Architecture of Qbox: A scalable first-principles molecular dynamics code

F. Gygi

We describe the architecture of Qbox, a parallel, scalable first-principles molecular dynamics (FPMD) code. Qbox is a C++/Message Passing Interface implementation of FPMD based on the plane-wave, pseudopotential method for electronic structure calculations. It is built upon well-optimized parallel numerical libraries, such as Basic Linear Algebra Communication Subprograms (BLACS) and Scalable Linear Algebra Package (ScaLAPACK), and also features an Extensible Markup Language (XML) interface built on the Apache Xerces-C library. We describe various choices made in the design of Qbox that lead to excellent scalability on large parallel computers. In particular, we discuss the case of the IBM Blue Gene/L™ platform on which Qbox was run using up to 65,536 nodes. Future design challenges for upcoming petascale computers are also discussed. Examples of applications of Qbox to a variety of first-principles simulations of solids, liquids, and nanostructures are briefly described.

Introduction

First-principles molecular dynamics (FPMD) is an atomistic simulation method that combines an accurate description of electronic structure with the capability to describe dynamical properties by means of molecular dynamics (MD) simulations. This approach has met with tremendous success since its introduction by Car and Parrinello in 1985 [1]. Because of its unique combination of accuracy and generality, it is widely used to investigate the properties of solids, liquids, biomolecules, and, more recently, nanoparticles. In correspondence with this success, FPMD has also become one of the most important consumers of computer cycles in many supercomputing centers. This, in turn, has motivated research on the optimization of the algorithms used in FPMD. The success of FPMD can be partially attributed to the fact that, in its original implementation, which was based on Fourier representations of solutions, the method readily benefited from mature and well-optimized numerical algorithms, such as the fast Fourier transform (FFT) and dense linear algebra. In the late 1980s, early implementations of FPMD on vector computers consistently achieved a sustained performance of about 50% of peak performance.

As a result of the need for simulations of increasing size and the availability of parallel computers, FPMD code design has been moving toward parallel implementations. In the 1990s and early 2000s, the FPMD community adapted to the change in computer architecture by developing code that scaled to a few hundred processors and maintained a similar ratio of peak performance. With the advent of larger, massively parallel computers, such as the IBM Blue Gene* supercomputer, the need to redesign FPMD applications for large-scale parallelism grew strong.

The Qbox project [2] was started in anticipation of the first Blue Gene machine, the IBM Blue Gene/L* platform [3], which was to be installed at the Lawrence Livermore National Laboratory. With 65,536 nodes and 131,072 processors, the Blue Gene/L architecture required a complete redesign of existing FPMD codes. The design of Qbox was carried out with specific attention paid to the requirement to distribute nearly all data structures on a platform as large as the Blue Gene/L platform. A careful

0018-8646/08/$5.00 © 2008 IBM

distribution of tasks on subpartitions of the machine, coupled with the use of highly optimized numerical kernels and libraries, eventually led to excellent scaling on the Blue Gene/L platform. This allowed Qbox to achieve an exceptionally high floating-point performance (207 Tflops, or 57% of peak) on the Blue Gene/L supercomputer [3], for which the Qbox team was awarded the 2006 Gordon Bell Award for peak performance [4].

In this paper, we focus on the software architecture of the Qbox code. After a brief description of the electronic structure problem, we present the high-level software abstractions used to solve it. We then discuss the parallelization approach and provide some examples of recent applications.

Electronic structure problem

The main computational task of FPMD is the calculation of the electronic structure [5]. In Qbox, this is done within the density functional theory formalism [6] by solving the Kohn–Sham (KS) equations [7]. The KS equations are a set of coupled partial differential equations whose solutions are the electronic orbitals. The electronic charge density depends on the electronic orbitals and, in turn, enters the definition of the potential acting upon the electrons in a nonlinear way. This feedback mechanism therefore requires a self-consistent approach in order to reach a solution in which the electronic charge density is consistent with the electronic potential. This is achieved by solving the KS equations repeatedly, starting from an initial guess of the electronic charge density and then recomputing the electronic potential until it is unchanged by further iterations.

