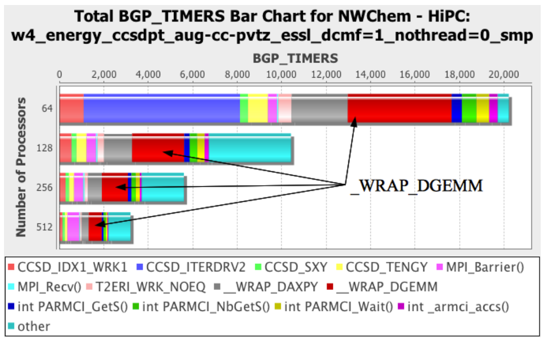


Figure 12. TAU's PerfExplorer shows relative efficiency plot for the strong scaling experiment. Note that the time axis is not the same in all figures.

The application timing perspective on BGP performance is presented in Table II. We see that in SMP mode, both interrupts and the CHT provide similar performance and scaling, although the CHT is slightly better. Because NWChem kernels lack of multithreading support, a free core is always available to handle interrupts (using multiple threads in ESSL was not found to improve performance significantly). While there is still a free core for interrupt-handling in DUAL mode, the performance advantage of using a CHT is more significant. This is clear for the case where 256 cores are used for computation: 128 nodes in DUAL mode performs almost identical to 256 nodes in SMP mode with the CHT, whereas interrupts are significantly slower with 128 nodes running DUAL mode compared to 256 nodes running SMP mode. One downside of using a CHT is that it competes with the compute process for locks on the communication fabric, DCMF, which means that MPI collectives can be slower when interrupts are used. However, the function-level analysis with TAU (not shown) reveals this to be a relatively minor issue, at least at the scales considered in this paper.

7. DISCUSSION AND FUTURE PLANS

Despite the importance of NWChem to HPC workloads, particularly on DOE supercomputers, it was previously impossible to characterize the communication behavior of NWChem because it uses ARMCI, rather than MPI, for communication. The analysis techniques demonstrated in this paper are already in use to understand how to optimize NWChem and the underlying Global Arrays runtime for larger systems. In addition, understanding NWChem's behavior on current supercomputer architectures has motivated design decisions involving both software and hardware of future systems.

In the future, we intend to extend this work in a number of ways. First, we will employ TAU's tracing capability to create a more detailed understanding of communication patterns within NWChem. Second, a detailed profiling interface for BLAS will be built in order to gather input parameters in addition to timings. Parameter histograms from BLAS calls will help us understand how NWChem can be multithreaded most effectively. For example, large MMM calls can utilize many threads at a time, whereas small MMM calls cannot. If there are many more small MMM calls than large ones, it will be necessary to develop a multithreaded runtime that can execute many such calls at one time within the same process, which motivates the development of a thread-safe implementation of Global Arrays (already in-progress). Another approach is to develop a different

Table II. Performance data on Blue Gene/P. All calculations were run using the ZYXT mapping and either SMP or DUAL mode. Both interrupts (i=1) or a communication helper thread (i=0) were used for passive-target progress. Jobs for DUAL mode using interrupts for 256 and 512 nodes were not run since it was obvious from the 128 node case that the performance was going to be much worse than with a communication helper thread.

mode	nodes	i	Tints	Triples	Total
SMP	64	0	1495.46	5105.27	17895.10
	64	1	1706.85	5099.81	20215.40
	128	0	748.55	2580.66	9177.00
	128	1	859.35	2576.60	10413.30
	256	0	376.19	1294.28	4896.80
	256	1	427.50	1292.47	5594.50
	512	0	190.80	647.01	3030.80
	512	1	209.33	646.03	3191.70
	nodes	i	Tints	Triples	Total
DUAL	128	0	379.62	1296.32	5293.10
	128	1	1029.78	1314.19	7444.30
	256	0	193.73	650.28	3338.00
	512	0	100.73	353.97	2202.20

implementation of MMM that does not use vendor BLAS libraries and is written specifically for multithreaded MMM calls involving small matrices. It is possible to optimize such operations in special cases, such as when all input arrays are already in cache.

The last area in which we hope to apply this work is to use the results of TAU traces and BLAS histograms to build a detailed performance model of NWChem's coupled-cluster capability. As scientific applications are often far too complex to be of use by computer scientists to develop new compiler and runtime technology, a detailed performance model and a representative skeleton application which mimics its behavior will allow in-depth study of quantum chemistry algorithms by non-chemists.

