QMCPACK

1.0

Generated by Doxygen 1.8.1

Mon Jul 23 2012 05:55:10

Contents

1 QMCPACK Documentation **1**

1.1 Introduction . . . 1

1.2 Major changes . . . 2

2 Getting started **3**

2.1 Required tools and libraries . . . 3

2.1.1 External packages . . . 4

2.2 Advanced: cmake . . . 4

2.2.1 Short introduction to cmake . . . 4

2.2.2 Customizing a cmake toolchain file . . . 5

3 QMCPACK I/O **7**

3.1 Main input file . . . 7

3.1.1 About XML in QMCPACK . . . 8

3.1.2 Reserved attributes and their implicit relationships . . . 9

3.2 Generic XML elements . . . 9

3.2.1 parameter . . . 9

3.2.2 include . . . 10

3.2.3 attrib . . . 10

3.3 simulation: Root element . . . 10

3.4 Defining physical entities . . . 11

3.4.1 qmcsystem . . . 11

3.4.2 simulationcell . . . 12

3.4.3 particleset . . . 14

3.4.3.1 group . . . 15

3.4.4 hamiltonian . . . 16

3.4.4.1 pairpot . . . 17

3.4.5 wavefunction . . . 17

4 Interface Utilities **19**

4.1 Quantum espresso . . . 19

4.1.1 Generating wavefunction files from PWSCF . . . 20

4.1.2 Using the converted files in QMCPACK . . . 20

4.1.3 Molecular calculations using PW codes . . . 21

4.2 Gamess . . . 21

4.3 wfconvert package . . . 21

5 QMC School 2012 **23**

5.1 Basics on Forge at NCSA . . . 23

5.2 Setups for QMCPACK . . . 23

6 Using PWSCF and QMCPACK to perform total energy calculations of condensed systems **25**

6.1 Introduction . . . 25

6.2 Initial DFT Calculations . . . 25

6.2.1 Pseudopotentials . . . 26

6.3 Creating and testing wavefunctions for QMCPACK . . . 26

6.3.1 Extracting Wavefunctions at Particular k-points . . . 27

6.4 Optimizing Jastrow Factors within QMC . . . 27

6.4.1 Testing Convergence of the Splines in QMCPACK . . . 31

6.5 Performing DMC Calculations . . . 32

6.5.1 Two-Body Finite Size Effects . . . 35

6.5.1.1 Creating and Using Supercells . . . 35

6.5.1.2 Two-Body Finite Size Corrections . . . 36

6.5.2 Putting it all Together . . . 37

6.6 Conclusion . . . 37

6.7 Acknowledgment . . . 37

7 Guide for the developers **39**

7.1 About QMCPACK . . . 39

7.2 Documentation . . . 39

7.3 How to add a page to this document . . . 40

8 Construction of a many-body wavefunction **41**

8.1 wavefunction . . . 41

9 Frequently Asked Questions **43**

9.1 General questions . . . 43

9.1.1 Compilers . . . 43

9.1.2 Parallelization in QMCPACK . . . 43
9.1.3 Numerical libraries . . . 43
9.2 Running QMCPACK . . . 44
9.2.1 Using multiple threads . . . 44
9.2.2 How to run QMCPACK with MPI . . . 44
9.3 Build problems . . . 44
9.3.1 Failed to link blas/lapack . . . 44
9.3.2 Failed to link MKL . . . 44
9.3.3 Failed to find XYZ package . . . 45

10 University of Illinois/NCSA Open Source License **47**

11 File Index **49**
11.1 File List . . . 49

12 File Documentation **51**
12.1 developers.html File Reference . . . 51
12.2 faq.html File Reference . . . 51
12.3 forge.html File Reference . . . 51
12.4 gettingstarted.html File Reference . . . 51
12.5 inputxml.html File Reference . . . 51
12.6 license.html File Reference . . . 51
12.7 othertools.html File Reference . . . 51
12.8 qmcpack_main.html File Reference . . . 51
12.9 solids_cecam2012.html File Reference . . . 51
12.10 wavefunctionbuilder.html File Reference . . . 51

Chapter 1

QMCPACK Documentation

1.1 Introduction

QMCPACK, an open-source QMC simulation code written in C++, implements advanced QMC algorithms for large-scale parallel computers. Designed with the modularity afforded by object-oriented architecture, it makes extensive use of template metaprogramming to achieve high computational efficiency through inlined specializations. It utilizes a hybrid OpenMP/MPI approach to parallelization to take advantage of the growing number of cores per SMP node. Finally, it utilizes standard file formats for input and output in XML and HDF5 to facilitate data exchange.

Main contributing authors are Jeongnim Kim and the members of the electron structure group of Profs. Ceperley and Martin at University of Illinois.

Special thanks to these developers who are responsible for core capabilities:

- K. Esler: einspline, CUDA port, improved numerical algorithms, tools
- J. McMinis: optimization and advanced QMC drivers
- M. Morales: implementation of advanced wavefunctions
- L. Shulenburger: tools

Different parts of the code are written by different people, whose names are usually signed at the top of each file. If you are interested in participating in QMCPACK development, email to Jeongnim Kim. QMCPACK is available under University of Illinois/NCSA Open Source License (OSI approved)

QMCPACK development at the Materials Computation Center was supported by the National Science Foundation under grant no. DMR-03 25939 ITR, with additional support through the Frederick Seitz Materials Research Laboratory (U.S. Dept. of Energy grant no. DEFG02-91ER45439) at the University of Illinois Urbana-Champaign and National Center for Supercomputing Applications.

Currently, the principal authors are funded by

- PetaApps, supported by the U. S. National Science Foundation
- ORNL LDRD

We acknowledge OLCF and ALCF for help and support in accessing their resources through INCITE program supported by US Department of Energy and NCSA, TACC and NICS for providing resources through NSF TeraGrid allocations.

1.2 Major changes

2012-06-01

- Energy/Linear optimization, Umrigar et al, PRL, 2007
- Efficient multi determinant updates, Clark et al, JCP 2011
- Supports for backflow
- Core functionalities ported on CUDA
- Bug fixes
- Tutorials are added

2008-07-22

- Numerous bug fixes.
- Support TrialWaveFunction cloning for multi-threaded applications
- EinsplineOrbitalSet uses Einspline library
- BsplinFunctor for One- and Two-Body Jastrow

2005-07-25

- updated doxygen documentations
- docs directory is added to cvs repository.
- To generate doxygen documentation on a local host,
  - cd docs; doxygen Doxyfile
  - doxygen/html/index.html is the main page
- Introduced groups that follow the directory structure
  - Orbital group
  - Orbital builder group
  - Many-body wave function group
  - Hamiltonian group
  - QMC Driver group
  - QMC Drivers using walker-by-walker update
  - QMC Drivers using particle-by-particle update
  - QMC Drivers for energy differences
  - QMC Application group

Chapter 2

Getting started

- Download source via svn
  - Developers' version:

    ```
    svn co https://subversion.assembla.com/svn/qmcdev/trunk qmcpack
    ```

  - Stable public version :

    ```
    svn co http://qmcpack.googlecode.com/svn/trunk/ qmcpack
    ```

- Build where everything is installed in standard directores, /usr, /usr/local Throughout this guide, we will use QMCPACK_HOME to denote the QMCPACK's top directory.

```
cd qmcpack/build
cmake ..
make -j4
```

Use any number of threads for parallel make. The example above used 4 threads with -j4 option. More on how to build QMCPACK can be found at wiki. If everything goes well, then you should see *qmcpack/build/bin/qmcapp*.

To run QMCPACK using 8 threads:

```
export OMP_NUM_THREADS=8
QMCPACK_HOME/build/bin/qmcapp input.xml
```

Here, input.xml is the main input file, a valid XML file. Tutorials have a number of common use cases of QMCPACK.

2.1 Required tools and libraries

- C/C++ compilers
- cmake, build utility, http://www.cmake.org/
- blas/lapack, numerical library, use platform-optimized libraries

On any platform, vendor provided numerical libraries should be used. Other factor to consider in selecting Blas/Lapack is the compilers.

2.1.1 External packages

In order to install QMCPACK, users have to install several required packages. These packages are included in standard Linux/cygwin distributions or can be downloaded by following the links. Installing these libraries with the source codes is straightforward. Because the header files are included by QMCPACK, it is important to install developers' version of each library. If these libraries are installed in standard directories, /usr /usr/local and /sw (Mac), no action is necessary. Alternatively, environment variables XYZ_HOME should be set. Here, XYZ stands for the name of package; the build utility can locate the libraries and use them.

With few exceptions, the build utility cmake will look for XYZ_HOME/include for the header files and XYZ_HOME/lib for the library files. When multiple environment variables apply to a library, e.g., blas/lapack, the library is searched according to the listed order.

External package	category	Env. variable	URL
hdf5	I/O	HDF5_HOME	http://www.hdfgroup.org/HDF5/
libxml2	I/O	LIBXML2_HOME	http://xmlsoft.org/
boost	C++ standard library	BOOST_HOME	http://www.boost.org
fftw	FFT library	FFTW_HOME	http://www.fftw.org

2.2 Advanced: cmake

2.2.1 Short introduction to cmake

cmake is a portable build system and is widely used for large-scale software development projects such as VTK. How *cmake* works is similar to *autoconfig/automake*. *cmake* uses *CMakeLists.txt* files (*cmake* scripts) to generate *Makefiles* with complete dependency analysis. *CMakeLists.txt* are equivalent to *Makefile.am*.

QMCPACK_HOME contains

```
CMakeLists.txt
src
  CMakeLists.txt
  QMCWaveFunctions/CMakeLists.txt
  QMCHamiltonians/CMakeLists.txt
  QMCApp/CMakeLists.txt
  ...
```

Always use *out-of-source* compilation with *cmake*, i.e., never build QMCPACK in QMCPACK_HOME. We can further separate the source (development) and build. One can build multiple executables in different locations by creating new directories and build QMCPACK in each directory.

```
/home/foo/build/gcc-real
/home/foo/build/gcc-complex
/home/foo/build/mpi-real
```

In each directory, e.g., */home/foo/build/gcc-real* (after secodeing the environments properly), execute

```
cd /home/foo/build/gcc-real
cmake /home/foo/src/qmcpack
make
```

There is no need to change sources or cmake files. If something goes wrong, simply remove the directory and start over.

2.2.2 Customizing a cmake toolchain file

Setting environment variables and modifying cmake files can be tedious and often lead to failure at the cmake (configure) stage. One can set up a script to set all the tool chains which will be noted as a `toolchain file` Using a toolchain file is particularly useful on the HPC systems that have different login and compute nodes and require cross compiling. Consult wiki

config directory has specialized toolchain files for the HPC systems the developers routinely use. They are regularly updated with the system changes. Users can edit a script for their environments. Note that where the toolchain file reside is not important. You can copy a reference toolchain file in the working directory and change the local copy as needed.

```
cd QMCPACK_HOME/build
cp ../config/TitanGNU.cmake .
cmake -DCMAKE_TOOLCHAIN_FILE=TitanGNU.cmake ..
cmake -DCMAKE_TOOLCHAIN_FILE=TitanGNU.cmake ..
make
```

Note that *cmake* is excecuted twice to ensure cmake variables are properly cached and updated.

Here, we discuss how to prepare a toolchain file for your environment using *config/TitanGNU.cmake* as an example. TitanGNU is a deca-petaflop system at ORNL running a varient of LINUX. As the name implies, GNU compilers and GNU programming environment is used and the same compiler options are tailored to the AMD processors. `CMake wiki` lists the cmake system variables with CMAKE_ prefix, such as *CMAKE_CXX_COPMILER* for C++ compiler.

