Section 1: Structure of code

qc-v<x>.<y> – main directory
Contains all folders and also script files which execute tasks.

QuasiSRC
Main source folder where all the source files are stored.

Info
Important files and directories.

Input
Contains input files and repository of input files. runQuasi.sh script copies input files from this folder to the folder where Quasi code is being built.

1.1 QuasiSRC

1. applications
Application directory. At present main.cc file, which is the main driver of QC code, is placed in nano-indentation subdirectory of applications directory.

2. applications/nano-indentation
It consists of main.cc and test.cc along with some make and configure related files. main.cc is compiled into executable “quasi” and test.cc is compiled into executable “test”.

We need to specify the correct location of the directory where libraries are built in Makefile.am. At present, I have built libraries in directory /usr/local. If you have built the libraries in some other <build-directory> change Makefile.am as follows

change -I$/usr/local/include to -I<build-directory>

main.cc is the main driver of QC code. It initializes various classes, reads from the input file, and also calls conjugate gradient minimizer.

test.cc is also the main driver QC code but it is for the purpose of debugging the input file, force and energy calculation, initial meshing of quasi. See quasi.ini.sample for various types of debugging flags implemented.

3. bin
This directory is generated by autoconf tools.

4. include

It consists of all the .h files of corresponding source .cc files in the src folder.

DataTypes.h contains a definition of various data structures and flags used in the quasi-code. If we want to add new unknown dof to each node of QC mesh, we need to modify

struct node_t {};

data corresponding to nodes.

5. mesh

The main file in this directory is mesh.cc.

mesh.cc is compiled into executable “mesh”. Use this executable to generate node input data, like NiAl.inp.gz in Input/NiAl, NiMn_LJ.inp.gz in Input/NiMn.

mesh.ini is the input file for mesh.cc

See mesh.ini.sample and also vast number of examples in Input/InputFilesRepository/Ar.

6. src

It consists of all the source files. If QC code is modified/corrected or new feature needs to be implemented, src and include are the two directories where changes will be needed.

Makefile.am is very critical here. If you have built the libraries, Boost, MPFR, Gmp, CGAL, Cmake, in the directory <build-directory>, modify the following in Makefile.am

-I$/usr/local/include to -I<build-directory>

Here is the brief summary of various source files

C++ Files

CGNonLinearSolver.cc

Conjugate gradient is implemented in this method. It also modifies the fixity masks of position and frequency degree of freedom. Therefore, if a new type of minimization or new unknown dof is added to data, we may need to modify this file as well.

C_Interface.cc

It consists of functions which help interaction between C++ objects and C functions.

CreateMesh.cc

Creates initial mesh and also remeshes the quasicontinuum.

CrossNeighborList.cc

Computed cluster data and also neighbor data if needed. Also, depending on the flag, it can update the location of cluster and neighbor data. See computeCrossNeighborlist() function.

Electrostatics.cc

Electrostatics related calculations are included in this class.

Element.cc

Operations on element data structures are included in this class. See DataTypes.h to know various types of data structures used in this code.

ForceEnergyCalculations.cc

Computes forces due to a different type of interactions like pairwise interaction, EAM interaction, entropic interaction, node-indentation interaction.

It has separate functions for the case of quasi-harmonic approximation.

Indent.cc

Indent related operations are included in this class. At present, we have not run any indentation problem.

Input.cc

It consists of two main functions mainInput() and quasiInput(). In mainInput(), we read from the init file, for example quasi.ini, initialize various classes.

In quasiInput(), depending on the flag, we will either read node data from .inp.gz file or from restart data.

Lattice.cc

Lattice related operations, like finding the location of lattice site, finding an element in which lattice site belongs, is put into this class.

MiscFunctions.cc

This contains functions which are used in the code.

Node.cc

Node data related operations are put into this class. If struct node_t is modified in DataTypes.h then we may need to modify Node.cc also.

NonLinearSolver.cc

CGNonLinearSolver.cc is a child of this class. This file can be left as it is.

Output.cc

This contains output related functions. At present, it outputs node related data and also restart data.

Since, we may output data in varieties of situations, for example, in between conjugate gradient iterations, after conjugate gradient iterations, output data just after creating initial mesh, output data to check force and energy at initial configuration, and also Conjugate minimization is of many different types, like fixing frequency and minimizing free energy wrt position, fixing position and minimizing free energy wrt frequency, minimizing free energy simultaneously wrt to position and frequency, minimizing alternatively.

