1 Time efficiency

During the preliminary phase of GPU acceleration, several factors prevented us from an efficient GPU implementation. Thus, in the new system, a great amount of effort was spent to address those problems, and now with simulation improvements and space efficiency, GPU acceleration is again implemented to the system, free of previous limitations.

1.1 Methodology

Methodology in converting CPU code into CPU-GPU code follows a similar pattern as demonstrated in the preliminary phase and thus will not be re-introduced here.

On top of the changes mentioned earlier, another attempt to minimize data transfer between CPU and GPU is made through initiating some data structures on GPU to reduce both memory requirement on CPU and data transfer required during simulation. For example, copies of $baby\_cl$ or $active\_rates$ may be created on GPU and no initial transfer of those structures from CPU to GPU will be needed during simulation process. Other necessary data transfer during simulation is now performed by CUDA managed memory. It allows data structure to be accessed on CPU when required.

1.2 Results

Simulation of CPU only code runs 1.6 to 1.9 times longer than the original system. Simulation of CPU-GPU code runs roughly 2 times longer than the original code for simulation of a parameter set, which does not fill all the parallel computational units on the GPU. However, parallel simulation of multiple parameter sets on GPU environment shows significant improvement in the new system. As a rough test case, it shows that when simulating twenty parameter sets simultaneously, the original code takes twenty times longer than single simulation, while doing so in the new system takes roughly double the time as running one simulation. Overall, simulation runtime of twenty parameter sets is five times faster than the same simulation through the original system, and with multiple GPU cards connected to the same work station, the new code can run efficiently on the station instead of relying on cluster.

The slight increase in runtime for CPU only code of the new system was a big concern for us and after profiling I found that the extra layer of indirection added to save memory requirement was a major source of time spent. Note that concentration levels are constantly accessed and updated in each time step for all species and all cells and thus number of memory access is very large. This is a price to pay for much smaller memory requirement on CPU or GPU and as discussed above, it is will be compensated through larger numbers of simulations running in parallel on the GPU.