Large-Scale Electronic Structure Calculations of High-Z Metals on the BlueGene/L Platform

Francois Gygi
Department of Applied Science
University of California, Davis
Davis, CA 95616
530-752-4042
fgygi@ucdavis.edu

Erik W. Draeger, Martin Schulz,
Bronis R. de Supinski
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551
{draeger1,schulz6,bronis}@llnl.gov

John A.Gunnels, Vernon Austel,
James C. Sexton
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{gunnels,austel,sextonjc}@us.ibm.com

Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
franzf@ece.cmu.edu

Stefan Kral, Christoph W. Ueberhuber, Juergen Lorenz
Institute of Analysis and Scientific Computing
Vienna University of Technology, Vienna, Austria
skral@mips.complang.tuwien.ac.at, c.ueberhuber@tuwien.ac.at
juergen.lorenz@aurora.anum.tuwien.ac.at

ABSTRACT
First-principles simulations of high-Z metallic systems using the Qbox code on the BlueGene/L supercomputer demonstrate unprecedented performance and scaling for a quantum simulation code. Specifically designed to take advantage of massively-parallel systems like BlueGene/L, Qbox demonstrates excellent parallel efficiency and peak performance. A sustained peak performance of 207.3 TFlop/s was measured on 65,536 nodes, corresponding to 56.5% of the theoretical full machine peak using all 128k CPUs.

Categories and Subject Descriptors
J.2 [**Physical Sciences and Engineering**]:– *Chemistry, Physics*.

0-7695-2700-0/06 $20.00 ©2006 IEEE

General Terms
Algorithms, Measurement, Performance.

Keywords
Electronic structure. First-principles Molecular Dynamics. Ab initio simulations. Parallel computing. BlueGene/L, Qbox.

1. INTRODUCTION
First-Principles Molecular Dynamics (FPMD) is an accurate atomistic simulation approach that is routinely applied to a variety of areas including solid-state physics, chemistry, biochemistry and nanotechnology [1]. It includes a quantum mechanical description of electrons, and a classical description of atomic nuclei. FPMD simulations integrate the Newton equations of motion for all nuclei in order to simulate dynamical properties of physical systems at finite temperature. At each discrete time step of the trajectory, the forces acting on the nuclei are derived from a calculation of the electronic properties of the system.

In this paper, we consider the case of FPMD simulations of heavy (or "high-Z") metals such as molybdenum or tantalum. In particular, we focus on high-accuracy electronic structure calculations needed to evaluate the energy of isolated defects. Such calculations are especially challenging since they require the inclusion of a large number of atoms in a periodic simulation cell. This in turn implies that a large number of valence electrons must be included in the calculation. Furthermore, high-accuracy calculations often require the use of additional tightly bound electrons (known as semi-core electrons) in the simulation.

The electronic structure calculation is the most time-consuming part of an FPMD simulation. It consists in solving the Kohn-Sham (KS) equations [2]. The KS equations are one-particle, non-linear PDEs that approximate the many-particle Schroedinger equation. The solutions of the KS equations describe electrons in the presence of an average effective potential that mimics the complex many-body interactions between electrons. In periodic solids the KS equations take the form

$$H_k \psi_{nk}(r) = \varepsilon_{nk} \psi_{nk}(r)$$

where H_k is the Hamiltonian, solutions $\psi_{nk}(r)$ are Bloch waves, *n* is a band index and *k* is a wave vector (also called *k*-point) confined to the first Brillouin zone of the crystal. The one-particle KS Hamiltonian H_k depends non-linearly on the electronic density $\rho(r)$ which in turn depends on the solutions $\psi_{nk}(r)$ through the relation

$$\rho(r) = \sum_{nk} |\psi_{nk}(r)|^2$$

The KS equations can therefore be solved independently for each value of *k*, which leads to a natural division of the computational work on a parallel computer. However, the dependence of the electronic charge density $\rho(r)$ on the solutions at all values of *k* introduces a coupling between solutions at different *k* values. This coupling appears in the dependence of the effective one-particle potential on the total electronic charge distribution $\rho(r)$ through the electrostatic and exchange-correlation potentials. In the systems considered here (metals in large supercells), the number of *k*-points needed is of the order of four to eight. Solutions of the KS equations are real if $k=0$ and complex otherwise. For this reason electronic structure computations involving multiple *k*-points are more costly than computations performed with a single *k*-point (if $k=0$) since they involve complex arithmetic.

