1 Compilation Challenges

In order to maximize the potential reach of the system, I also included a CPU only version of the system. This imposes another low-level challenge to the system, which is to compile the same functions in a different way with different compilers (*g++* for CPU and *nvcc* for GPU).

There are two main parts of my solution. On the simulation side, for functions to be run on both CPU and GPU, I defined a conditional macro called *CPUGPU_FUNC* for function declaration. If the system will run on GPU, it will be compiled with nvcc and when this is the case, I define *CPUGPU_FUNC* as *__host__ __device__*, which is the function declaration necessary for a function to run both the host (CPU) and the device (GPU). When the user wishes to compile the code only on CPU, *CPUGPU_FUNC* will be defined as blank, not changing anything to function declaration. This conditional macro is important since keyword *__host__ or __device__* would make C++ code invalid if included, but are necessary if using GPU acceleration. Another change was in regards to usage of CUDA managed memory, which is used in the new system to transfer data between CPU and GPU. Following a similar design patter, I added another macro called *STATIC_VAR* and declare it as *__managed__* on GPU and nothing for CPU only simulations. Through these two strategies, functions are declared based on the platform on which the simulation will run.

Turning to the model files, I am forced to create different files for different simulation environments since when a static variable is passed to the compiler, *g++* will make assumptions such as the location of the variable, such assumptions are not valid when *nvcc* is used and thus files have to created in versions, normal implementation files (*.cpp*) for *g++* and CUDA implementation files (*.cu*) for *nvcc*. To work with two versions of the implementation files, I added macro declarations inside the header file so when the header file is linked to the implementation file, it will be compatible to either of them. For a similar reason, two versions of the test files are created for CPU only and CPU-GPU simulations.

Files with the two kinds of macros mentioned above can only be included once among all the object files since multiple inclusions will result in multiple declarations of the macros. The location of inclusion is thus an important choice. For the CPU-GPU simulation, I could not place the macros it in the normal simulation implementation file ($simulation.cpp$) since it also serves as the super class for CUDA implementation file ($simulation\_cuda.cu$). Thus included in GPU accelerations as well. Therefore, the only place that will determine

if a simulation is CPU only or CPU-GPU simulation is the actual test file. One of the header files also includes implementation details and for that file in particular, it was placed inside simulation_cuda implementation file since that will never be included in a CPU only simulation.