CM30225 Parallel Computing
Assessed Courseork Assignment 1

December 17, 2016

1 Approach

In order to parallalise the relaxation problem we need to be able to split the matrix up into chunks then let each node relax its own chunk with as little communication between nodes as possible.

In order to chunk the matrix up we want to give each node as similar numbers of rows as possible to distribute work evenly. In my implementation first we calculate the rounded up value of the number of rows minus 2, because the border rows arn't relaxed, divided by the number of nodes. Each node is the allocated this many rows until there are no rows left in the matrix. For example if we run 5 nodes on a 100 x 100 matrix the rounded value is 20 so the first 4 nodes are allocated 20 and the last is allocated 18.

The least each node needs to know after each iteration is the values in the row to the left and right of its chunk. For example given a 6 x 6 matrix and 2 nodes, the first is allocated row 2 and 3 and the second rows 4 and 5, after one iteration all the first nodes needs is row 4 and the second node only needs row 3. Therefore after each iteration each node sends its outside rows to the appropriate nodes.

We can improve this by calculating the two outside rows before any others and sending them, so the node can be calculating the rest of the chunk while the communication happens. To do this MPI_Isend and MPI_Irecv are used as these are non blocking so the node won't wait for the other node to receive the value before continuing.

2 Testing

To correctness test the program it was split into three separate chunks which would be tested spectrally. The first is the ability to perform one iteration correctly and replace all cells with the average of their four neighbors. The second

is that the iterations will stop after the precision is reached, and finally that when more than one processor is used the output is the same.

To test the first case the program was run on a 5x5 matrix, surrounded by 10's, for one iteration and the answer captured. If the program works correctly the output should consist of 5 in each of the corners and 2.5 around the edge. As shown by the output below this is working correctly.

10	10	10	10	10
10	5.0	2.5	5.0	10
10	2.5	0.0	2.5	10
10	5.0	2.5	5.0	10
10	10	10	10	10

The next step is too check that the iterations stop when the precision is met. In order to test this the same 5x5 matrix was used. The precision was first set to 5 this should yield one iteration as the precision is met immediately, the precision was then halved to 2.5 which should yield 2 iterations. As shown by the result table below this worked as expected.

iteration	diffrence	precision	continue
1	5	5	0

iteration	diffrence	precision	continue
1	5	2.5	1
2	2.5	2.5	0

Finally to test that these conditions still hold when multiple threads are used a 10x10 matrix was run using 1, 2 and 4 processors and each where checked to make sure they were identical. below are the outputs from the runs with 1 and 4 processors.

Table 1: 1 Processor

10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
10.000	9.981	9.965	9.953	9.946	9.946	9.953	9.965	9.981	10.000
10.000	9.965	9.934	9.911	9.899	9.899	9.911	9.934	9.965	10.000
10.000	9.953	9.911	9.881	9.864	9.864	9.881	9.911	9.953	10.000
10.000	9.946	9.899	9.864	9.846	9.846	9.864	9.899	9.946	10.000
10.000	9.946	9.899	9.864	9.846	9.846	9.864	9.899	9.946	10.000
10.000	9.953	9.911	9.881	9.864	9.864	9.881	9.911	9.953	10.000
10.000	9.965	9.934	9.911	9.899	9.899	9.911	9.934	9.965	10.000
10.000	9.981	9.965	9.953	9.946	9.946	9.953	9.965	9.981	10.000
10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000

Table 2: 4 Processors

10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
10.000	9.981	9.965	9.953	9.946	9.946	9.953	9.965	9.981	10.000
10.000	9.965	9.934	9.911	9.899	9.899	9.911	9.934	9.965	10.000
10.000	9.953	9.911	9.881	9.864	9.864	9.881	9.911	9.953	10.000
10.000	9.946	9.899	9.864	9.846	9.846	9.864	9.899	9.946	10.000
10.000	9.946	9.899	9.864	9.846	9.846	9.864	9.899	9.946	10.000
10.000	9.953	9.911	9.881	9.864	9.864	9.881	9.911	9.953	10.000
10.000	9.965	9.934	9.911	9.899	9.899	9.911	9.934	9.965	10.000
10.000	9.981	9.965	9.953	9.946	9.946	9.953	9.965	9.981	10.000
10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000

Given these results it is concluded that the program correctly calculates the relaxation of a matrix with any number of processors.