Thus the electronic structure problem can be solved efficiently on a parallel computer if i) a one-particle (single *k*-point) KS problem can be solved efficiently on one fourth to one eighth of the machine, and ii) the solutions for all *k*-points can be combined efficiently to compute the total charge density. We show in this paper that both these conditions can be met and lead to efficient electronic structure calculations on the BlueGene/L platform.

We have used the Qbox code to perform electronic structure calculations of molybdenum on the BlueGene/L (BG/L) computer installed at Lawrence Livermore National Laboratory. Qbox is a C++ implementation of the FPMD method [3]. It uses the MPI message-passing paradigm and solves the KS equations [2] within the pseudopotential, plane wave formalism. The solution of the KS equations has been extensively discussed by various authors [1] and requires the capability to perform three-dimensional Fourier transforms and dense linear algebra efficiently. The implementation of these two operations will be discussed in detail below. Qbox was designed specifically for large parallel platforms, including BlueGene/L. The design of Qbox yields good load balance through an efficient data layout and a careful management of the data flow during the most time consuming operations.

The sample chosen for the present performance study contains 1000 molybdenum atoms, and includes a highly accurate treatment of electron-ion interactions. Norm-conserving semi-local pseudopotentials were used to represent the electron-ion interactions. A total of 64 projectors were used (8 radial quadrature points for *p* and *d* channels) on each atom to represent the semi-local potentials. A plane wave energy cutoff of 112 Ry was used to describe the electronic wave functions. Semi-core *p* electrons were included in the valence shell. Calculations including 1, 4 and 8 *k*-points were performed.

Simulations were performed using up to 65,536 nodes. Performance measurements were carried out by counting floating-point operations using hardware counters. Qbox realizes a 41% parallel efficiency between 2k and 64k CPUs for single *k*-point calculations. When using multiple *k*-points, we show that 56.5% of peak performance, or 207.3 TFlop/s, can be achieved on the full machine (65,536 nodes, 131,072 CPUs).

This kind of simulation is considerably larger that any previously feasible FPMD simulation. Our demonstration that BG/L's large computing power makes such large simulations feasible opens the way to accurate simulations of the properties of metals, including the calculation of melting temperatures, defect energies and defect migration processes, studying the effects of aging on the structural and electronic properties of heavy metals and the properties of materials subjected to extreme conditions[4].

2. KEY ASPECTS OF THE BLUEGENE/L ARCHITECTURE FOR FPMD

BlueGene/L (BG/L) presents several opportunities for efficient implementation of FPMD simulations. Details of the tightly-integrated large-scale system architecture are covered elsewhere [5], including aspects of it that are particularly relevant to Qbox [6]. Overall, LLNL's BG/L platform has 65,536 compute nodes and a total peak performance of 367TFlop/s.We briefly cover its general architectural aspects here, focusing on those related to recent or planned optimizations in Qbox.

Each compute node is built from a single compute node ASIC and a set of memory chips. The compute ASIC features two 32-bit superscalar 700 MHz PowerPC 440 cores, with two copies of the PPC floating point unit associated with each core that function as

a SIMD-like double FPU [7]. Achieving high performance requires the use of an extensive set of parallel instructions for which the double precision operands can come from the register file of either unit and that include a variety of paired multiply-add operations, resulting in a peak of four floating point operations (Flop) per cycle per core. Later in this paper, we discuss the DGEMM, ZGEMM and FFT implementations that allow Qbox to use these instructions and, thus, to achieve a high percentage of peak performance.

BG/L includes five networks; we focus on the 3-D torus, the broadcast/reduction tree and the global interrupt for Qbox optimizations. Integration of the network registers into the compute ASIC not only provides fast inter-processor communication but also direct access to network-related hardware performance monitor data. Due to limitations on deadlock-free communication, the MPI implementation uses the tree networks only for global (full-partition) collective operations. The torus network also includes broadcast support; however, currently only MPI communicators that consist of nodes that exactly form a rectangular and compact sub-prism can use it. Since Qbox MPI communication primarily occurs in library routines that use derived subset communicators, we can only make limited use of the tree network and hence torus performance dominates its communication costs in the absence of special optimizations.

3. COMPUTATIONAL KERNELS

When using the plane-wave representation, an efficient solution of the KS equations depends critically on two computational kernels: dense linear algebra and 3D complex Fourier transforms (FT). In the following section, we describe the optimized implementations of these kernels used in Qbox.

