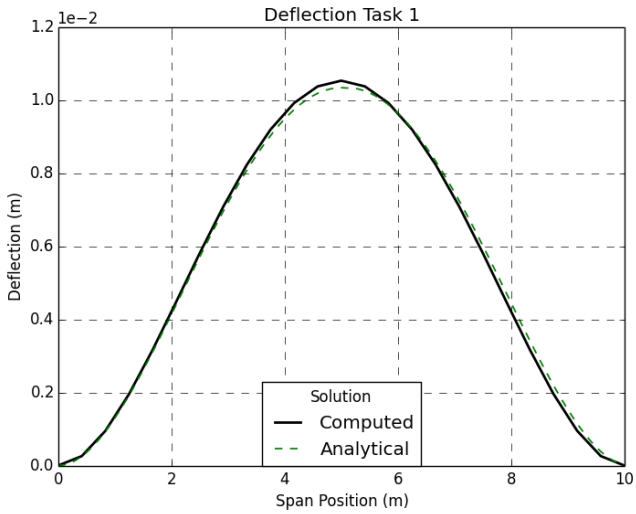
AE3-442 - High-Performance Computing Coursework

Giacomo Sammons – CID: 01252194

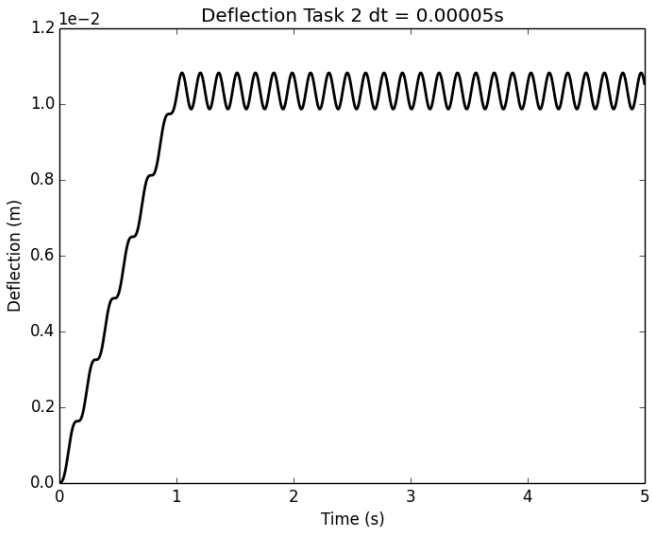
GitHub repository: https://github.com/sammonsg/HPC_coursework

The code for the solver was written from the very beginning with scalability, memory efficiency and minimal operation count. The linear solvers leverage as much as possible on the LAPACK library, which is more prone to vectorization on compile, as well as avoiding memory allocation and destruction within the iterator. This includes the switching of the pointers for the u_n and u_{n-1} pointers rather than performing deep copies, pre-calculation of matrices outside. On the parallel code, communication was kept at the mathematical minimum and MPI_barriers avoided. For ease of coding and readability, many helper functions were used, such as a sparse addressing function, which converts full matrix coordinates to banded coordinates, or one which swaps pointers to arrays.