The discretization of the KS equations using a plane-wave basis results in a large eigenvalue problem. (A full description of the plane-wave method is given in [5]). Although the matrix representing the KS Hamiltonian operator is dense in the Fourier basis, the calculation of a matrix-vector product can be performed efficiently by noting that the KS Hamiltonian is a sum of two operators (the kinetic and potential energy), which are respectively sparse in Fourier space and in real space. This property can be used to implement efficient iterative eigensolvers for the KS equations. As a consequence, the KS solutions must be transformed frequently between a Fourier representation and a real-space representation. This requires an efficient implementation of three-dimensional (3D) FFTs. The solutions of the KS equations must also be kept orthonormal during the simulation. This constraint is enforced using a variety of algorithms including the Gram–Schmidt procedure.[1] The orthogonal constraints require an efficient implementation of dense linear algebra operations for large matrices. Qbox relies on the Scalable Linear Algebra Package (ScaLAPACK) library [8] for the implementation of parallel linear algebra operations. ScaLAPACK is a parallel library of linear algebra functions written in Fortran. It relies on the Basic Linear Algebra Communication Subsystem (BLACS) library to implement communication between processes.[2]

[1]Although the modified Gram–Schmidt algorithm is known to have better numerical properties than the Gram–Schmidt algorithm, it is not needed here since the solutions of the KS equations are always kept nearly orthogonal during the simulation.

High-level abstractions and design

The high-level software components used in Qbox fall into three main categories. The first category consists of components providing a simulation infrastructure, such as a command interpreter and a suitable user interface. The second category (the `Sample` class hierarchy) includes components describing the physical system being simulated. A third category (the algorithm class hierarchy) consists of various algorithms that allow the user to modify and compute the time evolution of the physical system. Efforts were made in the design of Qbox to keep these categories as weakly coupled as possible. For example, the user interface infrastructure is independent of the problem of FPMD simulations and could be reused for other applications. Well-established guidelines of C++ software design (e.g., [10]) were adhered to as much as possible, in particular, in the goal of limiting dependencies between components and eliminating cyclic dependencies. Polymorphism was used to ensure flexibility and extensibility of the main code features, as illustrated in the next section. Multiple inheritance was avoided, however, on account of its potential complexity [11]. C++ templates are not used in Qbox except in the context of the Standard Template Library (STL). This choice was the result of insufficient support from compilers during the initial phase of the design. Attempts were also made to use C++ exceptions in some simple situations in which error recovery is manageable.

Simulation infrastructure

At the core of the Qbox simulation infrastructure is a command interpreter object (class `UserInterface`) that reads commands from a C++ stream (standard input or an input script) and executes them sequentially [12]. The functionality of Qbox is defined by a collection of command objects derived from an abstract `Cmd` base class. User-defined parameters (e.g., the simulation timestep or the wavefunction energy cutoff) are represented by a collection of objects (or *variables*) derived from an abstract `Var` class. The possibility of adding any number of commands and variables to the user interface facilitates the extension of Qbox to include new functionality. The

[2]Both ScaLAPACK and BLACS are available at [9].

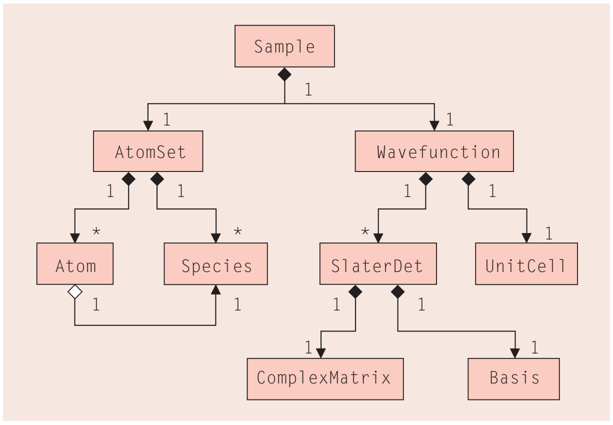
Cmd class has a pure virtual action ( ) member function that defines the functionality of the command. Similarly, the Var class has a pure virtual set ( ) member function that defines how a variable should be initialized by a user. This use of polymorphism allows for a complete decoupling of the user interface from the set of commands and variables. The ability of the user interface to read commands from a stream makes it possible to use Qbox interactively, in batch mode, or in client–server mode using UNIX**-named pipes.