- Setting system properties: only static libraries can be used for the compute nodes.

```
SET(CMAKE_SYSTEM_PROCESSOR "XT5")
SET_PROPERTY(GLOBAL PROPERTY TARGET_SUPPORTS_SHARED_LIBS FALSE)
```

- Setting compilers and compiler options

```
#set cmake variables for the compilers
set(CMAKE_C_COMPILER  /opt/cray/xt-asyncpe/5.05/bin/cc)
set(CMAKE_CXX_COMPILER  /opt/cray/xt-asyncpe/5.05/bin/CC)

#temporary variables for GNU compilers
set(GNU_OPTS "-DADD_ -DINLINE_ALL=inline")
set(GNU_FLAGS "-fopenmp -O3 -Drestrict=__restrict__ -finline-limit=1000
       -fstrict-aliasing -funroll-all-loops -Wno-deprecated ")
set(XT_FLAGS "-march=bdver1 -msse3 -D_CRAYMPI")

#set C/C++ compiler flags
set(CMAKE_CXX_FLAGS "${XT_FLAGS} ${GNU_FLAGS} -ftemplate-depth-60 ${GNU_OPTS}")
set(CMAKE_C_FLAGS "${XT_FLAGS} ${GNU_FLAGS} -std=c99")
```

- Setting search locations for external libraries

```
set(CMAKE_FIND_ROOT_PATH
  /opt/cray/hdf5/1.8.6/gnu/46
  /opt/fftw/3.3.0.0/interlagos
  /sw/xk6/boost/1.44.0/cle4.0_gnu4.5.3
  /ccs/proj/mat034/jnkim/xk6/gnu45/libxml2
  /opt/fftw/3.2.1
)
```

- Setting configuration properties: MPI, OpenMP, etc. *src/config.h* is created in *build* directory based on these properties.

```
SET(ENABLE_OPENMP 1) #enable OpenMP
SET(HAVE_MPI 1)      #have MPI

# SSE-related
SET(HAVE_SSE 1)
```

```
SET(HAVE_SSE2 1)
SET(HAVE_SSE3 1)
SET(HAVE_SSSE3 1)
SET(USE_PREFETCH 1)
SET(PREFETCH_AHEAD 12)
```

- Special settings for Cray XT: All these variables are set to prevent dynamic linking. Not needed on normal ∗NIX systems.

```
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_SHARED_LINKER_FLAGS "")

FOREACH(type SHARED_LIBRARY SHARED_MODULE EXE)
  SET(CMAKE_${type}_LINK_STATIC_C_FLAGS "-Wl,-Bstatic")
  SET(CMAKE_${type}_LINK_DYNAMIC_C_FLAGS "-static")
  SET(CMAKE_${type}_LINK_STATIC_CXX_FLAGS "-Wl,-Bstatic")
  SET(CMAKE_${type}_LINK_DYNAMIC_CXX_FLAGS "-static")
ENDFOREACH(type)
```

Chapter 3

QMCPACK I/O

3.1 Main input file

The input file for *qmcapp* (main application of QMCPACK) is a standard XML file. It handles essential parameters which have to be modified by the users and light-weight data to define the physical problem as clearly as possible. The large data associated with single-particle orbitals are usually generated by tools and stored in HDF5 file for the portability and efficiency.

```
<?xml version="1.0"?>
<simulation>
  <project id="TITLE" series="0"/>
  <include href="PTCLXML"/>
  <include href="WFSXML"/>
  <include href="HAMXML"/>
  <loop max="5">
    <qmc method="cslinear" move="pbyp" checkpoint="-1" >
      <cost name="energy">                0.05 </cost>
      <cost name="unreweightedvariance">0.00 </cost>
      <cost name="reweightedvariance">    0.95 </cost>
    </qmc>
  </loop>
  <qmc method="vmc" move="pbyp">
    <parameter name="timestep">1</parameter>
    <parameter name="samples">100000</parameter>
  </qmc>
  <qmc method="dmc" move="pbyp">
    <parameter name="timestep">0.02</parameter>
  </qmc>
</simulation>
```

(a) problem descriptions

(b) optimization

(c) vmc

(d) dmc

Figure 3.1: A main input xml file

The input file above has the main sections

- Project name for bookkeeping
- Description of the physical system
  - Electronic and/or ionic systems
  - Trial wavefunction
  - Hamiltonian

- QMC execution
  - optimization: loop is used to repeate VMC/optimization
  - vmc
  - dmc

Note that problem-dependent elements are defined in external xml files, e.g., ptcl.xml, and are included via *include* element.

3.1.1 About XML in QMCPACK

The hierarchical nature of XML is ideally suited to the object-oriented design of QMCPACK. Throughout this document, we will use a UML-based graphical notation to represent a XML node and its relationship with other XML nodes as shown here.

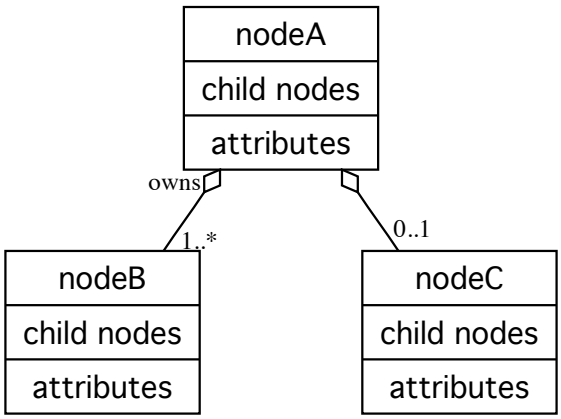

Figure 3.2: Graphical notations for a XML node.

The opendiamond denotes the owndership: nodeA owns nodeB and nodeC. The multiplicity of child nodes of one kind is specificed at the child-node end.

1 required

0..1 optional, maximum one

0..* optional, any number of nodes

1..* required, any number of nodes

QMCPACK uses DOM parser of libxml2 library. The entire input XML file is parsed and stored in a tree which is processed recursively. Parsing a XML file is similar to running a shell script. Each XML element (node) is mapped to a factory or builder function to instantiate the main computational objects or to execute a class member function to initialize of the object which owns the node. The order of XML nodes is critical to instantiate objects with proper ownerships and roles. All these features are exploited by the developers at the design stage of a particular class and algorithm in QMCPACK.

XML files can be manipulated by standard editors and tools including web browsers. New features can be easily added without breaking the logical structure of the existing input files in XML. QMCPACK requires a validating XML but does not enforce correctness with a schema. This is mainly because QMCPACK is evolving quite rapidly and the input file is subject to change when new capabilities are added. A schema for the tested and accepted features is presented in this document and will be updated so that the new features can be utilized by the users as soon as they become available.

3.1.2 Reserved attributes and their implicit relationships

These attributes are reserved to define the relationship between XML nodes.

id "unique" ID in a given XML file

name the name of an object. If *id* is not provided, *name* is used as the ID of the object.

ref refererence to an existing object

href external file name (relative path or full path)

type the object or engine to be used at the run time

target the name (ID) of the quantum *particleset* (e.g., electrons), or *wavefunction*

source the name (ID) of the source *particleset*

A XML element has a well-defined scope. Child nodes inherit the property of its parent node. When a XML node is referred by *ref*, *source* and *target*, the named XML node has to be defined before using it.

3.2 Generic XML elements

These elements are used by other nodes

parameter set the value of a named property

include include an external file

attrib set the value of an array

3.2.1 parameter

This is a genric way to define a property of a XML element and contains

```
parameter =
  children  : text for the value
  attribute : name, (units,condition)
```

For example, *parameter* sets the VMC variables:

```
<qmc method="vmc">
  <parameter name="timestep">0.5</parameter>
  <parameter name="blocks">10</parameter>
  <parameter name="steps">10</parameter>
  <parameter name="warmupsteps">10</parameter>
</qmc>
```

Remarks

- *parameter/@name* is the key to map a named property. The allowed *parameter* set and their usage are determined by their parent element.
- Attribute *units* and *condition* are reserved.

Warning

The type checking is handled by C++ *static_cast*<*T*> and does not report nor abort the execution upon failure.

3.2.2 include

include is used to include an external file *include/@href*.

```
include = attribute : href
```

An example below includes three external files. Typically, conversion tools generate ptcl.xml and wfs.xml. The actual names may differ.

```
<simulation>
  <include href="ptcl.xml"> <!-- define  ions and electrons -->
  <include href="wfs.xml"> <!-- trial wavefunction -->
  <include href="ham.xml">  <!-- hamiltonian -->
  <!-- now do QMC -->

</simulation>
```

The included file has to be valid XML with *qmcsystem* as its root element. The order of *include* is strict and only one level of include is allowed, i.e., the external files cannot have any *include* element.

3.2.3 attrib

This is a genric way to define an attribute of a particle set, such as positions.

```
attrib =
  children  : text for the value
  attribute : name, datatype, size
```

Remarks

- *attrib/@size* determines the size of the array
- the allowed *attrib/@datatype*
  - intArray : integers
  - realArray : reals
  - posArray : D-dim vectors
  - stringArray : strings

3.3 simulation: Root element

simulation is the root element of the main input file and contains everything about a QMC run.

```
simulation =
  children : project,
             (qmcsystem,particleset,wavefunction,hamiltonian,include)*,
             init?,
             loop?,
             qmc+
```

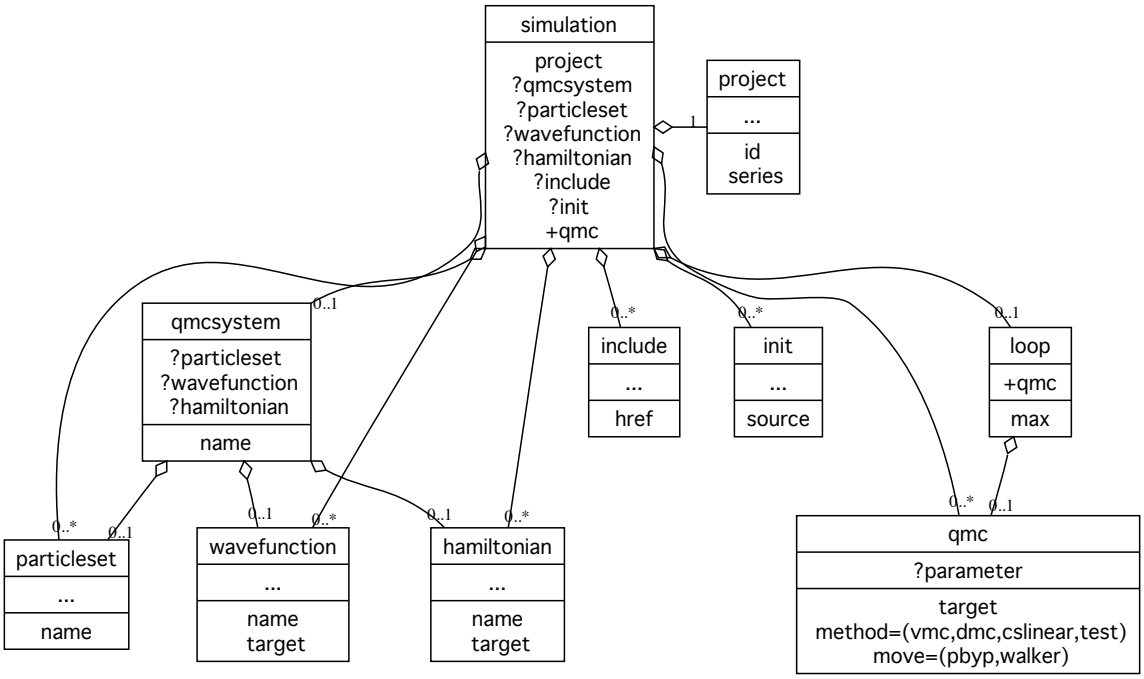

Figure 3.3: simulation root element

3.4 Defining physical entities

3.4.1 qmcsystem

A *qmcsystem* defines a system for a QMC simulation and contains

```
qmcsystem =
  children : (supercell,particleset,wavefunction,hamiltonian)*
```

