Exercise 1\_2 (Experiments on Poisson solver)

Q. 2.2 The figure below shows the experimental results for five different values of omega and the corresponding number of iterations. Results indicate that an omega of 1.93 gives the least number of iterations, 131 in this case.

|  |  |
| --- | --- |
| **Omega** | **iterations** |
| 1.90 | 220 |
| 1.93 | 131 |
| 1.95 | 166 |
| 1.97 | 281 |
| 1.99 | 830 |

Q2.3 The three tables below indicate the execution time (seconds) for different number of iterations, configurations, and grid sizes. (Comma is used to indicate the decimal point). A large number of iterations we’re chosen, i.e., starting from 1000 onwards, because the execution time is not modelled well by a linear model otherwise.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **config (200x200 gridsize)** | **1000** | **2000** | **3000** | **4000** | **5000** | **6000** | **7000** | **8000** | **9000** | **10000** |
| 4x1 | 0,32 | 0,62 | 0,85 | 1,05 | 1,28 | 1,48 | 1,70 | 1,93 | 2,13 | 2,34 |
| 2x2 | 0,37 | 0,68 | 0,97 | 1,20 | 1,45 | 1,74 | 1,99 | 2,24 | 2,49 | 2,75 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **config (400x400 size)** | **1000** | **2000** | **3000** | **4000** | **5000** | **6000** | **7000** | **8000** | **9000** | **10000** |
| 4x1 | 1,04 | 1,87 | 2,68 | 3,50 | 4,37 | 5,13 | 5,99 | 6,79 | 7,60 | 8,43 |
| 2x2 | 1,15 | 2,13 | 3,17 | 4,10 | 5,05 | 5,99 | 6,96 | 7,95 | 8,89 | 9,89 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **config (800x800 size)** | **1000** | **2000** | **3000** | **4000** | **5000** | **6000** | **7000** | **8000** | **9000** | **10000** |
| 4x1 | 3,49 | 6,73 | 9,97 | 13,22 | 16,42 | 19,67 | 22,93 | 26,17 | 29,33 | 33,17 |
| 2x2 | 4,13 | 7,85 | 11,67 | 15,47 | 19,26 | 23,08 | 26,93 | 30,73 | 34,58 | 38,58 |

Plotting the table values gives us the following figures: Observe that in all three scenarios, the 2x2 execution time is greater than the 4x1 configuration.

The corresponding values of alpha and beta computed for the three sizes are shown in the table below for the different configurations. These were obtained using a least squares linear fit to the data. Observe that the value of alpha does not show a consistent increase or decrease with the grid size, but beta is clearly increasing with the size. Furthermore, the 2x2 configuration has a greater value of beta compared to the 4x1 scenario, however the value of alpha does not show a clear pattern.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **size** | **4x1 (alpha)** | **4x1 beta** | **2x2 (alpha)** | **2x2 beta** |
| 200x200 | 0,158871 | 0,000220 | 0,150738 | 0,000261 |
| 400x400 | 0,229500 | 0,000820 | 0,213645 | 0,000966 |
| 800x800 | 0,150790 | 0,003265 | 0,206850 | 0,003822 |

Ex2.4

For this question, we can use a result from the previous question to limit the number of experiments. We observed that the execution times were increasing with the grid size, so we can limit ourselves to comparing the different configurations for just one particular size, say 200x200.

Furthermore, a 1x16 configuration is symmetric to a 16x1 configuration, so it is not necessary to experiment for each of them separately. Similar assumptions hold for 8x2 and 2x8.

Therefore, using a similar setup to Q2.3, we get the following values of alpha and beta for different configurations:

|  |  |  |  |
| --- | --- | --- | --- |
| size = 200x200 | 16x1 | 8x2 | 4x4 |
| alpha | 0,069573 | 0,076220 | 0,086300 |
| beta | 0,00010039 | 0,000104 | 0,000106 |

For a large number of iterations n, the value of beta plays a higher role in computing the execution time, because the time is estimated using the formula:

Since 16x1 has the smallest value of beta, it is also expected to have the lowest execution time. The figure below indicates this is indeed the case. So, choosing 16x1 or 1x16 configuration is the best choice.

Note the similarity with the result of the previous question. Even there, the square configuration of 2x2 was performing worse than the 4x1 configuration. This may be because a more “square” configuration has to communicate with more neighbours. In the present case, a processor has to communicate with 2, 3, and 4 neighbours in the worst case for the 16x1, 8x2, and 4x4 configurations respectively.

