INF 236 - Parallel Computing - Problem Set 1

Mateus de Oliveira Oliveira
Department of Informatics
University of Bergen
mateus.oliveira@uib.no

February 8, 2019

Updates

31/01:
- You may assume that the number of processes devides the size of the configuration. Specify your choice in the report.
- You may assume that the number of processes is even. Specify your choice in the report.
- Change configuartion size.

08/02:
- Add precisions for Branching Programs input.
- Correct the size of the input for the 2D Cellular Automata.
- Correct command for lyng.
- Add precisions for the time measure for Branching Programs

Instructions

1. The deadline for this Problem set is 17 February 2019 - 23:59 CET.
2. The exercises have a programming part, and a report intended to provide an intuitive explanation of the techniques you used and the experiments you have performed. The reports of all three exercises should be submitted as a single PDF file called

   `reportProblemSet1-XXXNNN.pdf`

   where XXXNNN is your UiB identifier.
3. Create a folder named ProblemSet1. In this folder create subfolders 1-Sequential, 1-Parallel, 2-Sequential, 2-Parallel, 3-Sequential,3-Parallel. The code of the sequential program and of the parallel program of each assignment should be located at its respective sub-folder.
4. The code of all programs should be readily compilable at the server lyng.ii.uib.no.
5. You are allowed, and encouraged to collaborate with colleagues, in groups of up to four people. Nevertheless, the reports and all programming exercises should be written/implemented individually. You should name the people you have collaborated with in the beginning of the report.

6. Please add the MIT license text `https://opensource.org/licenses/MIT` in the main file of each program you implement.

7. Submission: upload your program to a public repository in github/bitbucket. Send an email to emmanuel.arrighi@uib.no with copy to mateus.oliveira@uib.no containing your attached report, and a link to your publicly available source codes. You do not need to upload the report to your repository. Only the source codes, and experimental results, are required. Please do not send us a compressed file with your code/experiments.

8. Please do not share your source codes before the submission deadline.

1 1-Dimensional Cellular Automata

A 1-dimensional a cellular automaton of length-n is a pair $A = (f, C_0)$ where $f : \{0,1\}^3 \to \{0,1\}$ is a function, called the *transformation function of* A, and $C_0 \in \{0,1\}^n$ is a length-n Boolean string called the *initial configuration* of A. The configuration of A at a given time $t \geq 1$ is the length-n Boolean string $C_t \in \{0,1\}^n$, where for each $i \in \{0, ..., n-1\}$, the symbol $C_t[i]$ at position i is defined as follows.

$$C_t[i] = f(C_{t-1}[i-1 \mod n], C_{t-1}[i], C_{t-1}[i+1 \mod n]) \quad (1)$$

We use the following convention: $-1 \mod n = n-1$. Note that in class we used a different boundary condition. For simplicity, in this assignment we use the periodic boundary condition, according to which position $n-1$ is considered to be adjacent to position 0. The goal of this exercise is to implement a parallel algorithm that takes a length-n cellular automaton $A = (f, C_0)$ and a number $t \geq 1$ as inputs and outputs the string C_t. The input cellular automaton should be specified in two separate text files, and the number of iterations t should be specified directly in the standard input. The first text file specifies the transformation function of the cellular automaton, and has the following format.

```
000 A
001 B
010 C
011 D
100 E
101 F
110 G
111 H
```

Here, A,B,C,D,E,F,G,H are Boolean values, where the bit A specifies the value $f(000)$, the bit B specifies the value $f(001)$, and so on. The second text file specifies the initial configuration of the cellular automaton. This file contains one line with the length n of the initial configuration, followed by one line containing a Boolean string of length n.

```
n
BOOLEAN_STRING
```

Your program should be named 'Cellular1D-Parallel' and it should be called from the command line as follows.

```
Cellular1D-Parallel  file1  file2  t
```

The program should compute the string corresponding to configuration C_t, print it at the standard output, and terminate.

You can assume that the configuration can be evenly splited between processes. You can also do the questions without assuming that the number of processes divides the size of the configuration. For this, you can use the functions MPI_Scatterv and MPI_Gatherv. But since this solution is slightly more complicated, this is completely optional. Specify your choice in the report.

1. First implement a program called Cellular1D-Sequential which implements a straightforward sequential algorithm for computing configuration C_t when given the files file1, file2 and the number t as input.

2. Create a file middle64.txt containing the configuration $0^{32}10^{31}$. In other words, 32 zeros followed by a one and again by 31 zeros.