3.1 Linear Algebra

Dense linear algebra is implemented through the ScaLAPACK library [8]. ScaLAPACK performs a wide variety of matrix operations on large, distributed, dense matrices. It places some constraints on the data layout. ScaLAPACK is built upon the BLACS communication library, which itself invokes MPI functions. The performance of ScaLAPACK depends critically on the availability of efficient BLAS subroutines. In particular, the ScaLAPACK matrix multiplication function **`pzgemm`** makes extensive use of the BLAS3 **`zgemm`** kernel. We used a hand-optimized version of the **`zgemm`** kernel that we describe in more detail below.

3.1.1 Optimized ZGEMM library

While the performance of the **`zgemm`** routine is dependent upon taking advantage of the hardware features at each level of the memory hierarchy, missteps at the lower-levels have a greater impact than analogously suboptimal decisions that involve higher levels of memory. Similarly, the design and implementation of the optimized **`zgemm`** routine on BG/L is most easily understood when described from the bottom up.

3.1.2 Mathematical and Memory-based Operations: SIMD Vector Units

The peak computational flop rate of a BG/L processor is based upon the assumption that a SIMD FMA can execute during every cycle. If computationally intensive routines cannot take advantage of SIMD instructions (or FMAs), they will not evince more than 50% of the theoretical peak rate of the processor. Fortunately, general matrix-multiplication (C op= A*B) is dominated by FMAs and BG/L's relatively rich instruction set allows one to utilize the SIMD FMA instructions for all of the computations involved in **`zgemm`**. The only prerequisite to taking advantage of these instructions is to load the registers utilized for computations with useful data (i.e. not pad them or throw away half of their result). Because the input data, complex double-precision values in this case, is assumed to be aligned on 16-byte boundaries, the loads and stores of the C matrix are strictly SIMD. If this assumption regarding alignment were not made, the load-primary and load-secondary instructions would allow one to load the registers in the prescribed pattern. Further, since the number of computations involved in **`zgemm`** is an order of magnitude greater than the number of data moves (loads and stores), data reformatting, again using only SIMD instructions, can be employed on the A and B matrices. Data reformatting is standard practice in the area as it makes the matrices more cache-friendly, not simply appropriately aligned.

3.1.3 The Computational Kernel: Register-Based View

Traditionally, the matrix-multiplication computational core, or "kernel routine," is carefully written so as to respect intricacies of the architecture of the machine under consideration. Typically, and on BG/L, the most important considerations are: 1) the number of architected registers, 2) the latency of the levels of the memory hierarchy that are being targeted and, somewhat less importantly, 3) the bandwidth between the register file and the level of the memory hierarchy being targeted.

BG/L's cores each have 32 architected SIMD (length 2) floating-point registers. We used these registers to target a 4x4xK matrix multiplication kernel as our main computational workhorse. Our register blocking uses all 32 SIMD registers: eight for A operands, eight for B operands, and 16 for elements of the C matrix. The computation is composed as two rank-1 updates of the C-registers, yielding, simplistically, a 32-cycle latency between outer products.

3.1.4 L1 Cache Considerations

BG/L's L1 caches are 16-way, 64-set associative and use a round-robin replacement policy [5]. Because of the excellent latency and bandwidth characteristics of its L3/L2 cache, we considered the L1 cache optimizations secondary in the construction of the **`zgemm`** routine; we do not cover them in this paper due to space constraints. It is noted, however, that it is important to block correctly for the L1 cache in order to approach optimal performance for small matrix multiplications.

3.1.5 The L2 Cache and Pre-fetching

BG/L's L2 cache is considerably smaller than the L1 cache (2KB vs. 32KB). The L2 cache acts as a prefetch buffer for data that is streaming from higher levels of memory to the L1 cache. For sequential data accesses, this prefetch mechanism yields a latency that is less than that needed by our register kernel. In order to use this prefetch buffer effectively, algorithms should not use more streams than it can handle optimally. Since it can efficiently handle seven streams in normal mode, we can safely use one

stream for the reformatted A matrix and one stream for the reformatted B matrix.

3.1.6 L3 Interface

The theoretical peak bandwidth from the L3 cache is 5.33 bytes/cycle, which equates to fetching a quadword every 3 cycles (or an L1 cache line every 6 cycles). In our SIMD 4x4x2 register kernel, the inner loop of the code (the part that is neither loading nor storing C) requires exactly one SIMD (quadword) load every four cycles. Thus, it is not surprising that the inner loop of this routine executes at a rate between 95% and 100% of the peak rate of the machine once the data is in L3 and the L2 prefetch mechanism is engaged.