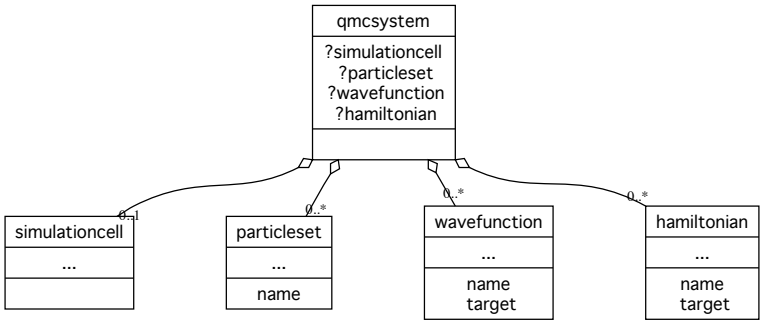

Figure 3.4: *qmcsystem* element

Remarks

- *qmcsystem* in the main input xml is not necessary but makes the input file more readable by grouping a number of elements to define a physical problem.
- It defines a local scope. For example, *supercell* applies to all the *particleset* in a *qmcsystem*.

qmcsystem is also the root of an external XML included by the main XML file by *include/@href* as shown in this wavefunction xml file.

```
<?xml version="1.0"?>
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="bspline" href="../lda.pwscf.h5" sort="1" tilematrix="
      3 0 0 0 3 0 0 0 1"
      twistnum="0" source="ion0" version="0.10" gpu="yes">
      <slaterdeterminant>
        <determinant id="updet" size="72">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
        <determinant id="downdet" size="72">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
      </slaterdeterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation speciesA="u" speciesB="u" size="8">
        <coefficients id="uu" type="Array"> 0.4733624685 0.3211249039 0.2092322
      955 0.1309323998 0.07945263393 0.04610358353 0.02334017812 0.009377732289</
      coefficients>
      </correlation>
      <correlation speciesA="u" speciesB="d" size="8">
        <coefficients id="ud" type="Array"> 0.6689003572 0.403606454 0.24007859
      85 0.1422242647 0.08386189425 0.04809711252 0.02406774346 0.01030826968</
      coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" source="ion0" print="
      yes">
      <correlation elementType="C" size="8">
        <coefficients id="eC" type="Array"> -0.8180912636 -0.6861867666 -0.5252
      400797 -0.3433298209 -0.2120687475 -0.119929026 -0.0528524441 -0.01515197638</
      coefficients>
      </correlation>
    </jastrow>
  </wavefunction>
</qmcsystem>
```

3.4.2 simulationcell

A *simulationcell* defines the supercell of a QMC simulation and contains

```
simulationcell = children : parameter+
```

As the name implies, *simulationcell* is only needed for the systems with periodic boundary conditions. Valid *parameter* elements of *simulation* are

name	datatype	default	definition
lattice	tensor (DxD)	$a(i,j) = \delta_{ij} 10^{10}$	simulation cell, $a(i,j)$ denotes i-th vector and the j-th component using C ordering

scale	real	1.0	scaling factor to lattice; multiplied to *lattice*
bconds	vector (D)	p p p	Required For each lattice vector. p for Periodic and n for non-periodic
LR_dim_cutoff	real	10	Used for the optimized breakup method for long-range potentials and Jastrow functions
rs	real	1	HEG density used to set the supercell for a homegenous electron gas.

The example below sets the supercell for the *particleset/@name='ion0'* and particleset/@name='e'. The details of the electronic and ionic systems are omitted.

```
<qmcsystem>
  <simulationcell name="global">
    <parameter name="lattice" units="bohr">
    20.521733294552881    0.000000000000000    0.000000000000000
     0.000000000000000   20.521733294552881    0.000000000000000
     0.000000000000000    0.000000000000000   20.521733294552881
    </parameter>
    <parameter name="bconds">p p p</parameter>
    <parameter name="LR_dim_cutoff">15</parameter>
  </simulationcell>
  <particleset name="ion0"/>
  <particleset name="e"/>
</qmcsystem>
```

Remarks

- When *simulationcell* is not provided, open boundary conditions are assumed. This is the default mode for the molecular systems. The wavefunction conversion tools for Quantum Chemistry codes using Gaussian-type orbitals do not insert *simulationcell* element.
- Specialized methods to evaluate the distance relationship between *particlesets* are selected based on the lattice type and boundary conditions.
- Only the images in the periodic directions are included in evaluating long-range interactions.
- Particle moves outside of the supercell in the open direction(s) are rejected with bspline SPOs. For instance, the z coordinate of an electron is restricted to [0,L_z) in a slab geomtry.

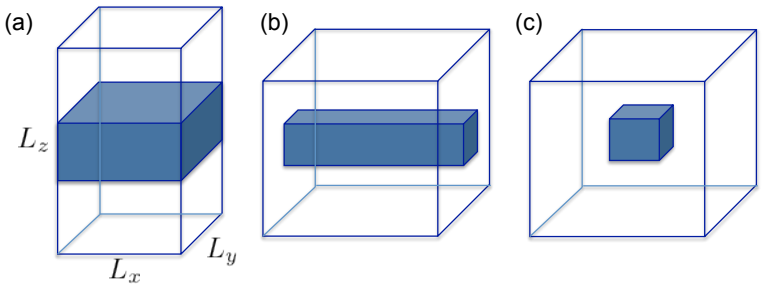

Figure 3.5: Valid supercell for (a) slab (ppn), (b) wire (pnn) and (c) molecule (nnn) geometry in 3D.

Warning

For the electronic structure calculations in the PW basis, the gemoetric center of the ionic coordinates in the open-boundary directions has to be close to the center, i.e., ions must be located in the shaded region of the valid supercells. It is because the values of the SPOs are not defined beyond the supercell. Since QMC uses the values and the gradients of the SPOs which are normally represented by bsplines on a grid, it is important to apply strict convergence conditions on the vacuum region for the PW calculations to minimize the contributions from image cells. Not only the total energy should be converged with respect to the supercell, all the SPOs should decay to zero at the open boundaries.

3.4.3 particleset

A *particleset* defines a set of particles and contains

```
particleset=
  children  : group+, attrib*
  attribute : name, size, random, random_source
```

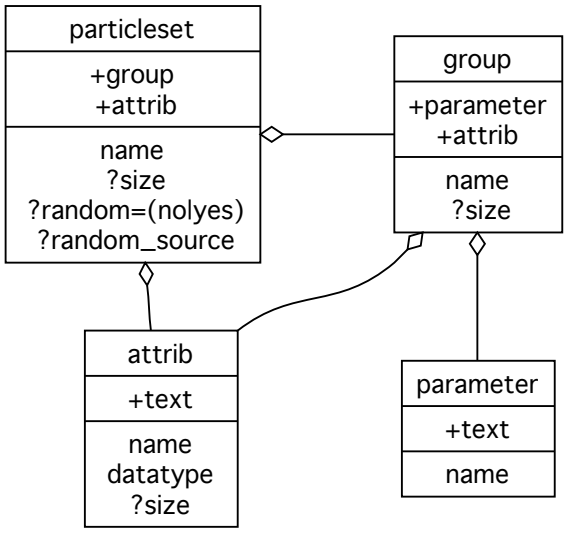

Figure 3.6: *particleset* element

Attributes of *particleset:*

name	datatype	default	definition
name	string	None	Name of this particle set. Referenced by other elements
size	integer	0	Number of particles, when size is missing, *group/@size* is used.

random	yes/no	no	Randomize the initial position (uniform)
random_source	string	None	Randomize the initial position using the named *particleset*

Remarks

- It is seldom necessary for the users to prepare *particleset* from scratch. The conversion tools from other packages will generate *particleset* elements.
- The HDF5 wavefunction file in ES-HDF format contains everything. It is possible to omit *particleset*. See the section on ES-HDF.
- If *particleset/@random="yes"*, the positions are randomly assigned with repsect to the supercell. This is recommended for bulk calculations.
- When *particleset/@random_source* is provided, the positions are randomly assigned around the particle positions of this *particleset*. Information such as the valence charge for each ion is used to make reasonable guess. This is useful with systems with open boundary conditions, e.g., molecules, surfaces.
- *particleset/@name='e'* is reserved for the electronic system.

The example below defines one *particleset* for an ionic system called *particleset/@name='ion0'* and another for an electronic system *particleset/@name='e'*. The electronic positions are randomly assigned using the positions of the ionic system with *particleset/@random_source='ion0'*. The order of the two *particleset* elements is important, since the electronic system depends on the ionic system.

```
<qmcsystem>
  <particleset name="ion0" size="3">
    <group name="O">
      <parameter name="charge">6</parameter>
      <parameter name="valence">4</parameter>
      <parameter name="atomicnumber">8</parameter>
    </group>
    <group name="H">
      <parameter name="charge">1</parameter>
      <parameter name="valence">1</parameter>
      <parameter name="atomicnumber">1</parameter>
    </group>
    <attrib name="position" datatype="posArray">
  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
  0.0000000000e+00 -1.4304287043e+00  1.1071569627e+00
  0.0000000000e+00  1.4304287043e+00  1.1071569627e+00
    </attrib>
    <attrib name="ionid" datatype="stringArray">
 O H H
    </attrib>
  </particleset>
  <particleset name="e" random="yes" random_source="ion0">
    <group name="u" size="4">
      <parameter name="charge">-1</parameter>
    </group>
    <group name="d" size="4">
      <parameter name="charge">-1</parameter>
    </group>
  </particleset>
</qmcsystem>
```

3.4.3.1 group

Defines a group of particles or species.

```
group =
  children  : parameter+, attrib*
  attribute : name, size
```

The properties of each group are defined by *parameter*. These parameters are predefined in QMCPACK and their uses are restricted. If an input file contains *parameter* unknown to QMCPACK, it is ignored.

name	datatype	default	definition
charge	real	0	Charge in atomic unit. q=-1 for the electrons
valence	real	0	Valence charge in AU
atomicnumber	integer	0	Atomic number in the periodic table

Remarks

- Default *group/@size='0'*.
- When *group/@size* is provided, the *attrib* elements within this group have the same size.

The example above shows two different ways to define *particleset*. The species of the ionic system are not grouped, while electronic system is grouped into u(p) and d(own) electrons. Below is identical to the previous example with the difference being that now the ionic system is grouped into O(xygen) and H(ydrogen) and the size is the sum of two groups.

```
<qmcsystem>
  <particleset name="ion0">
    <group name="O" size="1">
      <parameter name="charge">6</parameter>
      <parameter name="valence">4</parameter>
      <parameter name="atomicnumber">8</parameter>
      <attrib name="position" datatype="posArray">
      0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
      </attrib>
    </group>
    <group name="H" size="2">
      <parameter name="charge">1</parameter>
      <parameter name="valence">1</parameter>
      <parameter name="atomicnumber">1</parameter>
      <attrib name="position" datatype="posArray">
      0.0000000000e+00 -1.4304287043e+00  1.1071569627e+00
      0.0000000000e+00  1.4304287043e+00  1.1071569627e+00
      </attrib>
    </group>
  </particleset>
  <particleset name="e" random="yes" random_source="ion0">
    <group name="u" size="4">
      <parameter name="charge">-1</parameter>
    </group>
    <group name="d" size="4">
      <parameter name="charge">-1</parameter>
    </group>
  </particleset>
</qmcsystem>
```

3.4.4 hamiltonian

It defines a many-body Hamiltonian $\hat{H} = \sum_i \hat{h}$ for the target *particleset* to evaluate the local energy,

$$E_L(\mathbf{R}) = \frac{\hat{H}\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}.$$

$\mathbf{R}$ denotes the configuration of the target *particleset*. It contains

```
hamiltonian =
  children  : pairpot*, estimator*
  attribute : name, target
```

Remarks

The children of *hamiltonian* are divided into

- Physical operator, $\hat{h}$, whose value is added to the local energy.
- Estimators for an operator $O = \hat{O}\Psi_T/\Psi_T$ The kinetic operator is automatically added to the total $\hat{H}$. It is possible to overwrite the kinetic operator with a specialized kinetic operator.

