MAP55612
Poisson Exercise

Harold Hodgins

Hilary 2024

Background

The discretised Poisson equation was solved using the Jacobi method for a finite difference matrix. Two parallel implementations were developed using 1D and 2D domain decomposition. The parallel communication was achieved using OpenMPI in C.

1D Decomposition

This involves splitting the grid into strips of a given number of columns depending on the number of processors used. The strips on each process (we call this local grid) will contain neighboring ghost columns which are required for the five point stencil which carries out the Jacobi iterations. The update step requires the point to be updated, as well as its four nearest neighbors, one of which will be within the ghost column when the point to be updated is in the last columns of the local grid.

1D Gather Grid Function

The gather grid function for the 1D program works by using `MPI_Send` and `MPI_Recv` commands. These commands send the columns of the local grids to the corresponding spot in an array called `global_grid` on the root process (0). They send the columns due to the fact that the grids are stored conceptually as column-major order, so a strided data-type is not required here to send columns.

2D Decomposition

This involves splitting the grid along both the vertical and horizontal axes. This is more involved in terms of the parallel communication required as it is now necessary to send non-contiguous memory. There is now ghost rows as well as ghost columns. The sending and receiving of ghost rows, and rows in general for this program, requires the use of a strided data-type, which requires the use of `MPI_Type_Vector` .

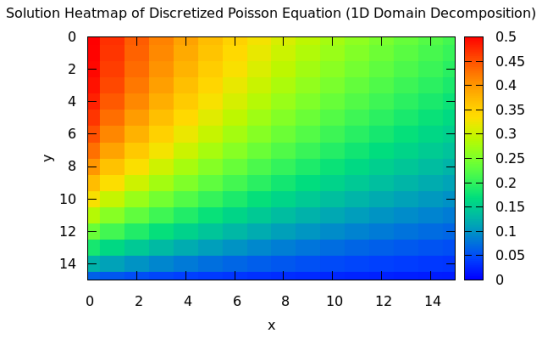
A customised strided data-type `column_type` was created which contained the specific number of elements in the rows of the local grids, and was used to send and receive the rows on each process.

2D Gather Grid Function

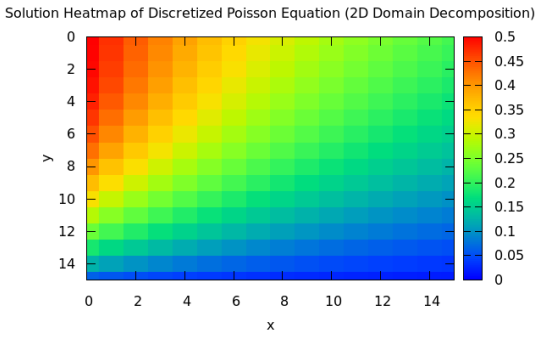
The 2D gather grid function was similar to the 1D version in that it worked by sending the columns of the local grids to the correct locations in the global grid on process 0. The only difference was that the row indexes had to be taken into account as well as the column indexes in order to send and receive the columns into the correct spot in the global grid, as the entire column was no longer being sent, but rather a segment of each column due to the extra dimension of decomposition.

Results

The 1D and 2D decomposition programs were ran on 4 processors for grid sizes of 15×15 as well as 31×31 (excluding boundary points). The plots below show heatmap plots of the global solution gathered onto process 0.

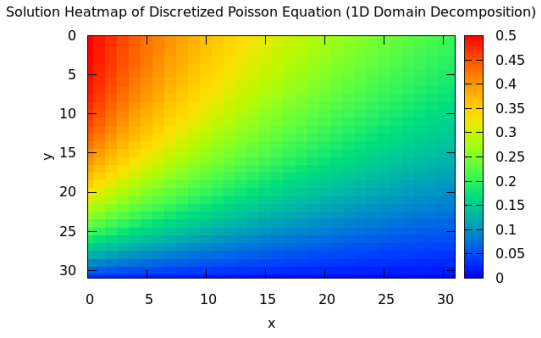


(a) 1D 15×15.