Given the frequent use of dense linear algebra operations in plane-wave electronic structure calculations, we have developed auxiliary components to facilitate such operations. The matrix classes ComplexMatrix and DoubleMatrix encapsulate the functionality of the ScaLAPACK parallel linear algebra library. These matrix classes form a simple implementation of a façade design pattern [13] for the ScaLAPACK library. No effort was made to overload algebraic operators for matrix operations. The complexity of operator overloading for matrices and the corresponding difficulties in managing hidden temporary objects have been described by Stroustrup [14]. The use of C++ templates to parameterize matrix types (complex or double) was also avoided Finally, the possibility of using a DoubleMatrix as a proxy for a ComplexMatrix (and the reverse) is provided in the form of special constructors for each matrix type.

Sample class hierarchy

Sample is a structure encapsulating all of the information needed to describe the physical system being simulated with Qbox. This includes the list of atomic species used (e.g., carbon and oxygen) and their associated pseudopotentials, the list of atomic positions and velocities, the definition of the electronic wavefunction, and the dimensions of the periodic unit cell. This information is broken into separate components. **Figure 1** shows the main components of the Sample class in Unified Modeling Language** (UML**) notation [15].

The AtomSet component encapsulates information about the set of atoms currently defined in the simulation, as well as a list of defined species. The Wavefunction component can include multiple Slater determinants (represented by the SlaterDet class) in order to represent the electronic structure at multiple k-points in the Brillouin zone. Each SlaterDet includes its own plane-wave Basis object defined at the relevant k-point in the Brillouin zone. The Fourier coefficients of the one-particle wavefunctions are arranged into a rectangular ComplexMatrix object that encapsulates the functionality of a ScaLAPACK distributed matrix of complex, double-precision numbers.



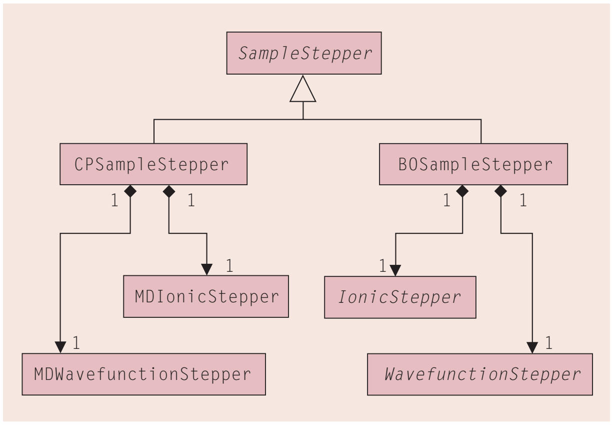
for position only

Figure 1

UML diagram of the main components of the Sample class hierarchy.

Stepper class hierarchy

The algorithms used to modify the state of a Sample during successive timesteps are generically called *steppers* and are defined in a separate class hierarchy. This separation effectively decouples the representation of a sample from the methods used to describe its evolution. The base class of all steppers is the abstract SampleStepper class (**Figure 2**). Qbox currently includes two types of steppers that encapsulate the algorithms for Car–Parrinello [1] and Born–Oppenheimer [5] MD, respectively. Algorithms for the modification of wavefunctions are further described by classes derived from the abstract WavefunctionStepper class. Similarly, algorithms for the modification of atomic positions are



for position only

Figure 2

UML diagrams of the SampleStepper hierarchy. Abstract classes are specified using italic typeface.

