QC Tutorial Guide version 1.4

R. E. Miller and E. B. Tadmor

www.qcmethod.org

October 2011

Contents

1 Introduction

This tutorial takes the user, step-by-step, through an input file for the QC method. The simulation that is performed by this input file, while relatively simple, illustrates the main features of QC and emphasizes the important steps in executing a successful QC simulation. In the last section, a brief description of two other examples provided with the QC release are also provided.

The input file to be examined in depth here is `gb_shear.in`, found in the QC subdirectory `GB-example/Shear`. The problem that this particular input file addresses is illustrated schematically in Fig. 1. The model consists of two fcc Al crystals. The boundary between them is a perfect twin boundary except for a step, the height of which is equal to three (111) interplanar spacings. The QC simulation gradually applies a uniform macroscopic shear strain on the bi-crystal, eventually causing the boundary to migrate. The mechanism of this migration is the nucleation of two Shockley partial dislocations at the grain boundary step, followed by the motion of the partials along the boundary.

1.1 QC Directories and Files

First the distribution file `QC.tar.gz` must be unzipped and untarred:

```
% gunzip QC.tar.gz
% tar xvf QC.tar
```

This will create the main QC directory `QC`. Within this directory are the following subdirectories:

- `QC/Code`: Main source code.
- `QC/Docs`: Documentation including this *Tutorial* and the *QC Reference Manual*
- `QC/Potentials`: A library of embedded-atom method potentials and related files. In this directory, each potential includes three files:

  - `potential.fcn` contains the potential itself in a format that QC can understand.
  - `potential.mat` contains a file that describes the most common material (lattice structure) associated with the potential. See a discussion of this file in Section 3.3.
  - `potential.ref` is a set of notes reporting various details about the potentials and their source.

  For a comparison of the performance of some of these potentials in fully atomistic simulations, the user may refer to [12].

- `QC/GB-example`: This directory contains the user file `user_gb.f` containing an example of the user-defined routines that must be written for a specific simulation. In this case, it is the simulation of a stepped grain boundary to be discussed within this *Tutorial*. The subdirectory `GB-example/Shear` contains the input and output from the specific run to be discussed below.
- `QC/Punch-example`: An example of nano-indentation by a knife-edged indenter.
- `QC/Friction-example`: Shearing contact between two nano-asperities.

By convention, the QC directory structure is as follows. Directory names are given an upper-case first letter, all files are lowercase. Each specific model should have its own parent directory on the same level as `QC/Code`, with `Makefile` and a `user_APP.f` file (where `APP` is the application name). Specific simulations of a model should be run in a subdirectory of the model's parent directory. In the example provided, the parent directory is `QC/GB-example` and the simulation input and output is in `QC/GB-example/Shear`.

The `user_APP.f` must contain five user-supplied Fortran 90 subroutines that are required by QC. These are:

- `user_mesh`: This routine produces the mesh. It is called by the `mesh` command and is discussed in Section 3.7 and in Section 2.11.1 of the *QC Reference Manual*.

- `user_bcon`: This routine applies the boundary conditions. It is called by the `bcon` command and is discussed in Section 4.7 and in Section 2.11.2 of the *QC Reference Manual.*

- `user_pdel`: This routine defines a scalar measure of the applied force for a given boundary condition. It is called by the `pdel` command and is discussed in Section 4.9 and Section 2.11.3 of the *QC Reference Manual.*

- `user_potential`: This routines computes the energy (and corresponding force and stiffness) associated with a user-specified external potential. These contributions are added to the total energy of the system and its derivatives. This is an alternative approach to applying boundary conditions. It is discussed in Section 2.11.4 of the *QC Reference Manual.*

- `user_plot`: This routine gives the user the opportunity to create specialized plots for the defined application. It is called by the `plot` command and is discussed in Section 2.11.5 of the *QC Reference Manual.*

Even if some of these routines are not necessary for a particular application (which is normally the case), the empty routines must be included in `user_APP.f`.

To facilitate the creation of new applications, it is recommended to take an existing application as a starting point and to modify it appropriately.

1.2 Running the example

If the convention described above is followed, then all simulations will be compiled and run in the manner discussed here. The commands given below assume a Unix/Linux operating system.[1] The example simulation presented in this tutorial is executed by first compiling the QC program from within the `QC/GB-example` directory:

[1]The code has been ported in the past to a Windows platform without difficulty, but this is not discussed here.

```
% cd ~/QC/GB-example
% make QCCOMPILER=COMP
```

Here `COMP` is the appropriate compiler for your machine. Currently, the following Fortran compilers are supported:

- Absoft Fortran 90/95 Compiler (`COMP=absoft`)
- GNU Fortran (gfortran) Compiler 4.2.0 or higher (`COMP=gnu`)
- G95 Fortran Compiler (`COMP=g95`)
- IBM XL Fortran Compiler (`COMP=ibm`)
- Intel Fortran Compiler (`COMP=intel`)
- NAGWare Fortran 95 compiler (`COMP=nag`)
- PathScale Compiler Suite (`COMP=pathscale`)
- Portland Group Inc. PGF95 Fortran 90/95 Compiler (`COMP=pgi`)
- Silicon Graphics MIPSpro Fortran 90 version 7.3 or less (`COMP=sgi`)
- Silicon Graphics MIPSpro Fortran 90 version 7.4 or more (`COMP=sgi7.4`)
- Sun Forte Developer Fortran 95 (`COMP=sun`)

Others can be used by adding an appropriate section to `Code/Makefile_common`. See Appendix A for more details on the QC Makefile architecture.

Successful compilation will produce an executable called `gb` in the `GB-example` directory. To run the example do:

```
% cd ~/QC/GB-example/Shear
% ../gb < gb_shear.in > gb_shear.out &
```

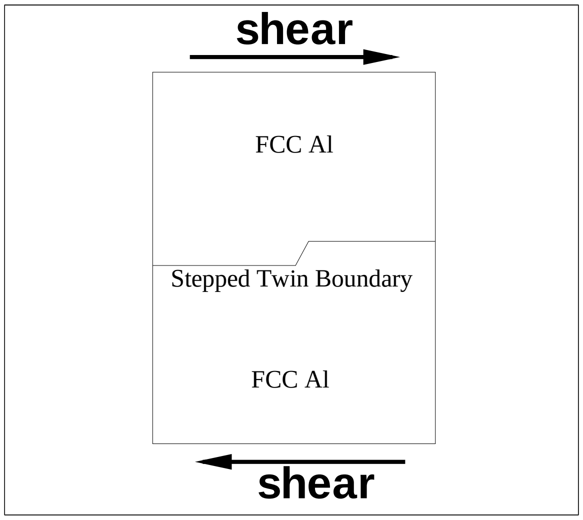

Figure 1: Schematic of the example simulation, shearing of a stepped twin boundary.

which reads `gb_shear.in` as standard input and sends all standard output to `gb_shear.out`. This example runs in a few minutes on a state-of-the-art workstation.[2]

All output files except the restart file discussed below are ASCII. The output files with a `.plt` extension can be viewed as mesh plots in Tecplot®, a commercial data visualization software package. Note that the main action of interest occurs between output files `gbshear_D008.plt` and `gbshear_D009.plt`, where the step migrates one atomic plane upwards due to two Shockley partial dislocations moving left and right along the boundary.

2 QC Input and Output

2.1 The QC Input File

The QC user interface is based on the simple command line interface originally provided with the program FEAP [11]. The advantages of this interface lie in its simplicity, ease of modification and portability (it requires no graphical capabilities and is written entirely in Fortran 90). Its main disadvantage is that is can be somewhat rigid in syntax and does not allow for easy debugging of faulty input. Commands are typically entered in a text input file (conventionally, but not necessarily, with a `.in` extension). The first action of the program

[2]See Appendix B for a list of execution times of the `GB-example` on a variety of computer architectures and using a variety of different compilers.

is to parse the input file into a temporary file called `qc.cmd`, in which all leading spaces and comment lines (denoted by a leading `%`) have been removed. The temporary file is deleted upon successful termination of the program[3], but faulty input files will often lead to an error message referring to the unit number corresponding to the `qc.cmd` file.

The commands take the general form:

```
command[,key][,data]
```

where `key` is used for some commands to activate various options. The `data` is generally a comma-delimited list of values required for the command, as explained in detail in the *QC Reference Manual*.

Note that for commands where no `key` is required or for which the default `key` is used, the commas must still delimit an empty field. In other words, such a command would appear as

```
command,,data
```

with the double-comma required.

Commands and keys are only unique up to the first four characters. Thus, `macr` and `macros` are the same as far as the program is concerned, but the latter form is recommended for better readability of input files. Commands and keys are not case-sensitive, but character data passed to commands are case-sensitive. To improve readability of the input file, indentation can be added using spaces, but tab characters must be avoided. No extra spaces should appear in the first 4 characters of the `key` field of the command line. Comment lines can be added anywhere in the input file by starting the line with a `%` character.

Commands are grouped into three simulation stages, which must appear in every input file for the QC program to run successfully. Each stage contains a series of commands, some of which are required and some of which are optional. The three stages (which must appear in this order) are: `head`, `macros`, and `stop`.

[3]The `qc.cmd` will not be deleted for some compilers that do not allow direct access to operating system calls from within Fortran. See Appendix A.1 for a discussion of the necessary `system` subroutine in module `mod_nonstandard`.

The `head` stage of the simulation is predominantly used for the initialization of simulation-specific variables, memory allocation, and the definition of the materials, constitutive laws and mesh. Normally, all of the `head` commands will appear in every QC simulation in the same order as presented in the following example, but with data and options appropriate to the simulation of interest.

The `macros` stage is the heart of the simulation, allowing the user a significant degree of flexibility in specifying, for example, the application of boundary conditions, the solution algorithm and the output generated. The `macros` stage also generally requires the greatest level of understanding and skill to produce meaningful results.

The final stage, `stop`, contains no commands and simply signals the program to gracefully terminate the simulation and stop execution.

The sample input file `gb_shear.in` appears in its entirety here:

```
head Shear Loading of a twin grain boundary with a step
2500,5000

% initialize flag settings
flag,NlocOn,T
flag,SurfOn,F
flag,GrainOn,T
flag,GhostOn,F

% initialize factor settings
fact,PROXFACT,2.0
fact,ADAPFACT,0.0
fact,CUTFACT,1.5
fact,epscr,0.15

% read in material definitions
mate,,1,../../Potentials/al_ea

% read in the no-adaption zones
zone,direct,,1
2
4
-100.d0 -50.d0
-100.d0 50.d0
-10000.d0 50.d0
-10000.d0 -50.d0
4
100.d0 -50.d0
10000.d0 -50.d0
10000.d0 50.d0
```

```
100.d0 50.d0

% read in grain information
grains,direct
2
1
-0.823028 2.32787 1.42552727
0.0 0.0 0.0
-1.0 -1.0 2.0
1.0 1.0 1.0
0.0
20.0
8
-200.d0 -6.88 F
-100.d0 -6.88 T
-2.46908 -6.88 T
0.0 0.1 T
100.d0 0.1 F
200.d0 .1 F
200.d0 200.d0 F
-200.d0 200.d0 F
1
2.46908  0.0 1.425527271
-0.05 0.0 0.0
1.0 1.0 -2.0
1.0 1.0 1.0
0.0
20.0
8
-200.d0 -6.8801 F
-200.d0 -200.d0 F
200.d0 -200.d0 F
200.d0 0.09999 F
100.d0 0.09999 T
0.0d0 0.09999 T
-2.46908 -6.880001 T
-100.d0 -6.880001 F

% read in constitutive information
cons,setf,1,../../Potentials/al_ea

% generate a simple coarse mesh
mesh,,8,8

end

macros
tole,,1.0d-6
proportional,,2,,0.,0.,1000.,1000.

% compute local/nonlocal status and automatically
% refine nonlocal regions
status
```

```
74     plot,bcon,gbshear,3,1.,1.
75     plot,disp,gbshear,0,1.,1.
76     plot,repatom,repatom

77     dtime,,0.005
78     loop,load,10
79        bcon
80        loop,,200
81           tang
82           solve,nr,1.0,10000,1
83           status,update
84           convergence,force
85        next
86        plot,disp,gbshear,0,1.,1.
87        report
88        pdel,,p-delta
89        restart,write,shear
90        time
91     next,load
92     end
93     stop
```

Note that the line numbers along the left margin are added only in this document for reference. In what follows, we will trace through these commands and identify the discuss the effect of each. Complete details regarding the functionality and options for each command, as well as all the remaining commands available to QC but not used in this example, are provided in the alphabetical listing of the commands in the *QC Reference Manual*.

2.2 QC Output

The QC method sends output in three main places: standard output, the `qc.log` file and any user-specified plot files for data visualization (see the `pdel` and `plot` commands in Sections 4.13 and 4.14 of the *QC Reference Manual*). The main output produced during the sequential execution of QC commands during a simulation is sent to standard output, commonly redirected to an output file by running the QC method as:

```
% qc < file.in > file.out
```

Here `file.in` is the input file and `file.out` will contain the QC standard output. The QC method writes to the `qc.log` file, on the other hand, whenever a potentially large list of data needs to be communicated to the user. This data is written to `qc.log` rather than to the standard output in order to maintain readability of the standard output file. Whenever there is an entry written to the `qc.log` file, it is referenced in the standard output.

