LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-CONF-211703

Large-Scale First-Principles Molecular Dynamics Simulations on the BlueGene/L Platform using the Qbox Code

F. Gygi, E. W. Draeger, B. R. de Supinski, R. K. Yates, F. Franchetti, S. Kral, J. Lorenz, C. Ueberhuber, J. A. Gunnels, J. C. Sexton

April 26, 2005

Supercomputing 05
Seattle, WA, United States
November 12, 2005 through November 18, 2005

Large-Scale First-Principles Molecular Dynamics Simulations on the BlueGene/L Platform using the Qbox Code

François Gygi, Erik W. Draeger, Bronis R. de Supinski, Robert K. Yates
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551

Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University

Stefan Kral, Juergen Lorenz, Christoph W. Ueberhuber
Institute of Analysis and Scientific Computing
Vienna University of Technology, Vienna, Austria

John A. Gunnels, James C. Sexton
IBM Thomas J. Watson Research Center
Yorktown Heights, NY

Abstract

We demonstrate that the Qbox code supports unprecedented large-scale First-Principles Molecular Dynamics (FPMD) applications on the BlueGene/L supercomputer. Qbox is an FPMD implementation specifically designed for large-scale parallel platforms such as BlueGene/L. Strong scaling tests for a Materials Science application show an 86% scaling efficiency between 1024 and 32,768 CPUs. Measurements of performance by means of hardware counters show that 37% of the peak FPU performance can be attained.

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 combines a quantum mechanical description of electrons with a classical description of atomic nuclei. The Newton equations of motion for all nuclei are integrated in time 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.

The electronic structure calculation is the most time-consuming part of an FPMD simulation, and limits the size of tractable systems to a few hundred atoms on most currently available parallel computers. It consists in solving the Kohn-Sham (KS) equations [2], a set of non-linear, coupled integro-differential partial differential

equations. Considerable efforts have been devoted over the past three decades to the development of efficient implementations of the electronic structure computation. In its most common implementation, the solution of the Kohn-Sham equations has $O(N^3)$ complexity, where N is the number of atoms.

In this paper, we report on FPMD simulations performed using the Qbox code on the BlueGene/L (BG/L) computer installed at Lawrence Livermore National Laboratory. Qbox is a parallel implementation of the FPMD method designed specifically for large parallel platforms, including BlueGene/L. Simulations were performed using up to 32,768 processors, including up to 1000 atoms. Performance measurements of strong scaling show that an 86% parallel efficiency is obtained between 1k and 32k CPUs. Floating point operation counts measured with hardware performance counters show that 37% of peak performance is attained when using 4k CPUs.

Predictive Simulations of Materials Properties

Although the applicability of FPMD is not limited to a specific class of physical system, we focus in this paper on an important issue of Materials Science, the simulation of metals under extreme conditions. Predicting the properties of metals at high pressures and high temperatures has been a longstanding goal of Materials Science and high energy density physics. Experimental measurements of these properties can be extremely challenging because of the difficulty to achieve and maintain extreme conditions of pressure and temperature. FPMD simulations offer the possibility of predicting these properties without relying on any empirical or adjusted parameters. The validation of the physical models used in FPMD simulations is an important step toward this goal, and requires large computational resources.

In this paper, we demonstrate the feasibility of FPMD simulations on large samples of transition metals. The sample chosen for the present performance study consists of 1000 molybdenum atoms (12,000 electrons) at ambient pressure conditions, 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 44 Ry defines the basis set used to describe the electronic wave functions. We choose to work at ambient pressure since this is the most CPU-intensive calculation: higher pressures involve smaller simulation volumes and are correspondingly less demanding.

This calculation 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 using the two-phase simulation technique [3], and the calculation of defect energies and defect migration processes.

Qbox: A Highly Scalable FPMD Simulation Implementation

Qbox is a C++ implementation of the FPMD method. It is developed at the Center for Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory

(LLNL). A detailed description of the code has been given elsewhere [4]. Qbox exploits parallelism using the MPI message-passing paradigm. 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. Qbox solves the Kohn-Sham (KS) equations within the pseudopotential, plane wave formalism. The solution of the KS equations has been extensively discussed by various authors and will not be repeated here [1].

