The idea of this project is to study how to take advantage of this parallelism and explore how much faster we can make these calculations.

The first steps I would suggest are to read as much of the bibliography as you can It is fine if most of the reading is a bit superficial as it is impossible to learn everything in detail. But you should make sure that the core bits are understood in full detail (i.e. you can replicate the ideas on your own and could take an exam) For example you could target understanding in full glorious detail all the relevant contents of the book by Smith (Numerical Solution of PDEs) i.e. the chapters 1 2 and 3 [or some other book that you like] And the content of the slides that I sent, again in full glorious detail. On the software side of things maybe you could learn a bit about the intel compiler and some intel numerical libraries that I have heard of but not used. The intel compiler is an add-on that you can use in Visual Studio and should be installed in the MSc computer lab. I believe these intel numerical libraries will have some ultrafast tridiagonal solver that you could try to find and then test. There is a technique called intrinsic which means that you can write AVX instructions in C++, you could practice with some baby examples. In my HPC slides there are demos of how to use multithreading, you could have a look at that and try some baby examples in Visual Studio

1.    This video (and 2 more that follow it) is about optimization using SSE (parallel registers in the cpu) intrinsics (instructions in C++ that directly access the SSE registers).  
<https://www.youtube.com/watch?v=GwII1AJzKN4>   
It uses only integer data but should provide with some ideas as to how to do this kin dof stuff  
The guy is quite rude and funny.

2.    This is an overview of AVX <https://www.youtube.com/watch?v=yAvuHd8cBJY>  
it is probably a bit too much as it includes programming in assembly which might be a bit too geeky (unless you like it) but still interesting to have a look at it.  
It is part of a series on x64 assembly <https://www.youtube.com/playlist?list=PL0C5C980A28FEE68D> which you might wish to have a look at for your interest. I would initially think that programming directly in assembly is a bit too crazy but again it depends on your interest.

3.    This video explains the importance of laying out data properly in memory so that it is cached and minimise the number of cache misses  
<https://www.youtube.com/watch?v=G92BCtfTwOE>   
It explains how by multiplying two matrices of size 1000x1000 is much faster if you transpose the second so that the matrix multiplication calculations use data that is together in memory and is cached together.  
It might be relevant for the vector calculations involved in the numerical methods above.

1.    Develop code for BS FD grid pricer (uses Thomas Algorithm)

2.    Develop code for Heat equation (easier than BS) 2 dim grid pricer

You can search online if there is anything professional available, if so make sure it is referenced in dissertation]

Maybe try some open-source “famous” quant libraries.

This will be your “vanilla” base case (not optimized).

3.    Timing code.  
Check <https://www.pluralsight.com/blog/software-development/how-to-measure-execution-time-intervals-in-c-->  
I have used queryPerformanceCounter but maye there are more modern things.  
Keep in mind that timing code is machine dependent and also dependent on other code running.  
Also you’lll need to run code several times with different data (VS C++ compiler is very clever and will not run same calculation twice)

4.    Continue reading about:

a.    ADI, multithreading, AVX, VS Optimization switches see <https://www.youtube.com/watch?v=6AhMyE3Q0wQ> and similar videos

b.    MPI, OpenMP, Intel Compiler…

c.     Using Intel compiler

**d.    Parallel versions of Thomas algorithm: cyclic reduction**

5.    Make a plan as to what you want to optimize: code con only be parallelized by the compiler if it is parallel friendly.  
For example: Thomas algorithm for tridiagonal systems is not parallelizable.

6.    Start thinking about **structure** of dissertation: chapters, sections, subsections, bibliography,…

a.    Introduction

                                          i.    BS equation, in one and 2 variables, relation with the heat equation

                                         ii.    Necessity of numerical methods: Montecarlo and FDG

                                        iii.    Finite difference grids in 1d and 2d

                                       iv.    Optimizations that can be considered:

1.    Multithreading

2.    OpenMP/MPI

3.    CUDA/GPU

4.    AVX: through compiler switches

b.    Work Done

                                          i.    Parallelizing tridiagonal systems

                                         ii.    Necessity of numerical methods: Montecarlo and FDG

c.     Conclusions

d.    Bibliography: try to use academically sound references (less blogs and random write-ups (which can be very useful to learn)).