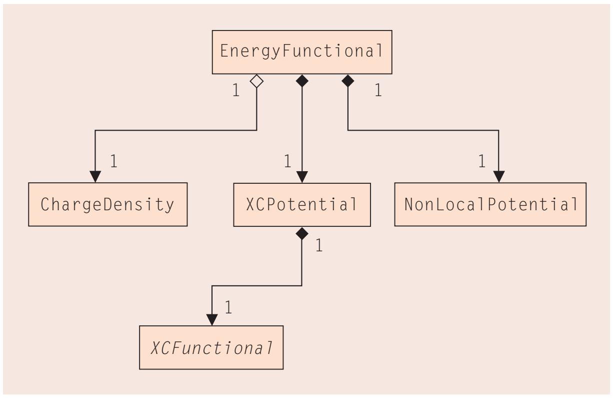


Figure 3

UML diagram of the main components used by the `EnergyFunctional` class. Abstract classes are specified using italic typeface.

encapsulated in classes derived from `IonicStepper`. This use of polymorphism leaves open the possibility of extending the set of algorithms by derivation of additional stepper classes. It also allows for the development of prototype algorithms (i.e., new derived stepper classes) while keeping the implementation of existing algorithms unchanged.

Examples of classes derived from `IonicStepper` include those that implement the Verlet algorithm for MD [16] or a steepest descent geometry optimization algorithm. Similarly, a number of wavefunction modification algorithms are implemented, including the constrained Verlet algorithm for wavefunctions used in Car–Parrinello dynamics [1] or variants of the steepest descent optimization with preconditioning, Anderson acceleration, or both [17].

The above stepper algorithms use the `EnergyFunctional` class (**Figure 3**), which includes the algorithms needed to compute the KS energy and its functional derivatives. `EnergyFunctional` relies on a number of auxiliary components, including `ChargeDensity`, `NonLocalPotential`, and `XCPotential`. The `XCPotential` class uses an instance of one of the classes `LDAFunctional`, `PBEFunctional`, or `BLYPFunctional`, which are derived from the abstract base class `XCFunctional`. Theses classes implement various types of density functionals [6] and can be used interchangeably. `EnergyFunctional` computes the energy of a given `Sample` as well as the derivatives of the KS energy with respect to its `Wavefunction` component, its `AtomSet` component, or its `UnitCell` component. These derivatives are, in turn, used by the various stepper classes to update the `Sample` using specific algorithms.

Parallel implementation

Qbox was designed for operation on massively parallel platforms. Current terascale and future petascale computers are typically characterized by the relatively limited amount of memory available on each node. For a simulation code to run on such platforms, it is necessary to distribute all significant data structures across various subparts of the machine. In Qbox, all the data describing electronic wavefunctions is distributed. Fourier coefficients of the wavefunctions are distributed according to both the plane-wave index and the band index, leading to a two-dimensional (2D) block matrix distribution. Data structures such as the list of atomic positions and forces or the list of atomic species use comparatively little space and are therefore replicated on each node.

An important consideration in the design of a parallel electronic structure code is that not all data structures should be distributed on the entire set of processors. The electronic structure problem involves tasks of various sizes. Most tasks cannot be performed on a single node (for lack of memory) and cannot be performed efficiently on a large number of nodes (e.g., 16K nodes) because the computation-to-communication ratio becomes unfavorable. For intermediate-sized tasks, a distribution on a subset of the nodes is more appropriate and leads to faster execution. The calculation of the electronic charge density from wavefunction Fourier coefficients provides an example of such a task. In that calculation, the partial contributions to the charge density from all one-particle wavefunctions must be computed and then accumulated. The 3D FFT used to compute one such contribution is too large to fit on a single node, yet too small to be distributed over many thousands of processors. We therefore adopt a mixed distribution in which nodes are arranged in a rectangular array, or *process grid*. Each column of the process grid hosts a set of electronic wavefunctions. Thus, 3D FFTs involve communication within columns of the process grid, whereas the accumulation of partial charge density contributions requires communication within rows of the process grid. This approach avoids, as much as possible, the use of global communications that involve all nodes. This arrangement contributes to the improvement of the scalability of the code.