BlueGene/L: A Scalable Architecture for Scientific Simulations

BlueGene/L, as designed and implemented by a partnership between Lawrence Livermore National Laboratory (LLNL) and IBM, is a tightly-integrated large-scale computing platform. Its compute node ASICs include all networking and processor functionality; in fact, a compute node uses only that ASIC and nine DRAM chips. This system-on-a-chip design results in extremely high power and space efficiency. The full details of the system architecture are covered elsewhere [5]; we give a brief overview with a focus on aspects that are particularly relevant to Qbox here.

Individual compute ASICs represent a custom design based on off-the-shelf component designs. They feature two 32-bit superscalar PowerPC 440 embedded cores that are clocked at 700 MHz. There are two modes to use the cores: communication coprocessor, in which one core is dedicated to servicing communication requests, and virtual node mode, in which two MPI tasks run on each node. Although the former sacrifices half of the compute power of the machine, it supports higher realized network bandwidth and was the original intent of the hardware design.

The chip associates two copies of the PPC floating point unit (FPU) with each core that function as a SIMD-like double FPU [6]. The double FPU is not two independent FPUs but instead supports an extensive set of parallel instructions for which the double precision operands can come from the register file of either unit. The instructions include a variety of paired multiply-add operations, resulting in a peak of four floating point operations (FLOPs) per cycle per core. An application's ability to use these SIMD-like parallel instructions is crucial to its overall performance. Due to limitations in the compilers currently available, Qbox currently only uses these instructions in the DGEMM and FFT implementations discussed later in this paper.

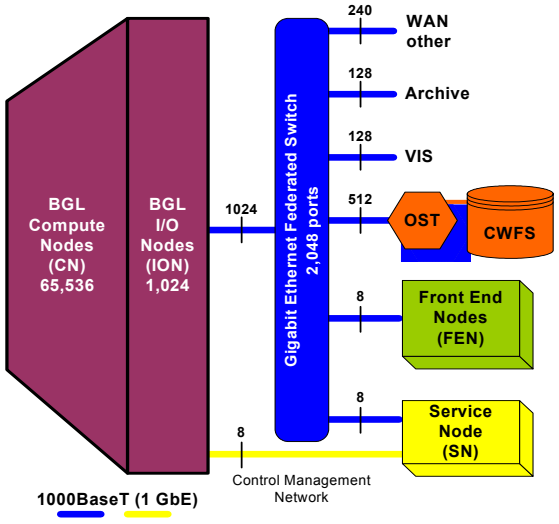

Figure 1: Integrated BlueGene/L Platform

The integrated BG/L platform significantly increases the scale of high-end computing platforms. The final machine, expected in the fall of 2005, will have 65,536 compute nodes for a total peak performance of 360TF; half of that machine is already installed and in use at LLNL. As shown in Figure 1, the final system will include 1024 I/O nodes that are nearly identical to the compute nodes. The difference is that they include gigabit ethernet (Gig-E) connections that

connect the system to a large parallel file system. They also connect the system to the front-end nodes, on which users compile their codes and launch their jobs.

The system includes five networks: three dimensional torus, tree, global interrupt, Gig-E I/O and control. The compute nodes communicate to the IO nodes over the tree, which is also used for a subset of MPI_COMM_WORLD collective operations. Point-to-point communication and collectives on subcommunicators use the torus network. Since almost all MPI communication in Qbox is through BLACS calls or ScaLAPACK routines that use derived communicators that represent a column or row of a process grid, we focus on the torus network. Each compute node has six bidirectional torus links with a raw hardware performance of 175 MB/s per link; measured MPI pingpong bandwidth is 150 MB/s. However, on-node message processing limits total realized MPI bandwidth over the torus links to less than 500 MB/s. The torus includes broadcast support that allows optimized implementations of many MPI collectives. Currently, those implementations are limited to communicators that form rectangular subprisms of the torus.

