Imperial College
London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF AERONAUTICS

High Performance Computing

Author:

Roger Gonzalez (CID: 00839673)

Date: March 26, 2017

Measures taken to improve efficiency:

- Banded Storage of stiffness matrix
- Creation and storage of only the diagonal for Mass matrix
- Minimization of message size being passed by MPI (task4)
- Minimization of arrays created by writing over arrays no longer necessary
- Repetitive use of generalized functions for linear algebra operations
- Separation of compilation and run in makefile

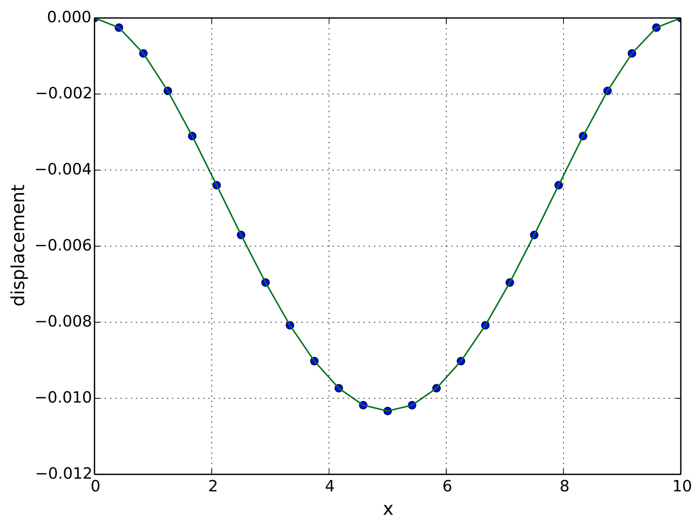

Figure 1: Comparison of Numerical and Analytical solution for beam (24 elements)

The Dynamic system exhibits a relationship between loading time and oscillation amplitude in both the explicit and implicit schemes. Figures 2. and 4. trace the path of the center point of the beam as it is loaded and then oscillates. Figures 3. and 5. show the relation in a log scale where it can be seen to behave like an exponential dampener.

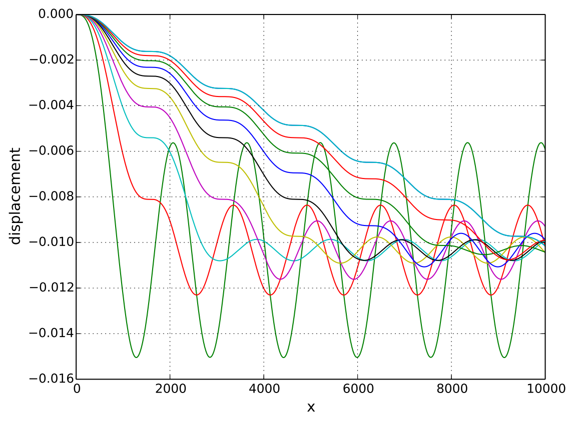

Figure 2: Centrepoint displacement with varying loading time

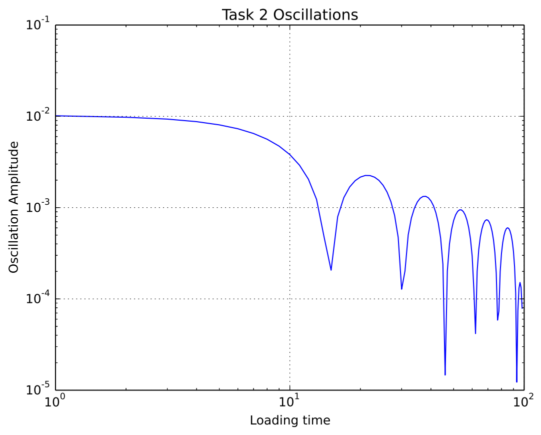

Figure 3: Oscillation amplitude with varying loading time (10 samples)

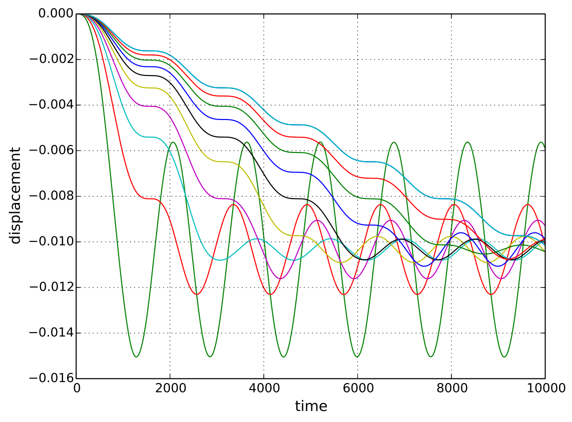

Figure 4: Centrepoint displacement with varying loading time (10 samples)