Another application of detailed performance models is to understand the anticipated performance of NWChem on hardware architectures that do not yet exist. In the short term this might involve understanding what parts of NWChem are readily amenable to acceleration with graphics processor units (GPUs) and which modules require redesign for such systems.

8. CONCLUSIONS

In this paper, we investigated the performance of an important module in the NWChem software package using TAU. Special consideration was paid to the use of one-sided communication operations and the optimal usage thereof on three platforms. The scaling of NWChem was analyzed for a variety of runtime parameters related to ARMCI communication system. It was demonstrated that seemingly similar platforms — Chinook and Fusion — showed vastly different behavior with respect to the dedication of cores to communication operations and to the use of pinned buffers. On the vastly different Blue Gene/P architecture, we compared interrupt to thread-assisted ARMCI communication and found that the communication helper thread is a more efficient means to achieving passive-target progress, although at the expense of losing a core per process. However, having shown that even on platforms which permit oversubscription that leaving one or more cores free to handle communication operations is optimal for many procedures in the NWChem CCSD(T) code, the use of a dedicated communication core on Blue Gene/P is further justified.

The role of a automatic instrumentation system such as TAU was invaluable for generating and analyzing performance data for a complex code such as NWChem. The NWChem code base is millions of lines and the relevant source even for a single method such as CCSD(T) is approximately 200K lines in addition to the tens of thousands of lines of code active in GA and ARMCI for a given interconnect. Without automated source instrumentation and profiling hooks to both MPI and ARMCI, it would not have been possible to reliably identify the performance issues described in this paper. As should be clear from the results, profiling one-sided communication can be more complex since, while the remote target is passive from a programmer perspective, passive-target progress requires significant resources on every node, especially since remote accumulate requires floating-point computation on top the memory operations required for packing and unpacking of non-contiguous messages. More rudimentary profiling techniques (e.g. `gprof`) are not useful for analyzing the behavior of an asynchronous agent such as the ARMCI data server. Thus, with increased interest in one-sided programming models in both GA and PGAS languages, advanced profiling tools such as TAU must become even more widely used.

9. RELATED WORK

9.1. Workload Characterization

Characterizing computational workloads is the first step in any quantitative approach to evaluating alternative implementations and ultimately improving performance. This includes system design, optimization of development tools and libraries, and prediction of performance and utilization on current and future systems. Each class of applications is characterized by kernel and application benchmarks that best capture the computation, memory access, and communication characteristics of that class. For example, NASA Advanced Supercomputing (NAS) parallel benchmarks [4] are representative of many codes of importance to NASA and have been used in numerous scenarios including tuning libraries and languages, and job scheduling on supercomputers. The DOE Advanced Scientific Computing (ASC) benchmarks [3] are of interest to U.S. National Nuclear Security Administration for large-scale modeling of nuclear materials. NAS and ASC benchmarks are representative of different classes of high-end applications equally important in terms of their utilization of supercomputer time. HPC applications across the spectrum of science domains share a need for performance benchmarking and robust workload characterization methods.

9.2. Performance Evaluation Tools

While the choice of TAU for the NWChem performance characterization provided us with robust instrumentation, measurement, and analysis capabilities, the methodology presented would also be supported by other performance tools for large-scale parallel systems. One important point to note is that the PARMCI API is tool independent and can be leveraged by other measurement support since it is part of the ARMCI distribution. For instance, PARMCI is being applied with Scalasca [16], especially for generating traces of one-sided communication that can then be visualized with tools such as Vampir [7].

The use of sampling-based measurement methods, such as supported by PerfSuite [29] and HPCToolkit [37] could also be helpful in exposing different thread performance behaviors and finer-grained multicore resource contention issues. Some support has been built into TAU for event-based sampling [39] and we will apply this technique in upcoming performance tests to investigate resource limiting performance factors.

9.3. One-Sided Programming Models

One-sided communication models communicate through *remote memory access* (RMA), a mechanism by which a process accesses the data in another process' memory without explicit synchronization with the remote process. This model extends the philosophy of shared memory programming models, which have been known to simplify programming as compared to message passing. The programming models designed using one-sided communication as the fundamental unit are said to be variants of global address space programming models. Partitioned global address space (PGAS) models enable differentiation between local and remote memory, enable incremental optimizations. UPC [54], Titanium [58], X10 [9], and Chapel [8] are examples of PGAS languages.

