README - Adaptive Mesh Equaixed Growth

General:
This folder contains source files for a phase-field model that can be used for equiaxed solidification. The model is based on that developed by Ofori-Opoku and Provatas [1] with the nucleation algorithm that was developed by Montiel *et al.* [2]. The model itself is solved using an adaptive mesh refinement (AMR) algorithm architecture which uses a finite difference scheme where an OpenMP parallel method has also been employed. Details of this particular design of the AMR algorithm can be found in the thesis of Greenwood [3]. This read-me text will serve as a guide to the general operation of the code and enough knowledge one can use to make some superficial changes to either the model being solved, outputting of data or including functions. Any interested user who would like to delve deeper into the understanding and operation, and perhaps make some non - trivial or superficial changes will require more assistance than what is contained here.

Compiling and running the code is simple. A one line script or on the command line will do, just be mindful to include an OpenMP flag. Something like *g++ -fopenmp *cpp -O3 -o equixTest* will suffice in generating the executable file name *equixTest*.

Flow of Algorithm:
The two most important files for someone who just wants to run this code is likely the main source file **main.cpp** and the header file **. initial.h**. The general flow of the program can be found in the **main.cpp** source file, while the bulk of the simulation variables, e.g. material parameters (melting temperature, partition coefficient, etc), model parameter (cooling rate, nucleation rate, number of nuclei, convergence parameters, etc), mesh size, Δx, Δt, etc., can be found in the header file **initial.h**. Currently the model's materials parameters are set to those of a Mg-Al binary alloy system where the solute is Al. When you wish to change the material characteristics, it should be a simple matter of going to the **initial.h** and making appropriate changes.

Following the general flow of the program, we first have some declaration and initialization, mostly surrounding nucleation arrays and initialization arrays for initial grain setup. After this, the mesh is initialized with what ever the initial conditions happen to be, which may be to setup some grains in an undercooled liquid or with just a liquid and await nucleation. After this block of code, the

time integration starts, which includes also the adaption processes, nucleation block and finally data output. One will know upon inspection that every function that is called from **main.cpp** is connected to the object *AdaptiveGrid*, defined and contained in the source file **grid.cpp** and it's corresponding header file. One can think of this as basically the workhorse of the code or more aptly the Solver Side, i.e, where the fields of interest are solved and updated, of this algorithm. If the objects and information contained in **grid.cpp** is the solver side, then it is safe to say that almost everything else belongs to the "brains", i.e., logic of the meshing algorithm. We can call this the Adaptor Side. I advise that you take some time and look generally over the entirety of the code to gain some familiarity, however, for all intent and purposes, unless you understand the algorithm and c++ coding well, I would avoid making any serious changes on the Adaptor Side of this code. Besides, as far as the actual modifications and functions, your focus should be on the functions in **grid.cpp**. Which is what I will discuss in next.

Meshing and Model Functions:
The adaption/re-adaption process is contained in the first part of the code in the time integration loop in the **main.cpp** file. The process calls three functions, of the grid, namely, updateGrid, setdx and createArray.

AdaptiveGrid→updateGrid
: This function call copies all necessary field values (phase-field, temperature, or concentration), i.e., order-parameter fields, from the solver side of the algorithm over to to adaptor side. After the fields are copied over, the mesh undergoes a re-griding, i.e., a readapting, process based on the current state of the system. Predominantly, the readaption process uses gradients of the fields to determine where new nodes and elements should be placed. After the readaption, the new size of the number of nodes/ghosts is also calculated.

AdaptiveGrid→setdx
: After the elements and nodes are created, some have increased in size, while others have decreased in size. The current layout, in terms of the size, i.e., Δx, of the new system is set.

AdaptiveGrid→createArray
: This is the final step in the adaption process, where the new sets of fields values, coordinates, and Δx values are copied from the adaption side to the solver size to resume the updating scheme until the next time the adaption process is invoked. Once these have been copied back, this function is also responsible for reassembling the boundary relationships between the new nodes.

The next set of code encountered is the nucleation block. The nucleation process is similar to that of the adaption process. In order for the system to nucleate a phase on the mesh, the current fields values and structure of the mesh

need to be known. The following is the sequence of functions, neglecting those ones that have already been discussed.