Contexts and distributed objects

Defining subparts of a set of parallel tasks can be implemented using Message Passing Interface (MPI) communicators. However, some libraries, such as ScaLAPACK, use context abstractions to represent groups of parallel tasks. ScaLAPACK relies on the BLACS context structure to define the subset of nodes on which distributed matrices are defined. BLACS contexts are organized as 2D rectangular process grids, which facilitate communications used in parallel matrix operations. In Qbox, a `Context` class was defined to encapsulate the concept of a BLACS context. Each Qbox distributed object holds a reference to a `Context` object that effectively determines how the object is distributed. Communication within a `Context` is performed using the BLACS interface, implemented as member functions of the `Context` class. The `Context` class is an example of the façade design pattern [13]. In order to simplify the management of distributed objects, it was decided that the `Context` on which an object is defined must be chosen at the time the object is created and cannot change during the lifetime of the object. Similarly, a `Context` cannot change size or shape during its lifetime. Since an instance of `Context` can be shared by multiple objects, the issue of ownership of a `Context` arises. This is solved by implementing the `Context` class as a reference-counted object. The implementation follows Meyers [18], although it can be simplified because a `Context` is logically immutable after its creation, therefore removing the need for the copy-on-write mechanisms usually necessary for reference-counted objects. `Context` objects provide a flexible and powerful tool for the design of parallel codes by allowing one to tailor the part of a computer on which a particular task is performed. This concept is expected to remain an important ingredient in future versions of Qbox for large parallel platforms.

XML I/O interface

Qbox adheres to the quantum-simulation.org (QSO) standard for the representation of FPMD simulation data [19]. The QSO specification provides Extensible Markup Language (XML) Schema definitions [20] for the representation of a simulation sample and for the representation of atomic species. Qbox uses the Apache Xerces-C C++ library [21] to implement the various XML parsers needed to input simulation data. Using a parser library, such as Xerces-C, presents a number of advantages. First, Xerces-C is well supported and provides a large number of features that reflect the latest development in the XML standard. Second, Xerces-C parsers have the capability of validating XML documents during the parsing process. This feature is used in Qbox in order to provide an error-checking mechanism and ensure that input documents conform to the QSO XML Schema definitions. Third, the Xerces-C library provides efficient C++ implementations of the SAX (Simple API for XML) and DOM (Document Object Model) interfaces. This ensures faster execution of the parsing process than would be obtained with parsers written in interpreted languages. Finally, the Xerces-C parsers include various types of network accessors that allow Qbox to use Web-based XML documents by specifying their Uniform Resource Identifier (URI) instead of a filename.[3]

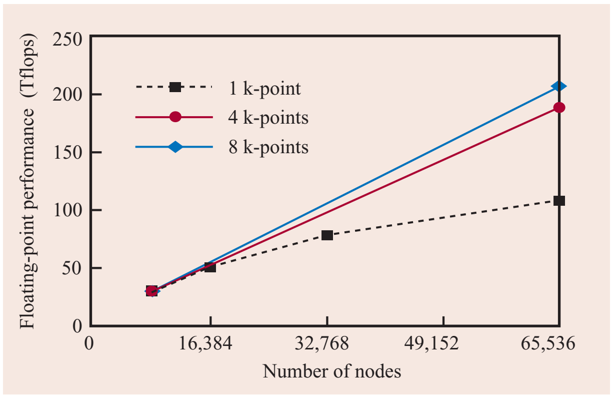
An XML document describing a sample contains all the information needed to instantiate a `Sample` object. The parsing process is implemented using the Xerces-C `SAX2XMLReader` class. A `StructuredDocumentHandler` class was defined in Qbox and derives from the SAX2 `DefaultHandler` class. It allows for a hierarchical invocation of specialized handlers for each component of `Sample`. Thus, the classes `SampleHandler`, `AtomSetHandler`, `SpeciesHandler`, and `WavefunctionHandler` are all derived from a `StructureHandler` abstract class and encapsulate the various phases of XML parsing needed to build the components of `Sample`.