3.4.4.1 pairpot

A pair potential in QMCPACK denotes any operator which has the form of

$$\hat{h} = \sum_i \sum_j V(\mathbf{R}_i, \mathbf{R^s}_j)$$

where the i and j run over the particles in the t(arget) *particleset* and s(ource) *particleset*. The s(ource) *particleset* can be the same as the t(arget) *particleset*, e.g., Hartree energy for the electrons. *pairpotential* node contains

```
pairpot = attribute : name, type, source, target, wavefunction
```

The attributes are

name	definition	default	choices
type	Pontential type	None	coulomb, pseudo
source	source *particleset*	e	any *particleset/@name*
target	target *particleset*	e	any *particleset/@name*
wavefunction	target Ψ_T	psi0	any *wavefunction/@name*

For a pseudopotential calculation, the many-body Hamiltonian,

$$\hat{H} = \sum_i^e -\frac{1}{2}\nabla_i^2 + \sum_{i<j}^e \frac{1}{|\mathbf{R}_i - \mathbf{R}_j|} + \sum_{i<j}^I \frac{Z_i Z_j}{|\mathbf{R}_i^I - \mathbf{R}_j^I|} + \sum_i^e \sum_j^I V_{pseudo}(\mathbf{R}_i, \mathbf{R}_i^I)$$

is given by

```
<hamiltonian name="h0" type="generic" target="e">
  <pairpot name="ElecElec" type="coulomb" source="e" target="e" physical="true"
      />
  <pairpot name="IonIon" type="coulomb" source="ion0" target="ion0"/>
  <pairpot type="pseudo" name="PseudoPot" source="ion0" wavefunction="psi0"
      format="xml">
    <pseudo elementType="C" href="C.xml"/>
  </pairpot>
</hamiltonian>
```

3.4.5 wavefunction

A trial many-body wavefunction takes a form $\Psi_T = \prod_k \psi_k(\{\alpha\}, \mathbf{R})$ and for the electronic structure calculations, it is generally written as

$$\Psi_T = \exp(-J_1)\exp(-J_2)\cdots\sum_{i=1}^{M} C_i D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi),$$

so called multi-Slater-Jastrow wavefunction.

wavefunction node contains

```
wavefunction =
  children  : jastrow*, determinantset*
  attribute : name, target
```

This example shows a Slater-Jastrow wavefunction with One-Body and Two-Body Jastrow functions:

$$\Psi_T = \exp(-J_1)\exp(-J_2)\underbrace{D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi)}_{determinant set},$$

where $\{\phi\}$ denotes the single-particle orbitals.

```
<?xml version="1.0"?>
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="bspline" href="lda.pwscf.h5" sort="1" tilematrix="1 0
       0 0 1 0 0 0 1"
      twistnum="0" source="ion0" version="0.10" gpu="yes">
      <slaterdeterminant>
        <determinant id="updet" size="8">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
        <determinant id="downdet" size="8">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
      </slaterdeterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation speciesA="u" speciesB="u" size="8">
        <coefficients id="uu" type="Array"> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
      <correlation speciesA="u" speciesB="d" size="8">
        <coefficients id="ud" type="Array"> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" source="ion0" print="
      yes">
      <correlation elementType="C" size="8">
        <coefficients id="eC" type="Array"> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
    </jastrow>
  </wavefunction>
</qmcsystem>
```

Chapter 4

Interface Utilities

As discussed before, fermionic wavefunctions are

$$\Psi_{AS} = \sum_{i=1}^{M} C_i D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi),$$

where $\{\phi\}$ denotes a set of single-particle orbitals (SPOs).

SPOs are obtained by other electronic structure codes, such as Quantum Espresso or GAMESS and they have to be put into the file formats for QMCPACK.

The most commonly used codes by the developers are PWSCF (pw.x of QE) and GAMESS and tools to support them are more developed than others. In this section, we describe how to use the available tools to prepare QMC runs.

A typical QMC workflow follows

- Run a DFT or QC calculation
- Convert output for QMCPACK
- Run QMC

The files generated by various converter tools are XML and HDF5 files. They are portable on any platform.

4.1 Quantum espresso

Download a distribution with PP/pw2qmcpack.f90 via subversion

```
svn co http://qmctools.googlecode.com/svn/dft/espresso-4.2/
```

It contains two additional files to the QE distribution

- PP/pw2qmcpack.f90
- clib/esh5_interfaces.c

Additionally, *PP/Makefile* is modified to compile *pw2qmcpack.f90* with HDF5 library. To build pw.x and pw2qmcpack,

- configure as instructed by QE documentations

  - will generate *make.sys* for the target machine

- edit make.sys: modify DFLAGS, CFLAGS and and add HDF5_HOME and HDF5_LIBS

```
HDF5_HOME = location of HDF5 library
DFLAGS    = [existing options]  -DH5_USE_16_API
CFLAGS    = [existing options] -std=c99 -I$(HDF5_HOME)/include
HDF5_LIBS = -L$(HDF5_HOME)/lib -lhdf5_hl -lhdf5
```

- make pw.x and pw2qmcpack.x

```
make pw; make pp
```

4.1.1 Generating wavefunction files from PWSCF

After fully converged PWSCF runs, execute *pw2qmcpack.x* as

```
pw2qmcpack.x < convert.in
```

to generate ESHDF file and XML file that is ready for use in QMCPACK.

convert.in needs only these options

```
&inputPP
    prefix= 'PREFIX',
    outdir='OUTDIR',
    write_psir=.false.
/
```

PREFIX and OUTDIR are the same as those used in the input file for pw.x. With *write_psir*=.false. (default), only the orbitals in the PW basis are written to reduce the file size and time to write the file. Make sure to compile QMCPACK with FFTW.

4.1.2 Using the converted files in QMCPACK

pw2qmcpack.xml generates

- $*$.ptcl.xml : describe ionic and electronic system
- $*$.wfs.xml : wavefunction
- $*$.pwscf.h5 : HDF5 file for the wavefunctions from planwave codes

.wfs.xml can be included in a main QMC input as

```
<include href="H2O.wfs.xml"/>
```

H2O.wfs.xml has the basic definition of a trial wavefunction which consists of

- slater determinant
- two-body Jastrow
- one-body Jastrow

```
<qmcsystem>
  <wavefunction name="psi0" target="e">
    <determinantset type="bspline" href="H2O.pwscf.h5" sort="1" tilematrix="1 0
       0 0 1 0 0 0 1"
      twistnum="0" source="ion0" version="0.10" gpu="yes">
      <slaterdeterminant>
        <determinant id="updet" size="32">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
        <determinant id="downdet" size="32">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
      </slaterdeterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation speciesA="u" speciesB="u" size="8">
        <coefficients id="uu" type="Array"> 0.0 0.0 0.0 0.0  0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
      <correlation speciesA="u" speciesB="d" size="8">
        <coefficients id="ud" type="Array"> 0.0 0.0 0.0 0.0  0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" source="ion0" print="
      yes">
      <correlation elementType="O" size="8">
        <coefficients id="eO" type="Array"> 0.0 0.0 0.0 0.0  0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
      <correlation elementType="H" size="8">
        <coefficients id="eH" type="Array"> 0.0 0.0 0.0 0.0  0.0 0.0 0.0 0.0</
      coefficients>
      </correlation>
    </jastrow>
  </wavefunction>
</qmcsystem>
```

For solid systems with periodic boundary conditions, there is no need to include ∗.ptcl.xml. All the data needed for QMC calculations - unitcell, ionic and electronic systems - are in ∗.pwscf.h5 and the attributes of *determinantset* are used to construct ionic and electronic particlesets.

4.1.3 Molecular calculations using PW codes

It is important to have sufficiently large vaccum regions in all directions to ensure that the SPOs go to zero at the cell boundary. In addition, the boudary conditions can be overwritten as

```
<simulationcell>
  <parameter name="bconds"> n n n </parameter>
</simulationcell>
```

This will remove any contribution from image cells to total energies.

The main input xml has to be modified.

```
<include href="H2O.ptcl.xml"/>
<include href="H2O.wfs.xml"/>
```

4.2 Gamess

4.3 wfconvert package

To support multiple formats of pseudopotentials and SPOs, K. Esler has developed a set of tools. They are distributed as a separate package and can be downloaded via subversion as

```
svn co http://qmctools.googlecode.com/svn/trunk/wfconvert
```

The distribution includes blitz header files and Common library. It uses the same build system based on *cmake* as QMCPACK and needs these libraries

- libxml2
- hdf5
- fftw3
- gsl

gsl (Gnu Scientific Library) is available as a LINUX package or as a module on HPC systems.

When the build is successful, two executables will be in bin directory.

- wfconvert
- ppconvert

ppconvert is a useful tool that can generate PPs files for other ES codes as well as for QMCPACK.

Chapter 5

QMC School 2012

5.1 Basics on Forge at NCSA

NCSA Dell NVIDIA Linux Cluster Forge Technical Summary

First time:

- Log on forge.ncsa.illinois.edu
- Set up common environment (only once)

```
ssh -Y your-id@forge.ncsa.illinois.edu
cp /uf/ac/jnkim/qmc/tcshrc_common ~/.tcshrc
cp /uf/ac/jnkim/qmc/modules ~/.modules
```

- Set up your directory to run exercise

```
mkdir yourname
cd yourname
```

Note that the default shell on forge is *tcsh* and it is not possible to change the login shell for the training account. However, you can start bash. In this case, copy */uf/ac/jnkim/qmc/bashrc_common* as

```
cp /uf/ac/jnkim/qmc/bashrc_common ~/.bashrc
```

For Monday's lab, copy /uf/ac/mcminis2/qmcpack_tutorial.tar in your working directory,

```
cp /uf/ac/mcminis2/qmcpack_tutorial.tar .
tar xvf qmcpack_tutorial.tar
```

5.2 Setups for QMCPACK

Pre-built binaries can be found in */uf/ac/jnkim/qmc/bin* and they are

qmcapp_real CPU, real wavefunctions in double precision

qmcapp_complex CPU, complex wavefunctions in double precision

qmcapp_cuda GPU, real wavefunctions in single precision

Some features of QMCPACK are fixed at the compiler time.

- data type of trial wavefunctions: real or complex
- CPU or GPU
  - With GPU, most of the calculations use single precision

To build QMCPACK yourself on forge, follow these steps

```
$svn co http://qmcpack.googlecode.com/svn/trunk/ qmcpack
$cd qmcpack/build
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake ..
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake ..
$make -j4
$ls bin
```

If all goes well, you will see *bin/qmcapp* in your directory. The defult build will create a binary using real wavefunctions on CPUs. Note that ForgeMvapich.cmake defines build properties that are only applicable to forge.

To build other versions with the prepared toolchain file on forge, create directories for each build.

- qmcpack
  - src
  - build
  - build_complex
  - build_cuda

Starting from the top directory,

- complex wavefunction

```
$mkdir build_complex
$cd build_complex
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake -DQMC_COMPLEX=1 ..
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake -DQMC_COMPLEX=1 ..
$make -j4
```

- CUDA version

```
$mkdir build_cuda
$cd build_cuda
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake -DQMC_CUDA=1 ..
$cmake -DCMAKE_TOOLCHAIN_FILE=../config/ForgeMvapich.cmake -DQMC_CUDA=1 ..
$make -j4
```

These are tools used to interface QE and QMCPACK:

```
$svn co http://qmctools.googlecode.com/svn/dft/espresso-4.2
$svn co http://qmctools.googlecode.com/svn/trunk/wfconvert
```

Chapter 6

Using PWSCF and QMCPACK to perform total energy calculations of condensed systems

6.1 Introduction

This tutorial session will focus on how to perform accurate QMC calculations on solids using pwscf for creating trial wavefunctions and QMCPACK for the QMC. The concepts and tools needed will be introduced in the context of calculating a pressure versus volume curve for carbon in the diamond structure. Due to the limited computational resources available for this tutorial, many of these calculations will be under converged, but complete results will be provided in the course of this document.