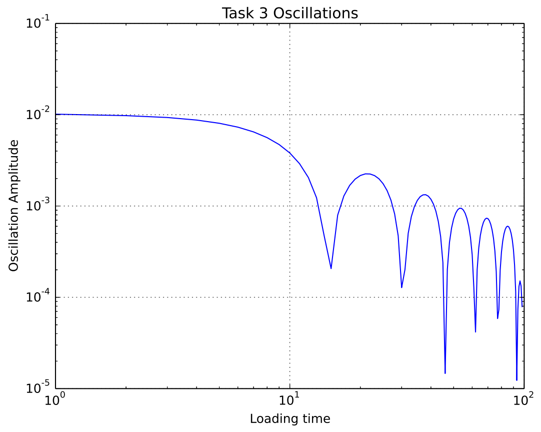

Figure 5: Oscillation amplitude with varying loading time

In order for the system to be solvable in multiple processes, each process needs to have a shared part with the other processes. To minimize the number of calculations done, it is found that for this particular system, the processes must share at least 4 equations. Implementing this way, means that the messages sent between processors are minimized in size. The Explicit scheme is tested with 1 and 2 process implementation. The run times, in seconds, are given in Table 1. for a variable number of elements. The benefits of parallelization are only evident when the number of flops becomes significant.

N_t	N_x	1 Process	2 Processes
10000	12	0.2	1.04
10000	24	0.3	1.05
20000	48	0.91	1.11
20000	100	1.86	1.17
100000	200	2.39	2.39
100000	400	68.34	9.09

Table 1: Run time (seconds) for the explicit scheme for increasing number of elements(Nx)

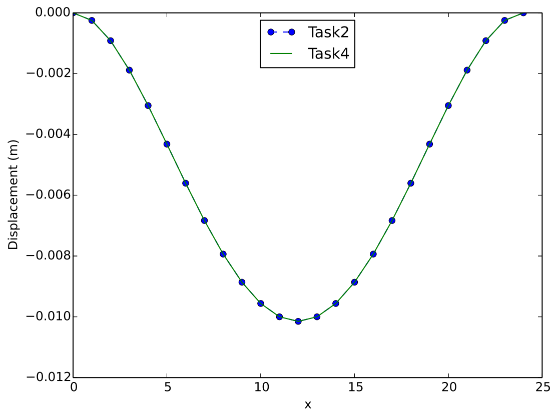

Figure 6: Explicit scheme in Serial and in Parallel (N=24)

Figure 7. presents the result from task 5 as compared to task 3 for the implicit scheme implementation. Implementation of ScalaPack is is done through a function and hence only works for this particular system. In the range tested, the benefit of larger parallelization is not evident, however for larger systems , $Nx > 2000$, performance improves significantly. This program does not efficiently scale to such a size. The aim of the program is achieved effectively and implementing almost all of the same functions as for task3.

N_x	1 Process	2 Process	4 Process
12	0.5	1.7	2.35
24	0.4	1.86	2.51
48	0.82	2.77	2.83
100	1.56	2.87	3.19
200	3.09	3.7	3.84
400	6.17	34.32	21.59
600	9.34	41.87	61.11

Table 2: Run time (seconds) for Implicit scheme with increasing number of elements

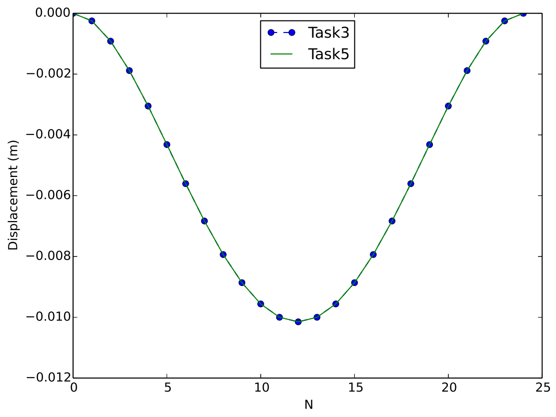

Figure 7: Implicit scheme in Serial and in Parallel (N=24)