Exercise 2

Ex 4.2

The table below shows the time split (seconds) between computations, communications with neighbours, global communication, and the idle time for a 1x4 and a 2x2 configuration. The number of iterations was kept fixed at 5000 for different grid sizes of 100, 200, and 400

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| config | size | Tot. time | compute | neighbour | global comm | idle |
| 1x4 | 100x100 | 1,268 | 1,100 | 0,038 | 0,095 | 0,036 |
| 1x4 | 200x200 | 3,362 | 3,123 | 0,046 | 0,120 | 0,073 |
| 1x4 | 400x400 | 12,770 | 12,251 | 0,064 | 0,278 | 0,177 |
| 2x2 | 100x100 | 1,285 | 1,080 | 0,067 | 0,101 | 0,037 |
| 2x2 | 200x200 | 3,380 | 3,096 | 0,082 | 0,135 | 0,067 |
| 2x2 | 400x400 | 12,830 | 12,220 | 0,124 | 0,296 | 0,190 |

Ex 4.3

When an grid is partitioned among processors, each processor gets 2/*P* points. The figure below shows the connections between the points along the border between processors. Though this figure shows only 2 points along the lattice it is understood that there are in general *N/sqrt(P)* points along the border.

Let us consider the number of points each of these processors would send to its neighbours (approximately, i.e., not counting the corner cases for points along the edges of a processor):

Processor 1: 2*N/sqrt(P)* points are sent to 2 and 4

Processor 2: 2*N/sqrt(P)* points sent to processors 1,3 and 5. One point sent to processor 4.

Processor 3: 2*N/sqrt(P)* points sent to processors 2 and 6. One sent to processor 5.

Processor 4: 2*N/sqrt(P)* points sent to 1, 5, and 7.

Processor 5: 2*N/sqrt(P)* points sent to 2, 4, 6, and 8. One point sent to 3 and 7 each.

Processor 6: 2*N/sqrt(P)* points sent to 3, 5, and 9. One sent to 8.

Processor 7: 2*N/sqrt(P)* points sent to 4 and 8. One sent to 5.

Processor 8: 2*N/sqrt(P)* points sent to 5, 7, and 9. One sent to 6.

Processor 9: 2*N/sqrt(P)* points sent to 6 and 8.

In general, if there are *P* processors, then there will be 4 in the corners, *sqrt(P)-2* along each of the 4 edges, and *P – 4\*(sqrt(P)-2) – 4* on the inside of the lattice.

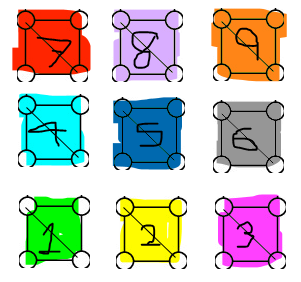
We can make a rough approximation on the total number of points communicated.

For each processor at the corner along the secondary diagonal (similar to 1 and 9), the total data sent is 4*N/sqrt(P)*

For corner processors along the primary diagonal (like 3 and 7), the total data sent is 4*N/sqrt(P) + 1.*

For points along the edges (like 2,4,6,8), the total data sent is *6N/sqrt(P) + 1*

For the remaining internal points, the data sent by each is *8N/sqrt(P) + 2*



Ex 4.4

The asymmetry exists because apart from the directly adjacent processors, a processor can only communicate with the ones that are located to the upper left corner, or the bottom right corner. The above figure illustrates this as well and shows how points are connected along the “primary” diagonal, i.e., 2 communicates with the bottom right point of 4, and 8 communicates with the upper left point of 6.

As a result, the central processor (i.e., number 5), communicates with 6 processors: 4 along the east, west, north, and south direction. The other two are along the northwest, and southeast direction.

For a corner point the answer depends on the corner. If it is located at the bottom left or the upper right, it communicates with 2 processors. Whereas if it is located at the upper left or the bottom right, it communicates with 3 processors.

Ex 4.5

1. To estimate the compute-to-communication ratio, multiple timers were started to measure the computation time, global communication, and neighbour-to-neighbour communication times. The table below shows the computation, total communication time (all in seconds), and the ratio per-processor for a 1x4 configuration with different grid sizes.

|  |  |  |  |
| --- | --- | --- | --- |
| size (nxn) | compute | communication | compute/communication |
| 100 | 1,10 | 0,13 | 8,54 |
| 50 | 0,12 | 0,06 | 1,93 |
| 25 | 0,02 | 0,04 | 0,44 |
| 38 | 0,05 | 0,03 | 1,04 |

The closest grid size approximation to make the compute-to-communication ratio one was found to be 38x38. This was determined empirically and the ratio came out to be 1.04

1. A similar procedure was applied to determine the number of processors required to make the ratio unity for a 1000x1000 grid size. All processors are assumed to be arranged on a

1 x K configuration where K is the number of processors.

|  |  |  |  |
| --- | --- | --- | --- |
| processor count | compute time | communication time | compute-to-communication ratio |
| 4 | 88,409 | 1,215865 | 72,71284 |
| 8 | 44,04538 | 1,209704 | 36,41005 |
| 16 | 20,427 | 1,465483 | 13,93875 |
| 32 | 9,766847 | 1,40793 | 6,937026 |

Due to resource limitations, it was not possible to spawn more than 32 processors. However, the above data was used to extrapolate the required number of processors. On taking the logarithm with base 2 for both the number of processors and the compute-to-communication ratio, we get the following plot.

On fitting a linear approximation and extrapolating this, we get the required number of processors is approximately 167.

Ex 4.6: To see the effect of using adapt, it is better to use an odd number of processors in the configuration because then the 100x100 points will not be evenly divided across each processor. Same holds for the 200x200 and the 400x400 grid sizes. The selected configuration was thus chosen to be 3x3

The table below shows the experimental results for the iterations, total time and the compute time (in seconds) when using adapt.

|  |  |  |  |
| --- | --- | --- | --- |
| With adapt | |  |  |
| size | iterations | tot time | compute time |
| 100x100 | 146 | 0,070 | 0,014 |
| 200x200 | 278 | 0,182 | 0,100 |
| 400x400 | 532 | 0,922 | 0,713 |

The above results do not differ by much when compared with the “without adapt” scenario shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| without adapt | |  |  |
| size | iterations | tot time | compute time |
| 100x100 | 141 | 0,069 | 0,013 |
| 200x200 | 274 | 0,192 | 0,098 |
| 400x400 | 529 | 0,919 | 0,709 |

Observe however that the number of iterations for convergence when using adapt is slightly larger than without adapt for all three cases. The total time and compute time are almost the same in both cases.