Throughout this tutorial, we will draw the reader's attention to segments of the standard output or `qc.log` file as they are generated by the various commands in `gb_shear.in`.

2.3 QC File Extensions

All files that are either read or written by the QC program during a simulation have standardized three-letter extensions to make is simpler for the user to identify the purpose of each file. Because the standard three-letter extension is required for every QC file, the user specifies a file by only the prefix, and the appropriate extension is automatically assumed by the program. For example, at line 14 of the input file `gb_shear.in`, a material is defined by reading a file `al_ea.mat`. The command includes only the prefix, `al_ea`, to this file:

```
mate,,1,../../Potentials/al_ea
```

Since the standard extension for files read by the `mate` command is `.mat`, the QC code will automatically append this extension before trying to open the file.

Table 1 presents a brief description of the function of all the QC file types, their extensions and the commands that access them. Further detail regarding the function of each file is provided in the appropriate sections of this tutorial and the *QC Reference Manual.*

3 Stage: `head`

The first line of every QC simulation takes the form of line 1 in `gb_shear.in`. The stage name `head`, followed by a user-specified header that identifies this particular simulation. The header may be up to 80 characters long, and must appear entirely on line 1. This line tells QC to begin the `head` stage, which in turn expects the second line of the input file to be of the form:

```
maxnp,maxel
```

This line is required for the allocation of an appropriate amount of memory. The entries

File Type	**Extension**	**Command**	**Function**
Material	`.mat`	`mate`	Definition of a material.
Special-Attribute Zone	`.zon`	`zone`	Geometry of the polygons defining "special-attribute" zones (e.g. zones in which mesh adaption is not allowed or zones where all all atoms must be nonlocal).
Interatomic Potentials	`.fcn`	`cons`	Grids defining the interatomic potentials, in DYNAMO [1] `setfl` or `funcfl` format.
Grain Geometry	`.geo`	`grai`	Grain definitions.
Proportional Load Table	`.prp`	`prop`	Table defining the dependence of the load variable `prop` on the `time`.
Load-Displacement	`.pde`	`pdel`	Output file containing the load vs. displacement data as defined for a specific simulation.
Binary Restart	`.res`	`rest`	Binary format restart file.
Text Restart	`.trs`	`rest`	Text format restart file (about 3 times larger than binary files, but portable. across platforms).
Tecplot® output	`.plt`	`plot`	Tecplot®-ready output, typically including the repatoms, mesh and output data.

Table 1: Standard QC files.

`maxnp` and `maxel` are, respectively, the largest number of representative atoms and the largest number of elements expected during the simulation. These can be difficult to predict if adaptive remeshing is being used, but programs which terminate due to either value being exceeded can be easily restarted with a larger allocation if the `restart` option described below (in Section 4.10) is used correctly. As a rule of thumb, `maxel` should be between 1.5 and 2 times larger than `maxnp`. In this example, QC will allocate storage for 2500 repatoms and 5000 elements.

The start of the `head` stage generates the following header in the standard output:

```
--------------------------------------------------------------------------
```

```
q u a s i c o n t i n u u m   m e t h o d   s i m u l a t i o n

Quasicontinuum (QC) Method: Mixed Continuum and Atomistic Simulation Package
QC Package distribution version 1.4 (November 2011)

Copyright (C) 2003 R. Miller, D. Rodney, M. Ortiz, R. Phillips,
                   V. B. Shenoy and E. B. Tadmor
Copyright (C) 2004, 2005, 2006, 2007, 2011  R. Miller and E. B. Tadmor

Visit the QC website at www.qcmethod.org

-----------------------------------------------------------------------------


Shear Loading of a twin grain boundary with a step
```

where the last line is an echo of the header provided on line 1 of the input file.

Following the memory allocation parameters on line 2, the input file can contain one or more of the `head` stage commands. For most simulations, the commands which appear and their order should be as presented in this example.

The general format for each command is

```
command[,key][,data]
```

where `key` is a command-specific key that distinguishes between the various options available to a command. A `key` that is common to several commands during the `head` stage is `direct`. Several commands have the option of either reading a large amount of data directly from the input file, or from another auxiliary file. If the `dire` key appears, the program will read the subsequent lines of the input file to find the data it needs. Otherwise, the command will expect the `data` on the command line to contain the names of one or more files in which to find the data. The `direct` option is useful for keeping all the input data for a simulation in one file. On the other hand, use of separate files can be useful if several simulations require the same data or if the number of data entries becomes long and affects the readability of the input file. This will be demonstrated by contrasting the `grains` and `materials` commands later in the following discussion.

3.1 Lines 3-7: `Flag` Settings

The `flag` command is used to set various true/false flags for the simulation. A detailed list of the possible flags, their effect on the simulation and their default values, is provided in the *QC Reference Manual* entry for the `flag` command. The `key` for the `flag` command identifies the variable to be set, and the `data` is either `T` or `F` as appropriate[4]. The `flag` commands can appear in any order and at any point in the `head` stage of the QC input file. In this example, the following flags are set:

- *Line 4.* `NlocOn=T`: Enables nonlocal repatoms. `NlocOn=F` runs a purely local simulation.
- *Line 5.* `SurfOn=F`: Free surfaces will not trigger nonlocality in this simulation, even though they should to correctly account for surface energetics. This approach can be used to reduce the computational effort in simulations where the surface energy effects are not important to the phenomena of interest. If `SurfOn=T` then repatoms within `PROXFACT*rcut` of active free surfaces will be made nonlocal. The significance of `PROXFACT` and `rcut` is explained below in Section 3.2. Note that for surface effects to be activated, both `SurfOn` and `NlocOn` must be set to `T`.
- *Line 6.* `GrainOn=T` The analogue of `SurfOn`, for active grain boundaries. For grain boundaries to trigger nonlocality, both `GrainOn` and `NlocOn` must be set to `T`. This is the case in this example, where we are interested in studying the details of the migration of a grain boundary step under applied shear stress.
- *Line 7.* `GhostOn=T` Activates the ghost force correction to mitigate spurious forces at the local/nonlocal interface.

Each call to the `flag` command generates two lines in the standard output of the form:

[4]The `flag` subroutine searches the `data` for the first incidence of either an "f" or a "t" in upper or lower case, so other common expressions for true or false can also be used.

```
**setting flag**
  NlocOn = TRUE
```

identifying which flag has been set and the value. Note that an unrecognized `key` sent to the `flag` command does not cause termination of the simulation, but rather leads to a warning issued to standard output of the form:

```
**setting flag**
  WARNING:  flag name "xxxx" not recognized
```

3.2 Lines 8-12: `Factor` Settings

The `fact` command is like the `flag` command except that it is used to set real-valued factors instead of true/false flags. A detailed list of QC's `factors`, their effect on the simulation and their default values is provided in Section 3.3 of the *QC Reference Manual*. The `key` for the `fact` command identifies the variable to be set, and the `data` is a double-precision real value to be assigned to the factor. The `fact` commands can appear in any order and at any point in the `head` stage of the QC input file. In this example, the following factors are set:

- *Line 9.* `PROXFACT=2.0`: `PROXFACT` is used to determine the range of nonlocal effects. During a simulation, a repatom is made nonlocal if any pair of elements within `PROXFACT*rcut` of the repatom, where `rcut` is the atomistic potential cutoff radius, have significantly different deformation tensors. A "significant difference" is determined by comparing the difference between the eigenvalues of the elemental right Cauchy-Green deformation tensor $\boldsymbol{C}$ to the factor `epscr` defined below[5]. Specifically, taking the eigenvalues of $\boldsymbol{C}$ in two elements a and b to be λ_k^a and λ_k^b ($k = 1 \ldots 3$), nonlocality is triggered if

$$\max_{a,b;k} |\lambda_k^a - \lambda_k^b| > \texttt{epscr}. \tag{1}$$

  As well, repatoms are made nonlocal if they are within `PROXFACT*rcut` of an active grain boundary or free surface. Note that the flags `NlocOn`, `SurfOn` and `GrainOn` must be set to true for these nonlocal effects to be activated.

[5] $\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F}$, where $\boldsymbol{F}$ is the deformation gradient. For details, see for example [4].

- *Line 10.* `ADAPFACT=0.0`: `ADAPFACT` is used during adaptive mesh refinement. Elements within `ADAPFACT*rcut` of any nonlocal repatom are targeted for adaption whenever the `adap` macro is called. This is in addition to adaption triggered by the error norm defined in the discussion of the `adap` command in Section 4.1 of the *QC Reference Manual*. In this example, since `ADAPFACT=0.0`, the QC simulation will not use proximity to a nonlocal repatom as a trigger for element adaption. For simulations where `ADAPFACT` is used, it is typically set to 2.0.

- *Line 11.* `CUTFACT=1.5`: Each nonlocal repatom requires a list of atoms which, in the deformed configuration, are within `rcut` of the repatom for energy and force calculations. Because the search for neighbors is a computationally expensive process, it is considerably faster to store a slightly larger sphere of neighbors, within `CUTFACT*rcut`, and only update the neighbor lists when the displacements since the last update are large enough that additional neighbors may have moved into range. In this example, the effective cutoff is made 50% larger than the actual cutoff radius. See the discussion of the `stat` command in Section 4.21 of the *QC Reference Manual* for more on this technique of neighbor list storage.

- *Line 12.* `epscr=0.15`: The factor `epscr` is used to determine if a repatom must be made nonlocal due to significant variations in the deformation gradients around the repatom. See [5] for details.

Like the `flag` command, each call to the `fact` command generates two lines in the standard output of the form:

```
**setting factor**
  CUTFACT = 1.500
```

to echo the user input. Note that an unrecognized `key` sent to the `fact` command does not cause termination of the simulation, but rather leads to a warning issued to standard output of the form:

```
**setting factor**
  WARNING:  factor name "xxxx" not recognized
```

3.3 Lines 13-14: `Material` Command

The `material` command is used to define the materials for the simulation. In QC, a "material" is very specifically defined to mean a Bravais lattice of three vectors and an atomic number for the atom associated with each Bravais lattice site. This version of the code is limited to simple Bravais lattices, i.e. those having a single atom at each Bravais site. The Bravais vectors are assumed to be in units of length (typically Å) and referred to the global x-y-z Cartesian coordinate system. The atom type is identified by its atomic number.

In this example, since the `direct` key has not been specified for the `material` command, the material definitions are expected to reside in separate files. The `data` following the command `key` (in this case, the double-comma since no `key` is present) is a `1` to indicate that one material file will be read in, followed by the name (without the `.mat` extension) of that file: `../../Potentials/al_ea`. The entire contents of this file appear here:

```
'Al(fcc) - ea'
0.d0 2.0160413497439759d0 2.0160413497439759d0
2.0160413497439759d0 0.d0 2.0160413497439759d0
2.0160413497439759d0 2.0160413497439759d0 0.d0
13
```

This data includes a `character(len=80)` string used to identify the material, followed by the three Bravais vectors, $\boldsymbol{a}_1$, $\boldsymbol{a}_2$ and $\boldsymbol{a}_3$. These vectors can be non-orthogonal but should form a right-handed basis set, *i.e.* $(\boldsymbol{a}_1 \times \boldsymbol{a}_2) \cdot \boldsymbol{a}_3 > 0$. The next line is the integer atomic number of the atom. In this example, an fcc lattice of aluminum is defined with lattice constant $a_0 \approx 4.032$ Å, the value ≈ 2.016 Å being $a_0/2$ in the Bravais vector definitions. This is the equilibrium, zero-temperature lattice constant for the Ercolessi and Adams potentials [3] used in this example. Note that one cannot, for example, use a generic "fcc aluminum" material file. This is because each set of interatomic potentials may give slightly different values for the lattice constants.

Note that the current release of QC does not permit complex lattice materials, although this is a topic of ongoing research in an effort to make the QC method more general. Thus, QC is currently limited to modeling crystals that can be defined by a single atomic number and Bravais lattice. It is possible, however, to model more than one atom type in the same simulation. For example, it is possible to look at the phase boundary between an fcc aluminum grain and a bcc Fe grain. On the other hand, it is currently not possible to look at a single crystal structure containing both Al and Fe.

The material command echoes the material data to the standard output, for this example, as follows:

```
 **reading material data**
   opening material data file ../../Potentials/al_ea.mat

   material information

   Mat # 1 : Al(fcc) - ea
   Bravais lattice vectors:
   a1 =    0.00000   2.01604   2.01604
   a2 =    2.01604   0.00000   2.01604
   a3 =    2.01604   2.01604   0.00000
   Atomic Species = 13
```

3.4 Lines 15-27: `Zones` in which mesh adaption will not be allowed.

In some simulations, it is desirable to restrict the automatic mesh adaption algorithm so that certain regions are not refined. For example, in grain boundary simulations, the re-arrangement of the atoms near the grain boundary which will occur to obtain the minimum energy boundary structure can induce large deformation gradients in the region. This can trigger repatoms to become nonlocal, which will further trigger adaption of the neighboring regions. These newly adapted regions will then rearrange to the minimum energy boundary structure, and the cycle will repeat. The result is that a grain boundary will fully refine itself along its entire length. This will be computationally expensive and is likely not desirable, as the reason for using the QC method to study a grain boundary would likely be its ability to atomistically resolve only select regions of the problem.

