Software Design and Architecture

Introduction  
From my understanding of the project, the goal is to create an intelligent simulation of atomic level interactions that occur in each material when inside a nuclear reaction. There are many factors to consider when implementing this solution, which is why each team has split up and divided the work. Each solution must take into account the physical side of things in order to create an accurate solution to the project proposal. The idea is that this work will be continued when we are finished with it and be used to further the research in the nuclear materials field by applying computer simulations and parallel GPU computing.

# Architectural Goals and Principles

The main goal is to provide an accurate simulation that can give tangible values that could be useful to someone researching in the field. Above all, this should be the main goal. Some side effects of this are methods of providing some visualization of this data, and ways for the app to accessible on a broad range of computers as per the project partner's request.

# System Overview

Architectural Patterns  
An effective architectural pattern that we might choose is the MVC pattern which divides the software up into components, one being the data, the other being the main function and lastly the visual representation of the data. Giving our users multiple ways to view the data that’s generated in our simulation allows us to deliver a complex solution with layers of abstraction to improve the user experience. This may also be useful in creating multiple data visualizations and a “tabbed” software that allows the user to view two sets of data at once and compare their results.

Component Descriptions

* UI: Provide some visuals to the user, and some meaning to data. In a project like this raw numbers are hard to interpret and gain meaningful knowledge from, so a visual representation like a graph, needs to be added.
* GPU Simulation / GPU Code: The purpose behind it is to provide the program with data. There needs to be some computational element to our simulation, and the GPU code either through CUDA or OPENCL will allow us to do that.
* Server: A possibility is to provide the users with a login to access the computational power of the OSU GPU farm and get their results quicker than they would running on a laptop.

# Data Management

We have already been experimenting with outputting csv data. This will most likely be the best method for graph representation. For user configuration and application data we have been using json files, which going forward would be the simplest and would make the most sense.

# Interface Design

An interface that we will need will most likely consist of a way to run the simulation, view the raw data, and the option to run multiple to compare the results. A tabbed system, or a way to panel the windows so that multiple graphs are on the screen at once will also be required.

# Considerations

## Security

There are very few security risks involved. There is no sensitive information that’s being delivered, and no authentication is needed.

Performance

The performance requirements need some kind of GPU at the very least to run the code locally, or alternatively we have access to the DGX server which is cloud based and no local GPU is needed. However, we are not certain on which path we will go down, for now we are assuming that a local GPU will be needed. The GPU allows the problem to be scaled up many iterations and do some kind of parallel computation. However, we are not sure where we will include the parallel section in the program.

## Maintenance and Support

The project partner expressed that this project would be passed to potentially another team in the nuclear materials field to be used and added upon. However, we are still partially unsure of how support will continue after this project.

Deployment Strategy

The target architecture as discussed with the project partner should be broad, he mentioned that he has a Mac so a start would be support on Windows and Mac. This makes it a little more difficult when implementing parallel as Mac has its own GPU API called Metal, which Windows has DirectX, but OpenCL and CUDA are also options for a Windows user. The project partner expressed that we could split the teams, one working on a Metal implementation and the others working on a Windows implementation.

Testing Strategy

I think a rough testing strategy that we might implement is the use of larger more general tests and unit tests, for example, we might use the unit tests to determine how the UI input is working as well as the parallel input. We might also use larger more general tests to ensure compilation across different OS and ensure that it’s uniform along each OS.

Glossary

CUDA – Compute Unified Device Architecture is Nvidia’s closed source API toolkit for parallel GPU computing.

OSU DGX – Is OSU’s GPU compute farm and houses many Nvidia GPUs for students and staff to run parallel tasks on.

GPU – Graphics Processing Unit