Imperial College London

Aeronautical Engineering

AE3-422 High-Performance Computing

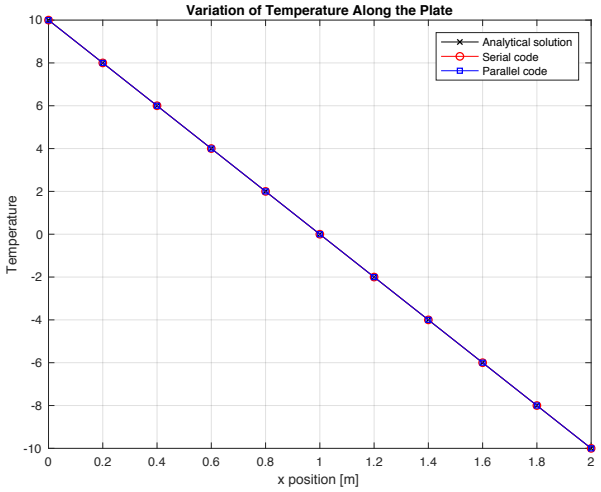
Coursework Assignment

Academic Supervisor:
Dr Chris Cantwell
Dr Omar Bacarreza

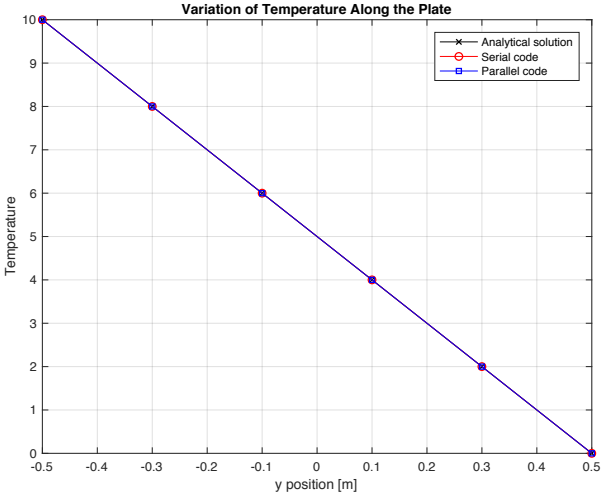
Trieu ho
CID: 01057749

March 23, 2018

1 Validation of Case 1 and 2



a) Case 1

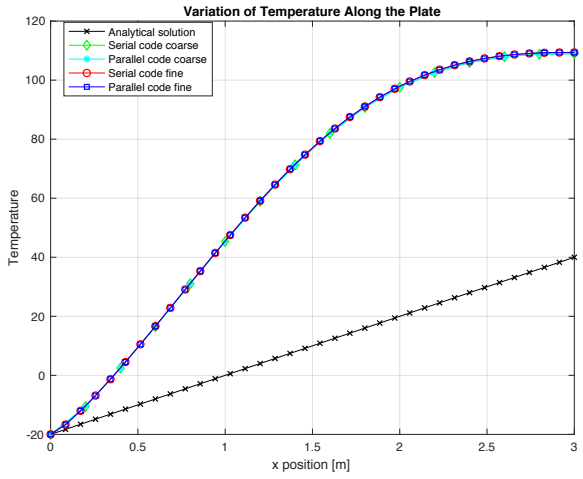


b) Case 2

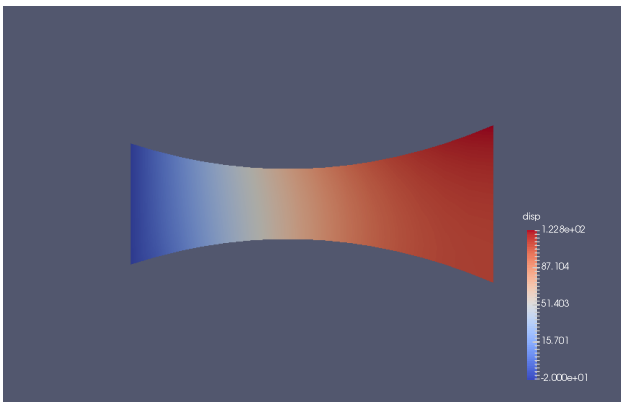
Figure 1: Temperature distribution found using parallel and serial code compared against the analytical solution.

We can see that the finite element solution for both the serial and parallel code represents the analytical solution well. This validates our code for cases 1 and 2. Since there is hardly any discrepancies between the serial and parallel code, we can just compare the time performance of the two codes and neglect any accuracy performance.

2 Case 3



a) Comparing serial and parallel code against analytical solution



b) Paraview diagram

Figure 2: Temperature distribution of case 3

Figure 2a shows that that the finite element solution is very different from the analytical solution. In addition, the parallel and serial code does not seem to have any discrepancies, even when the mesh size is improved.

3 Design Decisions

There were many considerations in mind when designing the code for optimised run time and memory usage. Firstly, with regards to the general coding theme, all matrices are stored as row-major arrays. This saves memory space as a multi-dimensional vector object are usually more costly to do operations with. Secondly, a lot of the matrix operations are handled using the Lapack and Blas external library, as these libraries offer optimised operation functions. I also divided the code up in different .cpp and .h files, where the functions are declared in the .h files and defined in the .cpp files. This greatly improves the readability of the code, but it does have a slight drawback in terms of run time.

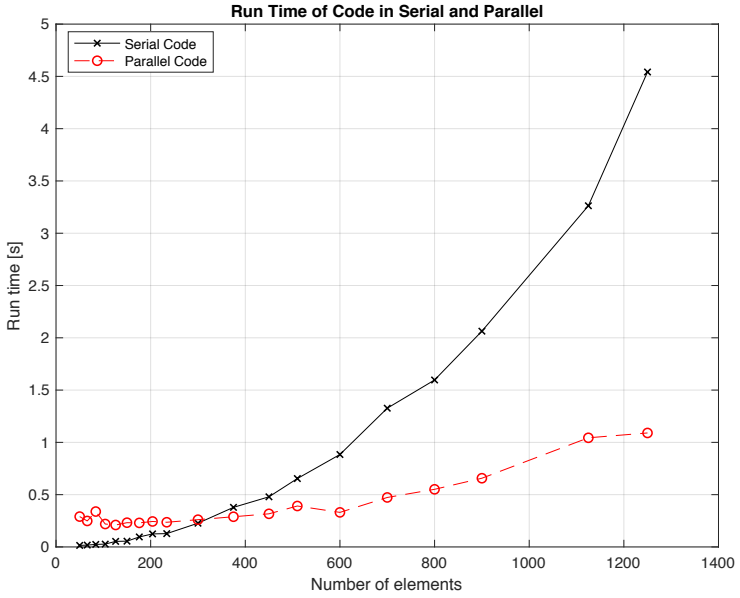


Figure 3: Comparing serial and parallel run time for case 1

In effort to improve the run time of the finite element program, parallel programming with multiple processors was used. In the mainp.cpp file, two processors were used. I first gave each processor half of the elements each and in parallel, they both found the respective element stiffness matrices. Next, we initialise two sparse matrices K1 and K2, both the size of the global stiffness matrix. So in parallel, these processors then translate the element stiffness matrices into their respective global matrix, K1 and K2. Processor 1, then sends K2 to processor 1, and then processor 0 sums up K1 and K2 in order to get the global stiffness matrix K. I then partitioned the global matrix and broadcasted KEE KEF and KFF to all processors. Finally, I tasked processor 0 to find the temperature solution and in parallel, processor 1 finds the solution to the reaction force.

In designing the parallel program, I tried to minimise the amount of communication between the two processors and split the work up as evenly as possible.