High Performance Computing
AE3-422

F. M. Grabner - 01220997

Imperial College London - Department of Aeronautics

Question 1

Matrices have been set up in banded format to minimise the memory utilised. Then the system is solved using Lapack[1] function DGBSV. However to leverage the Fortran routines in BLAS[2], arrays are in column-major storage format.

Figure 1 shows the analytical solution against computed solution using the static equations. The computed solution is practically identical to that of the analytical, for the coarse discretisation of 24 elements.

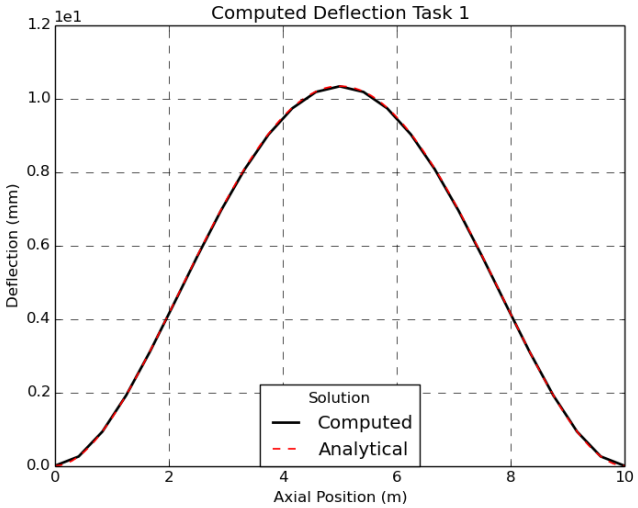
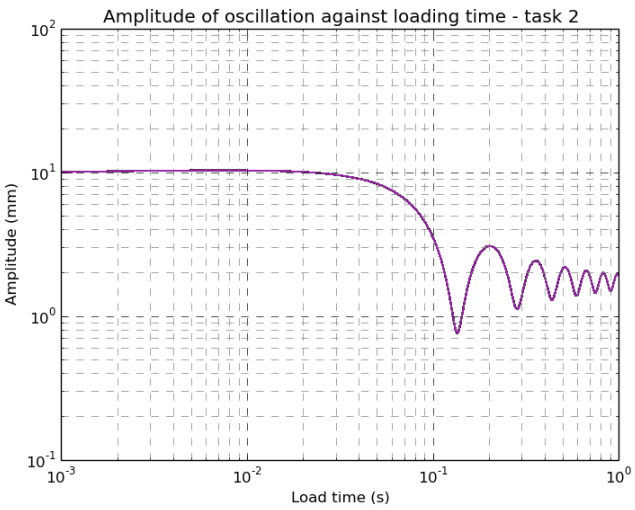


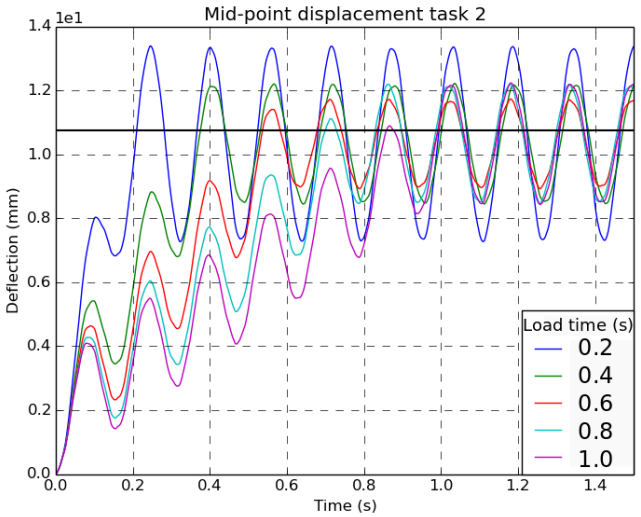
Figure 1: Static against analytical solutions, using $N_x = 24$.

Question 2

Matrix storage identical to task 1. Matrix-vector multiplication has been implemented using BLAS level 3 routine `dgbmv`, and then the explicit central-difference scheme is solved using `dgbsv`.