To prevent the scenario just described, special-attribute zones where adaption is not allowed to occur can be defined. Other possibilities for special-attribute zones can exist. In this case the index 1 at the end of the `zone` command on line 16 indicates that the defined zones are no-adaption zones. See Section 3.8 of the *QC Reference Manual* for more details.

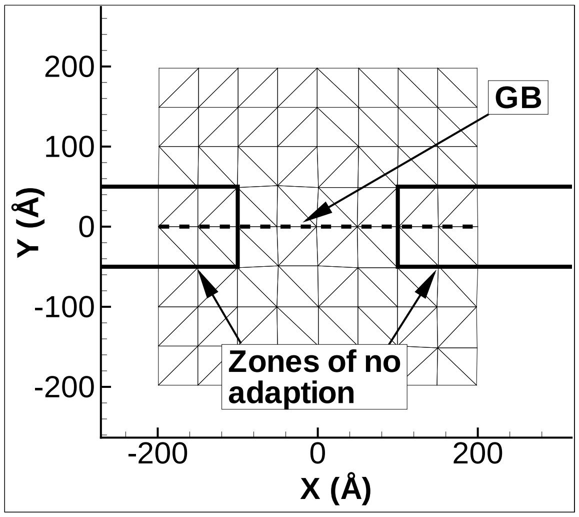

Figure 2: Zones where no adaption will take place, as defined by the `zone` command.

In this example, a zone is defined at each end of the segment of grain boundary where the deformation of interest is expected to occur. These zones are shown in Fig. 2. Since the `direct` key has been specified on line 16, the data for the `zone` command appear directly in the input file. These lines are

- *Line 17.* `nzones.` The number of zones to be defined.
- *Line 18.* `nvertices.` The number of vertices that will be used to define the first zone.
- *Lines 19-22.* `vertices.` The x and y coordinates of the vertices defining the polygon of the first zone, listed counter-clockwise around the zone. In this example, each of the two zones is a rectangle.
- *Lines 23-27.* The same details for the second zone.

3.5 Lines 28-62: `Grain` Definitions

A grain in the QC method is composed of a material, as defined by the `material` command, a rotation which orients the material relative to the global x-y-z axes, and a polygon in the x-y plane. Grain polygons cannot overlap, and all elements in the mesh (defined later) must be inside one or more grains (elements which occupy more than one grain will be assigned to the grain in which their centroid resides, hence the actual path of the grain boundary will always follow a set of contiguous element edges in the mesh).

In practice, it is advantageous to define the grain polygons to extend far beyond the space in which it is anticipated that the elements of the problem will be defined. In this way, the model mesh can be thought of as having been "cut-out" from a much larger region and errors of elements lying outside any defined grain can be avoided.

Note that even in a single crystal example, one grain must still be defined. In that case, the grain serves only to define the orientation of the crystal, as the material occupying it is normally defined by the `material` command to have the natural crystal orientation (for example, an fcc lattice is typically defined by the `material` command to have the $\langle 100 \rangle$ cube axes along x-y-z, so the `grain` command is used to re-orient the crystal as desired). In this way, a material like Ercolessi/Adams [3] fcc aluminum needs to be defined only once, using the file `al_ea.mat` as read by the `materials` command, and each simulation can use this material in conjunction with different grain definitions to produce different crystal orientations.

It is convenient to define the orientation of a grain using a two-step process. First a "natural" set of coordinates is defined based on the definition of the Bravais lattice vectors of the material which will make up the grain. So, for example, if the material is fcc, it may be convenient to choose the $[1\bar{1}0]$ and $[111]$ directions as the natural x and y directions of the grain. However, if the simulation of interest involves two grains with a small tilt angle between them, an additional rotation about the z axis is needed to rotate the grains relative to one another. Hence the second step is to define a rotation of the natural crystal orientation around the global z axis.

In this simple example, the twin-boundary being considered is naturally defined by the $\langle 111 \rangle$ and $\langle 112 \rangle$ crystallographic directions aligned parallel and perpendicular with the boundary plane as shown in Fig. 3(a). However, to consider the slightly different boundary shown in Fig. 3(b), one would need to specify the cumbersome crystallographic directions

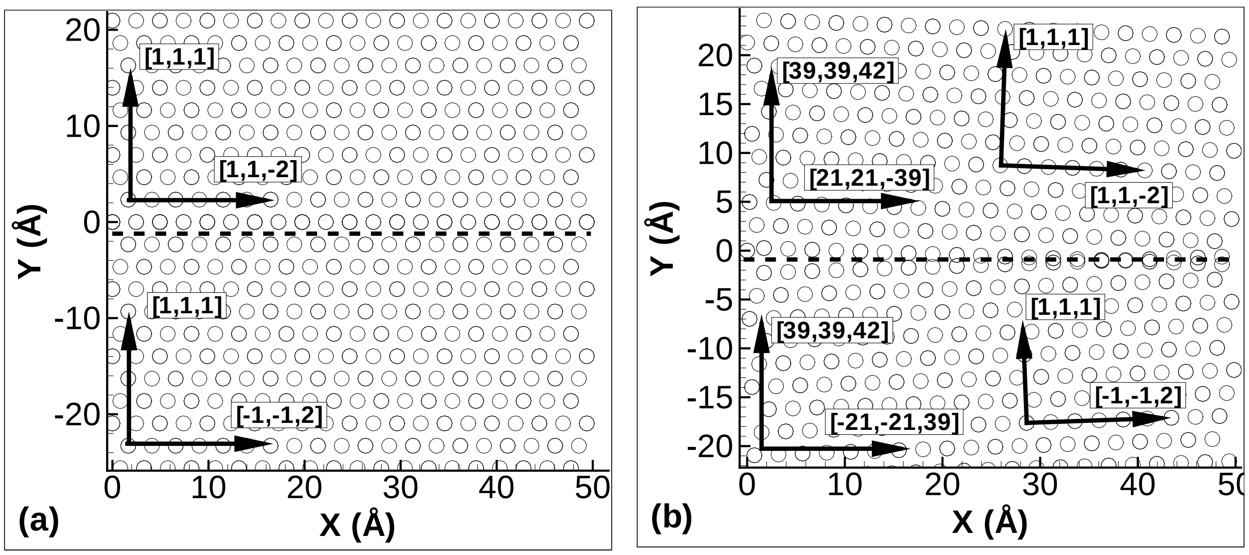

Figure 3: An example grain boundary highlighting details of its construction using the QC code.

$\langle 21, 21, 39\rangle$ and $\langle 39, 39, 42\rangle$ to correctly align these grains with the global x and y directions. However, the grains in Fig. 3(b) are the same as those in Fig. 3(a), but for a rotation of $\pm 2.03^o$ about the z-axis, so it is more convenient to first define the "natural" crystallographic orientation of Fig. 3(a), and then rotate these grains to their final orientation of Fig. 3(b).

In the example input file `gb_shear.in`, the key `direct` is specified, so that QC will expect the grain data to appear immediately following the `grains` command rather than in a separate file as was demonstrated with the `materials` command above. This data, in lines 30-62, includes the following entries:

- *Line 30.* `ngrains`. Specifies the number of grains that will be defined for the simulation. In this case, there will be two grains. Lines 31-46 define the first grain, while lines 47-62 define the second.
- *Line 31.* `material`. Specifies the material number (as defined by a previous `material` command) that will occupy the first grain.
- *Line 32.* `reference atom.` The coordinates (x, y, z) of one atom in the first grain, chosen as the `reference atom`, which will be used as the origin of the Bravais lattice

for the grain. The coordinates of the reference atom must be inside the grain polygon (defined below).

- *Line 33.* `shift vector.` The `shift vector` can sometimes be helpful in defining the grain such that the details of the grain *boundary* structure will be as the user wishes. The QC will build grain boundaries by constructing the individual grains. In the regions near the boundary, where atomic scale grain boundary structure is important, QC will simply include all atoms that are on the Bravais lattice defining the grain and whose reference (undeformed) position is inside the polygon defining the grain. In some cases, this may create a boundary structure with atoms that are non-physically close together. This will happen, for example, when two atoms within neighboring grains that are both geometrically "inside" their respective grains are both included in the model despite being only a fraction of an Angstrom apart across the boundary.

  The `shift vector`, $\boldsymbol{x}_s$ is one tool at the user's disposal to avoid this situation and build the desired grain boundary structure. The vector represents a small translation that will be applied to an atomic position before the test is performed to check if the atom is "inside" the grain and therefore to be included in the model. Specifically, in the process of building all the atomic positions in a grain "A", QC attempts to include an atom at position $\boldsymbol{x}$. To decide whether or not to include this atom, QC tests whether the point $\boldsymbol{x} + \boldsymbol{x}_s$ is inside the polygon defining grain A. If so, an atom is included at position $\boldsymbol{x}$. More detail about using the `shift vector` is provided when discussing Fig. 4 at the end of this section.

  Producing QC simulations in which the grain boundary structure is correct requires considerable skill and *a priori* knowledge of what the correct structure should be, usually gained from small-scale, direct atomistic simulations of the boundary structure. For more information on computing grain boundary structures using atomistics, see, for example, [6, 7].

- *Line 34* `‘‘Natural’’` x`-axis direction`. A vector, referred to the coordinate system used to define the Bravais lattice of the material (see the `material` command above) which points along the crystallographic direction that the user wishes to use to as the natural x-direction for the grain. In the example of Fig. 3, this is the $\langle 11\bar{2} \rangle$ direction rather than the $\langle 21, 21, \bar{3}9 \rangle$ direction.

- *Line 35* `‘‘Natural’’` y`-axis direction`. A vector, referred to the coordinate system used to define the Bravais lattice of the material (see the `material` command above) which points along the crystallographic direction that the user wishes to use as the natural y direction for this grain. This vector must be orthogonal to the x-axis direction defined on line 34.

- *Line 36,* `Rotation.` As illustrated in the example of Fig. 3(a) and Fig. 3(b), this is a rotation angle in degrees (positive anti-clockwise) about the z-axis. This rotation takes the material in the grain from the “natural” orientation defined by the crystallographic directions of lines 34-35 and rotates it to the final desired orientation with respect to the global x, y and z directions. In this particular example, the simple twin boundary being constructed requires no rotation from the natural orientation, so `Rotation=0.d0`.

- *Line 37,* `crystal radius`. Local energy calculations are performed by deforming a small crystallite according to the current deformation gradient and computing the energy of a representative atom at its center. This crystallite can be built in advance, and stored or built at each energy computation. The former approach requires a conservatively large crystallite to be stored, but is preferable to the latter approach which significantly increases the computational effort. The crystal radius specifies the size of the representative crystallite to be stored. Larger crystals will not significantly slow the computation but will increase the memory requirement of the simulation.

  Typically, as in this example, a crystal radius of about three times the cutoff radius of the atomistic potentials being used is a suitable choice.

- *Line 38,* `Number of Polygon Vertices`. The final data required for the grain definition is the polygon defining the region of space that will be occupied by the grain. Thus, this line indicates the number of vertices that will follow to define that polygon.

- *Lines 39-46,* `Polygon Vertices.` Each of these lines is the x and y coordinates of a vertex at the start of linear segment of the polygon defining the grain and the flag "T" or "F" indicating whether this segment is to be considered active during the simulation. If a segment is active, it will trigger nonlocality in its vicinity, defined as the region of space within `PROXFACT*rcut` of the segment. The vertices must be listed in order, going counter-clockwise around the polygon.

- *Lines 47-62.* The details of the second grain.

In this example, we have defined two adjacent grains, with the grain boundary lying approximately along the plane $y = 0$. The polygons defining the grain have a small jog in them near the origin, such that at the atomic scale there will be a step in this boundary of height 7.0 Å.