NWChem relies on GA as the underlying PGAS programming model. It provides a global view of a physically distributed array, supporting one-sided access to arbitrary patches of data. Being developed as a library, it is fully interoperable with MPI allowing a programmer to use MPI and Global Arrays API at the same time. ARMCI is the communication substrate providing the one-sided communication support for GA. It is a portable high-performance one-sided communication library that supports a rich set of remote memory access primitives. In order to analyze the communication characteristics of NWChem modules, we have developed support to generate TAU events when the ARMCI API is invoked.

9.4. Computational Chemistry Applications

Quantum chemistry codes have long targeted high-performance computing (HPC) platforms due to their substantial requirements for floating-point computation, storage, execution time and job number. Early HPC-oriented codes targeted Cray vector machines by exploiting BLAS calls in numerical kernels and pipelining array access for optimal memory and I/O utilization. Many codes originally written for vector machines transitioned to superscalar architectures without complete rewriting due to the continuity in performance provided by BLAS and the ever-increasing power of a single processor due to Moore's law.

Between the transition from vector to superscalar processors, relatively little effort was devoted to distributed-memory parallelization of quantum chemistry codes. NWChem was specifically designed for massively-parallel supercomputers and exploited a variety of new communication protocols — including TCGMSG [22] and GA — which were developed specifically in response to the challenge of parallelization quantum chemistry algorithms across distributed memory [12]. Other early efforts to develop parallel quantum chemistry codes include:

- COLUMBUS [32], which used GA,
- GAMESS [56], which used TCGMSG,
- Gaussian [53], which used Linda.

A more complete review can be found in Ref. [23].

Very recently, there has been significant effort devoted to parallel implementations of CCSD(T), including the two different implementations in NWChem [1, 27] and those in the Molpro [55], GAMESS [5], ACESIII [33], and CFOUR [21] packages. Given the ubiquity of parallelism in all modern computer hardware and the increasing interest in parallel quantum chemistry software development, particularly for the CCSD(T) method, our careful investigation of one implementation is timely.

ACKNOWLEDGMENTS

This work was funded by the U.S. Department of Energy's Pacific Northwest National Laboratory under the Extreme Scale Computing Initiative. The authors would like to thank PNNL EMSL Molecular Sciences Computing Facility (MSCF) for access to their computing resources.

This research used resources of the Argonne Leadership Computing Facility and the Laboratory Computing Resource Center at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. JRH acknowledges an Argonne Director's Postdoctoral Fellowship.

The research at the University of Oregon was supported by grants from the U.S. Department of Energy, Office of Science, under contracts DE-FG02-07ER25826, DE-SC0001777, and DE-FG02-09ER25873.

REFERENCES

1. E. Aprà, R. J. Harrison, W. A. Shelton, V. Tipparaju, and A. Vazquez-Mayagoitia. Computational chemistry at the petascale: Are we there yet? *Journal of Physics: Conference Series*, 180:012027 (6pp), 2009.