We will need to pass flags specifying the situation in which output of data is performed. Therefore, the flags and what each flag mean is important to perform output of data.

PairPotentials.cc

It contains information related to interatomic potential and it also computed force and energy due to interatomic potential.

QuadraturePoints.cc

Currently, three-point quadrature method is implemented for phase average calculation. See multiscale QC formulation to know more about quadrature method.

Quasicontinuum.cc

In a multi-lattice crystal, each lattice is assigned, one Quasicontinuum.cc class. This consists of meshing of each lattice, a list of nodes of mesh, list of elements of mesh and other data corresponding to the lattice in a multi-lattice crystal.

Quasicontinua.cc

This class contains all the Quasicontinuum.cc class instances. Quasicontinuum instance corresponding to any lattice in a model can be accessed through this class object.

QuasicontinuaForceFunction.cc

This class acts as a bridge between Quasicontinua.cc, ForceEnergyCalculation.cc and CGNonLinearSolver.cc class.

This is a child of SolverFunction.cc.

RunData.cc

This class consists of functions related to time calculation of various operations in a QC code. We have not updated it recently.

Shape.cc

This consists of functions related to interpolation.

SolverFunction.cc
This class is a parent of QuasicontinuaForceFunction.cc. It can be left as it is.

Void.cc
This class contains void related functions.

C Files
monitor.c
It is related to multi-threading.

numa.c
It is related to multi-threading and system specific.

range_search_generic.c
Functions related to the search of the element the lattice site belongs to in a QC mesh is included in this file.

read_pipe.c
Read and write functions are in this file. It can be used to read from .gz file or write to .gz file.

site_element_map.c
Lattice site and element related functions are in this file.

site_element_map_cache.c
We create a cache which consists of sites and its element data.

threads.c
Threading related functions.

Fortran Files
aluminium.f
Fortran subroutine to compute cubic spline function.

Note: C and Fortran Files can be left unchanged. As they do not deal with QC formulation. Also basic classes like SolverFunction.cc, NonLinearSolver.cc can be left unchanged.

7. system
It consists of configure and autoconf related files.

8. triangulation_server

We use Delaunay algorithm to mesh. This directory consists of all the files related to meshing. This directory can be left as it is as it does not deal with QC formulation.

1.2 Input

Consists of the input file directories of a different model. Bash script, depending on the flag testName, will copy files from one of the directories here to the place where quasi executable is present.

InputFilesRepository

This directory consists of a repository of input files. It contains input data of the model of different sizes.
Mainly see Ar folder which has a vast number of input data. To generate input data of any other model do as follows

1.2.1 Creating input files for QC Code

Step 1

Go to the directory where mesh executable is present. Suppose we have specified the testName = "NiMn-NoElectrostatics", then mesh executable will be in

QuasiBuild/NiMn-NoElectrostatics/mesh

Step 2

Copy mesh.ini from InputFilesRepository/<model>/ to the present directory

Step 3

Open mesh.ini file to edit and also edit the mesh.ini in InputFilesRepository/Ar/<desired-mesh-size>/

Let us call the mesh.ini of InputFilesRepository/Ar/<desired-mesh-size>/ as source-mesh.ini and mesh.ini in current directory as target-mesh.ini

Step 4

Copy the information related to Mesh Info section of source-mesh.ini to Mesh Info section of target-mesh.ini

Step 5

execute mesh as follows

$./mesh

Step 6

Create a new directory, in InputFilesRepository/<model>/<desired-mesh-size> and copy the output data of ./mesh to this folder.

1.3 Info

We keep various QC code related files and directory in this directory. For example, it consists of small code in CheckLibrary to check the functioning of boost and CGAL library.

Section 2: Libraries required

We need following libraries to run the QC code

1. Boost
2. Cmake
3. Gmp
4. MPFR
5. CGAL

Note: For CGAL, only the version supplied with QC code is found to work. If one is interested in trying another version of CGAL, he/she can try to debug and compile the code, but we do not guarantee if the code will work with another version.

CGAL at file can be found in Info directory.

Note: We have included tar file of each of the library in Info directory. If you would like to use a recent version of the library, download this from the internet and run the command as mentioned in “Buil libraries” section.