3.1.7 DDR Bandwidth

Since we have blocked the computation to run out of L3, BG/L's DDR bandwidth and latency might seem unimportant. However, they do impact the performance of matrix multiplication, especially for relatively small matrices. BG/L's DDR bandwidth is, at approximately 4 bytes/cycle, comparable to that of its L3 cache and is of great value in achieving high performance for this routine in some cases.

While the **`zgemm`** routine is blocked to take advantage of the L3 cache, a preliminary step copies and reformats the data in the A and B matrices. This step, typically, copies data from DDR to DDR or from DDR to L3. Although this is a negligible start-up cost with large matrices, this overhead may be a sizeable fraction of compute time with small matrices. Further, computation occasionally requires bringing data from DDR and keeping it in the L3 (or L1 in the case of small matrices) cache even for large matrices.

3.2 Fourier Transforms

Qbox takes advantage of the fact that many 3D FTs must be performed simultaneously (one for each electronic state), which eases the scalability requirements on individual 3D FT calculations. For the large systems of interest to this paper scaling of individual FT's beyond 512 tasks is unnecessary, since a sufficient number of transforms can occur simultaneously to utilize the entire machine fully. A custom parallel implementation of 3D FT was developed and optimized for BG/L and shows excellent scaling properties on up to 512 tasks.

3.2.1 FFTW-GEL for BlueGene/L

Qbox calls one-dimensional single-processor FFT kernel routines within its computation. Among other libraries, it can use the portable open-source library FFTW 2.1.5 [9]. FFTW-GEL for BG/L [10] is an FFTW 2.1.5 replacement for BG/L based on the SIMD FFTW replacement provided by FFTW-GEL [11].

Several BG/L specific optimizations were required to achieve good floating-point performance. FFTW-GEL for BG/L achieves good utilization of the two-way vector instructions for the double FPU by replacing the original scalar FFTW codelets with explicitly vectorized double FPU codelets. For BG/L, these vector codelets are generated using intrinsic functions and the C99 complex data type provided by the IBM XL C compiler for BG/L. Additionally, the Vienna MAP vectorizer [12] two-way vectorizes large computational basic blocks by a depth-first search with chronological backtracking to produce explicitly vectorized FFTW codelets with solely two-way vector instructions and a minimum of data reorganization instructions. MAP's vectorization rules that describe the variable and operation pairing encode machine characteristics such as the double FPU's special fused multiply-add instructions. SIMD instructions provide a large performance increase in FFTW-GEL (near two-fold speedup) when measured on a hot L1 cache (e.g. by transforming the same data multiple times). The increase that we observe in Qbox is smaller, since the data to be transformed far exceeds the size of the L1 cache, and memory bandwidth limits performance. A speedup of 20-25% was measured when comparing the FFTW-GEL library with the conventional FFTW 2.1.5 implementation in that case

4. NODE MAPPING STRATEGIES

Unlike many applications for which a simple 3D domain decomposition naturally maps to a 3D torus architecture, the KS equations do no exhibit any obvious way to map parts of the calculation to a torus. For this reason, we have explored various node mappings in order to optimize performance. This optimization must be carried out for each partition size, since the shape of a partition depends on its size. For example, a 4k-node partition consists of an 8x16x32 block of nodes, whereas a 16k partition is a 16x32x32 block. As a consequence, the optimal map for one partition size can differ substantially from the optimal map for another partition. This process is facilitated by the capability to specify a node mapping at run time.

The Qbox data layout distributes the degrees of freedom describing electronic wave functions on a two-dimensional process grid similar to the BLACS process grids [8]. Collective communications over MPI communicators that correspond to rows and columns of the process grid form the bulk of Qbox communication costs. Table 1, which provides Qbox communication timings, indicates that MPI_Bcasts and MPI_Allreduces over these communicators dominate those costs.

Experience shows that Qbox communication costs vary significantly with the mapping of MPI tasks to BG/L's torus topology. Good connectivity between communicators requires more complex node maps than the default XYZ, YZX or ZXY orderings. Figure 1 shows examples of four node mappings used on the full machine 64k-node partition. For a 64k-node calculation of 1000 molybdenum atoms with a single k-point, i.e. a single set of KS equations, we found that the mapping dramatically affected performance. The default mapping (shown in Figure 1a) resulted in a sustained floating-point performance of 39.5 TFlop/s. Attempts to minimize intra-column communication via a compact scheme (Figure 1b) did not improve performance, yielding 38.2 TFlop/s. Distributing each process column over a torus slice in a bipartite (Figure 1c) or quadpartite (Figure 1d) arrangement provided the highest overall performance: 64.0 TFlop/s and 64.7 TFlop/s. The fact that a 1.64 speedup can be achieved over the default node layout illustrates the critical importance of proper task layout on a machine like BG/L. The bipartite mapping was found to be the optimal mapping for partition sizes of 8k, 16k and 32k nodes.