Please note that this tutorial will be highly dependent on the setup-qmc.pl tool for generating pwscf and QMCPACK input files. This tool has been preconfigured and placed in your path. In general you will need edit the configuration file setup-qmc-conf.pm located in the utils directory of QMCPACK to specify the locations of several tools used throughout.

The general outline of the tutorial is follows. The necessary input DFT calculations for a QMC simulation will be discussed. Then a wavefunction will be generated from a DFT calculation for diamond. Next, VMC will be used to optimize Jastrow factors for the many body wavefunction. Then the proper convergence of the b-spline representation of the DFT wavefunction will be examined. With that in hand, DMC calculations will be discussed including the convergence of the time step. Finally, the finite size errors inherent in these calculations will be confronted.

6.2 Initial DFT Calculations

The first step in performing quantum Monte Carlo calculations is to perform a DFT calculation on the same system. This is necessary to provide the trial wavefunction that is necessary both for improving the efficiency of the QMC calculation (importance sampling) and for mitigating the sign problem. In this case we will be focusing on diamond, so we will start with a pwscf input file as given below:

```
&control
   calculation='scf'
   restart_mode='from_scratch',
   tstress = .true.
   prefix='diamond',
   pseudo_dir = './',
   outdir='out'
   disk_io='low'
   wf_collect=.true.
/
&system
   ibrav=2, celldm(1) = 6.700, nat=  2, ntyp= 1,
   nspin=1,
```

```
    degauss=0.001,
    smearing='mp',
    occupations='smearing',
    ecutwfc =210
    ecutrho =840
 /
 &electrons
    conv_thr =  1.0d-10
    mixing_beta = 0.7
 /
ATOMIC_SPECIES
 C  12.0107  C.ncpp
ATOMIC_POSITIONS {crystal}
 C 0.00 0.00 0.00
 C 0.25 0.25 0.25
K_POINTS AUTOMATIC
8 8 8 1 1 1
```

As the goal of the calculation is to produce an extremely accurate DFT wavefunction, ecutwfc and the k-points should be converged to a very high degree. For this purpose, it is often better to monitor convergence of the stresses (which depend linearly on the wavefunctions) than on the total energy which improves quadratically with the quality of the wavefunctions.

In each of your home directories, there should be a directory called diamond. This input file is included in the diamond directory and is called diamond-scf.in. Create a directory for the output file (mkdir out) and then run pwscf with the following command:

```
pw.x < diamond-scf.in > diamond-scf.out
```

6.2.1 Pseudopotentials

In the calculation on diamond, note that the cutoff is higher than is typical in modern electronic structure codes. This is due to two factors. The first is that the treatment of nonlocal pseudopotentials in QMC presently requires the use of norm conserving pseudopotentials rather than the PAW or ultrasoft formalisms which are now in use. The second reason for the high cutoff is due to the particular construction of this pseudopotential. It was created using opium with the only constraint in mind that it accurately reproduces the all electron results within LDA. This is often a practical way to generate pseudopotentials for QMC as the increased plane wave cutoffs do not result in increased run time within QMC and the DFT calculation is many orders of magnitude faster than the QMC. The only exception to this rule is when there is difficulty fitting the wavefunction into the memory of the computer during the QMC. This will be discussed more thoroughly in section Creating and testing wavefunctions for QMCPACK.

It may also be practical to use pseudopotentials available in various libraries distributed with DFT codes. The ppconvert tool which is a part of the qmcutils distribution can convert many of these to the xml format which QMCPACK understands. For the purpose of this tutorial, any norm conserving pseudopotential in UPF or NCPP format may be used for the DFT part of the QMC calculations.

6.3 Creating and testing wavefunctions for QMCPACK

The plane wave representation for the wavefunction used in pwscf is not a good choice for quantum Monte Carlo. This is because the largest computational effort involved in QMC is evaluating the trial wavefunction and related quantities when an electron is moved from $\mathbf{r}$ to $\mathbf{r}'$. With a plane wave basis, this requires evaluating all N of the basis functions for every move.

Rather than using these plane waves, QMCPACK calculations are typically performed using a b-spline representation of the wavefunction. Practically speaking, this involves extracting the plane waves from a pwscf calculation and then choosing a regular grid upon which to construct the spline representation. In this section you will construct the wavefunction from pwscf calculations and test that the splines are sufficiently dense within QMCPACK.

6.3.1 Extracting Wavefunctions at Particular k-points

There is a direct mapping from k-points in DFT calculations to non-periodic boundary conditions in quantum Monte Carlo (essential for twist averaging). For this reason and for others detailed in section Creating and Using Supercells, it is important to be able to extract the wavefunction at arbitrary k-points in the first Brillouin Zone of the system. setup-qmc.pl simplifies this process by creating input files for pwscf to get the wavefunction at any k-point using the charge density from the converged DFT calculation found in section Initial DFT Calculations. Following this pwscf calculation, the tool pw2qmcpack.x should be run to generate an hdf file which is then prepared for QMCPACK using wfconvert. Generate these files in the directory where pwscf has already been run by issuing the command

```
setup-qmc.pl --genwfn --kpoint 0  0  0 diamond-scf.in
```

This will generate an input file for pwscf called be-nscf.in. This input file should be identical to be-scf.in except for a few critical points. Instead of a scf calculation, the calculation will now be nscf so as to avoid changing the charge density from the previously converged calculation. Secondly, the wavefunction will be collected by a single process using wf_collect = .true. This should allow the mpi version of quantum espresso to be used where available. All symmetry is also turned off in order to avoid removing any k-points. Finally, the last section of the file: K_POINTS is changed to include only the point “0 0 0”.

Run this calculation with the following command:

```
pw.x < diamond-nscf.in > diamond-nscf.out
```

The wavefunction can now be extracted to hdf format using the pw2qmcpack.x tool that is included with the modified version of quantum espresso available from: (qmc-tools). All that is needed for this is a simple input file that tells pw2qmcpack.x where the wavefunctions from the pwscf calculation are and whether or not to generate the b-splines directly. Invoke pw2qmcpack.x as follows:

```
pw2qmcpack.x < diamond-pw2x.in > diamond-pw2x.out
```

Now this is converted to the eshdf format used by QMCPACK with the following command

```
wfconvert --nospline --eshdf diamond.h5 out/diamond.pwscf.h5 >& diamond-
      wfconvert.out
```

The nospline argument keeps the wavefunction in reciprocal space, which will be useful in convergence testing below, but can significantly increase the setup time in QMCPACK for large systems. wfconvert can also be invoked with the argument –factor xx which will generate the real space b-splines directory with a given mesh spacing.

6.4 Optimizing Jastrow Factors within QMC

The next step in performing QMC calculations of diamond is to produce as accurate a many body wavefunction as possible. Although many forms have been proposed for many body wavefunctions of periodic systems, we will stick to the standard Slater-Jastrow form whereby the Slater determinant obtained from DFT is multiplied by correlation factors which explicitly account for the approach of electrons to other electrons and of electrons to the ions in the system. A very general form of these Jastrow factors which is used in QMCPACK is as follows

$$\exp^{-J(\mathbf{R})} = \exp^{-\sum_i \sum_j f(|R_i - R_j|)}$$

where the function $f(x)$ is represented as a one dimensional functor.

It is not possible to optimize this function analytically, so we will optimize its form using variational Monte Carlo. There are several different formalisms for doing this optimization. Probably the simplest to understand conceptually is the

variance minimization approach. This technique takes advantage of the zero variance principle whereby the variance of a VMC simulation with any many body eigenstate as the wavefunction will be zero. Knowing this, VMC simulations are performed and the parameters which influence the Jastrow factors (in this case spline values) are varied in order to reduce the variance of the simulation.

Recently, successful techniques have been devised which also take into account the variational nature of VMC and work to minimize the total energy calculated by varying the parameters in the wavefunction. We will use one such approach to optimize Jastrow factors for our aluminum example. Again, we will use setup-qmc.pl to generate a QMCPACK input file. Invoke that script with the following command:

```
setup-qmc.pl --optwvfcn --splfactor 1.0 --wvfcnfile diamond.h5 \
   --vmctimestep 0.8 --vmcblocks 512 --vmcwalkers 128 --vmcsteps 5 \
   --optsamples 32768 --optloops 16 --onebodysplinepts 4 \
   --twobodysplinepts 4 diamond-scf.in
```

To run these calculations, it is necessary to have some understanding of how parallelism works in QMCPACK. QMCPACK takes advantage of distributed memory parallelism (such as between nodes of a supercomputer) via MPI and shared memory parallelism (such as within a node) using OpenMP. On modern supercomputers, this can result in large benefits because the communication is reduced linearly with the number of OpenMP threads used per node. In this case we will have logged directly into the nodes of the supercomputer and will run QMCPACK using only OpenMP parallelism. To do this, set the number of threads to 4 using the following command

```
export OMP_NUM_THREADS=4
```

Now QMCPACK should be invoked to find the optimal Jastrow factor. First move to the directory optimization and invoke QMCPACK as follows

```
qmcapp opt-diamond.xml >& opt-diamond.out &
```

Hybrid parallelism can also be used by invoking qmcapp via the standard mpi process on your supercomputer taking into account that the number of processes requested should not be the total number of CPU cores to use, but should likely correspond to either the number of processors on the nodes or to the number of nodes. As an example, to run QMCPACK on a computer with 4 nodes which each have 8 processors, commands similar to the following would be used:

```
export OMP_NUM_THREADS=8
mpirun -np 4 qmcapp opt-diamond.xml >& opt-diamond.out &
```

