Stuff to be said in presentation:

Project Motivation (good place to state initial problem)   
Clear Aims  
Analysis of tools and Methods  
Go into detail   
Preliminary results  
Progress made satisfactory? (no lol)  
Is project on time and can a reasonable outcome be made  
Summary / Conclusion  
Future Work

Important details:

Table of Contents between chapters  
Separation slide between chapters  
Section titles  
Title Slide (Catching title, name, student id, supervisor, the occasion)  
Needs to be 10 mins (no more, no less)  
No more than 7 lines of text per slide and 7 words per line  
Use colour scheme

Script:

The N-Queens problem originally started out as the eight queens puzzle. It was proposed by chess composer Max Bezzel in 1848 to chess players in Germany. The aim of the puzzle was to place eight queens on a standard eight by eight chess board where no two queens could threaten one another. The queen can move in any horizontal, vertical or diagonal direction any number of tiles as seen in the left picture. It’s troublesome to find 1 solution to the problem although finding every possible solution was the real goal. On the left you can see an example of 1 solution to this problem and in total there is 92 solutions to the 8 queens puzzle.

As you can see in the table, the number of solutions increases dramatically with each increase of n. n = 27 is the highest order board that has been completely enumerated. Many solutions to a board can be derived from another solution via symmetry. You can see on the right hand picture 2 solutions for the n = 4 board one of which was derived from the symmetry of the other. There are 2 solutions to each size of board, total solutions and fundamental solutions. Fundamental solutions exclude the symmetrically derived solutions while total include all.

The backtracking algorithm that I will be using constructs a tree as it looks to solve the solution. It works by added an element to the tree on every successful queen placement. If the algorithm finds itself unable to find a queen placement, it will backtrack to the previous element on the tree and find the next available spot. The image shown is an example of the tree generated while trying to solve n=4. The N-Queens algorithm is regarded as perfectly parallel meaning it is effortless to separate the problem into parallel tasks. I aim to take advantage of this and will be using a High-Performance Computer (HPC) to run on.

I split my aim into 5 distinct smaller aims. I hope to achieve all 5 of these aims to a high standard.

Before I think of creating a parallel N-Queens algorithm I had to create a serial one to understand the algorithm and to compare back to when I have successfully run on the parallel computer. I made the first prototype in Java as it is a language I know well. I then moved on and created a working serial prototype in C++ as this is the language I will use on the HPC.

I thought I would show off a couple of the more interesting function in the algorithm, so to start with I have the checkSquare function from the C++ version. This function is used to check if a square can be threaten by another queen. It needs to check the X, Y and both diagonal axis and checking if a queen lies at any point, if it does the function can stop and just return false as only 1 queen threatening is required to nullify the square.

Here is the backtracking function in Java. The parameters being passed into the function is y being the id of the column being currently being checked, the board with the queen positions in and number of solutions. To move onto the next column a queen must have been placed in the previous column. So if the id of the current column matches the board size, then the board must have been completed. If I wanted to find 1 solution we could stop the program here and return the board, which would display a single solution. As we are looking for all the solutions we can increment the total solutions by 1. At this point we could also print the board or store it elsewhere if we wanted to. After this point the program is going to look to back track to the previous state which would have all but 1 queen completed and see if it can find another valid solution. If you take a look at the parameters you can see why this problem is perfectly parallel as you can run this algorithm at any level of board completion assuming the parsed board follows all the rules meaning the parallel tasks do not need to communicate with each other.

As I said earlier the N-Queens algorithm is perfectly parallel meaning it can be simply split into parallel tasks. The way this will be done is through running the algorithm from boards with set queen locations such as the ones seen below. The more of these set boards I have the more parallel tasks I will have and each parallel take will take shorter to complete.

Unlike many parallel programs, I will not need to worry about shared memory management orrith memory communication as it is unnecessary. Each parallel task can just return the number of solutions found and we can add up all the totals after the program has finished executing on a personal machine.

I am yet to run my algorithm on the Swansea Sunbird HPC due to unforeseen difficulties with project files. More time was spent on the C++ program than expected as the language took longer to learn than initially anticipated. This puts me behind my schedule although with extra time put into the project over the coming months, I am certain I will be able to complete all five of my aims to it’s fullest extent.

In summary the overarching aim of my project is to successfully implement a N-Queens algorithm onto a parallel HPC in order to get a speedup over serial methods. So far I have set benchmarks by creating implementing 2 algorithms in 2 separate languages ready to compare against. I hope that in the next month I will have a program running on a parallel HPC and then the comparison and improvement process can begin. As a final remark here is 1 solution to the 27 queens problem in which there is over 200 quadrillion solutions to.