We have explored how BG/L's MPI implementation maps collective operations to the torus network through hardware performance counts of torus network events. We focus on the

performance of MPI_Bcast, since this dominates communication costs in Qbox (see Table 1). Figure 2 shows the communication pattern of a single broadcast on a 4x4 plane of BG/L nodes using three eight-node communicators. Broadcasts over a compact rectangle of nodes (left panel), which use the torus network's broadcast functionality, have the most balanced packet counts as well as the lowest maximum count. When we split the nodes across multiple lines resulting in disjoint sets of nodes (middle panel), the communication requires significantly more communication packets with less balanced link utilization. The node mappings in Figure 1c) and 1d), on the other hand, lead to a more balanced link utilization (as illustrated in Figure 2, right panel) and hence to higher overall performance.

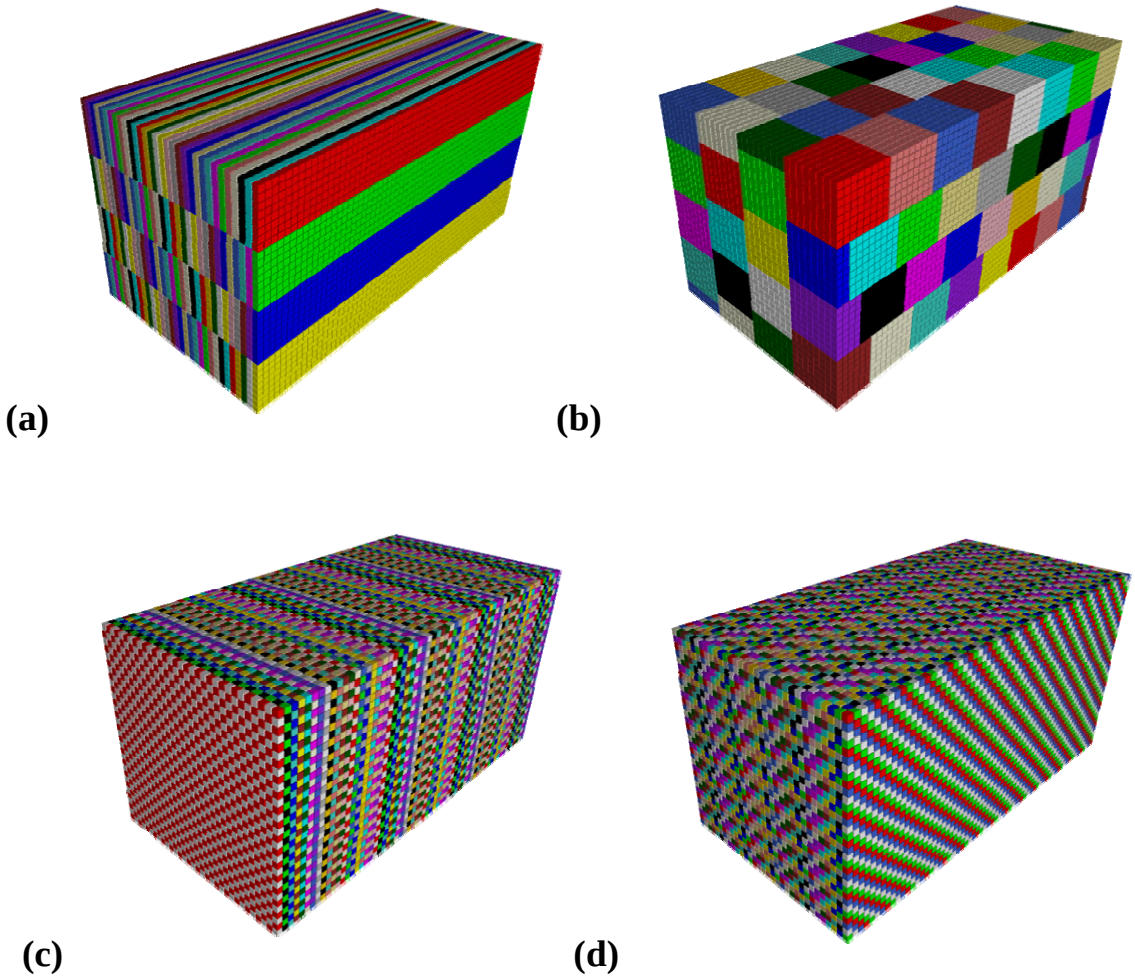

Figure 1. Illustration of different node mappings for a 64k-node partition. Each color represents the nodes belonging to one 512-node column of the process grid.