While QMCPACK is optimizing the wavefunction, take some time to examine the input file that was generated by setup-qmc.pl:

```
<?xml version="1.0"?>
<simulation>
  <project id="opt-diamond" series="1">
    <application name="qmcapp" role="molecu" class="serial" version="0.2">
      Optimization of jastrows for diamond
    </application>
  </project>
  <qmcsystem>
    <!-- define simulation cell -->
    <simulationcell>
      <parameter name="lattice">
        -3.35   0.00   3.35
         0.00   3.35   3.35
        -3.35   3.35   0.00
      </parameter>
      <parameter name="bconds">p p p</parameter>
      <parameter name="LR_dim_cutoff">15</parameter>
    </simulationcell>
    <!-- define ionic system -->
    <particleset name="ion0" size="2">
      <group name="C">
```

```
      <parameter name="charge">4.000000</parameter>
      <parameter name="valence">4.000000</parameter>
      <parameter name="atomicnumber">6.000000</parameter>
    </group>
    <attrib name="position" datatype="posArray" condition="1">
      0.00   0.00   0.00
      0.25   0.25   0.25
    </attrib>
    <attrib name="ionid" datatype="stringArray">
      C  C
    </attrib>
  </particleset>
  <!-- define electronic system -->
  <particleset name="e" random="yes" randomsrc="ion0">
    <group name="u" size="4">
      <parameter name="charge">-1</parameter>
    </group>
    <group name="d" size="4">
      <parameter name="charge">-1</parameter>
    </group>
  </particleset>
  <!-- define wavefunction for "e" -->
  <wavefunction name="psi0" target="e">
    <determinantset type="einspline" href="../diamond.h5"
      tilematrix="1 0 0 0 1 0 0 0 1" twistnum="0" gpu="yes" meshfactor="1.0">
      <basisset/>
      <slaterdeterminant>
        <determinant id="updet" size="4" ref="updet">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
        <determinant id="downdet" size="4" ref="downdet">
          <occupation mode="ground" spindataset="0">
          </occupation>
        </determinant>
      </slaterdeterminant>
    </determinantset>
    <jastrow name="J2" type="Two-Body" function="Bspline" print="yes">
      <correlation speciesA="u" speciesB="u" size="4">
        <coefficients id="uu" type="Array">  0  0  0  0  </coefficients>
      </correlation>
      <correlation speciesA="u" speciesB="d" size="4">
        <coefficients id="ud" type="Array">  0  0  0  0  </coefficients>
      </correlation>
    </jastrow>
    <jastrow name="J1" type="One-Body" function="Bspline" print="yes" source=
    "ion0">
      <correlation elementType="C" cusp="0.0" size="4">
        <coefficients id="C" type="Array"> 0  0  0  0  </coefficients>
      </correlation>
    </jastrow>
  </wavefunction>
  <!-- define hmailtonian for "e" -->
  <hamiltonian name="h0" type="generic" target="e">
    <pairpot type="pseudo" name="PseudoPot" source="ion0" wavefunction="psi0"
     format="xml">
      <pseudo elementType="C" href="../C.xml"/>
    </pairpot>
    <pairpot name="IonIon" type="coulomb" source="ion0" target="ion0"/>
    <pairpot name="MPC" type="MPC" source="e" target="e" ecut="60.0" physical
    ="false"/>
    <pairpot name="ElecElec" type="coulomb" source="e" target="e" physical="
    true"/>
    <estimator name="KEcorr" type="chiesa" source="e" psi="psi0"/>
  </hamiltonian>
</qmcsystem>
<loop max="16">
  <qmc method="linear" move="pbyp" checkpoint="-1" gpu="yes">
    <parameter name="blocks">   512 </parameter>
    <parameter name="warmupsteps"> 0 </parameter>
    <parameter name="steps">    5 </parameter>
    <parameter name="timestep">  0.8  </parameter>
    <parameter name="walkers">  128 </parameter>
    <parameter name="samples">  32768  </parameter>
    <parameter name="minwalkers">  0.5 </parameter>
    <parameter name="maxWeight">    1e9 </parameter>
    <parameter name="useDrift">  yes </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <cost name="energy">                   0.05 </cost>
    <cost name="unreweightedvariance">     0.0 </cost>
    <cost name="reweightedvariance">       0.95 </cost>
```

```
      <parameter name="MinMethod">quartic</parameter>
      <parameter name="GEVMethod">mixed</parameter>
      <parameter name="beta">  0.05 </parameter>
      <parameter name="exp0"> -16 </parameter>
      <parameter name="nonlocalpp">no</parameter>
      <parameter name="useBuffer">no</parameter>
      <parameter name="bigchange">5.0</parameter>
      <parameter name="alloweddifference"> 1.0e-4 </parameter>
      <parameter name="stepsize">4.0e-1</parameter>
      <parameter name="stabilizerscale">  1.0 </parameter>
      <parameter name="nstabilizers"> 3 </parameter>
      <parameter name="max_its"> 1 </parameter>
    </qmc>
  </loop>
</simulation>
```

The geometry of the calculation is read from the pwscf input file and the unit cell is placed in the lattice tag. The bconds tag can be changed so that slabs or wires can be handled natively without any spurious interactions from images, to do this, simply change p (periodic) for a given boundary condition to n (non-periodic). Following this, particlesets are specified giving information about the electrons and ions in the problem.

The determinantset controls how the wavefunction is represented in the problem. Here type="einspline" specifies that we are using a b-spline basis set and meshfactor="1.0" specifies a mesh spacing. The meshfactor is the spacing in terms of the density of the real space points corresponding to the plane waves used in the DFT calculation. A meshfactor of 0.5 will give the same number of real space points as plane waves in the cell. Other notable sections include the Jastrow factor (which in this case starts as being represented by a 4 point spline that is all 0), and the optimization block at the end of the file.

In general each QMCPACK input file will consist of some header information followed by a particleset, a wavefunction, a Hamiltonian and then any number of qmc sections which give the actions for QMCPACK to perform. Here we use a loop construct to repeat the optimization several times because the optimization needs to proceed somewhat self-consistently. At this point, take a minute to familiarize yourself with the QMCPACK users guide, which will provide a much more thorough explanation of how the input file is structured and what each element means.

This optimization will take about 10 minutes to run. However, the calculation will continuously write to files named opt-diamond.s0??.scalar.dat, with one for each iteration of the VMC optimization. There is another script, OptProgress.pl which can be used to monitor the progress of the optimization. The syntax to invoke it is as follows:

```
OptProgress.pl opt-diamond.s001.scalar.dat
```

When the calculation is finished, this will produce plot similar to this figure.

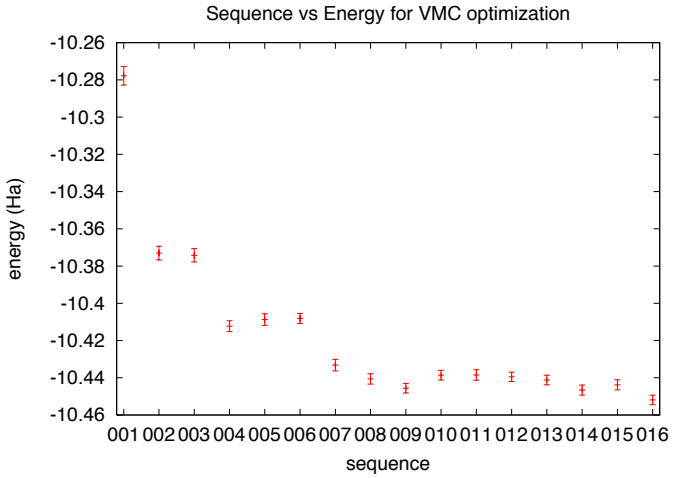

Figure 6.1: Total energy vs iteration number for optimization of Jastrow factors in diamond.

6.4.1 Testing Convergence of the Splines in QMCPACK

Now that the wavefunctions have been generated the proper density for the b-splines should be found. In the previous section we used a rather large mesh given by meshfactor="1.0" in the determinantset. This is typically larger than is necessary and as the memory needed to store the wavefunction goes as the cube of the meshfactor, this can cause calculations to be intractable.

In order to determine the smallest appropriate meshfactor, we will perform variational Monte Carlo calculations using the best Jastrow factor determined above. The quantities to monitor are the total energy which should decrease until saturating at large values of meshfactor, the variance which should decrease as meshfactor increases and the kinetic energy which depends only on the shape of the trial wavefunction including Jastrow factor.

To perform this convergence test, generate appropriate input files using setup-qmc.pl with the following command

```
setup-qmc.pl --splconv --wvfcnfile diamond.h5 --withjas \
   --vmcblocks 2000 --vmcwalkers 512 --vmcsteps 2 --vmctimestep 1.0 \
   --minfactor 0.4 --maxfactor 0.8 --factorinc 0.1 diamond-scf.in
```

This will generate input files with names like diamond-f0.40.xml in the directory bsplineConv. Run each of these calculations by changing to the directory bsplineConv and invoking qmcapp for each one of the input files in turn:

```
qmcapp diamond-f0.40.xml >& diamond-f0.40.out
qmcapp diamond-f0.50.xml >& diamond-f0.50.out
qmcapp diamond-f0.60.xml >& diamond-f0.60.out
qmcapp diamond-f0.70.xml >& diamond-f0.70.out
qmcapp diamond-f0.80.xml >& diamond-f0.80.out
```

When you have finished this (it should take several minutes) you can monitor the convergence of the results using the PlotBsplConv.pl utility distributed with QMCPACK:

```
PlotBsplConv.pl diamond-f0.40.s001.scalar.dat
```

Which will generate a plot which looks like Fig. SplConvFig. The convergence of the total energy can be somewhat misleading as the quadratic convergence of the energy with the wavefunction can mask differences in the wavefunction.

The variance of the local energy is a much more reliable metric of the spline's convergence. It requires many fewer samples to converge and is a direct measure of how the spline factor will influence the cost of the DMC calculations to come. The number of steps required to obtain a particular statistical error in a DMC calculation goes linearly with the variance. In this case we see convergence with a meshfactor of roughly 0.7, which we will use in our future calculations.

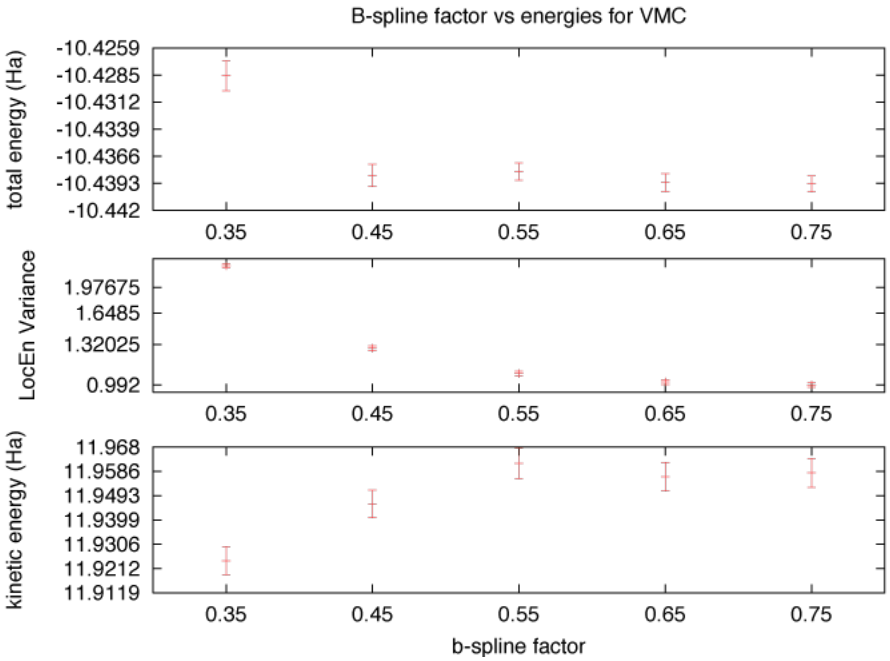

Figure 6.2: Total, kinetic energy and variance of diamond as a function of b-spline spacing from VMC.

6.5 Performing DMC Calculations

Diffusion Monte Carlo (DMC) calculations will consume the vast majority of the CPU cycles in your QMC projects. This method differs from VMC in that it samples the observables you specify for the lowest energy wavefunction consistent with the many body nodes that you provide in your trial wavefunction.

In performing accurate DMC calculations, it is important to converge several technical parameters. The first of these is the average population of walkers in the calculation. In general this needs to be large so that fluctuations in the population do not unduly influence the results of the calculations. For relatively good trial wavefunctions like the one used here, 1000 walkers should be more than sufficient, however for wavefunctions with larger variance, like all electron wavefunctions of first row solids, this may need to be 10's of thousands.

The second technical parameter that will need to be converged is the size of the time step which is used in the DMC propagation. This controls the accuracy of the Green's function which is used to propagate the walkers. A rule of thumb is that the acceptance ratio upon moving the walkers should be around 99%. However, this should be precisely converged, a task which can again be accomplished by the combination of setup-qmc.pl and QMCPACK. Here is the input line for setup-qmc.pl

```
./setup-qmc.pl --convdmctstep --wvfcnfile diamond.h5 --splfactor 0.7 \
   --vmcblocks 64 --vmcwalkers 1024 --vmcwarmupsteps 20 --vmctimestep 1.0 \
   --vmcsteps 5 --dmcwalkers 1024 --dmcsteps 5 --dmcblocks 1000 \
   --dmcwarmupsteps 200 --mindmctstep 0.01 --maxdmctstep 0.04 \
   --dmctstepinc 0.01 diamond-scf.in
```

The input file this produces will start with a short VMC calculation to produce a population of walkers more nearly

distributed according to the trial wavefunction. After that a series of DMC calculations are done with successively shorter time steps. The first calculation (series 002) starts with a time step of 0.04, followed by time steps of 0.03, 0.02 and 0.01. In order to analyze the results of these calculations, it is necessary to understand how DMC calculations are analyzed. The PlotDMC.pl script can show both the trace of the local energy as well as the population at each time step. Look at the first 1000 steps of the first DMC calculation:

```
PlotDMC.pl diamond-dmcTsteps.s002.dmc.dat 1 1000
```

This should produce a plot like the one below.