3 Scalability Investigation

3.1 Speedup

The speedup was calculated using a 10,000 by 10,000 matrix using a selection of nodes between 1 and 64. The results are given in the table below.

processors	Time (S)	Speedup on P processors
1	477	1
2	239	2
4	120	3.98
8	61	7.82
16	31	15.39
32	18	26.5
48	13	36.69
64	10	47.7

As the results show the speedup on P processors is very close to the number of processors used. This is true until over 16 processors, at this point the extra communication overhead added by using another processor starts to outweigh the extra processing power gained. Figure 1 shows this in graph form.

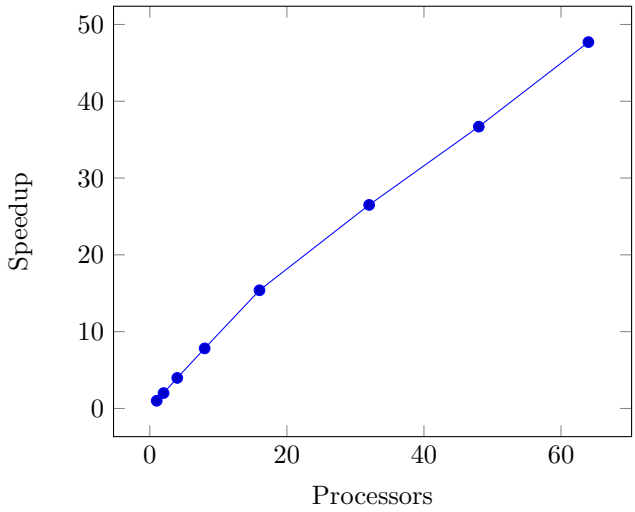

Figure 1: Speedup

3.2 Amdahls Limit

Amdahls Limit is the maximum limit speedup can reach, this is because a certain ammount of the computation must be done in serial and therefor adding more processors will not reduce the time this takes.
The only parts of the system that have to run in parallel is the checking if all the nodes have finished and the rebuilding of the matrix at the end of the computation. However the creation of the matrix and the swapping of the read and write matrix happens on all processors so adding more will not decrease this time. Also something to note is as the number of processors increases the amount of time taken to check if each processor has done and rebuild the matrix will increase as there are more processors to communicate with.
Due to the fact that the serial parts of the problem increase as the number of processors are added the theoretical amdahl limit was not calculated.

3.3 Slowdown

When smalled matrix's were used it was noticed that as more threads were added the time time taken to complete would increase. This is due to the overhead of communicating between processors is more than the time saved by splitting the problem up further. Slowdown can be observed when a 1000 by 1000 matrix is used, results are show in Figure 2 below.

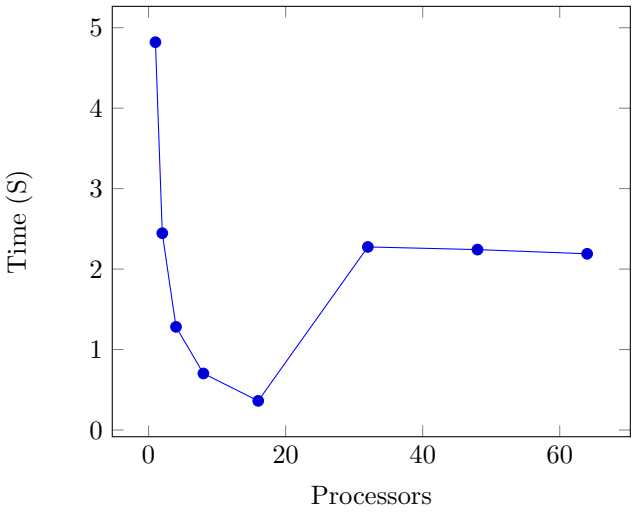

Figure 2: Slowdown

As Figure 2 shows when more that around 16 processors were used the amount of time required to complete the computation increased.

3.4 Gustafson's Law

Gustafson's Law states that if you increase the size of the problem then the amount of time spent on any sequential parts will be a less significant chunk of overall runtime. This means better speedup values can be obtained. To show this the speedup values were calculated for differing sizes of matrices to show as the size increase so does speedup.