Node mapping strategies

Unlike many applications for which a simple 3D domain decomposition naturally maps to a 3D torus architecture, the KS equations in the plane wave formalism do no exhibit any obvious way to map parts of the calculation to the torus. The data layout adopted in Qbox distributes the degrees of freedom describing electronic wave functions on a two dimensional process grid similar to the process grids used in the BLACS communication library [7]. MPI subcommunicators are defined appropriately in order to facilitate communication along rows and columns of the process grid. In order to obtain a good connectivity between subcommunicators, it is necessary to use node maps that are more complex than the XYZ, YZX or ZXY orderings provided by default. We have explored a variety of node mappings in order to optimize performance.

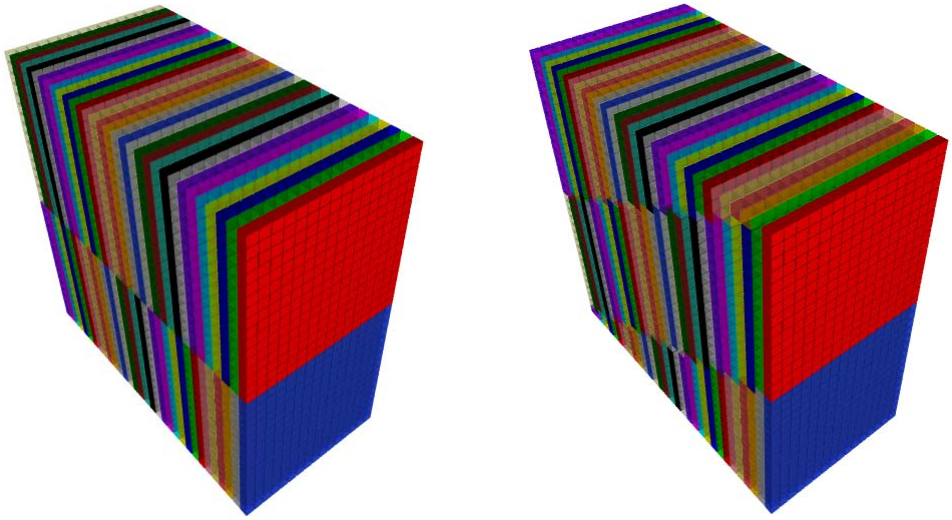

Figure 1 Illustration of two choices of node mappings for a 16k-node partition. The nodes associated to a column of the process grid are mapped to 3D torus half-planes. Each color represents the nodes belonging to one column. Half-planes corresponding to successive columns share a common face for maximal connectivity (e.g. column 0 is red and column 1 is green). Although the left mapping leads to good connectivity between most neighboring columns, some (e.g., columns 31 and 32) only share an edge. A twist in the mapping (right) maximizes local connectivity between columns and avoids such discontinuities.

This optimization must be carried out for each partition size, since the shape of a partition changes with size. For example, a 4k-node partition consists of a 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 using the BGLMPI_MAPPING environment variable. Figure 1 shows an example of two node mappings used on a 16k-node partition. The performance results reported in this paper were obtained with the best mappings that we developed to date, although our investigation of optimal node mapping strategies is ongoing.

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 sections, we describe the optimized implementations of these kernels used in Qbox.

Linear Algebra

Dense linear algebra is implemented through the ScaLAPACK library [7]. 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 **`pdgemm`** makes extensive use of the BLAS3 **`dgemm`** matrix multiplication kernel. We used a hand-optimized version of the **`dgemm`** kernel that we describe in more detail below.

Optimized DGEMM library

While the performance of the 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 **`dgemm`** routine on BG/L is most easily understood when described from the bottom up.

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% the theoretical peak rate of the processor.

Fortunately, general matrix-multiplication is dominated by FMAs and BG/L's relatively rich instruction set allows one to utilize the SIMD FMA instructions for the vast majority of computations involved in **`dgemm`**. 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). Fortunately, the load-primary and load-secondary instructions allow this, regardless of the relative alignment of the input data.

Further, since the number of computations involved in **dgemm** is an order of magnitude greater than the number of data moves (loads and stores), data copies can be employed to align data on 16-byte boundaries, enabling the use of SIMD loads almost exclusively.