AdaptiveGrid→temperature
: Calculates the current temperature profile in the system. For isothermal runs, this would just be the current temperature in the domain based on the structure of the mesh and the current time step and cooling rate.

AdaptiveGrid→findNucSites
: Using classical nucleation theory, this function uses Boltzmann statistics to calculate the probability of a nucleation event (only a single event per query of the system is allowed) at some point in the domain. If it's probable then this function will return the site coordinates, nucleus concentration and the nucleus radius.

AdaptiveGrid→nucleate
: Once the nucleation parameters have been determined, this function will then place the nucleus in the domain.

Next in the sequence of the code are the functions which calculate the different parts of the model. In this part of the code, the calculation of each portion of the model's equations of motion follow a pattern. First, buffer and boundary node values are calculated, then some field, its derivatives or a function of the filed is then calculated. The functions responsible for the boundaries and buffers are *AdaptiveGrid→updateGhosts* and *AdaptiveGrid→updateBC*. We now review these and each function where parts of the model are calculated.

AdaptiveGrid→updateGhosts
: As described in the thesis of Greenwood, not all the nodes in the domain are "real". Since the algorithm works by dividing the domain into smaller elements, sometimes when some of the nodes needed may fall in another element. This is where Ghost nodes come into play. This function will updates the Ghost nodes of the appropriate function using the surrounding information from other nodes.

AdaptiveGrid→updateBC
: This function updates the boundary conditions applied to the domain. In this code that could either be zero flux or periodic. This is called often since some functions depend on other functions or fields.

AdaptiveGrid→calcprePhi
: Given the equation of motion for the order parameter, this function calculates several of the functions, and derivatives of the order parameter that are needed for the numerical update. This particular function calculates things like gradients, anisotropy fields, diffusion interpolation functions, etc.

AdaptiveGrid→calcdPdt
: Calculates the time rate of change of the order parameter ($\partial_t \phi$), using the driving forces, gradients and interaction terms.

AdaptiveGrid→calcUnoise
: Thermal fluctuations are active in the equation of motion for the concentration field. This function determines those terms and their relation and behaviour.

AdaptiveGrid→calcpreC
: For the update of the concentration equation, several functions need to be calculated beforehand. Things to do with the anti-trapping flux for example, gradients in the

AdaptiveGrid→calcdCdt
: The time rate of chanhe, $\partial_t c$, is calculated in this function using those functions calculated calculated in the previous function.

Data Output and Visualization:
From time to time, something set by the user, the simulation will output to a file the current values of each of the fields. The typical name of the data file is "aaa100.dat", if for example the current status of the system at 100 time steps were outputted. The function that does this is *AdaptiveGrid→output*, from the source file **grid.cpp**. The data is in a single file in columns, where the value in each column corresponds to $x, y, \phi(x, y), c(x, y)$ respectively.

Inevitably one may wish to visualize the data. There's been a python script included, **plotad.py** (written by Sebastian Gurevich), that will read each data file from the output and generate png images that correspond to the order parameter, concentration and nature of the adaptive grid structure for each time step sequence. Once the script has been executed, it asks for several pieces of information in order to generate the images. The first is the file name, in the example above "aaa", then the first time step, last time step and increment between time steps. Next come the dimensions of the domain you wish to image, $x_{initial}$, x_{final} and $npoints_x$. Where if the whole domain is to be imaged $x_{initial} = 0$, $x_{final} = nx * size$ (from **initial.h**, and $npoints_x$ is the number of points used in the interpolation of the data, usually 1000 suffices but bare in mind that anything larger and it will take longer for the image to be generated. Analogously for the y-dimension as well.

References

[1] Nana Ofori-Opoku and Nikolas Provatas. A quantitative multi-phase field model of polycrystalline alloy solidification. *Acta Materialia*, 58(6):2155 – 2164, 2010.

[2] D. Montiel, L. Liu, L. Xiao, Y. Zhou, and N. Provatas. Microstructure analysis of {AZ31} magnesium alloy welds using phase-field models. *Acta Materialia*, 60(16):5925 – 5932, 2012.

[3] Michael Greenwood. *Ph.D. Thesis*. McMaster University, 2008.