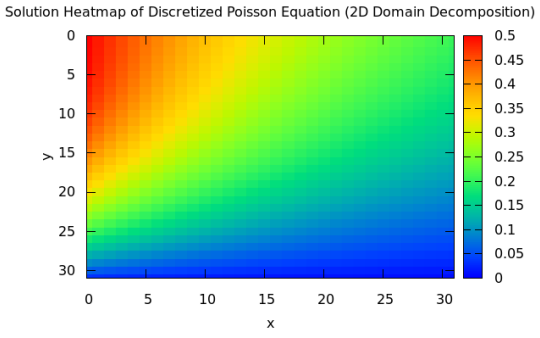


(b) 2D 15×15.

We note the resemblance between the two solutions which is a good sign as it shows both methods produce the same results. We also consider the plots for the 31×31 grids for each method.



(a) 1D 31×31.



(b) 2D 31×31.

Finally we can compare the above plots to the analytical solution for the 31×31 grid.

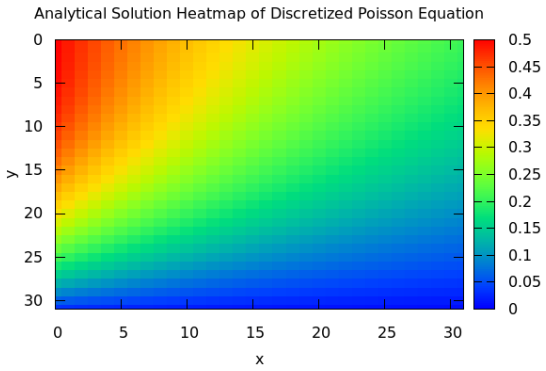


Figure 3: Analytical solution 31×31.

We note that the solutions are essentially identical, showing that the Jacobi method applied to a finite difference grid is an accurate method of solving the discretised Poisson equation.

The 2D decomposition program was ran on `seagull` (`link`) to make use of 16 processors. The iterative solver converged for each grid size. The 15×15 grid took 0.0062544s, and the 31×31 0.022443s. When comparing this to the running time for 8 processors, the 31×31 grid had a run time of 0.013888, and 0.014040s on 4 processors. Due to the running time being so low it is difficult to observe much speedup. However, if we are to increase the size of the grid to 100×100, running times of 0.117965s and 0.074587s were observed for 4 and 8 processors respectively. Significant speedup was not observed for more processors past 8, but this shows the strong scaling capability of the algorithm for larger grid sizes. The plot for the 100×100 grid was very high resolution, and it is plotted below.

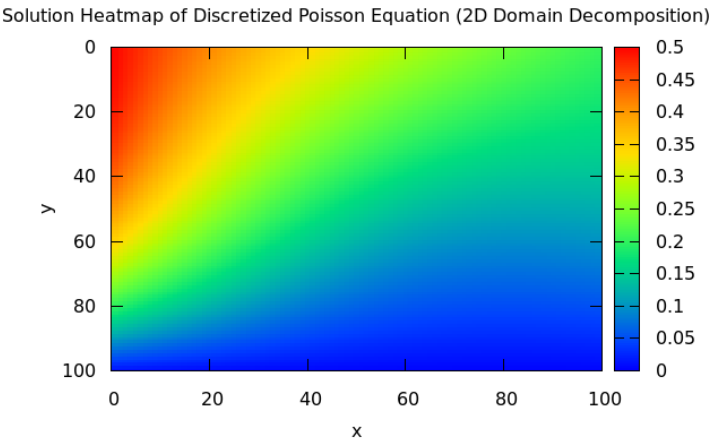


Figure 4: 2D 100×100.

Conclusion

It was found that OpenMPI is a powerful tool for creating parallel programs in C. The discretised Poisson equation was solved using 1D and 2D domain decomposition. The methods showed signs of strong scaling when larger grid sizes were used.