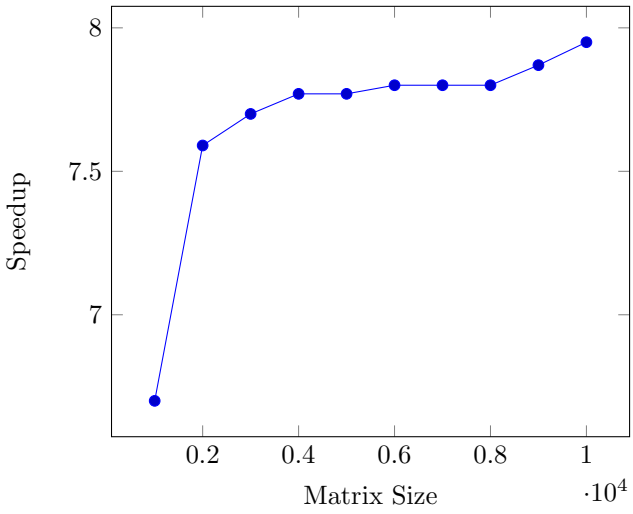

Figure 3: Gustafson's Law 8 Processors

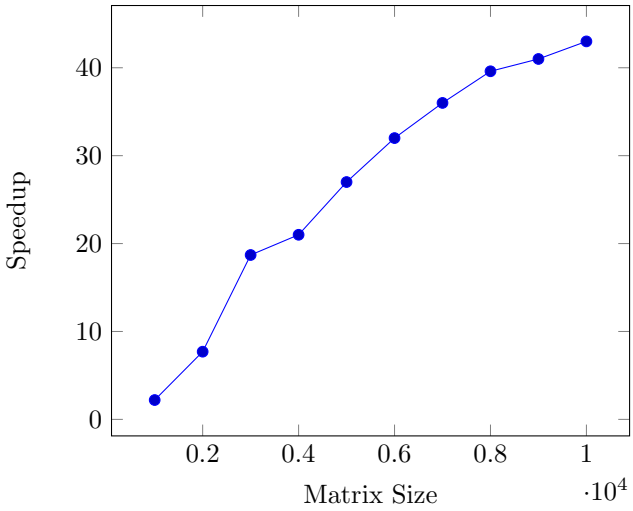

Figure 4: Gustafson's Law 64 Processors

As you can see from Figure 3 and 4 as Gustafson's Law suggests as the size of the problem is increased the speed up on P processors also increases, until the speedup reaches the number of processors used.

3.5 Efficiency

The efficiency was calculated for a number of processors and problem sizes, the results are shown in Figure 5.

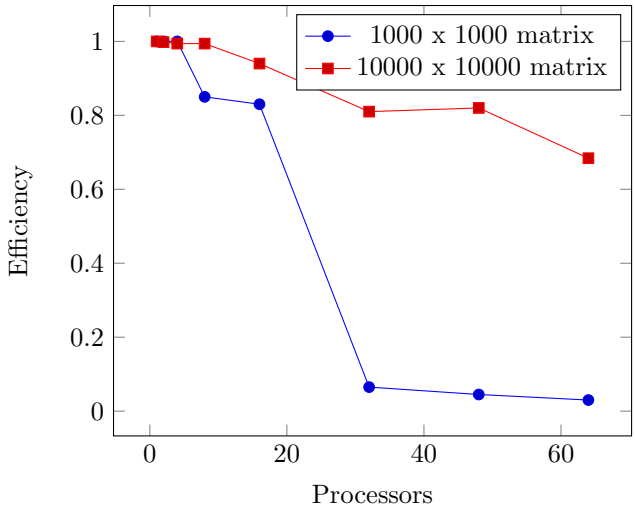

Figure 5: Efficiency

As expected from the results of the speedup investigation earlier the efficiency drops as more processors are added and the problem size is kept constant. This is because as some of the program has to be done in serial for that part every other processor is idle, therefore if more processors are added the cumulative time a processor is ideal increases.

3.6 Karp-Flatt

The Karp-Flatt metric is a measure for the sequential part of the program, the larger the Karp-Flatt metric the larger the sequential part of the problem. The metric was calculated for three different matrix sizes on different numbers of processors, it is expected that as more processors are added the Karp-Flatt will increase due to the longer time spent on communication. The results are shown in Figure 6.