The construction of this boundary is elaborated in Fig. 4, where in Fig. 4(a) we show a segment of the grain polygon, the reference atom of each grain, and the Bravais lattice vectors of grain 1 ($\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$) and grain 2 ($\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $\boldsymbol{b}_3$). The filled circles are the atoms built from the grain 1 Bravais lattice which are included in the model because they fall inside the polygon for grain 1. The hollow circles are similarly the atoms built for grain 2. The hollow square at the center of the figure illustrates the use of the shift vector, as it represents an atom that *could* potentially be part of grain 2, but is explicitly *not* included in the model. This is illustrated further in Fig. 4(b), which is a close-up of the hollow square. For $\boldsymbol{x}_s = 0$, this atom would be included as one of the repatoms in the lower grain. However, a previous atomistic simulation showed that to include this atom would lead to a grain boundary structure with higher energy. Thus, by using the shift vector shown, this atom was not included in the initial configuration of atoms around the grain boundary. The

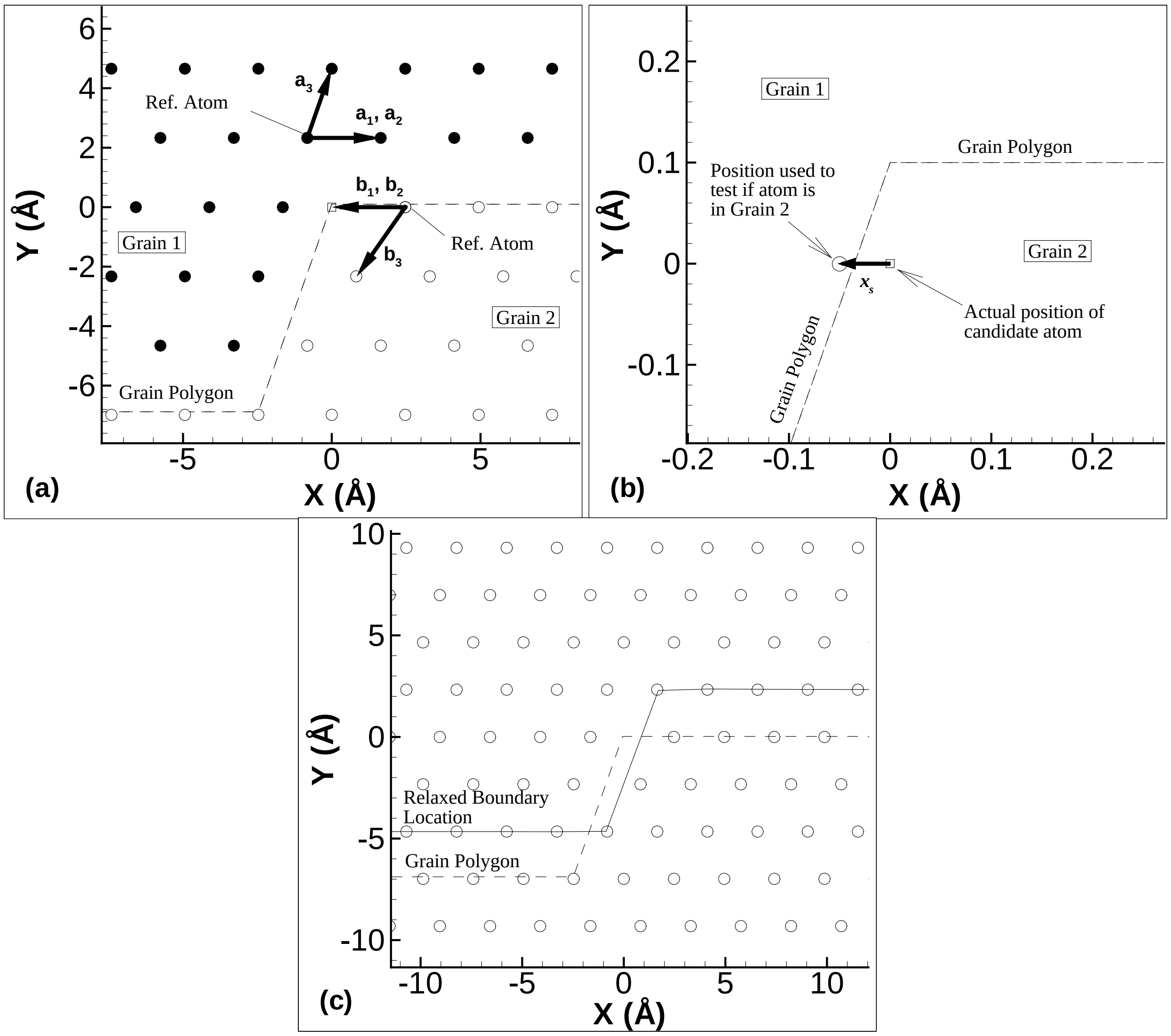

Figure 4: Details of constructing a grain boundary in QC. (a) The grains are defined by all Bravais lattice sites lying inside the grain polygon. The square atom at (0,0) is not included because of the user-defined `shift vector` as shown in (b). The final relaxed grain boundary structure is shown in (c), with the original grain polygons shown as a dashed line and the final location of the boundary shown as the solid line.

relaxed configuration of the grain boundary, under no applied stress, is shown in Fig. 4(c).

The `grai` command processes the user-defined grains by computing a number of geometric quantities. These are echoed to the user in the standard output. For the first grain in this example, the output is as follows:

```
==================================================
  Grain    1
    Material    1   Al(fcc) - ea
    Reference Atom at
       -0.82303   2.32787   1.42553
    Lattice Vectors
       -0.40825  -0.40825   0.81650
        0.57735   0.57735   0.57735
       -0.70711   0.70711   0.00000
    Rotation (degrees)    0.00000
    Number of vertices =    8
       -0.20000E+03   -0.68800E+01  F
       -0.10000E+03   -0.68800E+01  T
       -0.24691E+01   -0.68800E+01  T
        0.00000E+00    0.10000E+00  T
        0.10000E+03    0.10000E+00  F
        0.20000E+03    0.10000E+00  F
        0.20000E+03    0.20000E+03  F
       -0.20000E+03    0.20000E+03  F
    Cell Structure
    Wigner-Sietz Area :        5.7479617393249740
    Cell dimensions
        4.93827   6.98377   2.85111
    Cell Bravais Lattice sites :   6
        0.00000E+00    0.00000E+00    0.00000E+00
        0.24691E+01    0.00000E+00    0.14256E+01
        0.82305E+00    0.23279E+01    0.14256E+01
        0.32922E+01    0.23279E+01    0.00000E+00
        0.16461E+01    0.46558E+01    0.00000E+00
        0.41152E+01    0.46558E+01    0.14256E+01
    Radius of representative crystal :   19.957791
    Number of atoms in crystallite :  2093
==================================================
```

A similar output is produced for grain 2. This output includes the direct user input as well as computed quantities. For example, the `Lattice Vectors` include the user-specified x and y coordinate directions, as well as the z direction which is `(-0.70711, 0.70711, 0.00000)` in this case. Note that specification of non-orthogonal x and y coordinate directions will lead to an error message and program termination.

The `Cell Structure` of each grain includes a computed `Wigner-Sietz Area` (which is the 2D projection of the primitive unit cell of the Bravais lattice) and a non-primitive unit cell of atoms. This non-primitive cell is the smallest periodic cell that is both (a) orthogonal with the natural axes and (b) able to fully define the grain's crystal structure. In this example, this cell contains 6 atoms, as illustrated in Fig. 5.

A word of caution is necessary due to the essentially 2D nature of the QC program. In this example, the cell dimension along the out-of-plane direction, `dz=2.85111`, is the same for the two grains we have defined. This is sensible for grain boundaries like this one, which is a simple tilt boundary. Note that for many boundaries, the two grains will have incommensurate periodicities along the out-of-plane direction. The QC will correctly treat grains with different values of `dz`, but only insofar as it will correctly compute the energy and forces on the structure. It is not capable of relaxing the structure in the out-of-plane direction, which generally would require accommodations such as misfit dislocations with $x-$ or $y-$ line directions and other deformations where the displacement field varies along the z direction.

Finally, the user-specified crystal radius is used to compute and store the representative crystallite, and the `number of atoms` in this crystallite is echoed to standard output.

3.6 Lines 63-64: `Constitutive` information

The next segment of the input file defines the constitutive information. In the current implementation of the code, this effectively means the pair-functional embedded-atom method (EAM) atomic interactions[6]. There is currently only one module available for the QC `constitutive` command, designed to read EAM potentials from a file as described below. In principle, this module (`mod_poten_eam.f`) can be replaced to use any other form of atomistic interactions where the total energy can be written as a sum over individual atom energies (e.g. three-body interactions). See Appendix A for information on how to change

[6]Simple pair potentials, without the pair functional terms of the Embedded Atom Method, can be implemented by setting the electron density and embedding energy functions to zero.

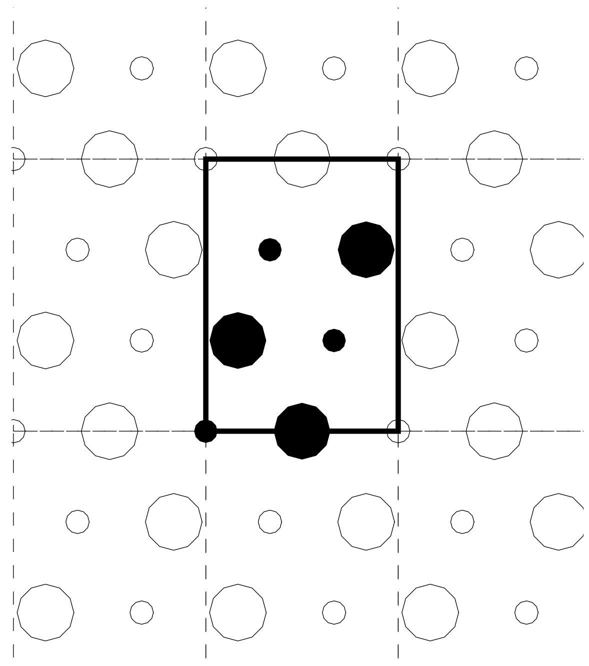

Figure 5: The filled circles and dark box are the minimum periodic cell for grain 1 of the example, while the hollow circles are the periodic copies that produce the crystal structure. Size of the atom indicates the two different planes of the structure in the out-of-plane direction.

potentials in the make file.

The existing `cons` module uses a general EAM potential in one of the formats defined by the EAM molecular dynamics code DYNAMO [1]. The module accepts a grid of discrete points to define the functions, which during the simulation are interpolated using Lagrange interpolations.

In this example, the key `setf` on line 64 tells the QC that there is a single file containing all interactions in DYNAMO "setfl" format, whereas `func` would indicate that a number of separate material files, in DYNAMO 'funcfl" format, will follow to define each material. In the latter case, pair potential interaction between atoms with different atomic numbers are approximated using the geometric mean of the homonuclear pair potentials [2]. The first item in the `data` is a `1` to indicate that one file is to be read in, followed by the filename (without the `.fcn` extension): `../../Potentials/al_ea`. Note that for the `setf` key, there can only be one file by definition, but if the `func` key is used there can be many. In this example, the file contains the EAM potential data for fcc Al developed by Ercolessi and Adams [3].

Note that all elements (as denoted by their atomic numbers) used previously to define the materials in the `mate` command at lines 13-14 must be assigned appropriate constitutive data by the `constitutive` command. Any undefined elements will cause the simulation to terminate with an error message.

The `cons` module allows any EAM functions to be used with QC once they have been converted to either of the `funcfl` or `setfl` formats. A utility program, `dynpots.f`, is provided with the QC download in subdirectory `Potentials/FortranSource` and can be used to convert any EAM functions to the `setfl` format. See Section 3.1 of the *QC Reference Manual* for more information.

The `cons` command writes the following to standard output:

```
 **reading constitutive information**
   opening constitutive information file ../../Potentials/al_ea.fcn

   constitutive information

   type  element     amass        alat     lattype      rcut
   ----  -------  ----------  ----------  --------  ----------
      1     13      28.00000     4.03200  fcc          5.55805
   grain reference energy, stress and moduli


   ==========================================================================
     Grain    1

     energy density =    -0.2050269616

     stress (should be zero)
        0.27302E-06  -0.88916E-17   0.17206E-23
       -0.88916E-17   0.27302E-06  -0.51617E-23
        0.17206E-23  -0.51617E-23   0.27302E-06

     elastic stiffness matrix (in grain c.s.) 11 22 33 12 13 23 convention
          0.79139     0.35167     0.36999    -0.02590     0.00000     0.00000
          0.35167     0.80970     0.35167     0.00000     0.00000     0.00000
          0.36999     0.35167     0.79139     0.02590     0.00000     0.00000
         -0.02590     0.00000     0.02590     0.19238     0.00000     0.00000
          0.00000     0.00000     0.00000     0.00000     0.21070     0.02590
          0.00000     0.00000     0.00000     0.00000     0.02590     0.19238
   ==========================================================================
```

Note the final output from the `cons` command is the cohesive energy density (energy per unit volume), initial stress and elastic moduli (in the compact 6×6 Voigt matrix) associated with a perfect, undeformed crystal of each previously defined grain. These data are a useful check of the input file. For example, finding the energy to be consistent with the known value from an atomistic simulation is a necessary (but not sufficient) condition for correct crystal structure and constitutive data. The initial stresses should be close to zero. If they are not, this is an indication that the lattice constants defined for the grain may be wrong. The elastic constants, in units of eV/Å^3 are often useful for comparing results to known elastic solutions.

3.7 Lines 65-66: `Mesh` Definition

The command `mesh` calls the user-defined subroutine `user_mesh` in `GB-example/user_gb.f` to generate the mesh to be used for the simulation. It can be followed by a number of data lines specific to that mesh generator if the `direct` key is provided as in this example. Another example of a mesh generator can be found in `QC/Punch-example/user_punch.f`, and a discussion of the input and expected output variables for this routine are discussed in detail in Section 2.11.1 of the *QC Reference Manual*. In broad terms, the `mesh` routine must define the following variables for the QC simulation:

- *Reference coordinates of the repatoms.* The positions of the initial set of repatoms to be used in the simulation. This does not discount the possibility of additional repatoms being added by the adaption routine later.