This analysis of Qbox communication led to several node mapping optimizations. In particular, our initial mappings did not optimize the placement of tasks within communicators. We have refined the bipartite mapping shown in Figure 1c to map tasks with a variant of Z ordering within a plane. This modification effectively isolates the subtree of a binomial software tree broadcast within subplanes of the torus. In addition, we observed that substantial time was spent in MPI_Type_commit calls in the BLACS library. The types being created were in many cases just contiguous native types, which allowed us to hand-optimize BLACS to eliminate calls in these cases. We are investigating other possible optimizations, including multi-level collective operations [13] that could substantially improve the performance of the middle configuration in Figure 2.

Table 1 Top 5 communication routines in Qbox measured using libmpitrace on 8k nodes, running 5 steepest descent iterations for 1000 molybdenum atoms with 1 (non-zero) k-point (complex arithmetic). The total run time was 2774.7 s, of which 571.4 s was spent in communication. Averages include the MPI_Bcast operations which take place once on startup and account for approximately 128 s of the total communication time.

	# calls	avg. msg size (bytes)	time (s)	% MPI	% total
MPI_Bcast	4048	950681.5	401.0	70.2%	14.5%
MPI_Allreduce	54256	35446.1	103.6	18.1%	3.7%
MPI_Alltoallv	5635	399.1	41.8	7.3%	1.5%
MPI_Reduce	940	186007.1	15.2	2.7%	0.5%
MPI_Barrier	2352	0.0	8.8	1.5%	0.3%

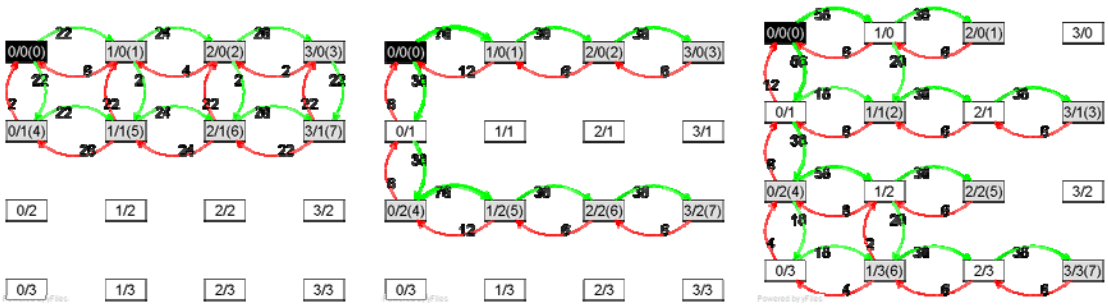

Figure 2: Communication pattern of a 4000 Byte MPI broadcast on an eight node communicator within a 4x4 node plane. The node labels indicate the coordinates of the node within the grid. Nodes in grey are part of the communicator (the rank is indicated in parenthesis), the node in black is the broadcast initiator, and nodes in white do not participate in the broadcast. Edges mark communication as measured using BG/L's hardware counters for torus traffic with the number showing the number of packets observed and the color indicating the direction of the traffic[14].

5. PERFORMANCE MEASUREMENTS

5.1 FPU Operations Count

Floating point operations were counted using the APC performance counter library. This library accesses the compute node ASIC's hardware performance counters to tracks several events including FPU, some SIMD and load and store operations. Counting can be limited to selected sections of the code by calling **`ApcStart()`**at the beginning and **`ApcStop()`**at the end of each section. In Qbox, these calls were placed around the main iteration loop, in order to get an accurate measure of sustained performance. Operation counts and total number of cycles used for each task are saved to individual files (one per task), and a cumulative report generated using the post-processing tool **`apc_scan.`**

BG/L's hardware counters do not include events for SIMD add/subtract, or multiply operations (although the fused multiply-add operations can be counted). Thus, some SIMD operations are not included in the count. For this reason, the floating-point performance cannot be extracted from a single measurement of the operation count. Instead, the number of cycles and the number of operations must be obtained from separate measurements using the following procedure:

1) Compile the code without SIMD instructions (i.e., use –qarch=440 with the xlC compiler), using unoptimized (non-SIMD) versions of the FFTW, DGEMM and ZGEMM libraries. Measure the total FPU operation count with this executable.
2) Recompile the code, enabling the SIMD instructions (using –qarch=440d). Obtain the total number of cycles and, thus, the total time with this executable. The FP operation count in this case is potentially inaccurate and should be discarded.
3) Divide the total FP operation count by the total time to compute the performance.