Ex2.5

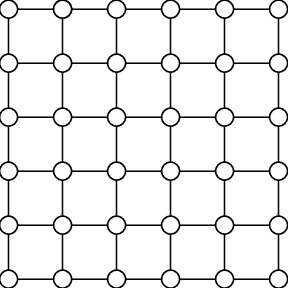
The table below shows the number of iterations for convergence for the 2x2 configuration and 4x1 configuration of processors using omega of 1.95

|  |  |  |
| --- | --- | --- |
| config | size | number of iterations |
| 2x2 | 200x200 | 397,00 |
| 2x2 | 300x300 | 784,00 |
| 2x2 | 400x400 | 1219,00 |
| 2x2 | 500x500 | 1678,00 |
| 4x1 | 200x200 | 397,00 |
| 4x1 | 300x300 | 785,00 |
| 4x1 | 400x400 | 1220,00 |
| 4x1 | 500x500 | 1679,00 |

For both 4x1 and the 2x2 configuration, there is a clear positive correlation between the grid size and the number of iterations for convergence. For understanding why this is the case, let us consider the number of steps it takes for an update to propagate across the grid when information is updated at a point. Refer to the figures shown below.

Step 1:

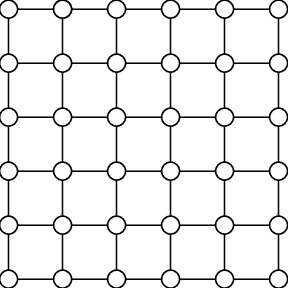
Update made to point marked in red





Step 2:

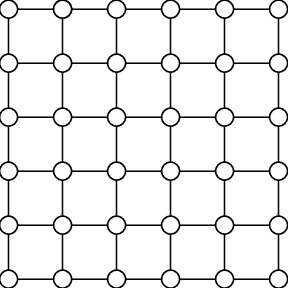
Propagation of update to neighbours (yellow)

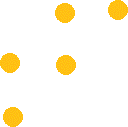




Step 3:

Step 3: Further propagation





So on. It is therefore clear that the time to propagate the update to all the grid points will vary linearly with the grid size.

Ex 2.6

The figures below show the absolute error with the number of iterations for the 2x2 and the 4x1 configuration with grid size 500x500 and omega = 1.95

Configuration = 2x2

|  |  |  |
| --- | --- | --- |
| iteration number | error (absolute difference) | log error |
| 200 | 1,81E-03 | -6,31E+00 |
| 300 | 1,51E-03 | -6,50E+00 |
| 400 | 1,28E-03 | -6,66E+00 |
| 500 | 1,06E-03 | -6,85E+00 |
| 600 | 8,64E-04 | -7,05E+00 |
| 700 | 7,03E-04 | -7,26E+00 |
| 800 | 5,74E-04 | -7,46E+00 |
| 900 | 4,70E-04 | -7,66E+00 |
| 1000 | 3,85E-04 | -7,86E+00 |
| 1100 | 3,15E-04 | -8,06E+00 |
| 1200 | 2,58E-04 | -8,26E+00 |
| 1300 | 2,12E-04 | -8,46E+00 |
| 1400 | 1,73E-04 | -8,66E+00 |
| 1500 | 1,42E-04 | -8,86E+00 |
| 1600 | 1,16E-04 | -9,06E+00 |

Configuration 4x1

|  |  |  |
| --- | --- | --- |
| iteration | error | log error |
| 200 | 1,70E-03 | -6,378304191 |
| 300 | 1,25E-03 | -6,688619749 |
| 400 | 1,04E-03 | -6,872388135 |
| 500 | 8,54E-04 | -7,065579364 |
| 600 | 6,96E-04 | -7,270160898 |
| 700 | 5,65E-04 | -7,478684827 |
| 800 | 4,58E-04 | -7,688641374 |
| 900 | 3,71E-04 | -7,899308495 |
| 1000 | 3,01E-04 | -8,108400293 |
| 1100 | 2,45E-04 | -8,314252347 |
| 1200 | 1,99E-04 | -8,522205733 |
| 1300 | 1,62E-04 | -8,727914223 |
| 1400 | 1,32E-04 | -8,932708635 |
| 1500 | 1,08E-04 | -9,133379331 |
| 1600 | 8,80E-05 | -9,338173743 |

Both figures indicate that the logarithm of the error varies linearly with the number of iterations. We thus conclude that the error must exponentially decay with the iterations.