The Computational Kernel: Register-Based View

Traditionally, the matrix-multiplication computational core, or "kernel routine," is carefully written so as to respect the architecture of the machine under consideration. Typically, and on BG/L, the most important considerations are: 1) the number of architected (useable) 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 accessible SIMD (length 2) floating-point registers. We used these registers to target a 6x6xK matrix multiplication kernel as our main computational workhorse. Our register blocking uses 30 SIMD registers: six for A operands, six for B operands, and 18 for C operands. The computation is composed as two rank-1 updates of the C-registers, yielding, simplistically, an 18-cycle latency between outer products and thus tolerates more than 18 cycles of latency for loads.

L1 Cache Considerations

BG/L's L1 caches are 16-way, 64-set associative and use a round-robin replacement policy [8]. 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 **dgemm** 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.

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 the 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. Alternate modes of use would have made three streams the most effective design point, and the routine was prototyped before final hardware decisions had been made.

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 6x6x2 register kernel, the inner loop (the part that is neither loading nor storing C) requires exactly one SIMD (quadword) load every three cycles. Thus, it is not surprising that the inner loop of this routine runs between 95% and 100% of the peak rate of the machine once the data is in L3 and the L2 prefetch mechanism is engaged.

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.

While the **`dgemm`** 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.

Fourier Transforms

Qbox takes advantage of the fact that many 3D FTs must be performed simultaneously (one for each electronic state) during an electronic structure calculation. This allows Qbox to dispatch the 3D transforms to subpartitions of the machine, and thus requires only moderate scalability of the 3D FT kernel. A custom parallel implementation of 3D FT was developed and shows excellent scaling properties on up to 512 tasks. Scaling beyond this size is not required since a sufficient number of transforms can occur simultaneously to use the entire machine. The 3D FT kernel requires an efficient implementation of single-node, one-dimensional complex FTs. We use the FFTW-GEL library that was optimized for BG/L.

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]. In this paper, we use FFTW-GEL for BG/L [10], an FFTW 2.1.5 replacement for BG/L based on the SIMD FFTW replacement provided by FFTW-GEL [11]. FFTW-GEL is available for AMD and Intel platforms and BG/L.

FFTW automatically optimizes FFT computation on a wide range of machines. FFTW's codelet compiler `genfft` automatically generates small FFT kernel routines called *codelets*, minimizing the number of arithmetic operations and ordering these for locality. In an initialization step at runtime, FFTW's *planner* finds the fastest recursive decomposition of a given FFT computation into these codelets by applying the Cooley-Tukey FFT algorithm and dynamic programming.

FFTW 2.1.5 codelets account for almost all of FFTW's computational work. They are generated as ANSI C straight-line code using real scalar arithmetic and feature complicated dependencies and (almost) no control flow. Compiling these codelets for BG/L's double FPU does not result in good utilization of the two-way vector instructions provided by the double FPU due to the complicated data dependencies which are beyond the vectorization analysis in the xlC compiler for BG/L. FFTW-GEL for BG/L solves this problem by replacing the original scalar FFTW codelets by 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.

At the heart of FFTW-GEL for BG/L is the Vienna MAP vectorizer [12]. MAP two-way vectorizes large computational basic blocks by a depth-first search with chronological backtracking. It produces explicitly vectorized FFTW codelet with solely two-way vector instructions and a minimum of data reorganization instructions. This process fuses scalar variables into vector variables, which requires fusing the corresponding scalar operations into vector operations. This creates a larger required search space since the scalar variables and operations can be fused in multiple ways. MAP's vectorization rules that describe the variable and operation pairing encode machine characteristics like the double FPU's special fused multiply-add instructions. As MAP commits to the first found solution, the order of MAP's vectorization rules can guide the vectorization process. It must be noted that the performance increase of FFTW-GEL due to SIMD instructions is large (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.

Performance measurements

We now describe the process used to measure the performance of Qbox on BG/L.