- *Element Connectivity.* The `mesh` routine must define the mesh using 3-node, linear, triangular elements, such that all repatoms are connected and there are no overlapping elements. This is facilitated by the inclusion of a constrained Delaunay triangulation routine [8] with the QC code. Details of how to use this triangulator are found in the comment lines of subroutines `delaunay` and `contri` in `Code/mod_mesh.f` and in the discussion of the `user_mesh` routine in Section 2.11.1 of the *QC Reference Manual* of

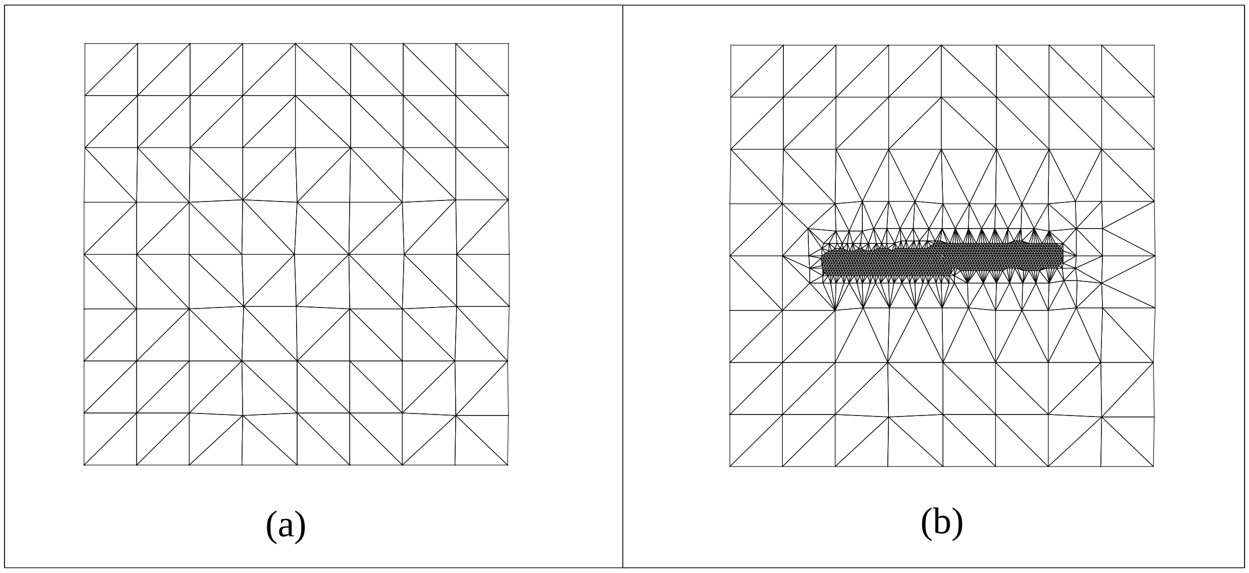

Figure 6: (a) Initial coarse mesh generated by the `mesh` command and (b) final mesh after the first `status` command.

the *QC Reference Manual*.

- *Boundary Constraints.* The `mesh` routine can also be used to identify nodes which have force or displacement constraints (boundary conditions) applied to them. The actually application of the forces or prescription of the displacements is left to the `bcon` command described later, where the constraints defined here can be modified if desired.

In this example, a very coarse initial mesh is defined, with the intention of using automatic mesh adaption to refine it. This approach requires the initial mesh to completely fill the region of space that the user intends for the model. Automatic mesh adaption later in a simulation will only add repatoms that lie inside one of the existing elements in the mesh. The initial mesh generated in this example is shown in Fig. 6(a), while Fig. 6(b) shows the result of mesh adaption that will be discussed below.

3.8 Line 67: `end`

The `end` command tells QC that the first stage, `head`, is finished.

4 Stage: Macros

The macros stage is initialized at line 68 by the macros command, and ends with the end command at line 93. The macros stage allows the user to control the simulation steps through such things as applying boundary loads, incrementing the time variable to change the applied loads, executing the solution algorithm, etc.

In this example, we have (during the head stage) set up the geometry to include a stepped twin boundary on the plane $y = 0$ and built a very coarse mesh to define the spatial extent of the model. In the macros stage we will first adapt this mesh so that it is appropriately refined in the region of interest near the step in the grain boundary, and then perform 10 load steps in which the externally applied shear strain on the outer boundary of the mesh is incremented by 0.5% per step. After the external strain has reached 4.0 %, the step migrates through a mechanism of two partial dislocations running along the boundary.

The main feature of the macros stage that differs from the previous head stage is the ability for the program to recognize nested loops in the command structure. These loops are demarcated by a pair of commands: loop to start the loop (and define the number of times the loop will be executed) and next to end to loop. Finally, a loop can be terminated before the loop counter has been fully exhausted by an appropriate convergence condition as identified by the conv command, for example at line 84. Loops may be nested to any depth. Details of the use and syntax of these loops appear in what follows.

Standard output from the initialization of the macros stage includes a summary of the important parameters (*i.e.* factors and flags) set during the head stage, followed by a list of the macros commands that have been read from the input file. The output for this example is as follows:

```
 **global parameter settings**

   Number of nodal points (max) =    2500
   Number of elements     (max) =    5000
   Number of spatial dimensions =       2
   Number of dofs per node      =       3
   Number of nodes per element  =       3
```

```
  Number of stress components  =       6
  Number of internal variables =      14

  Critical nonlocality strain  =    0.15000

  Proximal nonlocality factor  =     2.0000
  Adaptation nonlocal padding  =    0.00000
  Effective cutoff factor      =     1.5000

  Non-locality                               = ACTIVATED
  Ghost Force Correction                     = OFF
  Free surface triggers for non-locality     = OFF
  Grain Boundary triggers for non-locality = ACTIVATED
  Nodes constrained to Bravais sites         = ACTIVATED
  Prevention of Warped Elements              = OFF
  Mesh Adaption in deformed configuration  = OFF
  Dummy atoms allowed                        = ACTIVATED

**macro instructions**
[... list of instructions ...]

**start of macro execution**
```

For the remainder of the `macros` stage, each output line has a standard format, as illustrated here:

```
 #    comm  key       * command-specific output
                      * command-specific output
                      * command-specific output
```

where `#` is the command number, followed by the command itself (`comm`) and the `key`. The right-most column is reserved for command-specific output. Often, this is simply echoing the user input, but for many commands there is additional information presented. This field may also contain references to entries made in the `qc.log` file if appropriate. Output from user routines will conform to this output structure if generated through the standard QC output interface defined in the module `mod_output`. See Section 2.6 of the *QC Reference Manual* for details.

4.1 Line 69: Setting the convergence `tolerance`.

The `tole` command is used to set the desired convergence tolerance for the simulation. Because there are no `key` options for this command, the double-comma must appear to signify an empty `key` field. The `data` for this command is a double-precision real tolerance value. How this tolerance is used will depend on the solution algorithm that is specified later. For example, it may be used as a minimum change in atomic positions during a conjugate gradient step, or as a tolerance on the out-of-balance force norm.

4.2 Line 70: Defining the `proportional` loading schedule.

Loading is incremented in the QC method by a variable referred to as the `time`, despite the static nature of QC simulations. This `time` variable is further used to compute a proportional load variable `prop`. The `proportional` command is used to define the dependence of the `prop` variable on the `time` variable. There are no `key` options for the `prop` command, so the data follows the double comma. The data consists of an integer identifying the number of ordered pairs to be read in, followed by *either* a number of ordered pairs defining the `prop` vs. `time` function *or* a `filename` (without the `.prp` extension) from which to read these values. If the numbers are entered directly, as in this example, the filename must be explicitly passed as an empty character string, and thus the second double-comma in the input.

In this simple example, the function is linear, going from (0.0, 0.0) to (1000.0, 1000.0). QC will linearly interpolate to find values between the ordered pairs.

In defining the boundary conditions for the problem, the user specifies a set of normalized displacements and/or forces on boundary nodes, which the QC multiplies by the current value of `prop` to determine the current load levels. In this example, a uniform shear strain is applied as illustrated in Fig. 7. The strain is applied by using fixed displacement boundary conditions on the nodes along the top and bottom of the mesh. In addition, the initial displacements for each load step are the solution from the previous step plus a displacement associated with an incremental uniform shear strain. The normalized displacements on the fixed nodes $\bar{u}_x$ are as shown Fig. 7(a). At each application of the boundary conditions during the simulation, QC uses the current value of the `prop` variable to scale these variables. For

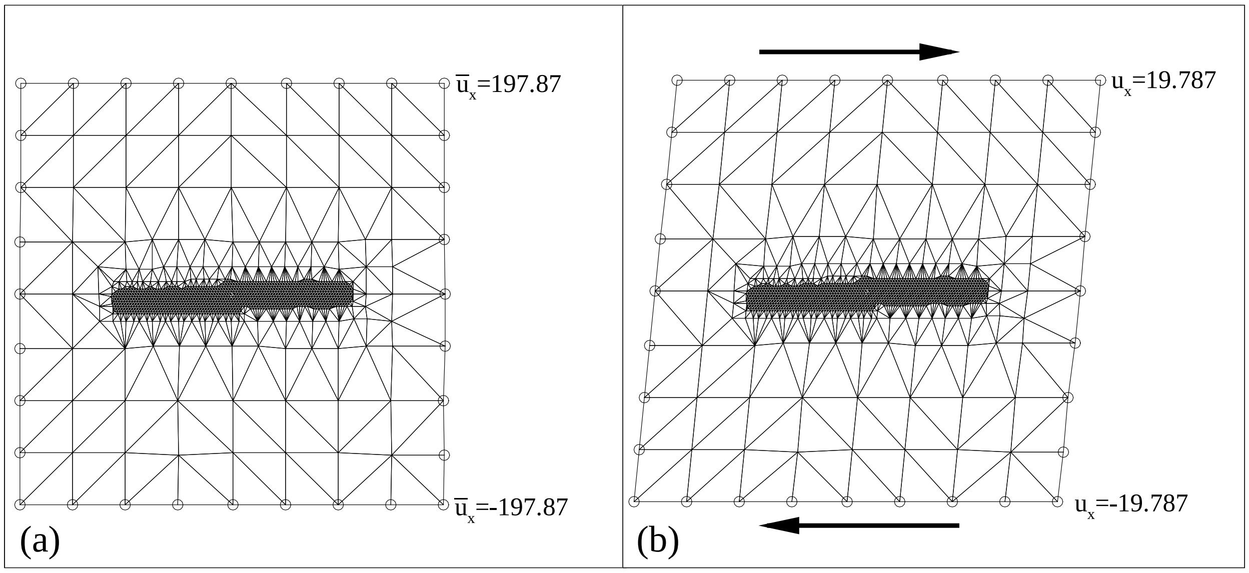

Figure 7: (a) Undeformed mesh and the normalized applied displacements stored at boundary nodes at various vertical positions. (b) Resulting displacements when `prop=0.10`.

instance, at `time=0.1` in this example, the proportional load table returns `prop=0.1`. Thus the applied displacements, `prop`$\times\bar{u}_x$, are as shown in Fig. 7(b). More detail about how to apply the boundary conditions in the QC method can be found in Section 4.2 of the *QC Reference Manual*.

4.3 lines 71-73: Computing the repatom `status`

The `status` command performs a number of automatic pre-processing steps that generate a data structure used during the simulation. The main function, as the name suggests, is to decide whether each repatom should have local or nonlocal status, but several other steps are performed as well. Generally, the `status` command should be called at the beginning of the `macros` stage before any major `macros` commands are invoked (`tole` and `prop` do not depend on `status` being previously called). It can also be called at other stages of the simulation to force a re-computation of the repatom statuses. However, the time spent in this computation is substantial and therefore `status` calls must be used judiciously. The functions performed by status are as follows.

After verifying that the user-defined mesh has all repatoms on valid Bravais lattice sites (if necessary, *i.e.* , if `NodesOnBSites=.true.`), the `status` command computes the *tessellation* of the mesh that will be used to assign sectors of each element to appropriate repatoms. It then determines the local/nonlocal status of each repatom and computes the initial neighbor list of each nonlocal repatom. Details of these processes can be found in [9, 5]. Standard output from this command includes a report of the number of local and nonlocal atoms:

```
* Status recomputed:  #Nonlocal =   746  #Local =   387
```

The `status` command then checks that each nonlocal repatom is representing only itself in the problem, *i.e.* that its tessellation cell contains only one atom. If this is not the case, the `status` command iteratively adapts the mesh by subdividing all elements which touch nonlocal repatoms. After each mesh refinement, statuses are recomputed and the process is repeated until all nonlocal repatoms represent only themselves. The effect of this adaption is shown in Fig. 6(b). In Fig. 6(a), the initial mesh created by the previously discussed `mesh` command is shown prior to adaption. In Fig. 6(b), the final mesh is shown. Notice that the active segment of the grain boundary has triggered a fully refined mesh along its length, but the inactive segments have not. In this way, full atomic-scale boundary structure and details are only included in the region of interest near the grain boundary step.

Next, `status` assigns each element in the mesh to one of the grains previously defined. This is achieved by determining the grain in which the element's centroid resides, and assigning the element to this grain. As a result, the actual grain boundary in a coarse mesh may not follow the user-specified grain boundary very closely. This can be improved, if desired, through careful mesh design and the use of constrained element edges. See the discussion of the `user_mesh` routine in Section 2.11.1 of the *QC Reference Manual* for more details.

Finally, if the ghost force correction is being used (`GhostOn=.true.`), the status command computes and stores the ghost forces for the current mesh, statuses and displacement field. See Section 4.8 of the *QC Reference Manual* for more details.