Ex 2.8

The table below shows the number of iterations for convergence and the time for different number of sweeps. The chosen configuration was 4x4 with an omega of 1.95 and a grid size of 200x200

|  |  |  |
| --- | --- | --- |
| sweeps | number of iterations | time (seconds) |
| 1 | 233 | 0,087 |
| 2 | 119 | 0,090 |
| 3 | 105 | 0,092 |
| 4 | 100 | 0,099 |
| 5 | 97 | 0,110 |
| 6 | 100 | 0,122 |
| 7 | 97 | 0,120 |
| 8 | 96 | 0,135 |
| 9 | 95 | 0,149 |
| 10 | 94 | 0,150 |
| 11 | 94 | 0,162 |
| 12 | 92 | 0,159 |
| 13 | 95 | 0,190 |
| 14 | 91 | 0,180 |
| 15 | 90 | 0,202 |
| 16 | 90 | 0,194 |
| 17 | 90 | 0,195 |
| 18 | 88 | 0,211 |
| 19 | 87 | 0,231 |
| 20 | 87 | 0,224 |

The figure below shows that initially the number of iterations required for convergence drops rapidly and then it slows down. This may be because increasing the number of sweeps between communications leads to re-computation using stale data and does not propagate the updated values across the grid. However, we observe that the time for convergence increases as we increase the number of sweeps. This is a result of increasing number of computations without much improvement in terms of convergence.

Ex 2.9

The problem with the naïve if implementation in the Do\_step is that the parity condition is checked for all x and y after entering the for-loop. Hence this can be improved by incrementing the index of y by 2 instead of 1. Furthermore, for every alternate value of x, i.e., the row, the starting y index for that row must be shifted by 1 or zero depending on the parity of the first element.

The table below shows the execution time by avoiding the check and after the modification to the condition for a fixed number of iterations (5000)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| config (omega = 1.95) | size | iterations | avoiding check | with parity check |
| 2x2 | 200x200 | 5000,00 | 1,04 | 1,110 |
| 2x2 | 400x400 | 5000,00 | 3,03 | 3,680 |
| 2x2 | 800x800 | 5000,00 | 11,10 | 13,660 |
| 4x1 | 200x200 | 5000,00 | 0,96 | 1,090 |
| 4x1 | 400x400 | 5000,00 | 2,99 | 3,650 |
| 4x1 | 800x800 | 5000,00 | 10,99 | 13,460 |

As expected, the check-avoiding version has a lower execution time. The performance improvement becomes more significant as the grid size increases: this is also expected because the time spent in the loop increases with the size.

Ex 2.11

The time spent in communication is the sum of overhead time per-transfer and the actual transfer time. For uniformity, the 4x1 configuration has been chosen for differing grid sizes and 10 trials (by maintaining a counter variable). In exchange\_borders (), the only thing required to be measured is the elapsed time for the x-y direction along with the size of the data transferred. This is done by timing the send-recv for either the left-to-right or right-to-left communication since both are equal by symmetry arguments.

|  |  |  |  |
| --- | --- | --- | --- |
| **config = 4x1** |  |  |  |
| grid size | no. transfers | total time (seconds) | size (bytes) |
| 200x200 | 10000 | 0,048 | 1600 |
| 400x400 | 10000 | 0,056 | 3200 |
| 600x600 | 10000 | 0,062 | 4800 |
| 800x800 | 10000 | 0,072 | 6400 |

The linear model can be estimated to be:

A linear fit gives us the following values:

Overhead = 4 microseconds

Bandwidth = 2.051 giga bytes per second

Ex 2.12

A modified function was implemented for reducing the number of points to communicate. (See the function Exchange\_Borders\_Half\_Data for the code). The code keeps track of the size of data to transfer and receive for each of the four directions, taking into account the parity of the first element to be transferred. If the first element has a parity different from that to be sent, the indices to be transferred are shifted ahead by 1. Checks have also been made to ensure that the sum of the number of points received along a border and the points sent sum up to the row/column dimension. This function takes the parity as input to know what kind of points are to be transferred (red/black).

To test the performance, 4 processors were tested in the 4x1 and 2x2 configurations and the total execution time was noted. Omega was chosen as 1.95. The earlier communication version and the modification were tested for 10000 iterations for different grid sizes. On comparing the two results, we see that the reduction in communication has resulted in reducing the execution time by almost up to 43 percent in the best-case scenario. This is quite a significant improvement, so it is definitely worth the additional programming complexity

|  |  |  |  |
| --- | --- | --- | --- |
| **config (200x200 size)** | **transferring red and black** | **transferring only required** | **percent reduction** |
| 4x1 | 2,34 | 1,69 | 27,78 |
| 2x2 | 2,75 | 1,77 | 35,64 |

|  |  |  |  |
| --- | --- | --- | --- |
| **config (400x400 size)** | **transferring red and black** | **transferring only required** | **percent reduction** |
| 4x1 | 8,43 | 5,78 | 31,44 |
| 2x2 | 9,89 | 5,92 | 40,14 |

|  |  |  |  |
| --- | --- | --- | --- |
| **config (800x800 size)** | **transferring red and black** | **transferring only required** | **percent reduction** |
| 4x1 | 33,17 | 21,81 | 34,24 |
| 2x2 | 38,58 | 22,16 | 42,56 |