FPU operations count

We counted floating point operations using the APC performance counter library. This library allows the user to access the compute ASIC's hardware performance counters. APC tracks several events including FPU operations, some SIMD operations and load and store operations. APC can limit counting to selected sections of the code by calling **`ApcStart()`** and **`ApcStop()`** functions before and after these sections. Operation counts are saved in binary format in separate files for each task at the end of the run. The post-processing program **`apc_scan`** then produces a cumulative report of operation counts, as well as of the total number of cycles used. By default, the APC library limits the number of data files to 16. We use the APC_SIZE environment variable to obtain one file per task, since the number of operations performed on each node is not strictly identical, especially during ScaLAPACK calls.

We instrumented Qbox with APC calls around its main iteration loop. The initialization phase that involves opening and reading input files and allocating large objects was not included in the performance calculation. Since molecular dynamics simulations are typically run for thousands of iterations, our performance measurements are representative of the use of Qbox in actual simulations.

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 may not be included in the count. For this reason, the FP performance cannot be extracted from a single measurement of the operation count and of the total number of cycles used. 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). 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.

This procedure is further complicated in the case where some libraries are hand coded to contain SIMD instructions, either because they are written in assembly language, or because their high-level source code contains SIMD intrinsics. In this case, the number of FP operations cannot be measured by disabling the SIMD operations as in step 1) above. This applies in our case to the FFTW-GEL library and to the DGEMM optimized library.

In the case of the FFTW library, we tested the effect of the SIMD instructions in FFTW-GEL by comparing its performance to the FFTW 2.1.3 library compiled without the SIMD option. These measurements indicate that the FFTW-GEL performance is 20-25% higher than the standard FFTW 2.1.3 library when used on multiple transforms of size comparable to those done in other Qbox runs. Since the total amount of time spent in the FFTW library is a relatively small fraction of the total time, we use the FP operation count obtained with the FFTW-GEL library event though the actual operation count is 25% higher. Our performance estimation is therefore an underestimation of the actual performance. We plan to complete more accurate measurement of the exact (higher performance) numbers in the coming month and include them in the final submission.

In the case of the DGEMM optimized library, we observed that the operation count reported by the APC library does indeed correspond to the total number of FP operations, by comparison with a standard implementation of the BLAS library. This last property is due to the fact that the DGEMM optimized library uses essentially only SIMD fused multiply-add operations, which are properly counted by the APC library.

Results

We report in Table 1 our measurements of the performance of Qbox for a simulation of molybdenum including 1000 atoms and 12000 electrons.

Simulations were performed on partitions of increasing sizes on the 32k-node LLNL BG/L platform, using co-processor mode. The problem size was kept constant for all partition sizes. The time per iteration reported is the wall-clock time needed to compute a single steepest-descent iteration on electronic states. Times are reported for the best node mapping at each partition size.

nodes	time/iteration (s)	speedup	frac speedup	agg. FP rate (TF)	peak FP fraction
1024	1040	1.00	1.00	0.89	0.31
2048	449	2.32	1.16	1.95	0.34
4096	210	4.95	1.24	4.13	0.36
8192	121	8.60	1.07	7.11	0.31
16384	66	15.76	0.98	13.76	0.30
32768	38	27.37	0.86	22.02	0.24

Table 1 Qbox performance data for a molybdenum simulation including 1000 atoms and 12000 electrons. The fractional speedup represents the fraction of ideal speedup obtained with respect to the 1024-node partition. The aggregate FP rate and the peak fraction are measured with the APC performance counters and represent a lower bound (see text). The peak FP rate for 32k nodes in coprocessor mode is 91.75 TF.

We observe superlinear scaling with 2k-8k CPUs. This is attributed to 1) a reduction in the amount of data residing on each node, and thus leading to better use of the cache, and 2) the fact that optimal node mappings lead to more efficient communication on the 2k- and 4k-node partitions than on the 1k-node partition.