4.4 Lines 74-76: Generating `plot` files.

The `plot` command generates various output files which are in a format that can be directly read by Tecplot®. There are several `key` options for the `plot` command that allow a variety of data to be plotted. For example, on line 74, the `bcon` key will produce a plot of the repatoms and mesh, with data at each repatom identifying whether it is a constrained boundary node or a free node.

Following the `key`, the first entry in the `data` for the `plot` command is a `fileprefix`, to which will be appended a unique file identifier. For this example, the `plot` on line 74 will produce the file `gbshear_B001.plt`. This incorporates four elements into the filename: the user's chosen file prefix, `gbshear`, a letter code, `_B`, identifying the type of plot produced, an automatically incremented 3-digit number uniquely identifying this file, and the `.plt` file extension identifying this as a plot file. If the `plot` command is called again with the `bcon` key during the simulation, the label will be incremented to `gbshear_B002,plt`, and so on. This automatic increment in the file name is of the greatest use for `plot` commands inside loops (discussed below) where it is not possible for the user to give each output file a unique file prefix. The letter codes for the types of plot files found in this example are `_B` for the `bcon` plot-type, `_D` for the `disp` plot-type on line 75 and `_R` for the `repa` plot-type on line 76. A complete listing of the plot-types is provided in Section 4.14 of the *QC Reference Manual.*

Each call to the `plot` command leads to a line in standard output of the form:

```
  6   plot  disp     * Opening formatted file: gbshear_D001.plt
```

to inform the user of the filename currently being used. Cross-referencing between standard output and the names of the generated output files allows the user to determine which files are from which stages of the simulation.

Details of the numerous `key` options for the plot command are provided in Section 4.14 of the *QC Reference Manual.* In this example, Line 74 produces a mesh with boundary condition data at each node, line 75 produces a mesh with the displacements at each node,

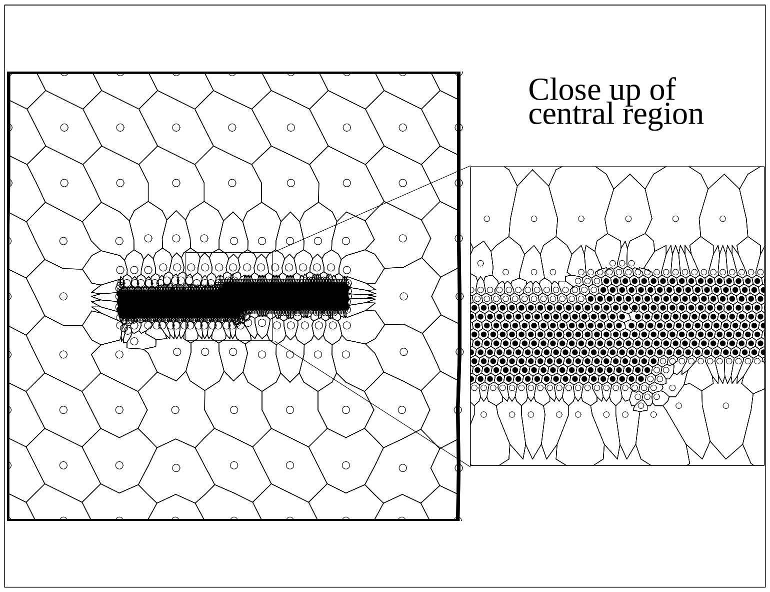

Figure 8: The repatoms and the tessellation which determines the regions of the crystal that each atom represents. Open circles are local repatoms, Filled circles are nonlocal repatoms.

and line 76 produces a file that graphically presents the details of the repatoms, statuses and tessellation.

The result of line 76 is the file `repatom_R001.plt`, which contains four Tecplot® "zones". These are shown in Fig. 8. The first zone, shown as filled circles, is the reference configuration of all non-local repatoms. The second zone, shown as open circles, is the reference configuration of all local repatoms. The third zone is plotted using a mesh in which triangular elements are collapsed to lines to draw the tessellation, *i.e.* it shows the region of the crystal that is represented by each repatom. The final zone presents the model boundary, and is shown in the figure as the bold line.

4.5 Line 77: Changing the time step, `dtime`.

The `dtime` command simply assigns a numerical value to the `dt` variable in QC, which is the current value of the time step. In subsequent calls to increment the `time` variable, `dt` will be used as

```
time = time + dt
```

4.6 Lines 78-91: Load-stepping loop.

The `loop` command takes the form

```
loop,[label],nloop
```

where the `label` is an optional identifier that improves the readability of the input file and `nloop` is the number of times the loop will be executed (unless it is terminated before completion due to a convergence signal from the `conv` command). The `loop` command must be followed, somewhere later in the input file, by a `next` command as on line 91. Note that the `next` command also has an optional `label`, and it convenient to use this label in the `loop` and `next` commands to identify where loops start and end. The QC program completely disregards the `label` for these two commands: it is only used to improve input file readability. Indentation, as in this example, can also be used to this end.

The loop between lines 78 and 91 serves to run the simulation 10 times, incrementing the `time` variable in each loop by steps of 0.005 from 0.000 to 0.045. The loop applies the boundary conditions, minimizes the energy using the Newton-Raphson solver, produces output and then increments the time before repeating the process. The commands within this `loop` are discussed in the following.

4.7 Line 79: Applying the boundary conditions: `bcon`

The command `bcon` calls the user-defined routine `user_bcon` which applies the boundary conditions for a given level of the proportional load variable `prop`. Often, the `bcon` routine is used to modify the displacements of *all* the repatoms (not just the boundary atoms) to set them to what the user may judge as a good *initial guess* to the correct equilibrium solution. Because QC problems can be highly nonlinear, this step can be considerably useful to ensure rapid convergence and to avoid non-physical solutions.

In this example, the boundary conditions are simply a fixed displacement on the top and bottom faces of the bi-crystal, consistent with a uniform shear strain equal to the current value of the `prop` variable. The initial guess is the solution from the last converged load step, *plus* the uniform shear strain associated with the strain increment that is applied (0.005). This modification to the displacement field is also made by the `user_bcon` subroutine. More detail about how the QC method treats boundary conditions can be found in the `bcon` entry in Section 4.2 of the *QC Reference Manual*.

4.8 Lines 80-85: The Newton-Raphson Solver Loop

This loop is the standard format for use of the Newton-Raphson (NR) solver provided with QC. The `loop` command specifies the maximum number of NR iterations that will be attempted before aborting the solution. A value of 200 iterations is quite large and should usually be adequate. However, critical load steps, such as those in which dislocations are nucleated and propagate over long distances, can often take in excess of 100 NR iterations to reach convergence.

Each pass through the solver loop updates the tangent stiffness matrix using the `tang` command. This matrix is essentially the second derivative of the energy functional with respect to all repatom displacements. The matrix is then inverted by the `solve` command, which also uses this inverted matrix to compute a NR increment to the displacement vector. The actual step is set using a line search routine with adaptive backtracking (see the file `mod_solve.f` for details). Standard output from these two commands are as follows

```
 11         tang          * Bandwidth optimization
                            maximum node degree =   133
                            stiffness matrix memory =  1161768

 12         solv  nr      * Newton-Raphson step
                            en = 0.30193E+01     rn = 0.26308E+01     (INITIAL)
                            en = 0.27630E+01     rn = 0.17306E+01     stp =  0.39686E+00
```

where the output from the `tang` command provides information on the size of the stiffness matrix and the output from `solv` allows the user to watch the convergence progress of the load step. The output includes the current energy (`en`) and out-of-balance force norm (`rn`). The final entry, `stp`, is an indication of the efficiency of the NR method. The `solv` command computes a displacement increment based on a linearized approximation to the energy functional. It then applies the largest fraction of that increment possible without an increase in the value of the actual energy functional. Thus, in a truly linear region of the configuration space, this `stp` would always be equal to unity, while a nonlinear problem may have to take significantly smaller fractions to account for differences between the linearized energy functional and the actual energy functional being minimized.

Line 83, the `status` command, is used to recompute the local/nonlocal status of the repatoms, but because the `key=update`, status is only re-computed if the displacements have changed significantly since the last status computation. “Significant” displacements are defined specifically in the `status` command details in Section 4.21 of the *QC Reference Manual*, based on a conservative estimate for the possibility of either new large deformation gradients or large motion of atoms such that neighbor lists must be updated. The output of each call will include the observed and allowable displacements as follows:

```
13         stat  upda     * Status update check:  Observed =  0.553     Allowed =  2.779
```

When the displacements since the last neighbor-list update are larger than the allowed displacement, the output signals this and indicates the new statuses and neighbor-lists:

```
13         stat  upda     * Status update check:  Observed =  3.957     Allowed =  2.779
                          ** Update Triggered **
                          * Status recomputed:  #Nonlocal =   747   #Local =   386
                          * Generating atom lists.
```

The last command of the NR solver loop is the `conv` command on line 84. Here, the `key` is `force`, so the `conv` command will check for convergence based on the smallness (compared to the `tolerance` set earlier) of the out-of-balance force norm. If the solution is converged, the loop is terminated and the simulation moves to line 86 of the input file. Otherwise, the loop starting at line 80 is repeated.

4.9 Lines 86-88: Output

Upon reaching line 86, the solution has converged for the current time step (unless the maximum number of NR iterations has been exceeded). Thus, lines 86-88 produce three items of output for each load step: a plot of the deformed mesh, a report of the model energy and an entry in the load-displacement file.

The first output is a plot of the deformed mesh of the repatoms and elements, together with the displacements at each nodal point. The filename `gbshear_D` will be augmented by a numerical label after each output, `gbshear_D001.plt`, `gbshear_D002.plt`, `gbshear_D003.plt`, etc.

The second item, produced by the `report` command at line 87, is written to standard output and lists the current value of the strain energy, external work and the out-of-balance form norm:

```
 17      repo           * Current energy and forces:
                          Strain energy             =      2.038179179
                          External work             =      0.000000000
                          Potential energy          =      2.038179179
                          Out-of-balance force norm =     0.6840353005E-08
```

Note that the `External work` will only be non-zero for simulations with applied external loads, in contrast to the current simulation which uses prescribed displacement boundary conditions.

The third item of output for each load step is produced by the `pdel` command at line 88, which adds a line to the output file `p-delta.pde`. Note that the filename specified in the command line is `p-delta`, and the `.pde` extension is automatically appended by the program. This output contains two real numbers: the current value of the proportional load `prop`, and a user-defined scalar measure of the applied force. Typically, this is useful for producing applied force vs. displacement, or $P - \delta$, curves.

For a given simulation, the user can determine a suitable scalar measure of the applied force and write the `user_pdel` routine. In this example, the routine is found in `QC/GB-example/user_gb.f`, where the scalar measure is the sum of the absolute values of the x-component of out-of-balance forces on the nodes along the top face of the bi-crystal. Although not especially accurate, this can be used as an estimate of the applied stress by dividing by the area (which in this case is the width 397.5 Å times the depth 2.85 Å). In a

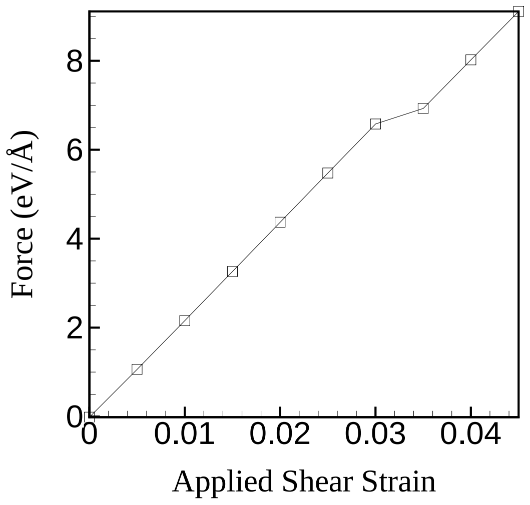

Figure 9: Applied force versus shear strain of the bicystal in this example.

displacement boundary condition problem like this example, there are no applied loads, and instead several nodes are constrained to a certain displacement. However, this is equivalent to applying a force to each of these nodes to exactly cancel out the nodal forces which arise from the derivatives of the strain energy functional. Thus, an equivalent set of applied forces can be determined from the out-of-balance forces on constrained nodes.

At the end of the simulation, the contents of the `p-delta.pde` file can be plotted as in Fig. 9. Note the drop in the load at the critical displacement, at which point the step in the grain boundary migrates.

More details about how to write the `user_pdel` routine can be found in Section 2.11.3 of the *QC Reference Manual*.

4.10 Line 89: Writing a `restart` file

The `restart` command with a `write` key will generate a restart file, with the name taken from the `data` of the command line. In this case, the file will be named `shear.res`, based on the file prefix `shear` provided in the command input and the `.res` default extension for restart files. If a file `shear.res` already exists, it will be moved to a file `shear_LAST.res` before the new `shear.res` is written, effectively keeping up to two options for future restarts.

Note that the file `shear_LAST.res` will be overwritten if it already exists.

The restart files produced by this command are binary files that contains all of the essential data to restore a QC simulation from where it was at the time the file was written, and can be read by another QC simulation using an appropriate `restart,read` command. There is also an option to produce ASCII restart files if portability between platforms is required.