2. E. Aprà, A. P. Rendell, R. J. Harrison, V. Tipparaju, W. A. de Jong, and S. S. Xantheas. Liquid water: obtaining the right answer for the right reasons. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–7, New York, NY, USA, 2009. ACM.
3. ASC sequoia benchmarks. https://asc.llnl.gov/sequoia/benchmarks/.
4. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks—summary and preliminary results. In *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 158–165, New York, NY, USA, 1991. ACM.
5. J. L. Bentz, R. M. Olson, M. S. Gordon, M. W. Schmidt, and R. A. Kendall. Coupled cluster algorithms for networks of shared memory parallel processors. *Computer Physics Communications*, 176(9–10):589–600, 2007.
6. S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A Portable Programming Interface for Performance Evaluation on Modern Processors. *International Journal of High Performance Computing Applications*, 14(3):189–204, Fall 2000.
7. H. Brunst, D. Kranzlmüller, and W. E. Nagel. Tools for Scalable Parallel Program Analysis - Vampir NG and DeWiz. *Distributed and Parallel Systems, Cluster and Grid Computing*, 777, 2004.
8. B. Chamberlain, D. Callahan, and H. Zima. Parallel programmability and the chapel language. *Intl. J. High Performance Computing Applications (IJHPCA)*, 21(3):291–312, 2007.
9. P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *Intl. Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 519–538. ACM SIGPLAN, 2005.
10. J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 53:1–53:11, New York, NY, USA, 2009. ACM.
11. K. M. Dixit. The SPEC benchmarks. *Parallel Computing*, 17(10-11):1195 – 1209, 1991. Benchmarking of high performance supercomputers.
12. T. H. Dunning Jr., R. J. Harrison, and J. A. Nichols. NWChem: Development of a modern quantum chemistry program. *CTWatch Quarterly*, 2(2), May 2006.
13. E. J. Bylaska et al. NWChem, a computational chemistry package for parallel computers, version 5.1.1, 2009.
14. M. P. I. Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, University of Tennessee, Knoxville, TN, USA, May 1994.
15. M. J. Frisch, B. G. Johnson, P. M. W. Gill, D. J. Fox, and R. H. Nobes. An improved criterion for evaluating the efficiency of two-electron integral algorithms. *Chemical Physics Letters*, 206(1-4):225–228, 1993.
16. M. Geimer, F. Wolf, B. J. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The SCALASCA performance toolset architecture. In *Proc. of the International Workshop on Scalable Tools for High-End Computing (STHEC)*, pages 51–65, Kos, Greece, June 2008.
17. P. M. Gill. Molecular integrals over gaussian basis functions. In J. R. Sabin and M. C. Zerner, editors, *Advances in Quantum Chemistry*, volume 25, pages 141–205. Academic Press, 1994.
18. K. Goto and R. A. v. d. Geijn. Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34:12:1–12:25, May 2008.
19. J. R. Hammond. *Coupled-cluster response theory: parallel algorithms and novel applications*. PhD thesis, The University of Chicago, Chicago, IL, USA, June 2009.
20. J. R. Hammond. Scalability of quantum chemistry codes on Blue Gene/P and challenges for sustained petascale performance. Poster at Supercomputing, 2009.
21. M. E. Harding, T. Metzroth, J. Gauss, and A. A. Auer. Parallel calculation of CCSD and CCSD(T) analytic first and second derivatives. *Journal of Chemical Theory and Computation*, 4(1):64–74, 2008.
22. R. J. Harrison. Moving beyond message passing. experiments with a distributed-data model. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 84(4):363–375, 1993.
23. R. J. Harrison and R. Shepard. Ab initio molecular electronic structure on parallel computers. *Annual Review of Physical Chemistry*, 45(1):623–658, 1994.
24. T. Helgaker, P. Jørgensen, and J. Olsen. *Molecular Electronic-Structure Theory*. Wiley, Chichester, 1st edition, 2000.
25. IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1):199–220, January 2008.
26. R. Kobayashi and A. P. Rendell. A direct coupled cluster algorithm for massively parallel computers. *Chemical Physics Letters*, 265(1-2):1 – 11, 1997.
27. K. Kowalski, J. R. Hammond, W. A. de Jong, P.-D. Fan, M. Valiev, D. Wang, and N. Govind. Coupled cluster calculations for large molecular and extended systems. In J. R. Reimers, editor, *Computational Methods for Large Systems: Electronic Structure Approaches for Biotechnology and Nanotechnology*. Wiley, March 2011.
28. S. A. Kucharski and R. J. Bartlett. Recursive intermediate factorization and complete computational linearization of the coupled-cluster single, double, triple, and quadruple excitation equations. *Theoritical chemistry accounts: theory, computation, and modeling*, 80:387–405, 1991. 10.1007/BF01117419.
29. R. Kufrin. Measuring and Improving Application Performance with PerfSuite. *Linux Journal*, 135:62–70, July 2005.
30. S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, M. Blocksome, A. Faraj, J. Parker, J. Ratterman, B. Smith, and C. J. Archer. The deep computing messaging framework: generalized scalable message passing on the Blue Gene/P supercomputer. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 94–103, New York, NY, USA, 2008. ACM.
31. K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, and C. Rasmussen. A tool framework for static and dynamic analysis of object-oriented software with templates. In *Proceedings of SC2000: High Performance Networking and Computing Conference*, Dallas, November 2000.