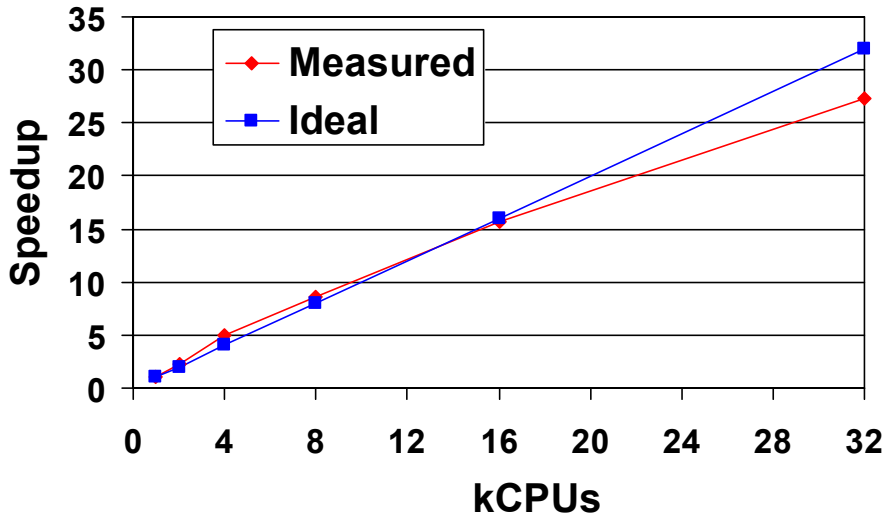

Figure 1 Strong scaling of Qbox for a simulation of 1000 molybdenum atoms. A scaling efficiency of 86% is achieved between 1k and 32k CPUs.

Ongoing Research and Anticipated Results

The results presented in this paper are current as of April 25th, 2005 and represent lower bounds on the performance of Qbox on BG/L. Ongoing developments in both application and system software will improve some of these results. In the following, we discuss some likely sources of performance improvement.

Generation of SIMD instructions using the xlC compiler

As of the submission date, the ability of the BG/L xlC compiler to generate vectorized (SIMD) instructions has not yet reached its full potential. Further improvements in the compiler are likely to improve the performance of Qbox. The current performance of Qbox is however obtained principally through the use of the hand-optimized DGEMM library, and (to a lesser extent) of the FFTW-GEL library. Thus, we expect that improved generation of SIMD instruction will not drastically change the performance figures.

Optimized BGL MPI implementation

The current version of the BG/L MPI library provides an MPICH implementation of most functions, and optimized implementations of many MPI collective operations that exploit BG/L's torus and tree networks. We found that for some partition sizes (in particular the full machine, 32k node partition), some optimized implementations had to be disabled in order to obtain correct results. Further developments will likely allow us to remove this constraint, and increase performance over the numbers presented here.

Furthermore, the current BG/L MPI implementation does not yet fully exploit the possibility of using the I/O co-processor for communication. We expect that the future availability of concurrent computation and communication when using co-processor mode will further enhance the performance of Qbox on BG/L.

Conclusion

We have demonstrated the feasibility and excellent scalability of the Qbox code for unprecedented large-scale First-Principles Molecular Dynamics on the BlueGene/L platform on up to 32k CPUs. 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 (12000 electrons) is excellent. A speedup of 24.7 is achieved when increasing the partition size from 1k nodes to 32k nodes, which corresponds to a parallel efficiency of 86%. The use of hand-optimized libraries for linear algebra and Fourier transform operations dramatically improves the effective floating point performance, which lies between 24% and 37% of peak performance depending on partition size. 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 modeling in the near future.

Acknowledgement

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. S. Kral was supported by LLNL subcontract No. B539881. F. Franchetti and J. Lorenz were supported by the Austrian Science Fund FWF.

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. T. Ogitsu, E. Schwegler, F. Gygi, and G. Galli, Phys. Rev. Lett. 91, 175502 (2003).
4. F. Gygi, “Qbox: a large-scale parallel implementation of First-Principles Molecular Dynamics” (LLNL preprint, 2005).
5. N. R. Adiga et al., “An overview of the BlueGene/L supercomputer” SC2002 – High Performance Networking and Computing, 2002.
6. 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.
7. 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).
8. These caches are not coherent in “co-processor mode” operation, but the advantages and disadvantages of this property are beyond the scope of this paper.
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.