

Meeting the Challenges of High-Dimensional Quantum Dynamics

Draw McCormack

Theoretical Chemistry, Free University, Amsterdam
 Web: <http://www.maniacalextext.com>
 Email: da.mccormack@few.vu.nl

Software Crisis?

The traditional, every-man-for-himself approach to writing scientific software will eventually lead to a stagnation of progress, just as it did in the business world two decades ago. While tens of thousands of lines of unstructured Fortran code may be comprehensible, as algorithms develop in complexity, and programs stretch to hundreds of thousands of lines, more attention will need to be paid to the software development process if further headway is to be made.

Scientific software is typically written with little attention paid to design. For small programs, this is often the fastest route to results, but as software grows over time, it can make it difficult for programmers to find their way around. Bad programming can lead to dependencies that traverse the entire code base, making it more and more difficult to maintain and extend. You may already be acquainted with the problem: have you ever experienced a sinking feeling when asking yourself the question "Will making a change here cause problems elsewhere in the program?"

Other symptoms of the problem are that there is relatively little sharing of scientific code (i.e. software reuse), and that scientific codes tend to be extremely specialized. Most codes implement a single method; if a different method is required, usually a new program is written. This can be extreme: often a different program will exist for each system studied. It is not unusual to have one program for solving the time-dependent (TD) Schrodinger equation for a two-dimensional (2D) system, and a completely different program for a 4D system.

Solutions

Solutions to these problems are known. They were developed by computer scientists when the business world suffered its software crisis in the 1980s. Techniques such as object-oriented programming and generic programming are now well established, and have proven themselves. They are even starting to penetrate scientific fields. For example, the latest version of Fortran, Fortran 2003, supports object orientation.

Object oriented programming (OOP) can be used to improve high-level structure. Programs are broken down into largely independent pieces, interacting via well-defined interfaces. Writing a large OO program is similar to writing lots of small programs, because the implementation of each part can be changed without fear of breaking the whole package.

Because OO software tends to be better organized, there are more opportunities for code reuse. Instead of writing a completely new program, a new method can be added to an existing program, leveraging the existing code base, without requiring wholesale changes throughout. Extending an OO program often only requires a few lines of existing code to be changed.

Periphery

'Periphery' is a C++ framework for Quantum Dynamics. It has been written in a modern style, incorporating up-to-date programming techniques and libraries. OOP is used for high-level organization, and generic programming for low-level, performance-critical code.

The objective of Periphery is to provide a reusable platform for quickly developing and testing new Quantum Dynamics methods. The similarities between computational methods are often far more numerous than the differences. Even unrelated methods will be implemented using the same representations (i.e. bases and grids), and practically all Quantum codes make extensive use of the mathematics of Linear Algebra. It is the intention that these aspects of method development be provided by Periphery, in a powerful, but easy to use, framework.

Software Engineering

Most scientific software development pays little heed to the lessons learned in the field of Software Engineering, but the techniques it offers can result in more extendible and maintainable code, which ultimately saves time. The process used to develop Periphery draws on these techniques:

Object-Oriented Design

The chances of success in any endeavor are generally improved by planning and design. Software development is no different.

Unit Testing

Just because a program compiles does not mean it works. Unit tests test code at a low level, improving robustness. They also act as a form of documentation, showing how certain constructions are used.

Tracing

A Logging/Tracing framework has been used throughout Periphery. This can be used to follow program flow. Different levels of logging are available, from basic program flow, to detailed debugging output.

Source Control Management (SCM)

Source Control Management systems like CVS and Subversion allow multiple developers to work on a single body of code, and maintain a history of the code. Developers can return to any version of any file, and determine what changes have been made. Periphery uses the Subversion SCM system (subversion.tigris.org).

Integrated Development Environment (IDE)

Periphery is developed in an IDE, which eases code navigation and searching, compilation, and debugging.

Advanced Programming

Contemporary C++ offers many features that not only facilitate constructions not possible with other programming languages, but can also be used to simplify algorithm development by hiding complexity. Periphery makes full use of these aspects of the language and its libraries.

For example, automatic garbage collection is utilized throughout using a technique known as 'reference counting'. This functionality, which comes from a widely-used C++ package called Boost (www.boost.org), greatly simplifies memory management. Another example is the Standard Template Library (STL) that is included in all C++ distributions. This package includes convenient data containers such as dynamic arrays and dictionaries, as well as algorithms for sorting, for example.

A primary concern of many scientific developers is performance. C++ provides for generic programming (through the 'template' construct), which allows high performance code to be written, without sacrificing expressiveness. Generic programming allows code to be written for generic types; the types used are determined at compile time, allowing the compiler to perform aggressive optimizations.

