Parallel Programming Exercise　6 - 13

Author:	邱亮茗 (r12942159@ntu.edu.tw)
Student ID	R12942159
Department	Graduate Institute of Communication Engineering

(If you and your team member contribute equally, you can use (co-first author), after each name.)

1 Problem and Proposed Approach

(Brief your problem, and give your idea or concept of how yo
u design your program.)

The problem involves implementing a parallel program to simulate Conway's Game of Life, a cellular automaton on a grid where each cell is either "alive" or "dead." The state of each cell evolves across iterations based on its neighbors' states, with the rules:

1. A **dead cell** with exactly 3 live neighbors becomes alive.
2. A **live cell** with 2 or 3 live neighbors remains alive.
3. A **live cell** with fewer than 2 or more than 3 live neighbors dies.

To design the parallel program, we can break it into the following steps:

1. **Input Parsing**:
    - Read the m×n matrix from a file to initialize the grid.
    - Parse j and k from command-line arguments.
2. **Parallelism Strategy**:
    - Divide the grid into smaller chunks, and assign each chunk to a parallel process (or thread).
    - Each process will compute updates for its assigned cells independently based on the current grid state.
    - Use a barrier or synchronization mechanism to ensure all cells update simultaneously after each iteration.
3. **Algorithm**:
    - For j iterations:
        1. Compute the next state for all cells in parallel.
        2. Synchronize processes after computation to update the grid state.
        3. Print the grid state every k iterations.

    - Rules are applied locally, so each process only needs to know its assigned cells and their immediate neighbors.
4. **Communication**:
    - Exchange border rows or columns between adjacent processes to ensure correct neighbor computations for cells at the boundaries.
5. **Output**:
    - Format the output grid for readability, displaying the grid at specified iterations.

2 Theoretical Analysis Model

(Try to give the time complexity of the algorithm, and analyze your program with iso-efficiency metrics)

Time complexity of the algorithm

1. Initialization and Distribution:

- Each process calculates its portion of the matrix. This involves:
    - Reading the rows and columns: O(1) per process.
    - Distributing rows: Requires communication overhead proportional to the matrix size and the number of processes.

2. Simulation of Game of Life Rules:

- Per Cell Update:
    - Each cell checks its eight neighbors, resulting in a constant-time operation: **O(1)**.
- Per Submatrix Update:
    - Each process updates n*n*m/p cells (rows divided among p processes).
    - Time complexity per iteration per process is **O(n*n*m/p)**.

3. Communication Overhead:

- Each process communicates its boundary rows to its neighbors:
    - 2 rows per process, with O(m) data for each row.
    - Communication cost: **O(m)** per process per iteration.
- Aggregating results for periodic display introduces additional communication overhead, **O(n*m*logp)**.

Combining computations and communication:

- **T(p)=O(n*n*m/p)+O(n*m*logp).**

Iso-efficiency Analysis

To(n, p) = O(n*n*m/p)+O(n×m*logp) = p*n*m*logp

Isoefficiency relation: n*n*m >= C(p*n*m*logp) => n >= Cplogp

M(n) = n*n

⇨ **M(f(p)) / p =** $M(Cp\log p)/p = C^2p^2\log^2 p/p = C^2p\log^2 p$

⇨ **The parallel system has poor scalability**

3 Performance Benchmark

(Give your idea or concept of how you design your program.)

Table 1. The execution time

Processors	1	2	3	4	5	6	7	8
Real execution time	2.441730	1.254572	0.851103	0.652997	0.528011	0.465550	0.40991	0.393317
Estimate execution time	2.441730	1.220865	0.813910	0.610433	0.488346	0.406955	0.348819	0.305216
Speedup	1	1.9462	2.8689	3.7393	4.6244	5.2448	5.9567	6.2080
Karp-flatt metrics		0.0276	0.0228	0.0232	0.0203	0.0287	0.0291	0.0412

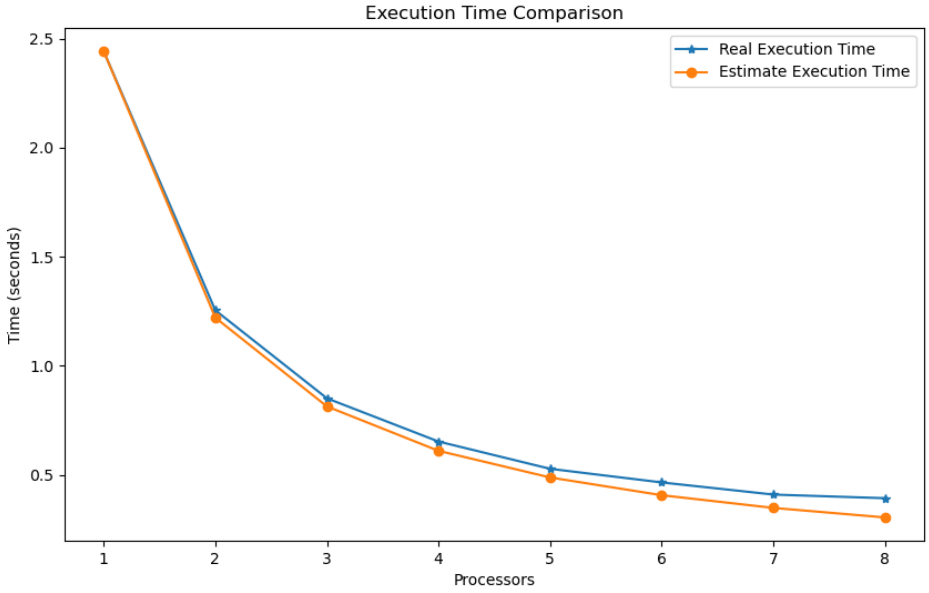

Figure 1. The performance of diagram

4 Conclusion and Discussion

(Discuss the following issues of your program

1. What is the speedup respect to the number of processors used?

   The speedup increases as the number of processors increases but with diminishing returns, indicating that overhead or inefficiencies grow as more processors are added.

2. How can you improve your program further more?

   Improve Cache Usage: Fit computations into the L1/L2 cache where possible.

   Algorithm Optimization: Explore parallel algorithms with better scalability.

3. How does the communication and cache affect the performance of your program?

   Communication Overhead: Increased communication reduces scalability and speedup. As the number of processors increases, the cost of synchronizing and transferring data grows.

   Cache Effect: Cache misses significantly slow down execution because data has to be fetched from slower memory. And if the problem size does not fit in the processor's cache, performance drops due to frequent memory accesses.

4. How does the Karp-Flatt metrics and Iso-efficiency metrics reveal?

   Karp-Flatt Metrics: In this, ε is relatively low for smaller processors, showing efficient parallelism initially. However, it increases with more processors, highlighting diminishing returns.

   Iso-efficiency Metrics: For this program, the increasing gap between estimated and real execution times indicates growing inefficiencies due to communication and overhead. To maintain efficiency: (1) Increase problem size proportionally with the number of processors. (2) Reduce communication overhead as processors scale up.

)

Appendix(optional):

(If something else you want to append in this file, like picture of life game)

程式碼是參照姜任懋同學完成的。