2.1 Where to build the libraries?

1. If you are a desktop user and have administrative rights over Linux OS, you can build the libraries in the default location. Default location is

<build-directory> = /usr/local

2. If you are building QC in a supercomputer and also if using a desktop with no administrative rights, you can build the libraries in your home folder.

I will suggest to create a new directory “Softwares” and “Softwares/local” in your home directory and use it to build the libraries

<build-directory> = /home/<user-name>/Softwares/local

2.2 Modifying Makefile.am

Make sure that correct <build-directory> is put in following Makefile.am

1. QuasiSRC/applications/nano-indentation/Makefile.am

2. QuasiSRC/src/Makefile.am

Change -I/usr/local to -I<build-directory>

2.3 When to run autoreconf command?

In case you have modified Makefile.am and/or configure.ac then you must run following autoreconf command to generate updated configure and makefile files.

autoreconf -fvi -I<location-of-QuasiSRC>/system

You may run this command also if ./configure does not work.

2.4 Build libraries

Use the following command to build library:

Let <build-library> be the location where we are building the library.

Note: If <build-library> = /usr/local, you will need to use “sudo” infron of each command

1. Cmake

$tar -zxf cmake-3.5.0.tar.gz

$cd cmake-3.5.0

 # if building in user's home folder, e.g. /home/<user-name>/Softares/local

$./configure --prefix=<build-directory>

$./configure --prefix=<build-director>

$make

$make install

 # or if building in /usr/local

$sudo ./configure --prefix=<build-directory>

$sudo ./configure --prefix=<build-director>

$sudo make

$sudo make install

 # export the path

export PATH=$PATH:<build-director>/bin

2. Boost

$tar -zxf boost_1_60_0.tar.gz

$cd boost_1_60_0

 # if building in user's home folder, e.g. /home/<user-name>/Softares/local

$./bootstrap.sh –prefix=<build-directory>

$./b2 install

 # or if building in /usr/local

$sudo ./bootstrap.sh –prefix=<build-directory>

$sudo ./b2 install

 # export paths

$export PATH=$PATH:<build-directory>/include

$export PATH=$PATH:<build-directory>/lib

3. GMP

$tar --lzip -xf gmp-6.1.0.tar.gz

$cd gm-6.1.0

 # if building in user's home folder, e.g. /home/<user-name>/Softares/local

$./configure --prefix=<build-directory>

$make

$make install

 # or if building in /usr/local

$sudo ./configure --prefix=<build-directory>

$sudo make

$sudo make install

4. MPFR

$tar -zxf mpfr-3.1.4.tar.gz

$cd mpfr-3.1.4

 # if building in user's home folder, e.g. /home/<user-name>/Softares/local

$./configure --prefix=<build-directory> --with-gmp=<build-directory>

$make

$make install

 # or if building in /usr/local

$sudo ./configure --prefix=<build-directory> --with-gmp=<build-directory>

$sudo make

$sudo make install

5. CGAL

$tar -zxf CGAL-4.4.tar.gz

$cd CGAL-4.4

 # if building in user's home folder, e.g. /home/<user-name>/Softares/local

$cmake -DCMAKE_INSTALL_PREFIX=<build-directory>
-DBUILD_SHARED_LIBS=TRUE

$make

$make install

 # or if building in /usr/local

$sudo cmake -DCMAKE_INSTALL_PREFIX=<build-directory>
-DBUILD_SHARED_LIBS=TRUE

$sudo make

$sudo make install

Note: If Cmake and Boost are installed in a directory other than /usr/local, you may need to export the PATH to these libraries to install GMP, CGAL, MPFR. You can create a bash script file, see

Info/build-library-local.sh

Section 3: Running quasi

3.1 Building QC Code

1. Check if <build-directory> in Makefile.am in nano-indentation and src directory is correct.

2. Check if configure file is present in QuasiSRC directory.

3. Check the value of parameters in quasi_script_parameters.txt

4. Check if libraries like Boost and CGAL are functioning.

In Info/CheckLibrary/ directory, there are two simple programs which use CGAL and Boost. See if these files compile correctly and output of the executable is correct.