32. H. Lischka, H. Dachsel, R. Shephard, and R. J. Harrison. Parallel computing in quantum chemistry - message passing and beyond for a general ab initio program system. In *HPCN Europe 1994: Proceedings of the nternational Conference and Exhibition on High-Performance Computing and Networking Volume I*, pages 203–209, London, UK, 1994. Springer-Verlag.
33. V. Lotrich, N. Flocke, M. Ponton, A. D. Yau, A. Perera, E. Deumens, and R. J. Bartlett. Parallel implementation of electronic structure energy, gradient, and hessian calculations. *The Journal of Chemical Physics*, 128(19):194104, 2008.
34. A. Malony and S. Shende. Performance Technology for Complex Parallel and Distributed Systems. *Distributed and parallel systems: from instruction parallelism to cluster computing*, pages 37–46, 2000.
35. J. D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.
36. J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995.
37. J. Mellor-Crummey. Hpctoolkit: Multi-platform tools for profile-based performance analysis. In *5th International Workshop on Automatic Performance Analysis (APART)*, November 2003.
38. V. Morozov and J. R. Hammond. unpublished results, 2010.
39. A. Morris, **A. Malony**, S. Shende, and K. Huck. Design and Implementation of a Hybrid Parallel Performance Measurement System. In *International Conference on Parallel Processing (ICPP 2010)*, Sept. 2010.
40. J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. *Lecture Notes in Computer Science*, 1586, 1999.
41. J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: a portable "shared-memory" programming model for distributed memory computers. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 340–349, New York, NY, USA, 1994. ACM.
42. J. Nieplocha, B. Palmer, M. Krishnan, H. Trease, and E. Aprà. Advances, applications and performance of the Global Arrays shared memory programming toolkit. *International Journal of High Performance Computing Applications*, 20, 2005.
43. R. W. Numrich and J. Reid. Co-array Fortran for parallel programming. *SIGPLAN Fortran Forum*, 17(2):1–31, 1998.
44. K. Parzyszek. *Generalized portable shmem library for high performance computing*. PhD thesis, Iowa State University, Ames, IA, USA, 2003. AAI3105098.
45. K. Raghavachari, G. W. Trucks, J. A. Pople, and M. Head-Gordon. A fifth-order perturbation comparison of electron correlation theories. *Chemical Physics Letters*, 157:479–483, May 1989.
46. S. Shende, A. Malony, and R. Ansell-Bell. Instrumentation and Measurement Strategies for Flexible and Portable Empirical Performance Evaluation. In *Proceedings Tools and Techniques for Performance Evaluation Workshop, PDPTA*, volume 3, pages 1150–1156. CSREA, 2001.
47. S. Shende, A. Malony, J. Cuny, K. Lindlan, P. Beckman, and S. Karmesin. Portable Profiling and Tracing for Parallel Scientific Applications using C++. In *Proceedings 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98)*, pages 134–145, 1998.
48. S. Shende and A. D. Malony. The TAU parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311, Summer 2006.
49. J. P. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford parallel applications for shared-memory. *SIGARCH Comput. Archit. News*, 20:5–44, March 1992.
50. G. L. Steele Jr. Parallel programming and parallel abstractions in Fortress. In *14th Intl. Conf. on Parallel Architecture and Compilation Techniques (PACT)*, page 157, 2005.
51. V. Tipparaju, E. Apra, W. Yu, and J. Vetter. Enabling a highly-scalable global address space model for petascale computing. In *CF '10: Proceedings of the 7th ACM conference on Computing frontiers*, New York, NY, USA, 2010. ACM. TO APPEAR.
52. Transaction processing performance council. http://www.tpc.org/.
53. D. P. Turner, G. W. Trucks, and M. J. Frisch. Ab initio quantum chemistry on a workstation cluster. In *Parallel Computing in Computational Chemistry*, volume 592 of *ACS symposium series*, pages 62–74. American Chemical Society, 1995.
54. UPC Consortium. UPC language specifications, v1.2. Technical Report LBNL-59208, Lawrence Berkeley National Lab, 2005.
55. H.-J. Werner, P. J. Knowles, F. R. Manby, M. Schütz, P. Celani, G. Knizia, T. Korona, R. Lindh, 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, A. Hesselmann, G. Hetzer, T. Hrenar, G. Jansen, 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, T. Shiozaki, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, M. Wang, and A. Wolf. Molpro, version 2010.1, a package of ab initio programs, 2010. see http://www.molpro.net.
56. T. L. Windus, M. W. Schmidt, and M. S. Gordon. Parallel algorithm for integral transformations and GUGA MCSCF. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 89(1):77–88, 1994.
57. A. T. Wong, R. J. Harrison, and A. P. Rendell. Parallel direct four-index transformations. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 93:317–331, 1996. 10.1007/BF01129213.
58. K. A. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A high-performance Java dialect. *Concurrency - Practice and Experience*, 10(11-13):825–836, 1998.