More details are provided in Section 4.18 of the *QC Reference Manual.*

4.11 Line 90: Incrementing the `time` variable

The final command in the loop is to increment the `time` variable, which is achieved by a call to the `time` command. If no `key` or `data` options are provided to the `time` command, it simply performs the operation:

```
time=time+dt
```

More detail can be found in Section 4.25 of the *QC Reference Manual.*

4.12 Line 92: `end`

The `end` command simply signifies the end of the `macros` stage of the simulation.

5 Stage: `stop`

The final stage, `stop`, is executed at line 93. `stop` closes any open files and terminates the simulation.

6 Other Important Features

A few important features of QC are not demonstrated by the example discussed above. These are discussed in this section.

6.1 Models with Multiple, Unconnected Domains

In some problems, such as nano-indentation, wear, or friction studies, it may be desirable to build a model with separate domains that are brought into contact. The code is capable of handling such models, with some limitations. The main caveat is that contact between such regions will only be recognized reliably by the code between nonlocal portions of the grains defining the contacting regions. Contact between two local regions or a local and nonlocal region can only be made through an element, *i.e.* , within a single connected region.

The example provided in `QC/Friction-example` illustrates this capability by modeling the collision between two small asperities on the surfaces of two separately meshed grains. To produce such a model, it is necessary to assign each repatom in the initial mesh to a `region`, numbered from one to the number of unique regions, `nregion`, in the model. By default, it is assumed that there is one region containing all the repatoms unless the user specifies otherwise in subroutine `user_mesh`. See Section 2.11.1 of the *QC Reference Manual* for more details on this feature.

6.2 Generating Multiple Plots in a Single Output File

The `plot` command in the example described above produces a series of sequentially numbered files, each containing the plot for a different load step. It is sometimes more convenient to put all of these plots in a single file, an option which is available in QC. See Section 4.14 of the *QC Reference Manual* for details of this option.

7 Other Examples

In this tutorial, the specific example of shearing a bi-crystal (found in `QC/Code/GB-example`) was discussed in detail. The following other examples are provided with the QC program.

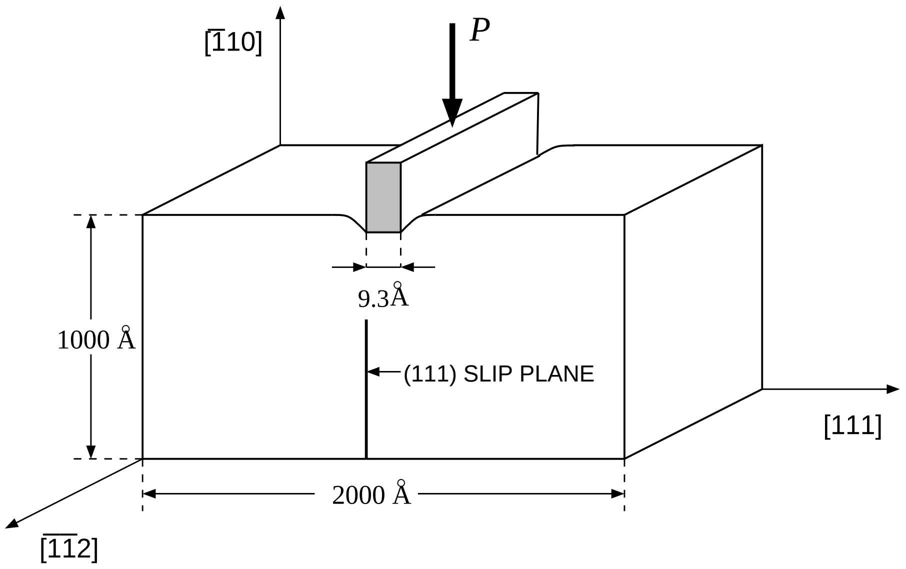

Figure 10: Schematic representation of the nanoindentation model for indentation into a ($\bar{1}$10) plane.

7.1 Nano-Indentation of an Aluminum Single Crystal by a Square Punch

This example, found in directory `QC/Punch-example`, simulates nanoindentation into single crystal aluminum. The aluminum is modeled using the Ercolessi-Adams EAM potential. A rigid knife-like flat indenter, 9.31 Å wide, is driven down into the ($\bar{1}$10) surface of the crystal. The x-axis direction is the [111] direction and the out-of-plane direction is the [$\bar{1}\bar{1}$2] direction (see Fig. 10). Perfect stick conditions between the indenter and crystal are assumed. The results of this simulation have been reported in [10].

As in the grain boundary example of directory `QC/GB-example`, the meshing routine initially generates a uniform mesh. An active grain boundary segment is defined in the grain geometry file punch.geo along the line where the indenter will be applied. The active segment is defined to be a little larger than the indenter (24 Å) to allow for surface relaxation effects in the vicinity of the indenter. A `status` command at the start of the `macros` stage forces the region in the vicinity of the active segment to be fully refined down to the atomic scale.

The solution algorithm is straightforward:

1. Relax out surface effects of the initial model:

```
loop,,50
   tang
   solve,nr,1.0,10000,1
   status,update
   convergence,force
next
```

2. Increment the time and apply boundary conditions: all nodes on the surface in contact with the indenter are moved down to a position `time` below their reference y-coordinates.

```
time
bcon
```

3. Relax the model given the new BC's.

```
loop,,200
   tang
   solve,nr,1.0,10000,1
   status,update
   convergence,force
next
```

4. Adapt the mesh to allow for defect nucleation and defect motion. There are two important points regarding this adaption step:

   (a) It is normally important to do the relaxation step in (3) before calling mesh adaption. If the model is not relaxed there may be large non-physical gradients in certain areas of the model which will drive adaption where it is not needed.

   (b) The status command called before the adaption step is important. During the relaxation loop in (3) the command `status,update` is repeatedly called to check whether the auxiliary lists are still up-to-date. If they are not, a `status` calculation is triggered. In practice this triggering often occurs due to an anomalous relaxation step which takes the model far from the relaxed configuration. The result may be the creation of a new non-physical nonlocal region in some portion of the model at the end of the relaxation step. By calling status after the relaxation, anomalous nonlocal regions will become local again and will not drive adaption.

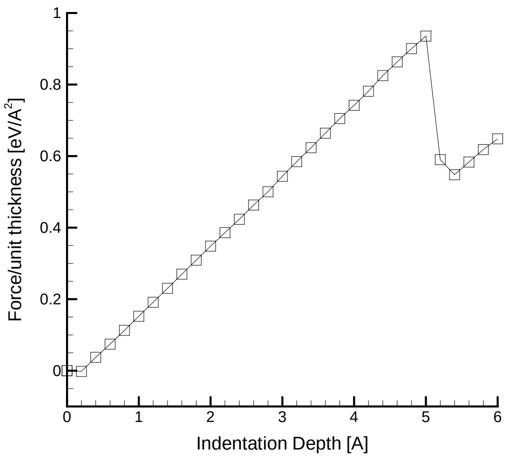

Figure 11: The force per unit thickness in the out-of-plane direction versus indentation depth for the nanoindentation simulation. The force per unit thickness is obtained by dividing the total force on the indenter given in the `pde` file by the repeat distance in the out-of-plane direction of the crystal structure (4.938Å for this model).

The adaption command itself is called with coarsening activated so such regions, if they exist, will be coarsened out.

```
status
adapt,,0.075,1
```

5. Relax adapted model.

```
loop,,200
   tang
   solve,nr,1.0,10000,1
   status,update
   convergence,force
next
```

6. Generate output and return to (2) for the next load step.

```
pdelta,,punch
plot,disp,punch,1,1.,1.
```

This algorithm is run for 30 load steps. At the 26th load step a dissociated edge dislocation is nucleated from the right indenter tip and at the 27th load step a second dissociated edge dislocation is nucleated from the left indenter tip[7]. The load versus indentation depth

[7]Note: the load step at which nucleations occur and even the order of nucleation (left vs. right) can depend on the platform that the code is run on due to nonlinearity and chaos effects in the solution. The results quoted above were obtained on an SGI Origin 200 machine running MIPSpro 7 Fortran 90.

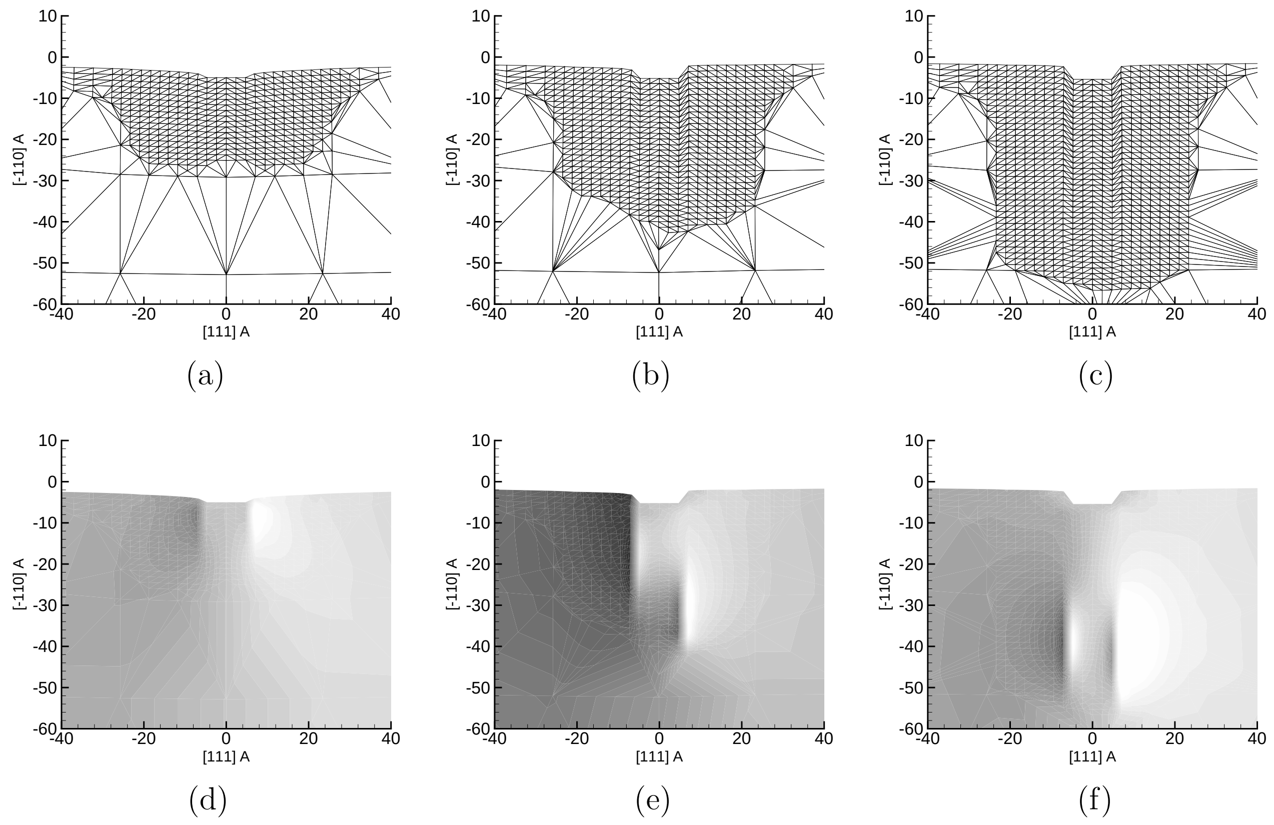

Figure 12: Three snapshots of the mesh under the indenter (a) immediately before the first emission (`punch_D025.plt`), (b) after the emission of dissociated dislocation from the right indenter tip (`punch_D026.plt`) and (c) after emission of dissociated dislocations from both ends of the indenter (`punch_D027.plt`). Frames (d), (e) and (f) show the corresponding `UZ` contour plots.

curve for this simulation is plotted in Fig. 11. The sudden drop in the load at the instant of dislocation nucleation is clearly visible. The nucleated dislocations are most easily seen by plotting the displacement plots with `UZ` contours displayed (see Fig. 12). The out-of-plane displacements in the stacking fault regions between the partials are a clear fingerprint of the location of the dislocations.

It is important to point out that the adaption in this simulation is insufficient to obtain unconstrained values for the nucleation load. The location of the local/nonlocal boundary remains too close to the indenter in this case and may retard the nucleation event. It is definitely insufficient to track the motion of the dislocations into the bulk after nucleation. The dislocations will move down and be trapped by the local/nonlocal boundary. A more rigorous approach is to carry out several relaxation/adaption steps within a single load step to create sufficient meshing for a nucleation event to occur if it needs to and then to allow the defect to move away. One way to do this is to loop until adaption creates no new elements:

```
% Relaxation - Adaption Loop
loop,relax,100

   % Relax
   loop,,200
      tang
      solve,nr,1.0,10000,1
      status,update
      convergence,force
   next

   % Adapt
   status
   adapt,,0.075,1

   conv,elements

next,relax
```

The `conv,element` command will terminate the loop when no new elements have been generated since the previous convergence check. The above code will prevent boundary effects on nucleation, but may significantly slow the simulation. This is particularly true if the nucleated defect needs to travel a large distance into the bulk to reach its equilibrium distance. In many cases sufficient accuracy may be obtained by performing a limited relaxation/adaption loop to allow the defects to nucleate unimpeded but not to follow their motion into the bulk.