The full set of methods needed to read a sample document is integrated into a `SampleReader` class, which can be reused in various postprocessing codes. For example, a visualization program that produces isosurface plots of electronic wavefunctions uses instances of the `Sample` and `SampleReader` classes. Keeping these classes independent from the rest of Qbox simplifies the development of postprocessing tools.

Keeping all the `Sample` information in a single XML document facilitates the management of simulation data, as there is no need to ensure consistency between separate files containing parts of the simulation data. However, this also implies that a sample document can become very large. Sample files for simulations that include hundreds of atoms can rapidly grow to sizes of 2 GB to 5 GB or more. This can lead to unacceptably long parsing times, since the parsers implemented in the Xerces-C library are inherently sequential. In order to alleviate this problem, we have developed a parallel XML parsing strategy for FPMD sample information. The approach is based on the observation that the bulk of the data describing an FPMD sample consists of the representation of the wavefunctions. It is possible to preprocess this information in order to reduce the size of the XML document, which can then be parsed efficiently using the Xerces-C library. The XML sample document is first read into memory and then preprocessed to produce a reduced XML document in a character buffer. The buffer is then parsed by the Xerces-C parser. The combination of parallel reading of XML documents and parallel preprocessing of XML data considerably reduces the time needed to load a sample file. For example, a sample document of 8.8 GB can be loaded and parsed in about 50 seconds on 128 nodes of a Blue Gene/L platform.

The I/O performance issue will likely become a significant problem for FPMD simulations on petascale computers. Parallelization of the I/O process may

[3]For example, the document *www.quantum-simulation.org/examples/species/hydrogen_pbe.xml* can be used by Qbox to define the species "hydrogen."



for position only

Figure 4

Qbox strong scaling results for 1,000 molybdenum atoms for various k-points sampling schemes. The eight-k-point calculation reaches a performance of 207 Tflops on 131,072 processors (65,536 nodes), or 57% of the peak performance.

alleviate the problem. It is, however, likely that new developments, such as application-specific compression algorithms, will be needed to remove that bottleneck.

Implementation and parallelization

The implementation of Qbox on the Blue Gene/L platform benefits from highly tuned libraries that were specially written to exploit the single-instruction, multiple-data (SIMD) unit of the IBM PowerPC* 440 processor. In particular, single-node, hand-coded matrix multiplication subroutines achieve over 95% of the peak performance [4]. These subroutines are included in Qbox at link time and allow ScaLAPACK to achieve near ideal performance in distributed matrix operations. Similarly, we use a specially developed version of the Fastest Fourier Transform in the West (FFTW) library [22, 23]. Libraries such as ScaLAPACK and BLACS are deployed on a Blue Gene/L machine in a straightforward manner. Other libraries, such as Xerces-C, require a modified build process in order to be cross-compiled on the front end of the Blue Gene/L system.

The Qbox parallelization strategy, and in particular the avoidance of large-volume global communication, is well suited to the torus network Blue Gene/L architecture. The organization of communication into separate stages along columns and rows of the process grid makes it possible to achieve good performance with a limited bisection bandwidth. The parallelization strategy consists of first distributing the electronic wavefunctions corresponding to different k-points (or boundary conditions) to separate groups of processors. A substantial part of the operations needed to update electronic wavefunctions are performed separately on different k-points and do not involve communication between these processor groups. Each processor group is further divided into a 2D rectangular process grid. Within a process grid, groups of electronic orbitals are assigned to each column while the Fourier coefficients of the orbitals are grouped by rows. This data layout leads to a minimum amount of communication between processors during the computation. For example, Fourier transforms of electronic orbitals involve communication with process grid columns only, whereas the calculation of the total electronic charge density involves communication within process grid rows and between processor groups only. Communication involving all processors simultaneously is avoided. This organization of communication is a major factor in the excellent scalability observed with Qbox on the Blue Gene/L platform. More details about the parallelization strategy are given in [24].

