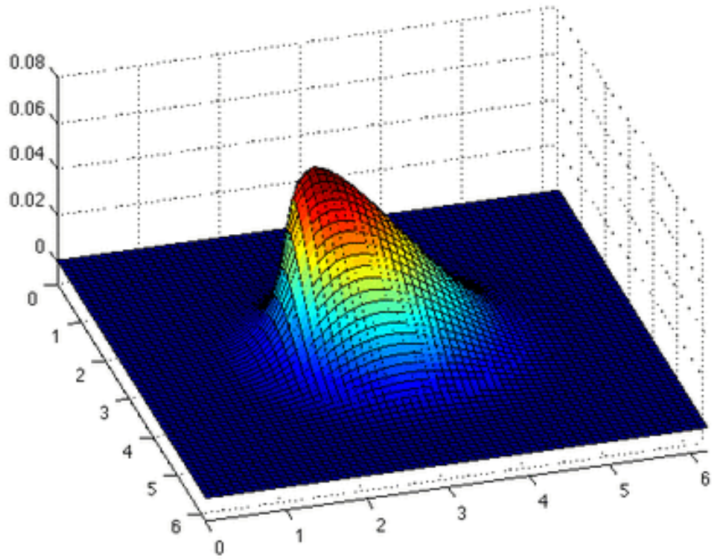
High Performance Computing

Department of Aeronautics - Spring Term 2019



Academic responsible:	Dr. Omar Bacarreza
	Dr. Dr. Chris Cantwel
Department:	Department of Aeronautics
Course:	H410 MEng in Aeronautics
Module:	AE3-422 High-Performance Computing
Academic year:	2018/2019
Student:	Javier Leguina Peral
CID:	01196703
Personal tutor:	Dr. Andrew Wynn

Department of Aeronautics
Imperial College London
London SW7 2AZ
United Kingdom

March 20, 2019

1 Introduction

This project had as an aim the numerical computation of Burgers' partial differential equation. This expression rules the behaviour of many phenomena such as fluid dynamics or non-liner acoustics. To implement the solution, an explicit integration scheme was carried out for both series and parallel programmes. For both cases, the characteristics of the problem had to be inputted through the Linux command line. The program was compiled and run on the Imperial College servers, accessed through a remote gateway.

Moreover, the code was developed on *C++* through the *CodeLite* platform. The main package used to perform the parallelisation of the code was the *C++* standard class *"mpi.h"*, provided as a Message Passing Interface.

2 Code verification an plot of results

In order to prove the correctness of the code, the method detailed in the report for this project was also implemented in MATLAB. This code was run to match the requirements set out: $T = 1$, a grid size of 10×10 with 2001 points in each direction and 4000 time steps. With these specifications, the result of the velocity field could not be properly visualised in a plot, so a reduced solution to the velocity field has been shown. This can be seen in Figure 1.

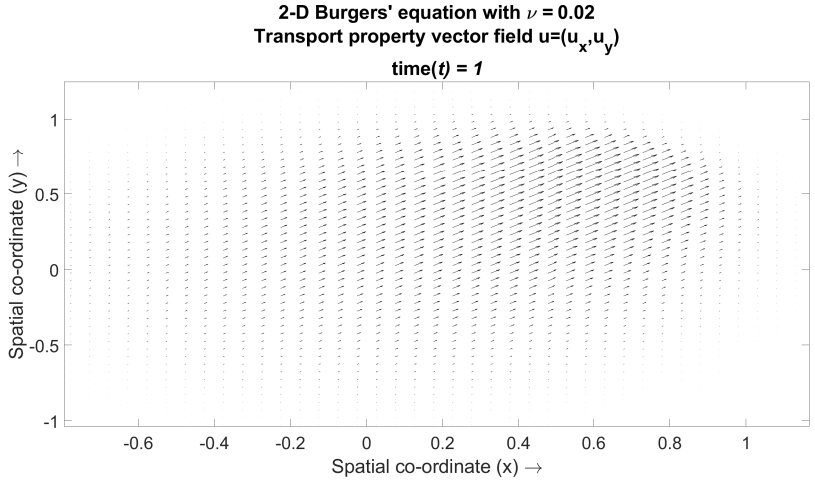


Figure 1: MATLAB computed result of the velocity field.