(a) Amplitude of oscillations.



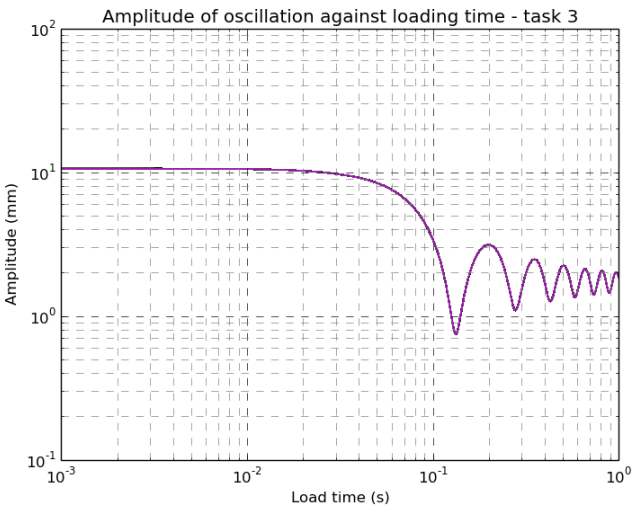
(b) Oscilation with different loading factors.

Figure 2: Grid Independence Test.

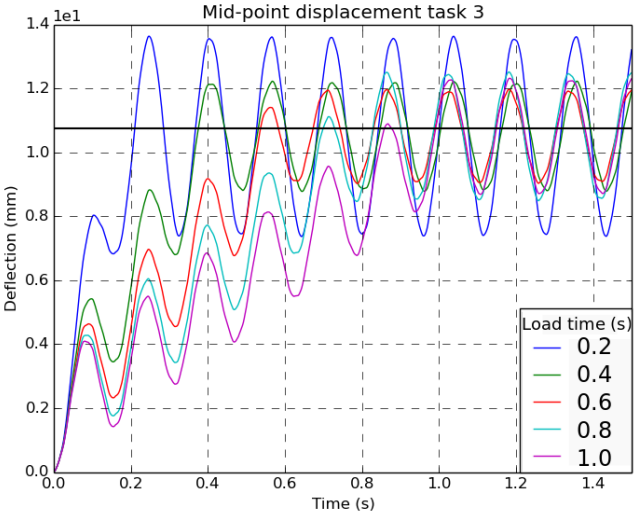
Figure 2b shows the deflection at beam midpoint against time, for a number of different loading times. In figure 2a the amplitude of oscillations as the loading time is increased from $0 - 1(s)$ in steps of size $\Delta t = 0.001s$ is shown. In general as load time increases, the amplitude of oscillations are seen to decrease. However a number of peaks and troughs exist which are of significantly higher and lower amplitudes. The peaks of amplitude likely correspond to the natural frequency modes of the system, and thus we see that a time of $0.8s$ exhibits lower amplitude oscillations than that of $1s$.

Question 3

Matrix storage identical to task 1. Matrix-vector multiplications have been implemented using for loops, as the identity matrix can be stored as a vector and using compiler flags `-O2` or `-O3` produces faster code than the BLAS routine `dgbmv` for the diagonal matrix, as the compiler performs some loop-unrolling and the call time is saved. The matrix system was solved using `dgbsv`.



(a) Amplitude of oscillations.



(b) Oscilation with different loading factors.

Figure 3: Grid Independence Test.

Using the implicit Newmark scheme the beam exhibits an almost identical response to that of the explicit integration scheme. Again some sort of frequency response is seen in that certain frequencies have significantly lower amplitude than others however the general trend is to an oscillation at order of magnitude $\approx 1mm$. Due to the implicit scheme the solution is found using far fewer timesteps.

Question 4

Matrix storage as in task 1. In task 4 the beam was solved across multiple processors. Parallel linear algebra solver Scalapack[3] was **not** used. To perform the matrix multiplication BLAS routine `dgbmv` was used with an overlap region which allowed contributions to the solution to be accounted for across processes. These contributions were then passed across nodes using MPI[4] at the end of each timestep. The solution was then found simply by inversing the diagonal mass matrix, as this the requires simply multiplication against the right hand side of the equation. However the inverse matrix was stored in an independent array as division is a costly operation,.