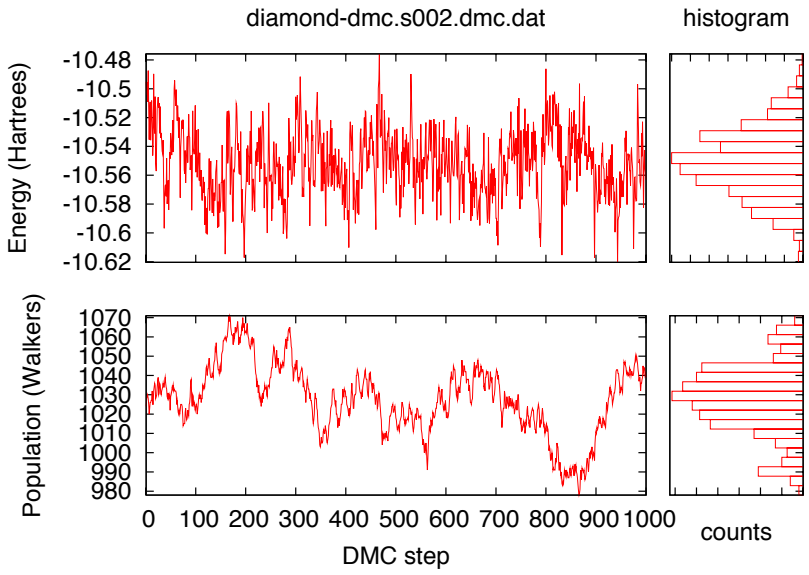

Figure 6.3: First 1000 steps of DMC calculation with time step of 0.04

At the beginning of the local energy plot, there is a transient as the population of walkers equilibrates from sampling Ψ_T^2 to the mixed distribution $\Psi_T\Phi$. After this transient, the local energy sampled should be approximately Gaussian (note recent work by R. Hood which suggests it is not exactly Gaussian) and the population should also be Gaussian. Any large deviations from this suggest that the sampling may not be ergodic. Now look at the full calculation with the transient removed:

```
PlotDMC.pl diamond-dmcTsteps.s002.dmc.dat 200
```

This produces a figure like below.

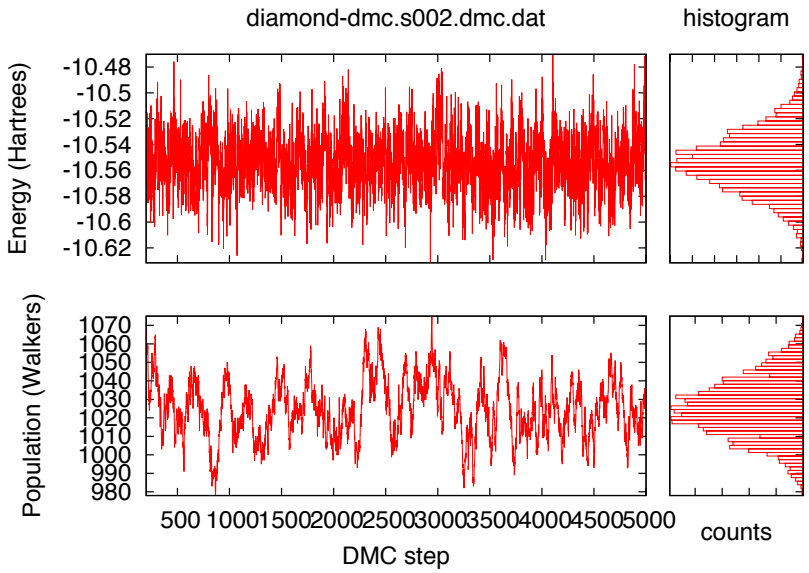

Figure 6.4: DMC calculation with time step of 0.04, first 200 steps removed to avoid transient

The spikes in the population histogram notwithstanding, this calculation appears to be producing reasonably good statistics. All of the various quantities which are reported in the scalar.dat files may now be examined using the energy.pl tool:

```
energy.pl diamond-dmcTsteps.s002.scalar.dat 50
```

Note that the first 50 blocks have been thrown out to avoid the transient (each block contains 5 steps so this is slightly more than in the plots above). This produces the output below.

```
LocalEnergy             =         -10.55142 +/-           0.00090
Variance                =             0.533 +/-             0.019
LocalPotential          =          -22.3342 +/-           0.0064
Kinetic                 =           11.7828 +/-           0.0066
LocalECP                =           -1.1639 +/-           0.0028
NonLocalECP             =           -5.7078 +/-           0.0081
IonIon                  =       -12.8639956 +/-        0.0000029
ElecElec                =           -2.5984 +/-           0.0018
MPC                     =           -2.3339 +/-           0.0020
KEcorr                  =      0.0355832891 +/-     0.0000000083
BlockWeight             =              5122 +/-              12
BlockCPU                =           0.25460 +/-           0.00049
AcceptRatio             =          0.970214 +/-          0.000033
Efficiency              =          20116.754 +/-             0.000
----------------------------------------------------------------------
Corrected Energy        =         -10.25132 +/-           0.00090
```

All energies are given in Hartrees and the BlockCPU is given in seconds. Serial correlations are accounted for by using the blocking technique. This tool is used extensively by the other analysis tools.

With those preliminaries in hand, we will use the PlotTstepConv.pl tool to analyze the convergence of the DMC calculations with respect to the time step. Run the tool as follows:

```
PlotTstepConv.pl diamond-dmcTsteps.xml 50
```

Note that rather than actual data files, this tool directly analyzes the QMCPACK input file to determine the data to plot. The output of the tool is shown:

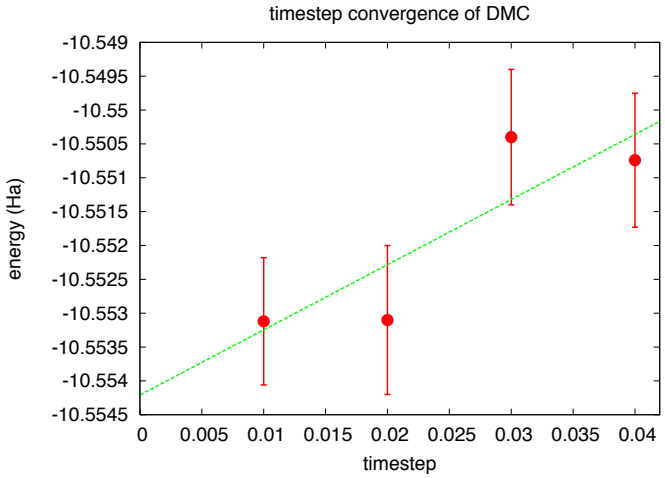

Figure 6.5: Total DMC energy as a function of time step for diamond

The trend line that is drawn may be wildly inaccurate especially if large time steps are included. In this case, there appears to be a very weak dependence on the time step and a time step of 0.02 appears to be a good candidate for future DMC calculations.

6.5.1 Two-Body Finite Size Effects

The other source of finite size effects comes from the spurious interactions of electrons in the simulation cell with their periodic images in neighboring cells. In DFT calculations these errors are not present because the exchange correlation functionals which deal with these interactions are all derived from calculations on the electron gas that are converged to the limit of infinitely large simulation cells. In this section we will introduce strategies for the mitigation of these two-body finite size effects.

6.5.1.1 Creating and Using Supercells

The simple solution to reducing the two-body finite size effects is to increase the size of the simulation cell. This can be done in two ways. First, the simulation cell specified in the DFT calculation can be explicitly made into a supercell. The author generally avoids this approach because the size of the wavefunction increases as the cube of the multiple of the number of primitive cells. In practice the required memory will rapidly outstrip what is available on the nodes of your supercomputer and the calculations will become intractable.

The other approach to generating supercells is to use Bloch's theorem and the repeated zone scheme to generate the wavefunction only in the primitive cell. This recognizes that the wavefunction for each band is just a periodic function times a simple phase factor anywhere in space. Thus, by generating the wavefunction in the primitive cell for the k-points corresponding to these phase factors, the full supercell's wavefunction can be reconstructed by using a vastly smaller amount of data. The memory required for this approach scales only linearly with the number of copies of the primitive cell.

Supercells should also be chosen so as to minimize the distance between points in the supercell and their images. As an example, a long skinny supercell will have much larger finite size errors than a cubic one with the same volume. setup-qmc.pl handles the search for an optimal supercell by searching through all possible combinations with a given number of primitive cell copies and determining the radius of the largest sphere which fits inside these trial supercells. The supercell which fits the largest sphere is then chosen. All of the methods available to setup-qmc.pl take as an optional argument –supercellsize, and will then produce DFT or qmc simulations adapted to this supercell.

The choice remaining is how large of a supercell should be used. Supercells containing all numbers of copies of the primitive cell are not equal in terms of their computational efficiency. As an example, multiples of the primitive cell that can be made into a cube are advantageous because the number of electrons necessary to get a certain distance between points in the supercell and their images (thereby minimizing finite size effects) is relatively small. The tool getBestSupercell.pl can be used to search a range of possible supercell sizes and report on the best ones in terms of efficiency. It is invoked as follows:

```
getBestSupercell --min 2 --max 32 --maxentry 5 diamond-scf.in
```

min and max control the range of supercells which should be searched and maxentry specifies the size of the search within each of these supercells. Due to the sub-optimal algorithm used by this tool, it is rather slow but fortunately only has to be run once for any given primitive cell geometry.

In the case of diamond, the output of this command searching for supercells having between 2 to 64 copies of the primitive cell shows particularly advantageous supercells for the following multiples: 4, 14, 32, 66 and 108. A finite size scaling study for this system should do calculations on those supercells which searching for convergence.

Now to do calculations on a supercell which has 4 copies of the primitive cell, you would use the following string of commands.

```
setup-qmc.pl --genwfn --supercellsize 4 diamond-scf.in
pw.x < diamond-S4-nscf.in > diamond-S4-nscf.out
pw2qmcpack.x < diamond-S4-pw2x.in > diamond-S4-pw2x.out
wfconvert --nospline --eshdf diamond-S4.h5 \
   out/diamond.pwscf.h5 >& diamond-S4-wfconvert.out
setup-qmc.pl --optwfn --supercellsize 4 --splfactor 0.7 \
   --wvfcnfile diamond-S4.h5 --vmctimestep 0.8 --vmcblocks 512 \
   --vmcwalkers 128 --vmcsteps 5 --optsamples 32768 --optloops 16 \
   --onebodysplinepts 4 --twobodysplinepts 4 diamond-scf.in
cd optimization-S4
qmcapp opt-diamond-S4.xml >& opt-diamond-S4.out
cd ..
setup-qmc.pl --dmc --supercellsize 4 --wvfcnfile diamond-8twists.h5 \
         --walkers 1024 --vmcblocks 64 --vmcsteps 5 --vmctimestep 1.0 \
         --dmcblocks 500 --dmcwarmupsteps 200 --dmcsteps 5 --dmctstep 0.02 \
         diamond-scf.in
cd dmc-S4
qmcapp diamond-S4-dmc.xml
```

Note that as above in the twist averaging example, it is not necessary to reconverge the dmc timestep and the meshfactor when changing supercell size.

6.5.1.2 Two-Body Finite Size Corrections

While the supercell approach is guaranteed to reduce the finite size effects, it is expensive. The cost of DMC calculations scales as the cube of the number of electrons. Given that we are often looking for per atom quantities, this reduces the actual cost to the square of the number of electrons. However, this still becomes extremely computationally demanding quite quickly.