3. Create a file mod2.txt containing the transformation specified by the following rows.

```
000 0
001 1
010 0
011 1
100 1
101 0
110 1
111 0
```

4. For visualization purposes, create a Boolean matrix with 100 rows where the first row is the configuration $0^{32}10^{31}$ and the t-th row is configuration C_t. Plot this matrix as a black and white image, where 0 is white and 1 is black (using any program of your choice for this).

5. Parallelize your program Cellular1D-Sequential using MPI (Or simply implement a parallel version from scratch). The program should be general enough to work with any number of processes but may assume that the number of processes devides the size of the configuration. In other words, the program should retrieve the number of processes and adjust its execution according to this number. Name the program Cellular1D-Parallel.

6. Let $0^{2^k}10^{2^k-1}$ be the configuration formed by 2^k zeros followed by a one and by $2^k - 1$ zeros. Let p be the number of processes. Execute your parallel program with $k \in \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$ and $p \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$. To start more processes than what the machine have you may need to use the option '-N n' with 'n' the nomber of processes by core. For example 'mpiexec -n 12 -N 2' will start 24 processes on 12 cores of the machine. You don't need this on lyng. For each such pair (m, p), measure the time $T(m, p)$ to compute configuration $C_{1000000}$, i.e the configuration at step $t = 1000000$. Organize the computed values $T(m, p)$ in a table with rows corresponding to values of m and columns corresponding to values of p.

7. Write a brief report explaining the technique you used to parallelize your program, containing the results of your experiments (Ex. Picture and Table), and some remarks on the results obtained.

2 2-Dimensional Cellular Automata

A 2-dimensional cellular automaton of length n is a pair $A = (f, C_0)$ where $f : \{0,1\}^9 \to \{0,1\}$ is a function called the transformation function of the automaton and $C_0 \in \{0,1\}^{n \times n}$ is a Boolean $n \times n$ square matrix called the *initial configuration* of A. For each $t \geq 1$, the configuration of A at time t is the $n \times n$ matrix inductively defined as follows.

$$C_t(i,j) = \begin{array}{lll} f(& C_{t-1}(i-1,j-1), & C_{t-1}(i-1,j), & C_{t-1}(i-1,j+1), \\ & C_{t-1}(i,j-1), & C_{t-1}(i,j), & C_{t-1}(i,j+1), \\ & C_{t-1}(i+1,j-1), & C_{t-1}(i+1,j), & C_{t-1}(i+1,j+1) \quad) \end{array} \tag{2}$$

where addition $+$ and $-$ on indices i, j are performed modulo n. In other words, $i \pm 1$ and $j \pm 1$ are shortcuts for $i \pm 1 \mod n$ and $j \pm 1 \mod n$.

The goal of this exercise is to implement a parallel algorithm that takes a length-n 2D cellular automaton $A = (f, C_0)$ and a number $t \geq 1$ as inputs and outputs the $n \times n$ matrix C_t. The input cellular automaton should be specified in two separate text files, and the number of iterations t should be specified directly in the standard input. The first text file specifies the transformation function of the cellular automaton, and has the following format.

```
000000000 A_0
000000001 A_1
000000010 A_2
...
111111111 A_{511}
```

Here, $A_0, ..., A_{511}$ are Boolean values, where for each $i \in \{0, ..., 511\}$, the bit A_i specifies the value $f(\text{binaryexpansion}(i))$. The second text file specifies the initial configuration of the cellular automaton. This file has one line containing the number n followed by n lines, each of which has a Boolean string of length n.

```
n
BOOLEAN_STRING_0
BOOLEAN_STRING_1
...
BOOLEAN_STRING_{n-1}
```

Your program should be named "Cellular2D-Parallel" and it should be called from the command line as follows.

You can assume that the configuration can be evenly splited between processes. You can also do the questions without assuming that the number of processes divides the size of the configuration. For this, you can use the functions MPI_Scatterv and MPI_Gatherv. But since this solution is slightly more complicated, this is completely optional. Specify your choice in the report.

```
Cellular2D-Parallel  file1  file2  t
```

The program should compute the matrix corresponding to configuration C_t, print it at the standard output, and terminate. Print each line of the matrix followed by a blank line at the standard output.

1. First implement a program called Cellular2D-Sequential which implements a straightforward sequential algorithm for computing configuration C_t when given the files file1, file2 and the number t as input.

2. Create a file gameOfLife.txt containing the transformation corresponding to Conway's game of life. This game is specified by the following rules:

   (a) Any live cell with fewer than two live neighbors dies, as if by underpopulation.

   (b) Any live cell with two or three live neighbors lives on to the next generation.

   (c) Any live cell with more than three live neighbors dies, as if by overpopulation.

