**Parallel Programming Exercise 6 – 13**

|  |  |
| --- | --- |
| **Author:** | 蕭彧 ([r11922138@ntu.edu.tw](mailto:r11922138@ntu.edu.tw)) |
| **Student ID** | R11922138 |
| **Department** | Computer Science |

# Problem and Proposed Approach

This programming exercise is to write the mpi version of John Conway’s game of life. The rules in textbook is a little ambiguous. The clearer rules are as follows:

Initially, there is a grid with some cells which may be alive or dead. The next generation of cells are based on the following rules:

1. Any live cell with fewer than two live neighbors dies as if caused by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Since each iteration needs previous iteration’s output, the functional partitional is weak in this case. I choose domain partition to spread tasks between processors. I use rowwise block striped to decompose the big state matrix to small sub matrices. Each processor calculate its own state matrix.

# Theoretical Analysis Model

mpfr\_d is time needed to do mpfr divide

mpft\_a is time needed to do mpfr add

λ is the message latency

β is network bandwidth

Sequential execution time: (n - 1)(mpft\_a + mpfr\_d)

Reduction time: (*λ* + d/β) ⎡log *p*⎤

Expected execution time: ⎡(n - 1)(mpft\_a + mpfr\_d)/p⎤ + (*λ* + d/β) ⎡log *p*⎤ + (mpft\_a) ⎡log *p*⎤

# Performance Benchmark

I pick process 0’s execution time, since it need to do the gather. The message latency is λ = 0.001521 sec, and the network bandwidth is β = 1658 MB/sec

Table . The execution time

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Processors | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| Real execution time | 2.05646 | 1.58445 | 1.29312 | 1.17022 | 1.0924 | 1.54012 | 0.961593 | 0.970369 |
| Estimate execution time | 1.995 | 1.45152 | 1.2485 | 1.0851 | 1.0651 | 1.0054 | 0.9245 | 0.9351 |
| Speedup | 1 | 1.298 | 1.59 | 1.76 | 1.88 | 1.34 | 2.14 | 2.12 |
| Karp-flatt metrics | - | 0.541 | 0.505 | 0.506 | 0.5 | 0.74 | 0.45 | 0.47 |

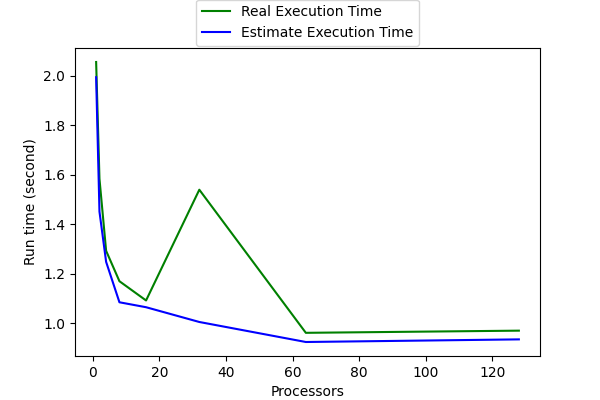


Figure . The performance of diagram

# Conclusion and Discussion

The concept of this problem is quite same as the previous pi calculation, but harmonic progression here. Each processor calculates the partial sum and then do a gather to get final output. The obstacle this time is how to cope with the arbitrary precision. I spend most of the time learning the api of mpfr and how to transfer mpfr instance between processors.

In the performance figure, we can see we did have good parallelism compared to previous problem. This is mainly because the problem size is bigger this time, so we can see nearly two-fold speedup when we double the processor amounts. Network latency doesn’t play an important role as before.