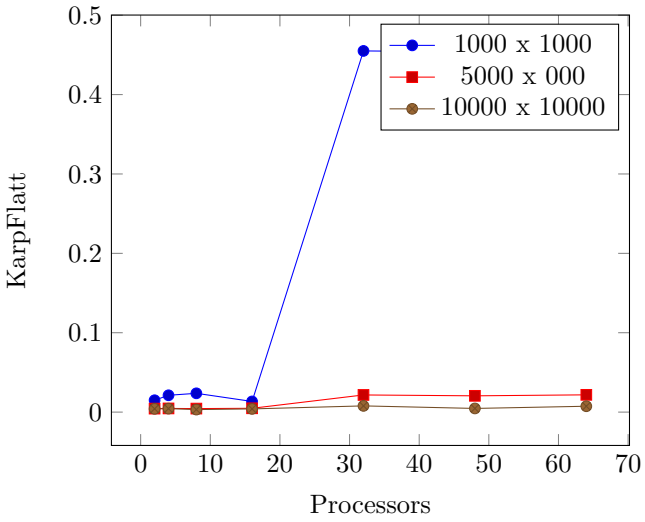

Figure 6: 0.01 precision

As the graph shows what we expected is true, as the number of processors is increased the Karp-Flatt metric also increases. The increase on smaller problem sizes is much larger than those on larger problems. This is because the proportion of the time spent doing sequential tasks is much larger on smaller problem sizes.

3.7 Parallel Overhead

The overhead was calculated to evaluate the work efficiency of the parallel program. The equation below was used to calculate the overhead.

$$Overhead \ = \ p \ \times \ time \ on \ P \ processors \ - \ time \ on \ 1 \ processor$$

This gives the total parallel overhead for a given number of processors in seconds. This total efficiency was divided by the number of processors to give the overhead per thread in seconds. The results from this are shown in Figure 7.

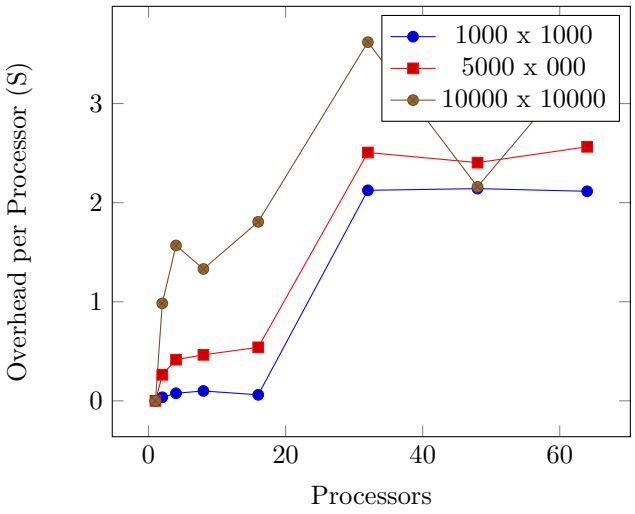

Figure 7: 0.01 precision

As shown in figure 7 as more processors are added the parallel overhead per thread increases. This is expected as when more processors are added there is more communication needed. Also as the problem size is increased the overhead also increases this is due to the fact there is more data to transfer between processors thus communication times increased.

3.8 Isoefficiency

The isoefficiency was calculated in order to work out how well the program scales. The isoefficiency tells you how much the problem size has to increased by in order to keep the efficiency constant. In order to work out the isoefficiency an efficiency of 0.8 was chosen and the number of cores needed to reach this efficiency for different problem sizes was found. These results are shown in the table below.

Problem size	0.8 efficiency
1000	8
3000	16
5000	21
7000	26
9000	30

The results in the table are shown in graphical form in Figure 8.

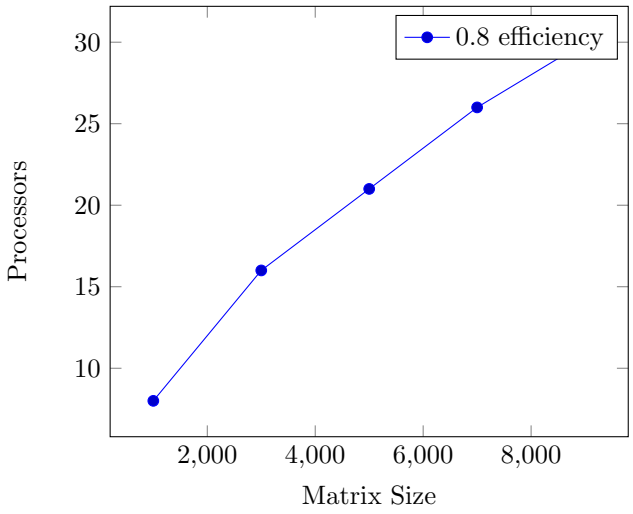

Figure 8:

A line of best fit can be drawn through those points with an equation of $y = 0.0027x + 6.7$. This shows that to maintain a given efficiency the matrix size must be increased by approximately 350 in each direction when another processor is added.

3.9 Conclusion

In conclusion the scalability is very good for the implementation given. The main reason for this is that the isoefficiency has a linear relationship and that if a new processor is added the matrix size only needs to be increased by 350 in both directions to maintain efficiency.