While this procedure requires running two simulations to get a single measurement, it uses BG/L's hardware counters to measure floating-point performance rigorously.

5.2 Results

Although many first-principles calculations require multiple simultaneous solutions of the KS equations, each representing different electronic spin states, k-points in the Brillouin zone, or imaginary-time slices in a path integral, the parallel efficiency of a calculation with a single set of KS equations is the most general measure of how successfully a FPMD code has been parallelized.

Thus, we first present results of calculations using a single k-point (k=0) before showing results with the multiple k-points needed for high-Z metals.

Table 2 shows strong-scaling results of Qbox for a simulation of 1000 molybdenum atoms and 12,000 electrons, for a single k-point. Simulations were performed on partitions of increasing size on the 64k-node LLNL BG/L machine. The problem size was kept constant for all partition sizes. The time per iteration reported is the wall-clock time needed to perform a single steepest-descent iteration on electronic states. Times are reported for the best node mapping at each partition size. All calculations were run in co-processor mode, although it should be noted that the DGEMM library was written to utilize both CPUs on a node even in coprocessor mode as discussed above.

Table 2. Qbox performance data for a molybdenum simulation including 1000 atoms and 12000 electrons with a plane-wave cutoff of 112 Ry, for a single k-point (k=0). The fractional speedup represents the fraction of ideal speedup obtained with respect to the 2048-node partition. The aggregate FP rate is measured with the APC performance counters as described in the text.

nodes	time/iteration (s)	speedup	frac speedup	agg. FP rate (TFlop/s)
2048	725.3	1.00	1.00	4.76
4096	348.6	2.08	1.04	9.86
8192	190.8	3.80	0.95	18.20
16384	108.2	6.70	0.84	32.63
32768	73.9	9.81	0.61	47.80
65536	55.3	13.12	0.41	64.70

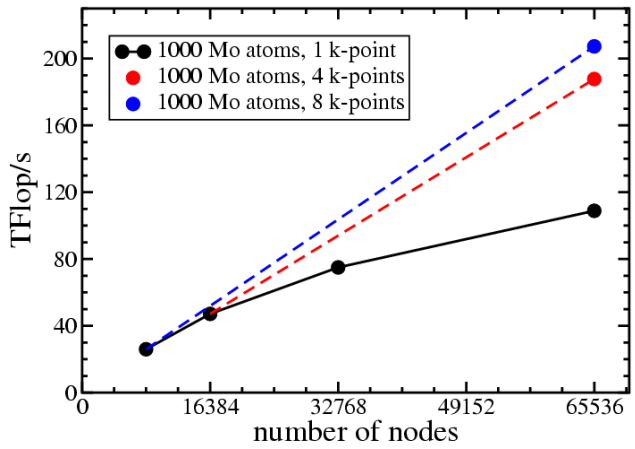

Figure 3. Strong scaling results for 1000 molybdenum atoms with 1 (non-zero) k-point. Also shown is the sustained performance on the full machine (64k nodes) with multiple k-points. Dashed lines indicate perfect scaling between the measured full machine result and the equivalent individual k-point calculations.

Calculations over a single k-point are often sped up by choosing the high-symmetry k=0 point (Γ-point) for which the imaginary part of the complex wave function is known to be zero by symmetry. In addition to decreasing the dimension of the wave function matrices by a factor of two, the linear algebra routines for matrices of doubles can be used, including DGEMM. In the case of multiple (non-zero) k-points, this symmetry does not hold, and complex linear algebra routines, including **`zgemm`**, are required. Performance results for simulations of 1000 molybdenum atoms with multiple k-points are shown in Figure 3 and Table 3. Excellent floating-point performance is achieved, primarily due to the highly optimized ZGEMM library used for single-node matrix multiplication and the fact that each k-point calculation takes place on 8k and 16k nodes where parallel efficiency per k-point is still quite high. A sustained performance of 207.3 TFlop/s was observed when 8 k-points were used, which corresponds to 56.5% of the theoretical full machine peak perfomance. This level of performance indicates an extremely efficient use of BG/L's computational resources, especially in view of the fact that these calculations were run in coprocessor mode with only the ZGEMM library making use of the second processor.