Performance on the Blue Gene/L supercomputer

The data distribution described above allows Qbox to scale well to very large numbers of processors. In particular, on the Blue Gene/L platform, simulations were run on up to 131,072 processors (65,536 nodes) at the Lawrence Livermore National Laboratory. In the largest problem solved using Qbox, the electronic structure of a sample of 1,000 molybdenum atoms was computed, including 12,000 electrons described by electronic wavefunctions at eight separate k-points. The scalability of Qbox for that application is shown in **Figure 4**. The floating-point performance achieved in that simulation was 207 Tflops, currently the highest recorded for a scientific application.

The high performance obtained on the Blue Gene/L machine can be attributed in part to the careful organization of interprocessor communications and to the availability of high-performance, hand-coded, single-node kernels for matrix operations. These kernels are described in more detail in [4].

Application examples

Because of the generality of density functional theory, Qbox can be applied to the simulation of a wide variety of physical systems. We mention a few applications briefly that illustrate that diversity. FPMD simulations are now routinely used to explore the properties of materials in extreme conditions, that is, conditions that are difficult to reproduce experimentally [25]. Recently, in a study of the high-pressure properties of carbon, Qbox was used to perform two-phase simulations [26, 27] that allow one to pinpoint the location of phase transitions in the pressure–temperature phase diagram [28]. FPMD simulations are also used increasingly to complement experiments performed on nanoparticles. In a recent investigation, Dal Negro et al. used Qbox to compute the electronic and optical properties of silicon nanocrystals [29]. Qbox has also been used extensively to explore the properties of liquid water [30, 31].

Conclusions and outlook

The architecture of Qbox has made it possible to use tens of thousands of processors to perform first-principles simulations. Such large-scale simulations are expected to greatly enhance our ability to understand matter on the atomic scale. Future implementations of FPMD on large-scale parallel computers will provide an even more powerful tool for detailed explorations of the properties of materials. New implementations will also have to face considerable challenges arising from the increasing complexity of future computer architectures. In particular, it is becoming apparent that future increases in computer power will come not from an increase in processor clock frequency, but rather from the multiplication of functional units, for example, in the form of multicore processors. Using such new architectures efficiently may prove challenging and will likely require substantial changes in the design of parallel FPMD codes in the future.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of The Open Group or Object Management Group, Inc., in the United States, other countries, or both.