A limitation of the current QC code is that fully-refined slip traces are left in the wake of dislocations. This means that as a dislocation moves away from its nucleation site the number of elements in the mesh is greatly increased and execution is slowed. Mesh coarsening will not remove these slip traces in the current implementation. A future revision where this limitation will be addressed is planned.

To run the nanoindentation example do the following:

```
% cd ~/QC/Punch-example
% make QCCOMPILER=COMP

% cd Indent

% ../punch < punch_indent.in > punch_indent.out &
```

The complete run takes about 30 minutes on an SGI Altix with the Intel compiler.

7.2 Shearing Contact Between Two Nano-Asperities

As mentioned in Section 6.1, the directory `QC/Friction-example` is the shearing of two nano-asperities across one another. The example illustrates the use of multiple, separate meshed regions, in this case one for each of the two crystals coming into contact.

The solution algorithm is straightforward, simply moving the upper grain relative to the lower grain in increments of 0.2 Å and holding the upper edge of the upper crystal and the lower edge of the lower crystal fixed to this displacement during relaxation. The problem runs in about two and a half hours on an SGI Altix computer and produces a series of 200 load steps and the lateral force vs. displacement curve. This example also demonstrates the feature of writing multiple outputs to a single file, rather than a separate file for each load step, as mentioned in Section 6.2. The displacement plot after each load step is appended to a single file called `shear-relaxed_D003.plt`

To run the friction example do the following:

```
% cd ~/QC/Friction-example

% make QCCOMPILER=COMP

% cd Shear

% ../fric < fric_shear.in > fric_shear.out &
```

A Working with the QC Makefiles

The makefile architecture of QC helps the user to compile the QC code. It requires GNU Make, which should be available on all machines.[8] On certain installations, GNU Make is

[8]For the benefit of older machines that may not have GNU Make installed by default (e.g. SGI Origin 200), we also supply the old-style make files used in previous versions of the code in directories `OldMakes`

run by `gmake` and not m̂ake.

Typing `make` in an application directory (e.g. `GB-Example`), causes the main code in directory `Code` to be compiled along with the application-specific file in the application directory (`user_gb.f` in `GB-Example`). Dependencies are automatically determined using the Perl script `Code/sfmakedepend`[9], which is supplied with the code. If some source files change, only the necessary files are recompiled. No further user interaction is required for this. The uptodate dependency list is stored in files called `fortran.d` in both the `Code` and application directory. These should not be deleted. Perl is freely available for most architectures.[10] In case you do not have (or do not want) Perl, the necessary `fortran.d` files are supplied with the distribution. These will work as long as you do not change the dependencies between the QC files.

Each application directory has a single file called `Makefile` which is executed by the `make` command. This is a very short file that is easy to adapt to new applications. For example, for the `GB-example` the `Makefile` contains:

```
# GB-Example Makefile

SRC_FREE        =
SRC_FIXED       = user_gb.f
EXECUTABLES     = gb
LIBRARIES       =
CODEPATH        = ../Code
QCPOTEN         = eam

include $(CODEPATH)/Makefile_common
```

The first line is a comment. The variables `SRC_FREE` and `SRC_FIXED` refer to Fortran 90 free format and Fortran 77 fixed format. The application file (`user_gb.f` in this case) should

placed in the application and `Code` directories. To use these make files, copy them into the parent directory containing each `OldMakes` directory. The compilation command is then, `make -f makeAPP.COMP`, which should be executed in the application directory. Here `APP` is a particular application (e.g. `gb`) and `COMP` is the compiler. For a list of currently supported compilers see Section 1.2.

[9]`sfmakedepend` is a freeware script generously provided by Kate Hedstrom at the Arctic Region Supercomputing Center. It can be downloaded at `http://people.arsc.edu/~kate/Perl/`

[10]Perl may be obtained at `www.perl.com`. The `sfmakedepend` script requires version 5.6.0 or higher of Perl. If you have an older version, install the latest version in a new directory and change the first line of `sfmakedepend` to point to it. Note that sometimes (e.g. on Sun systems) the default version of Perl is an old one, but a newer version is installed elsewhere. Likely places to look are `/usr/local/perl` and /opt/perl.

be placed in one or the other based on its format. The next variable `EXECUTABLES` contains the name of the executable that will be generated by make (`gb` in this case). `LIBRARIES` will normally be left blank. `CODEPATH` contains the path to the directory containing the QC code. If the directory structure described in Section 1.2 is adopted, this will be `../Code` for all applications. `QCPOTEN` refers to the inter-atomic potential to be used. This is explained below. The last line links in the main QC make file `Makefile_common` that performs most of the work. This file is located in the `Code` directory. It will not need to be modified for most applications unless a new compiler or potential are being added as described below. A third make file called `Makefile` also located in the `Code` directory contains a list of the QC files. It should not be modified.

The calling format for the make statement is

```
% make [QCCOMPILER=COMP] [QCPOTEN=POT] [QCDEBUG=1]
```

The optional compilation variables following the `make` statement are explained below.

A.1 Specifying the compiler

The compiler can be specified by setting the environment variable QCCOMPILER:

- for csh/tcsh: `% setenv QCCOMPILER COMP`
- for sh/bash: `% export QCCOMPILER=COMP`

or calling make with a compiler specification:

```
% make QCCOMPILER=COMP
```

where `COMP` is replaced by the appropriate compiler. See Section 1.2 for a list of compilers currently supported by QC. The default is the `intel` compiler. This can be changed in `Code/Makefile_common`. Other compilers can be used by adding an appropriate section to `Code/Makefile_common` of the following form:

```
ifeq ($(QCCOMPILER),ibm)
    FC           = xlf90
    FFLAGS       = -I../Code
    FFLAGS_FREE  = -qfree=f90
    FFLAGS_FIXED = -qfixed
    FFLAGS_OPT   = -O2
    MOD_NONSTANDARD = mod_nonstandard_ibmxlf.f
    LDFLAGS      =
else
```

and adding an additional `endif` to the end of the list. Here `ibm` is the compiler name (`COMP`) that should be changed to that of the new compiler. `FC` is the system command used to execute the compiler. In addition the following compiler-specific options must be defined:

- `FFLAGS`: Append the directory `../Code` to the module files search path. (This assumes the directory structure described in Section 1.1.) This is necessary for compiling the user file in the application directory that depends on modules in `Code`. (Typically this is `-I../Code`).
- `FFLAGS_FREE`: Assume Fortran 90 free form.
- `FFLAGS_FIXED`: Assume Fortran 77 fixed form.
- `FFLAGS_OPT`: Non-aggressive optimization setting. Note that the aggressive optimization offered by some compilers must be used with care since it does not guarantee the correctness of the compiled code. (Typically this is `-O2`).

For some compilers it is necessary to specify a compiler-specific module `mod_nonstandard`. This module contains all of the non-standard Fortran extensions used by QC. Currently there are two: `flush` for flushing output buffers and `system` for issuing operating system commands from within a Fortran program. The names and calling formats for these routines are the same for most Fortran compilers, however some either do not support one or both of these options or have different names and/or calling formats.[11] For these cases the generic file `mod_nonstandard.f` supplied with the QC code may be copied to a new name and

[11]These routines are not necessary for QC to function correctly.

modified appropriately. The name of the new `mod_nonstandard` file is then given by the `MOD_NONSTANDARD` variable in the compiler definition. If the compiler works with the generic `mod_nonstandard.f` file, this line may be removed. Finally, the variable `LDFLAGS` should contain any additional libraries that must be available at link time. This may be required to link to the nonstandard routines in `mod_nonstandard.f`. (For example, the Absoft compiler requires the library `libU77.a` for this, so `LDFLAGS = -lU77`).

A.2 Specifying the inter-atomic potential file

In similar fashion to the compiler specification, it is also possible to specify the inter-atomic potential file that is to be compiled with the code by setting the environment variable `QCPOTEN`:

- for csh/tcsh: `% setenv QCPOTEN POT`
- for sh/bash: `% export QCPOTEN=POT`

or calling make with a compiler specification:

```
% make QCPOTEN=POT
```

where `POT` is replaced by the potential file designator. Currently, there is only one option, `POT=eam` for the EAM potential `mod_poten_eam.f` supplied with the distribution code. (See Section 3.6 for details.) As noted above, the `QCPOTEN` variable is normally set in the application `Makefile`. Over-riding this selection is possible in certain cases, but this should be done with care since most applications will fail if the potential is switched to an incompatible one.

The `QCPOTEN` variable provides the user with the option of adding on new potentials and selecting them in a convenient manner. New potentials can be used by adding an appropriate section to `Code/Makefile_common` of the following form:

```
ifeq ($(QCPOTEN),mypot)
    MOD_POTEN = mod_poten_mypot.f
else
```

and adding an additional `endif` to the end of the list. Here `mypot` is the new potential name (`POT`) and `mod_poten_mypot.f` is the new potential file written by the user. At this point there is no documentation on preparing new potentials except for the comments that appear in `mod_poten_eam.f`.

A.3 Cleaning up

When changing to a new compiler or to a new potential, the user must

```
% make clean
```

which will remove all architecture and compiler-dependent files (the executables, library (`libmdlr.a`), object files (with `.o` extension), and module files (with `.mod` extension). For a more extensive cleanup that also includes `*~` files left by the emacs editor, do

```
% make veryclean
```

The `make clean` and `make veryclean` should be executed in the application directory. They will remove the necessary files in both the application and `Code` directories.

To help the user remember to perform a `clean` when necessary, the current compiler and potential are stored in the hidden files, `.compiler` and `.poten`, in the `Code` directory. If the user changes compiler or potential without performing a `make clean`, an error message is printed and `make` terminates.

A.4 Debugging applications

By default, code optimization is turned on and debug support is turned off. To change this, use

```
% make QCDEBUG=1
```

or set the environment variable QCDEBUG to 1:

- for csh/tcsh: `% setenv QCDEBUG 1`
- for sh/bash: `% export QCDEBUG=1`

References

[1] M. S. Daw and S. M. Foiles. Dynamo version 8.7. FORTRAN code, 1994.

[2] M.S. Daw and M.I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, 29:6443–6453, 1984.

[3] F. Ercolessi and J.B. Adams. Interatomic potentials from first-principles calculations – the force-matching method. *Europhys. Lett.*, 26:583, 1994.

[4] L. E. Malvern. *Introduction to the Mechanics of a Continuous Medium.* Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

[5] R. E. Miller. *On the Generalization of Continuum Models to Include Atomistic Features.* PhD thesis, Brown University, 1997.

[6] D.N. Pawaskar, R. Miller, and R. Phillips. Structure and energetics of long-period tilt grain boundaries using an effective hamiltonian. *Phys. Rev. B*, 63:214105–214118, 2001.

[7] J. D. Rittner and D. N. Seidman. $< 110 >$ symmetric tilt grain-boundary structures in fcc metals with low stacking-fault energies. *Phys. Rev. B*, 54(10):6999–7015, 1996.

[8] S. W. Sloan. A fast algorithm for generating constrained delaunay triangulations. *Computers and Structures*, 47(3):441–450, 1992.

[9] E. B. Tadmor. *The Quasicontinuum Method.* PhD thesis, Brown University, 1996.

[10] E. B. Tadmor, R. Miller, R. Phillips, and M. Ortiz. Nanoindentation and incipient plasticity. *J. Mater. Res.*, 14(6):2233–2250, 1999.

[11] O. C. Zienkiewicz. *The Finite Element Method*, volume 1-2. McGraw-Hill, London, 4th edition, 1991.

[12] J.A. Zimmerman, H. Gao, and F.F. Abraham. Generalized stacking fault energies for embedded atom fcc metals. *Modeling Simul. Mater. Sci. Eng.*, 8:103–115, 2000.

B QC Timing Study

The QC program has been successfully compiled and run on a large number of computer platforms and compilers. Below is a table of execution times of the `GB-example` described in this tutorial. No attempt was made to optimize performance by tinkering with compiler switches. In all cases, the codes were run with the standard optimization setting `-O2`. The `GB-example` does not require much memory, so this factor most likely did not play a role in the tests. All of the runs are on a single processor.

Computer	Processor	Compiler	Time [s]
IBM BladeCenter H+	2.6 GHz AMD Opteron 2218	Intel v9.1	179
IBM BladeCenter H+	2.6 GHz AMD Opteron 2218	Pathscale v2.5	179
IBM BladeCenter H+	2.6 GHz AMD Opteron 2218	Portland v6.2-5	207
Linux cluster	2.2 GHz AMD Opteron 275	Pathscale v2.5	213
Dell Precision 390	2.9 GHz Intel Core 2 X6800	Portland v6.0	241
Linux box	2.8 GHz Intel Pentium D	NAG v4.1	304
Apple MacBook Pro	2.3 GHz Intel Core 2 Duo	g95 v0.9	317
SGI Altix 350	1.5 GHz Intel Itanium	Intel v9.0	378
Linux box	2.8 GHz Intel Pentium D	gfortran v4.1.2	409
IBM Power4 System	1.7 GHz Power4	IBM XL	439
SGI Origin 200	225 MHz MIPS R10000	MIPSpro v7.3	1854
Compaq Proliant ML350	800 MHz Pentium III	Absoft v3.0	1954