For this reason, several two-body finite size correction schemes are available which attempt to estimate the two-body finite size errors and correct them. Typically we use two of these, the Model Periodic Coulomb interaction to correct the potential energy and the scheme of Chiesa, Ceperley and Martin to correct the kinetic energy. The necessary quantities are calculated automatically using the tools in this tutorial, with energy.pl for instance giving a line: Corrected Energy which includes both of these corrections.

While these corrections are not perfect for the smallest cell sizes, they do give a significantly better convergence of the total energy with cell size.

6.5.2 Putting it all Together

Although it will be far too computationally demanding for this tutorial, one should converge the parameters of interest with respect to the supercell size before performing detailed studies. In the case of diamond, this would involve repeating the calculations as specified above but with different –supercellsize's. Additionally, multiple supercell twists should be included via the –kgrid option.

Once this has been done, the tool PlotFsConv.pl can be used to produce plots using the command

```
PlotFsConv.pl --minsize 14 diamond-scf.in
```

Note that the extrapolation of both the calculations with the 2 body correction and without go to the same value in the infinite system limit. If were not the case, it would suggest that there were difficulties (for example timestep convergence or insignificant twist averaging) and the finite size corrections should not be trusted.

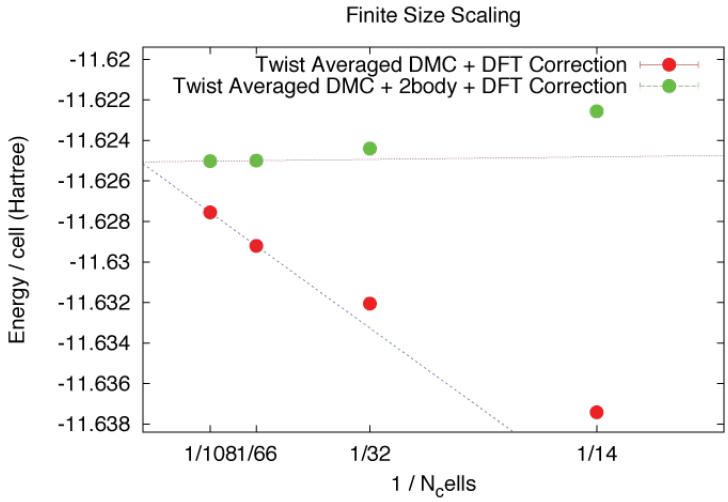

Figure 6.6: Total DMC energy vs supercell size. Convergence is achieved with a supercell that contains 66 copies of the primitive cell.

6.6 Conclusion

This tutorial has covered using pwscf and QMCPACK to perform DMC calculations on diamond. The convergence of technical parameters including the b-spline mesh, dmc timestep, twist averaging and supercell size are specifically addressed. At this point, the reader should know how to perform converged DMC calculations on similar condensed systems.

6.7 Acknowledgment

This tutorial was created with support from Sandia National Laboratories.

Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE-AC04-94AL85000.

Chapter 7

Guide for the developers

7.1 About QMCPACK

7.2 Documentation

Download doxygen-related files:

```
svn co  https://subversion.assembla.com/svn/qmcpack/pubs/qmcpack-ug/doxy
       qmcpack-doc
```

In qmcpack-doc directory, modify Doxygen.ug file

```
STRIP_FROM_PATH       = full-path-of-qmcpack-doc
```

and run doxygen

```
doxygen Doxygen.ug
```

These are created by doxygen:

html index.html, view with any browser

latex refman.tex, use pdflatex to generate a pdf file

One can change the location of these directories by setting OUTPUT_DIRECTORY in Doxygen.ug, e.g.,

```
OUTPUT_DIRECTORY      =  /home/foo/public_html/
```

Currently, these pages are used in Doxygen.ug

qmcpack_main.html Main Page

gettingstarted.html Getting and building QMCPACK

inputxml.html About I/O of QMCPACK

solids_cecam2012.html Tutotial given at CECAM QMC school 2011

developers.html This document for the developers

faq.html Frequently asked questions

doxygen is a powerful tool that can generate both good html and latex documentation. Check out doxygen document for more information.

7.3 How to add a page to this document

- Create a file mypage.html and simply start a page as

```
/** \page a_page A new page

Say anything in plain text, html or latex

*/
```

- Add mypage.html to INPUT of Doxygen.ug

```
INPUT                  = \
qmcpack_main.html \
gettingstarted.html \
inputxml.html \
solids_cecam2012.html \
developers.html \
mypage.html \
faq.html \
license.html
```

- Run doxygen

```
doxygen Doxygen.ug
```

Each page becomes a new page in html and a chapter in latex. Use page_template.html as a template for a new page.

Chapter 8

Construction of a many-body wavefunction

8.1 wavefunction

detemrinantset element instantiates a Fermionic wavefunction that will be added to its parent wavefunction as

```
<wavefunction target="e">
  <determinantset>
  </determinantset>
</wavefunction>
```

Fermionic wavefunctions are

$$\Psi_{AS} = \sum_{i=1}^{M} C_i D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi),$$

where $\{\phi\}$ denotes a set of single-particle orbitals (SPOs). A SPO can represented by a linear combination of a basis set, e.g., atom-centered spherical orbitals, as $\phi_i = \sum_\alpha C_{i,\alpha}|\alpha\rangle$.

The relationship among various classes are schematically given here:

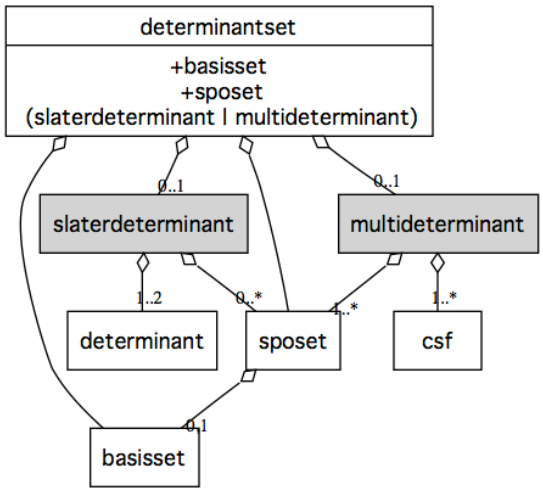

Figure 8.1: Fermion wavefunction

There are several builder/factory classes involved. The map between xml node and the responsible class is given

determinantset SlaterDetBuilder

basisset create BasisSetFactory

sposet create BasisSetFactory

backflow prepare BackFlow transformation

slaterdeterminant create SlaterDet or SlaterDetWithBackflow

multideterminant create MultiSlaterDeterminant

Within a scope, there is only one instance of each object, .e.g, a BasisSetFactory is used for both *basisset* and *sposet*.

Chapter 9

Frequently Asked Questions

9.1 General questions

9.1.1 Compilers

Use one of these compilers

- GNU 4.2.x higher : older GNU compilers do not support OpenMP
- Intel compilers
- IBM XL compilers

There is no plan to support other compilers by the developers.

9.1.2 Parallelization in QMCPACK

We use hybrid method MPI-X where X=OpenMP or Cuda. OpenMP is enabled by default, unless the compilers do not support OpenMP. In this case, we strongly recommend getting newer compilers.

MPI is optional but using it is highly recommended.

9.1.3 Numerical libraries

Blas/Lapack are essential for any scientific applications. On any platform, vendor provided numerical libraries should be used. Other factor to consider in selecting Blas/Lapack is the compilers.

These are used by the developers

1. MKL : Intel compilers on x86 platforms
2. atlas : GNU compilers on x86 platforms
3. LibSci : GNU compilers on Cray systems
4. Lapack/ESSL : on IBM systems

9.2 Running QMCPACK

9.2.1 Using multiple threads

Set OMP_NUM_THREADS:

```
export OMP_NUM_THREADS=12
./qmcapp input.xml
```

9.2.2 How to run QMCPACK with MPI

Check out what MPI library is used on your platform.

Often, using mpirun is all you need:

```
export OMP_NUM_THREADS=6
mpirun -np 2 ./qmcapp input.xml
```

9.3 Build problems

9.3.1 Failed to link blas/lapack

If the automatic detection fails with cmake, set LAPACK environment. For example in bash,

```
export LAPACK="-L/usr/lib64 -llapack -lblas"
```

and run *cmake* again.

On IBM systems, use ESSL and mass(v) libraries. Since ESSL does not support important functions of LAPACK used in QMPACK, it is always necessary to build and link LAPACK as well as ESSL library. Setting LAPACK environment should work, e.g.

```
export LAPACK="-L${LAPACK_LIBDIR} -lnameoflapack -lessl -lmass -lmassv"
```

Note that the name of LAPACK library and mass libraries (and where they are located) can vary on different systems. Consult the system documentation. If all these fail, check IBM Power running Linux as an example to make a toolchain file that works on your system.

9.3.2 Failed to link MKL

In general, toolchains files are updated to reflect the changes in MKL and synchronizing with the repository should fix the problems.

With Intel Composer XE 2011, MKLROOT is used to search header files and set the link flags. Check out your environment.

```
$ env | grep MKLROOT
MKLROOT=/opt/intel/composer_xe_2011_sp1.8.273/mkl
```

If problems persist, try these:

1. Check out Intel Math Kernel Library Link Line Advisor
2. Select sequential for "Select sequential or multi-threaded layer"
3. Set the result in "Use this link line:" to MKL environment variable

9.3.3 Failed to find XYZ package

CMakeCache.txt in your build directory can be edited to "fix" problems with any library.

When a library is not found, CMakeCache.txt has *SOMETHING-NOTFOUND* as

```
//Path to a file.
XYZ_INCLUDE_DIR:PATH=SZLIB_INCLUDE_DIR-NOTFOUND

//Path to a library.
XYZ_LIBRARIES:FILEPATH=SZLIB_LIBRARIES-NOTFOUND
```

Simply search and replace the path name for the reliated library.

- HDF5

```
//Path to a file.
HDF5_INCLUDE_DIR:PATH=/usr/local/gnu-4.6.3/hdf5-1.8.9/include

//Path to a library.
HDF5_LIBRARIES:FILEPATH=/usr/local/gnu-4.6.3/hdf5-1.8.9/lib/libhdf5.so
```

- libxml2

```
//Path to a library.
LIBXML2_LIBRARIES:FILEPATH=/usr/lib/x86_64-linux-gnu/libxml2.so

//Path to a file.
LIBXML_INCLUDE_DIR:PATH=/usr/include
```

- FFTW

```
//Path to a file.
FFTW_INCLUDE_DIR:PATH=/usr/local/gnu-4.6.3/fftw3.3/include

//Path to a library.
FFTW_LIBRARIES:FILEPATH=/usr/local/gnu-4.6.3/fftw3.3/lib/libfftw3.a
```

Chapter 10

University of Illinois/NCSA Open Source License

Developed by: Jeongnim Kim Condensed Matter Physics, National Center for Supercomputing Applications, University of Illinois Materials computation Center, University of Illinois http://qmcpack.cmscc.org/

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

developers.html . . . 51
faq.html . . . 51
forge.html . . . 51
gettingstarted.html . . . 51
inputxml.html . . . 51
license.html . . . 51
othertools.html . . . 51
qmcpack_main.html . . . 51
solids_cecam2012.html . . . 51
wavefunctionbuilder.html . . . 51

Chapter 12

File Documentation

12.1 developers.html File Reference

12.2 faq.html File Reference

12.3 forge.html File Reference

12.4 gettingstarted.html File Reference

12.5 inputxml.html File Reference

12.6 license.html File Reference

12.7 othertools.html File Reference

12.8 qmcpack_main.html File Reference

12.9 solids_cecam2012.html File Reference

12.10 wavefunctionbuilder.html File Reference