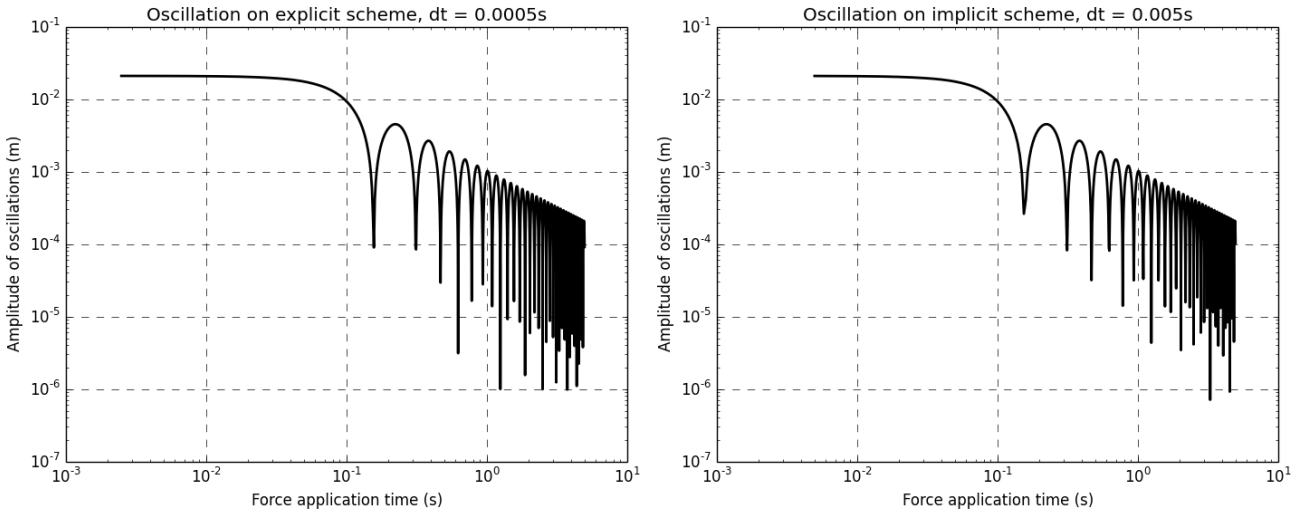
The beam under investigation for this coursework is loaded both with a distributed force and a concentrated one. As such, we may use the theoretical results derived from linear mechanics and compare them to the solution obtained in the static solver, as described in task 1 of the coursework. Below is an overlay plot of the calculated solution against the theoretical one. It can be observed that they mostly match except in the middle region. The case was run using 24 elements and as such, discretization may have let error affect the solution.



The static solver fails to capture the transients that occur due to inertia of the beam. For this, an explicit 2nd order in space method was used. The force is applied linearly from t = 0 to full values at t = 1. If we were to track the deflection of the center point, it would be possible to observe the oscillatory motion that occurs throughout time. This is due to jerk and jounce rates when the force rate switches between linearly increasing and constant. Below is a plot of the deflection/time.



The oscillations may depend on the rate of application of force, so for both the explicit and implicit scheme the amplitude of these oscillations was measured, starting from the moment when the full force was achieved. It can be observed from the two log-log plots below. The presence of peaks and troughs may be explained as the consequence of the timing of the changes in force applied. The initiation of the force constitutes the first impulse which puts the system in oscillatory state, then the second, based on the timing relative to the cantilever's undamped frequency, may either increase or decrease the amplitude. Thus the observed troughs. The peaks represent the maximum amplitude possible, and it decreases because the strength of the impulses is weaker as the force application rate is decreased.



In order to improve the performance of the solver, the explicit scheme was rewritten to use MPI and manual communication of the nodes. For the implicit solver, MPI was not sufficient, as there was a step to solve Ax = b. So ScaLAPACK was implemented. As expected, the overhead of exchange of information through network causes both solvers to run slower in parallel than the serial code. The scope of multiprocessor is large domain though, where the overhead is comparatively smaller than the problem being solved. As such, an analysis of speed was done for different element counts. The results are available below in a table. The implicit code ran on 200'000 iterations while the explicit one on 2-million iterations for 1s. Both parallel solvers were run on 4 cores on a 2.9GHz Intel I7-6920HQ. It can be observed that the explicit solver is far more scalable than the linear solver, but this is due to the nature of the matrix operations performed on the explicit scheme. The implicit solver shows relative improvements with increase in number of elements but is still inferior up to 768 elements.

ELEMENTS	SERIAL EXPL. (S)	2-NODE EXPL. (S)	RELATIVE SPEED	4-NODE EXPL. (S)	RELATIVE SPEED
24	3.015	2.31	131%	2.297	131%
48	6.226	3.961	157%	3.25	192%
96	12.151	7.68	158%	5.013	242%
192	24.281	14.145	172%	8.48	286%
384	49.768	26.215	190%	15.925	313%
768	103.15	56.454	183%	33.783	305%

ELEMENTS	SERIAL IMPL. (S)	2-NODE IMPL. (S)	RELATIVE SPEED	4-NODE IMPL. (S)	RELATIVE SPEED
24	1.304	5.382	24%	8.27	16%
48	2.917	7.968	37%	11.102	26%
96	5.767	12.777	45%	13.286	43%
192	12.55	22.151	57%	19.368	65%
384	27.207	42.708	64%	31.759	86%
768	54.558	80.374	68%	57.709	95%