Unfortunately, the *C++* code failed to produce a suitable text file that could be imported to MATLAB. For this reason only a sample solution can be provided. This is shown in figure 2. These results however, diverged slightly from what was obtained when the *C++* code was run.
Furthermore, in order to perform a suitable validation of the results, the velocity fields calculated in MATLAB and *C++* were compared. One of the most significant observations was that, following the solution shown in Figure 2, results on the 2^{nd} and 3^{nd} quadrants were consistently smaller than those on the left mid-plane. Knowing that energy results were mostly off target, this observation pointed towards later in the code as the location of the errors. A table with the calculated energy values for the diffusion case can be found in Table 1.

Table 1: Values obtained for the energy in various cases.

Energy Values - Diffusion	$N_x = N_y = N_t = 10$	$N_x = N_y = N_t = 100$
$P_x = P_y = 2$	4.75×10^{-6}	2.05×10^{-7}
$P_x = P_y = 3$	1.42×10^{-8}	5.25×10^{-8}
$P_x = P_y = 4$	4.47×10^{-5}	2.17×10^{-7}

Upon comparing the results of the *C++* code and the MATLAB script, it was noticed that most of the divergence between calculations occurred in the implementation of the trapezoidal method for the integration of the modulus of the velocity vectors. This phenomenon is a direct cause of the lack of consistency in the figures displayed in Table 1. While the reference value for the energy was found

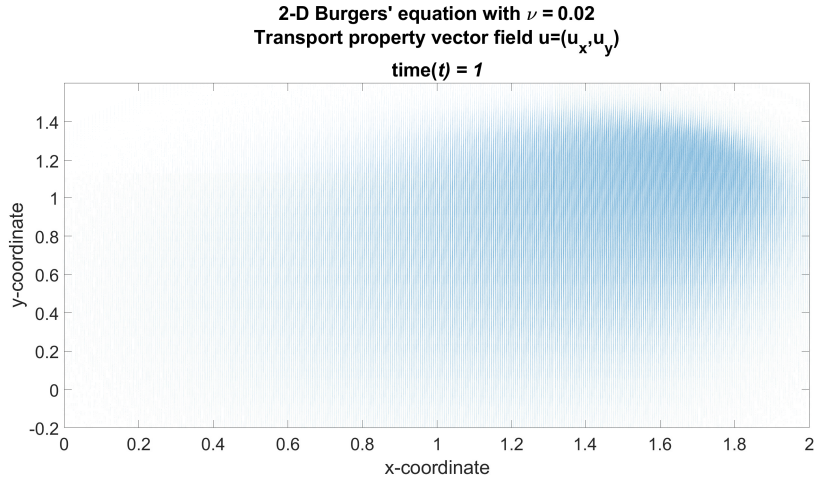


Figure 2: C++ numerical sample results for 2001 grid points in each direction with a final time of T = 1 with 4000 time steps.

to be approximately around the 10^{-1} order of magnitude, the variation between 10^{-8} and 10^{-5} seen in the table suggests the methods used were not robust.

Most notably, as the number of data points decreased, the value of the energy approximated the estimated value, with $N_x = N_y = N_t = 5$ yielding an energy value of $E = 0.298$ for $P_x = P_y = 2$ and $E = 0.00778$ for $P_x = P_y = 3$.

3 Performance Considerations

In order to discuss the performance of the code, a brief introduction to the methods used must be done. In order to locate each process, a relation between the rank and the row and column that particular sub-matrix occupied in the general matrix was developed: $row = floor\left(\frac{rank}{Px}\right)$ and $column = rank - Px \times row$. That, coupled with the identification of the starting indexes in y and x for each of the elements that composed each sub-matrix, gave enough information to properly divide and characterise each of the partitions of the matrix.

One of the main performance considerations revolves around the need that the code performed the integration for only 1 process. In order to do this, a second class analogous to *Burgers.h*, *BurgersSingle.h*, was included. This package allowed the computation of the velocity field in that particular case. Nevertheless, this had to be accompanied by a conditional statement that diverted the code towards the implementation needed, either series or parallel. This required two extra files that take away some of the performance, as it implies two more files must be compiled and executed.

On the other hand, due to restrictions in the manipulation of the classes and the initialisation of variables, the deletion and clearing of dynamic memory could not be performed, as it continuously resulted in errors. This point has a high computational cost, especially when initialising variables that will not be used. This was the case for numerous variables in numerous processes.

Finally, as it has already been said, the code lacked robustness as the number of operations increased. This, coupled with the observation that, for lower number of processes the resulting velocity field was closer to the MATLAB - reference result, suggest the use of the Message Passing Interface was not optimal. Overall, a better performance could have been accomplished with a more rational use of the classes provided and the MPI.

Javier Leguina Peral
March 20, 2019