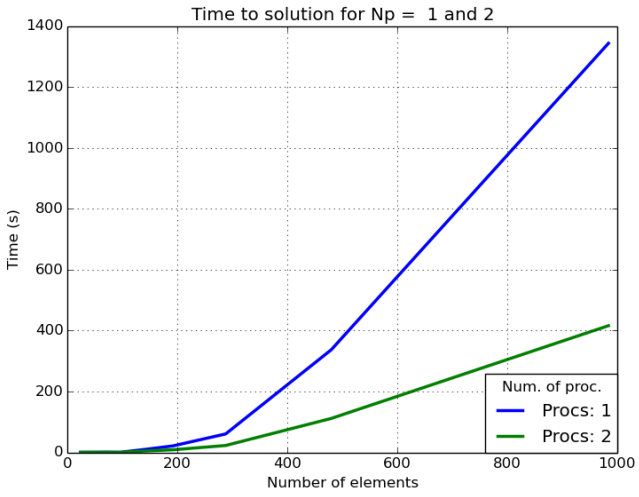


Figure 4: Time to solution as problem size increases with 1 and 2 processors.

Above in figure4 the time to solution for different problem sizes is displayed. Clear to see that, except for small problems of less than 200 elements, the MPI routines are significantly quicker. The strong scaling is not seen to be linear, this is due to the problem size not being kept uniform. The decision was taken not to spend too long finding the minimum timesteps to keep the larger simulations stable and rather to simply match increase in N_t with increases of N_t.

Question 5

Matrix storage in banded as always, however with a buffer of zeros on top for use with Scalapack. Task 5 splits the Newmark method across 1, 2, and 4 processors. This problem allowed use of Scalapacks `pdgbsv` routine. Implementing this requires equal block sizes across processes. However the definition of the problem makes this difficult, in order to leverage Scalapack all ranks were filled except the last rank which received any left-over nodes. Then the end zone is padded with zeros and 1s across the diagonal for matrices, filling simply with 0s for vectors. This meant the system was solved easily and just the final points could be discarded.

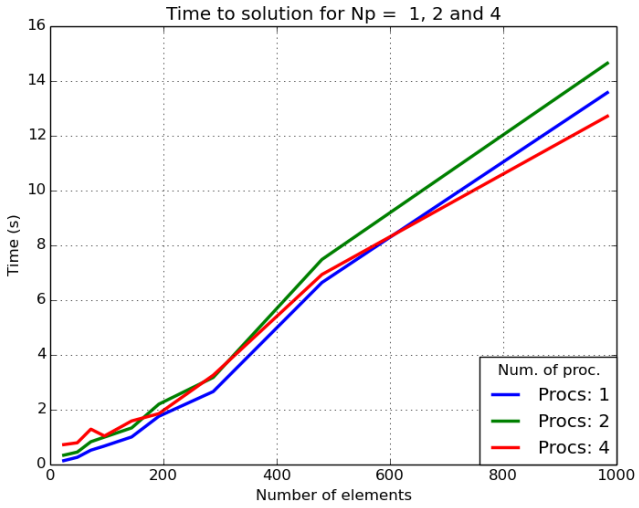


Figure 5: Time to solution as problem size increases with 1, 2 and 4 processors.

Figure 5 shows the time to solution for 1,2 and 4 processors. For this implicit scheme the timesteps were kept constant at $N_t = 10,000$ for all sizes of N_x. The parallel programs only begin to become faster as $N_x >> 500$. This indicates that whilst scalapack efficiently hides communication in the Blacs sublayer, communication takes a large chunk out of the performance, in comparison to serial BLAS routines. However as the problem becomes larger this effect reduces.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft.*, (3):308–323, 1979.

[3] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[4] MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2, September 4th 2009. available at: `http://www.mpi-forum.org` (Dec. 2009).