   (d) Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

3. Parallelize your program Cellular2D-Sequential using MPI (Or simply implement a parallel version from scratch). The program should be general enough to work with any number of processes. In other words, the program should retrieve the number of processes and adjust its execution according to this number. Name the program Cellular2D-Parallel.

4. For each n in the set $\{1024, 2048, 4096, 8192, 16384, 32768, 65536\}$ construct a random $n \times n$ initial configuration. For each such a configuration and each $p \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$, measure the time $T(n, p)$ to compute configuration C_{10000}, i.e., the configuration at step $t = 10000$. To start more processes than what the machine have you may need to use the option '-N n' with 'n' the nomber of processes by core. For example 'mpiexec -n 12 -N 2' will start 24 processes on 12 cores of the machine. You don't need this on lyng.

   Organize the computed values $T(m, p)$ in a table with rows corresponding to values of m and columns corresponding to values of p.

5. Write a brief report explaining the technique you used to parallelize your program, containing the results of your experiments (Ex. Picture and Table), and some remarks on the results obtained.

3 Branching Programs

Let S be a finite set. An n-variable S-branching program of length m is a tuple $P = (S, \odot, F, X, T)$ where $\odot : S \times S \to S$ is an associative operation, $F \subseteq S$ is a subset of S called the final elements of P, $X = \{x_1, ..., x_n\}$ is a set of Boolean variables, and T is a sequence of triples

$$T = (i_1, I_0^1, I_1^1)(i_2, I_0^2, I_1^2)...(i_m, I_0^m, I_1^m),$$

where $i_j \in [n]$ and $I_0^j, I_1^j \in S$. Given a Boolean assignment $a : [n] \to \{0, 1\}$ of the variables $x_1, ..., x_n$, where $a[i]$ is an assignment of x_i, the value of P at a is defined as

$$\text{val}(P, a) = \begin{cases} 1 & \text{If } I_{a[i_1]}^1 \oplus I_{a[i_2]}^2 \oplus \cdots \oplus I_{a[i_m]}^m \in F \\ \\ 0 & \text{Otherwise.} \end{cases} \tag{3}$$

The goal of this exercise is to take a branching program P and an assignment a as input and to compute $\text{val}(P, a)$. The branching program is specified in a text file with the following format.

```
n
m
k
r
```

```
A_0 B_0 C_{0,0}
A_0 B_1 C_{0,1}
...
A_{k-1} B_{k-1} C_{k-1,k-1}
F_1
F_2
...
F_r
i_1 I^1_{0} I^1_{1}
i_2 I^2_{0} I^2_{1}
...
i_m I^m_{0} I^m_{1}
```

Here, n n is the number of variables, m the length of the branching program, k is size of the set S, and r is the size of the subset F. Elements of S will be integer, $S = \{0, 1, \ldots, k-1\}$ Since S has k elements, the operation $\oplus$ is defined on k^2 pairs. The value of each such pair (i, j) is specified by the number $C_{i,j}$. The last lines $F_1, ..., F_r$ are the final elements of the branching program.

1. Implement a sequential program named Branching-Sequential that takes as input a file *bp.txt* specifying a n-variable branching program P, and a string $a \in \{0,1\}^n$ specifying an assignment of the variables of the branching program, and returns the boolean value $\text{val}(P, a)$. The program should be called as follows.

   ```
   Branching-Sequential bp.txt assignment
   ```

2. Show that if there exists an unlimited number of processes, $\text{val}(P, a)$ can be computed in time $O(\log m)$.

3. Write a parallel version of the program **Branching Sequential**. Name this program Branching-Parallel. The program should be called as follows.

   ```
   Branching-Parallel bp.txt assignment
   ```

   You should organize the computation in such a way that the parallel time is as minimum as possible with respect to a given number of processors. For instance, by organizing the computation into a tree-like structure.

4. Create random instances of pairs of branching program/assignment with m in the set $\{1000, 2000, 4000, 8000, 16000, 32000, 64000\}$. Assume that $n = m$ for this exercise, and that $S = \{0, ..., 59\}$ is an encoding of the group A_5 (alternating group), and that $\oplus$ is an encoding of the multiplication table of this group. Therefore, the only thing that is random is the sequence of triples $i_j I_0^j I_1^j$.

   For each number p of processes in the set $\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$, measure the time $T(m, p)$ between the end of the dispatching of the data to all the processes (after the call to the Scatter function) until the end of the computation. Organize the experimental results into a table whose rows correspond to m and whose columns correspond to p.

5. Write a brief report describing the technique you used to parallelize your program. This report should also contain a small description of your results.