A simple example is that of a wavefunction type. A Fortran program may include separate, but almost identical, code for complex and real wavefunctions. In C++, code dealing with the wavefunction could be written generically, and the type (i.e. complex or real) substituted automatically by the compiler. Effectively, the compiler does the duplication, reducing the amount of source code, and the burden of making future changes in two places in the program.

Periphery also provides facilities for simplifying I/O. A simple interface is included to the Hierarchical Data Format (HDF5) library, which allows binary data to be stored in a hierarchical file that internally resembles a file system. High-level archiving is also provided, which allows the developer to store complex data structures on file with a single function call, and retrieve them just as easily.

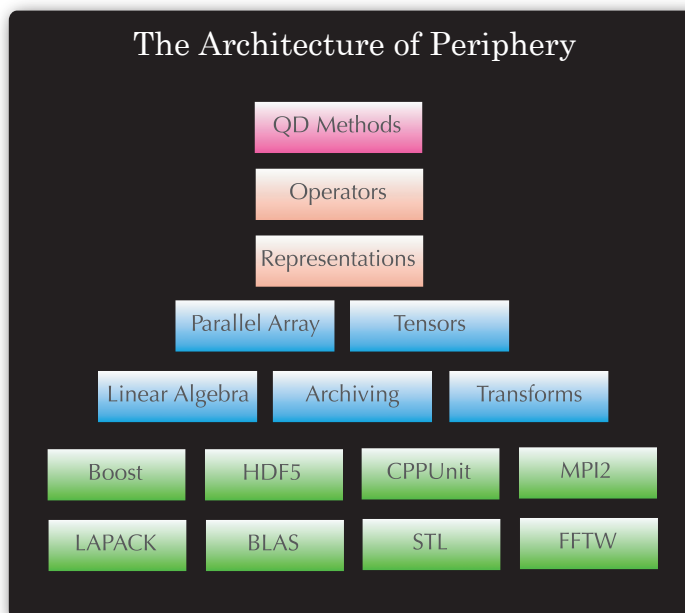
Under Development

Periphery is already being used to develop and test new Quantum Dynamics methods. Both time-dependent and time-independent methods have been implemented.

A current line of research centers on dynamic culling of basis functions during wavepacket propagations. By generating wavelet basis sets, which are localized in phase space, only 'relevant' basis functions need be included in the propagation at each point in time. Simultaneous Diagonalization (SD) is used to generate the wavelet basis from a conventional basis or grid.

Early tests indicate that the number of basis functions needed may scale relatively favorably with system dimensionality. For example, for reactive scattering of hydrogen molecules on a Pt surface, the number of wavelets required scaled approximately as 10 to the n, where n is the dimensionality. A wavepacket method utilizing a static, direct product grid scaled as 30 to the n, and a dynamically-culled DVR grid scaled as 20 to the n. The Dynamically-Culled Wavelets (DCW) method may provide an option for performing calculations on high-dimensional quantum systems with strong correlations, for which methods like MCTDH are known to become less efficient.

Relevant References: B. Poirier and A. Salam, JCP 121 (2004) 1690; R. Daves and T. Carrington, JCP 122 (2005) 134101-1.



Is C++ Too Hard?

The following source code applies an exponential operator to a multidimensional wavefunction. It is equivalent to many, many lines of Fortran code, with embedded loops and complicated data structures. The code presented here is quite generic, in that it can be used with a wavepacket of any dimensionality.

```

void MultiDimWaveletWavepacketSolver::performExponentialKineticOperation(
  SplitOperatorPropagator const & const propagator,
  const double &timeStep,
  const complex<double> &exponent ) {
  for ( size_t dim = 0; dim < representations_.size(); ++dim ) {
    // Determine kinetic operator matrix
    RealMatrix kinMatrix = representations_[dim].secondDerivMatrix();
    kinMatrix *= -0.5 * hbar * hbar / masses[dim];

    // Transform to diagonal coords, so that exponential operator can be evaluated
    RealMatrix evecs;
    RealVector evals;
    Diagonalize(kinMatrix, evals, evecs);
    ComplexVector diagExpElements = exp( exponent * evals );
    ComplexMatrix expKinMatrix(kinMatrix.dimensions());
    expKinMatrix = ComplexZero;
    SetDiagonalElements(expKinMatrix, diagExpElements);

    // Transform the exponential kinetic operator to the original representation
    expKinMatrix = MatrixMultiply( evecs, MatrixMultiply( expKinMatrix,
      Transpose(evecs) ) );

    // Apply it to wavefunction
    ComplexMatrixTrans kinTrans(expKinMatrix);
    waveFunction_ = kinTrans.forwardTransform( waveFunction_, dim );
  }
}
  
```