References

1. R. Car and M. Parrinello, "Unified Approach for Molecular Dynamics and Density-Functional Theory," *Phys. Rev. Lett.* **55**, 2471–2474 (1985).
2. Qbox, Gygi Research Group, University of California, Davis; see *http://eslab.ucdavis.edu*.
3. N. R. Adiga, G. Almási, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, et al., "An Overview of the BlueGene/L Supercomputer," *Proceedings of the ACM/IEEE 15th Annual Supercomputing Conference* (SC2002), Baltimore, MD, 2002, p. 60.
4. F. Gygi, E. W. Draeger, M. Schulz, B. R. de Supinski, J. A. Gunnels, V. Austel, J. C. Sexton, et al., "Large-Scale Electronic Structure Calculations of High-Z Metals on the Blue Gene/L Platform," *Proceedings of the ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006; Gordon Bell Prize for Peak Performance; see *http://sc06.supercomputing.org/schedule/pdf/gb104.pdf*.
5. R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods*, Cambridge University Press, Cambridge, UK, 2004.
6. R. M. Dreizler and E. K. U. Gross, *Density Functional Theory*, Springer-Verlag, Berlin, Germany, 1990.
7. W. Kohn and L. J. Sham, "Self-Consistent Equations Including Exchange and Correlation Effects," *Phys. Rev.* **140**, No. 4A, A1133–A1138 (1965).
8. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, et al., *ScaLAPACK Users' Guide*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
9. The NetLib Repository; see *http://www.netlib.org/*.
10. J. Lakos, *Large-Scale C++ Software Design*, Addison-Wesley Professional, Boston, MA, 1996.
11. S. Meyers, *Effective C++: 50 Specific Ways to Improve Your Programs and Design*, Second Edition, Addison-Wesley Professional, Boston, MA, 1997.
12. F. Gygi, *Qbox User Guide*, V 1.30.1, November 2006; see *http://eslab.ucdavis.edu/software/Qbox/doc/QboxUserGuide.pdf*.
13. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, Boston, MA, 1995.
14. B. Stroustrup, *The C++ Programming Language*, Third Edition, Addison-Wesley Professional, Boston, MA, 1997.
15. M. Fowler and K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Second Edition, Addison-Wesley Professional, Boston, MA, 1999.
16. D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Second Edition, Academic Press, San Diego, CA, 2001.
17. D. G. Anderson, "Iterative Procedures for Nonlinear Integral Equations," *J. ACM* **12**, No. 4, 547–560 (1965).
18. S. Meyers, *More Effective C++: 35 New Ways to Improve Your Programs and Designs*, Addison-Wesley Professional, Boston, MA, 1996.
19. Web standards for first-principles simulations; see *http://www.quantum-simulation.org*.
20. *XML Schema Part 0: Primer Second Edition*, D. C. Fallside and P. Walmsley, Eds., W3C Recommendation, October 28, 2004; see *http://www.w3.org/TR/xmlschema-0/*.
21. The Apache XML Project, Xerces-C++, Version 2.7.0; see *http://xml.apache.org/xerces-c/*.
22. M. Frigo and S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Seattle, WA, 1998; see *http://www.fftw.org/fftw-paper-icassp.pdf*.
23. S. Kral, FFTW-GEL; see *http://www.complang.tuwien.ac.at/skral/fftwgel.html*.
24. F. Gygi, "Large-Scale First-Principles Molecular Dynamics: Moving from Terascale to Petascale Computing," *J. Phys. Conference Series* **46**, 268–277 (2006).
25. F. Gygi and G. Galli, "*Ab Initio* Simulation in Extreme Conditions," *Materials Today* **8**, No. 11, 26–32 (2005).
26. T. Ogitsu, E. Schwegler, F. Gygi, and G. Galli, "Melting of Lithium Hydride Under Pressure," *Phys. Rev. Lett.* **91**, No. 17, 175502–175506 (2003).
27. A. B. Belonoshko and L. S. Dubrovinsky, "Molecular Dynamics of NaCl (B1 and B2) and MgO (B1) Melting: Two-Phase Simulation," *American Mineralogist* **81**, No. 3/4, 303–316 (1996).
28. A. A. Correa, S. Bonev, and G. Galli, "Carbon under Extreme Conditions: Phase Boundaries and Electronic Properties from First-Principles Theory," *Proc. Natl. Acad. Sci.* **103**, No. 5, 1204–1208 (2006).
29. L. Dal Negro, S. Hamel, N. Zaitseva, J. H. Yi, A. Williamson, M. Stolfi, J. Michel, G. Galli, and L. C. Kimerling, "Synthesis, Characterization, and Modeling of Nitrogen-Passivated Colloidal and Thin Film Silicon Nanocrystals," *IEEE J. Selected Topics Quantum Electr.* **12**, No. 6, 1151–1163 (2006).
30. E. Schwegler, J. C. Grossman, F. Gygi, and G. Galli, "Towards an Assessment of the Accuracy of Density Functional Theory for First Principles Simulations of Water. II," *J. Chem. Phys.* **121**, No. 11, 5400–5409 (2004).
31. M. Allesch, E. Schwegler, F. Gygi, and G. Galli, "A First Principles Simulation of Rigid Water," *J. Chem. Phys.* **120**, No. 11, 5192–5198 (2004).

Received March 23, 2007; accepted for publication May 25, 2007

François Gygi *University of California at Davis, Department of Applied Science, 3013 Engineering III, Davis, CA 95626 (fgygi@ucdavis.edu)*. Dr. Gygi is Professor of Applied Science at the University of California at Davis and is the architect of the Qbox code. He holds a Ph.D. degree in physics from the Ecole Polytechnique Fédérale, Lausanne, Switzerland, and he pursued his research career at AT&T Bell Laboratories, IBM Zurich Research Laboratory, and the Lawrence Livermore National Laboratory. Dr. Gygi is the recipient of the 2006 ACM/IEEE Gordon Bell Award for Peak Performance for Qbox simulations on the Blue Gene/L computer.