Table 3. Qbox performance data for a molybdenum simulation including 1000 atoms and 12000 electrons with a plane-wave cutoff of 112 Ry, as a function of number of k-points. Note that the single k-point calculations presented here are not calculated at k=0, as in Table 2, but instead use the same complex treatment as multiple k-points for the sake of accurate comparison.

nodes	# of k-points	time/iteration (s)	agg. FP rate (TFlop/s)
65536	1	127.13	108.8
65536	4	289.43	187.7
65536	8	526.91	207.3

6. CONCLUSION

We have demonstrated the feasibility of unprecedented large-scale First-Principles Molecular Dynamics, and the excellent scalability of the Qbox code on the BlueGene/L platform on up to 64k nodes. Our experiments indicate that a careful choice of node mapping is essential in order to obtain good performance for this type of application. Strong scalability of Qbox for a Materials Science problem involving 1000 molybdenum atoms, with 12000 electrons is excellent. The use of hand-optimized libraries for linear algebra and Fourier transform operations dramatically improves the effective floating point performance. This early application of First-Principles Molecular Dynamics demonstrates that the exceptional computing power provided by the BlueGene/L computer can be efficiently utilized and will have an important impact in the area of first-principles prediction of materials properties in the near future.

7. ACKNOWLEDGMENTS

UCRL-PROC-220592. Work performed under the auspices of the U. S. Department of Energy by University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

8. REFERENCES

[1] R. Car and M. Parrinello, Phys. Rev. Lett. 55, 2471 (1985). For a review, see e.g. M. Parrinello, “From Silicon to RNA: the Coming of Age of First-Principles Molecular Dynamics” Sol. St. Comm. 103, 107 (1997).

[2] W. Kohn and L.J.Sham, Phys. Rev. A140, 1133 (1965).

[3] F. Gygi, “Qbox: a large-scale parallel implementation of First-Principles Molecular Dynamics” (http://eslab.ucdavis.edu).

[4] For a review, see e.g. F. Gygi and G. Galli, *"Ab initio simulations in extreme conditions"*, Materials Today **8**, 26-32 (2005).

[5] N. R. Adiga et al., “An overview of the BlueGene/L supercomputer” SC2002 – High Performance Networking and Computing, 2002.

[6] F. Gygi, E.W. Draeger, B.R. de Supinski, R.K. Yates, F. Franchetti, S. Kral, J. Lorenz, C.W. Ueberhuber, J.A. Gunnels, J.C. Sexton, "Large-Scale First-Principles Molecular Dynamics Simulations on the BlueGene/L Platform using the Qbox Code," SC2005, 2005.

[7] L. Bachega, S. Chatterjee, K. Dockser, J. Gunnels, M. Gupta, F. Gustavson, C. Lapkowski, G. Liu, M. Mendell, C. Wait, T.J.C. Ward, “A High-Performance SIMD Floating Point Unit Design for BlueGene/L: Architecture, Compilation, and Algorithm Design” PACT, 2004.

[8] L.S.Blackford, J.Choi, A.Cleary, E.D’Azevedo, J.Demmel, I.Dhillon, J.Dongarra, S.Hammarling, G.Henry, A.Petitet, K.Stanley, D.Walker, R.C.Whaley, “ScaLAPACK Users’ Guide” SIAM, Philadelphia, (1997).

[9] M. Frigo and S. G. Johnson: FFTW: an adaptive software architecture for the FFT, Proceedings of ICASSP 1998, Vol.3, pages 1381-1384

[10] J. Lorenz, S. Kral, F. Franchetti, C. W. Ueberhuber: Vectorization techniques for the BlueGene/L double FPU, IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005, pages 437-446

[11] S. Kral: FFTW-GEL Homepage: http://www.complang.tuwien.ac.at/skral/fftwgel.html

[12] Franchetti, S. Kral, J. Lorenz, C. W. Ueberhuber: Efficient Utilization of SIMD Extensions, Proceedings of the IEEE Special Issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, 2005, pages 409–425.

[13] N.T. Karonis, B.R. de Supinski, I. Foster, W. Gropp, E. Lusk, J. Bresnahan, "Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance," 14th International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun, Mexico, May 1–5, 2000.

[14] Graphs in Figure 2 were generated with YED (http://www.yworks.com/).