Execute the following command to build the quasi

$./runQuasi.sh

3.2 quasi_script_parameters.txt

We need to specify following parameters to run and build quasi:

1. runFlag

   0 – run quasi in QuasiBuild folder

   1 – run test in QuasiBuild folder

   2 – only make and/or configure

2. makeClean

   0 – do not run make clean in QuasiBuild folder

   1 – run make clean before executing make

3. config

   0 – do not configure the QuasiBuild

   1 – configure the QuasiBuild. To build quasi in any new folder, we have to first configure the folder by running

$cd <location-where-QC-will-be-built>

$<location-of-QuasiSRC-directory>/./configure

4. libraryFlag

   0 – libraries are built in location /usr/local

   1 – libraries are built in location specified by libraryDir

5. libraryDir

   location where libraries are built.

   It is used only if libraryFlag is set to 1.

   Example: /home/prashant/Softwares/local

6. testName

   format: <model>-<comments>

   Example: NiMn-Electrostatics, NiMn-NoElectrostatics, NiMn-Shear

   NiAl-Electrostatics, NiAl-NoElectrostatics, NiAl-Shear

runQuasi.sh will create the directory by name “testName” if it does not exist and build quasi inside this directory.

If we are creating this directory first time or reconfiguring the quasi, we first need to execute the following command

$cd <location-where-QC-will-be-built>

$<location-of-QuasiSRC-directory>/./configure

7. debugMode

   0 – no debug

   1 – outputs the name of various folders, files, and flags to check the reading of parameters by runQuasi.sh and to also check if required folders and libraries are present.

8. quasiBuildFlag

   0 – use default location to build QC

      the default location is QuasiBuild in the main directory

   1 – use directory specified here as QuasiBuild and build QC inside that directory.

9. quasiBuildHere

   location of the directory where we want to build quasi.

Example: /home/<user-name>/<directory-name>

10. quasiRunOutFlag

0 – output the QC debug data to shell

1 – output the QC debug data to the directory QuasiBuild/testName/OUTPUT.

Note: libraryFlag, libraryDir, quasiBuildFlag, quasiBuildHere, quasiRunOutFlag, debugMode are flags which one does not need to change often.

Once the directory in which libraries are built is fixed, the value of libraryFlag and libraryDir is fixed.

Decide where you want to build the QC, and keep using the same directory. We can leave the quasiBuildFlag to 0 and put any directory in quasiBuildHere (as it would not matter when quasiBuildFlag is 0).

3.3 Simple way to build and run QC

If one does not want to use runQuasi.sh script and quasi_script_parameters.txt, we mention the simple way to build and run QC.

1. Check if <build-directory> in Makefile.am in nano-indentation and src directory is correct.

2. Check if configure file is present in QuasiSRC directory.

3. Check if libraries like Boost and CGAL are functioning.

In Info/CheckLibrary/ directory, there are two simple programs which use CGAL and Boost. See if these files compile correctly and output of the executable is correct.

Step 1: Create a new directory inside QC directory, or in any other location. Let us assume this directory name is QuasiBuild.

Create another directory inside QuasiBuild which may or may not consists of the model name. Let us assume that we have created directory NiMn-NoElectrostatics inside QuasiBuild.

Step 2: Configure the build directory

Let us assume that QuasiSRC is the location of QuasiSRC directory of QC Code. Run configure in the build directory.

$cd QuasiBuild/NiMn-NoElectrostatics

$QuasiSRC/./configure

Step 3: Running the QC Code

Copy the input files from Input/NiMn/ to QuasiBuild/NiMn-NoElectrostatics/applications/nano-indentation

Run following command to run QC

$cd QuasiBuild/NiMn-NoElectrostatics/applications/nano-indentation

$./quasi

3.4 QC Input files

1. init file: quasi.ini

This file is read in Input::mainInput() function. It consists of parameters required in QC code. We have tried to explain the parameters and their use in quasi.ini file itself.

See quasi.ini.sample and quasi.ini in other directories in Input directory to know how one should modify quasi.ini for different models.

2. <model>.inp.gz

This file is read by Input::quasiInput() function. It consists of initial node data, their lattice site, and their initial location. It also consists of fix_mask which specified if particular dof is fixed or free corresponding to the node.

Example: NiMn_LJ.inp.gz, NiAl.inp.gz

3. restart.gz

This file is read by Input::quasiInput() function if restart flag is set to 1.