# Assignment 1 : Parallel Random Number Generation

1. **The description of the program**

In this assignment, variable a, m and c are set as global variables, which are 1664525, 42294967296 and 1013904223 respectively given by the handout.

**The main steps and strategy for parallelizing and generating random numbers:**

**Step 1.** The master processor (rank = 0) generates seeds sequentially, which has the same numbers with the amounts of processors, as the original seeds of other processors, and puts them into an array with long\* type.

**Step 2.** The master processor calls the MPI\_Send method to deliver seeds to its own processor.

**Step 3.** The master processor generates the random numbers by leapfrog method and then calculate PI value.

**Step 4.** Slave processors receive their own original seed by calling MPI\_Recv method.

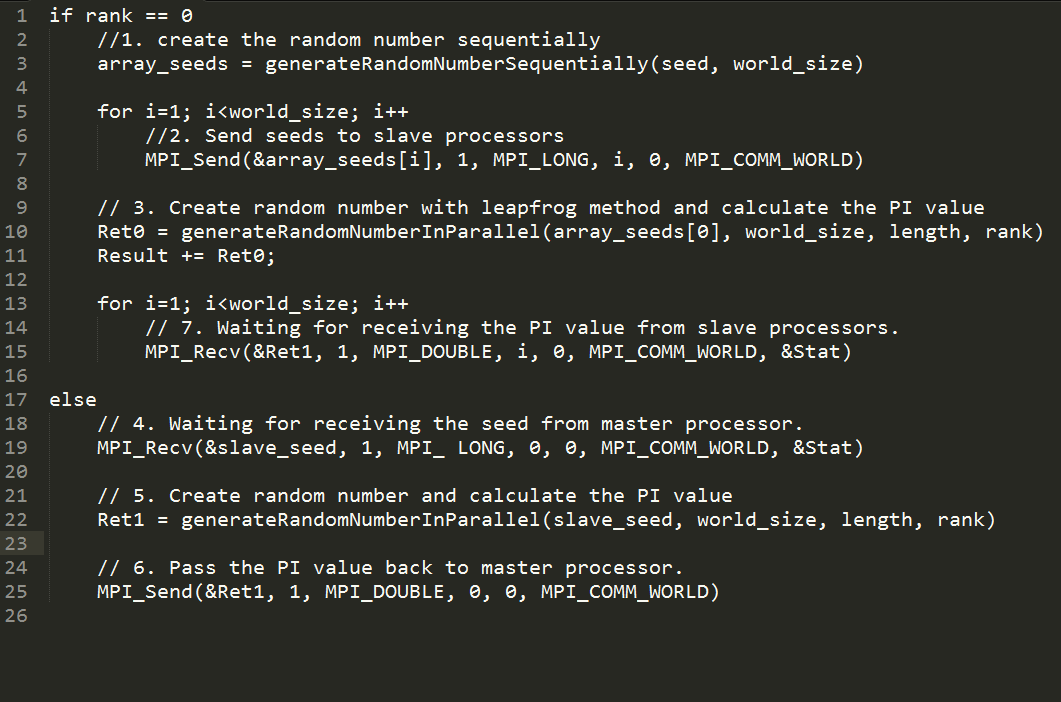
**Step 5.** The slave processor separately generates the random numbers based on their own seeds and calculate the PI value.

**Step 6.** The slave processor send the respective result to the master processor

**Step 7.** The master processor waits for receiving the PI with MPI\_Recv.

**Step 8.** The master calculate the average of all the return value and get the final PI value.

1. **Pseudo code**



1. **The comparison of performance**

|  |  |  |
| --- | --- | --- |
| **The amounts of random number** | **Parallel Method**  **(4 processors)** | **Naïve Method** |
| 1010 | 16.947380 | 90.12 |
| 109 | 12.013179 | 63.92 |
| 108 | 1.202400 | 6.39 |
| 107 | 0.181387 | 0.64 |

No surprise, the speed of parallelizing is far higher than the one of sequentially generating when the same length of numbers are calculated.

1. **The estimate of performance based on Amdahl’s Law**

Amdahl’s law: **Speed-up = Ts / Tp** , where

**Ts**: The total time of serial execution.

**Tp**:The execution time after parallelizing.

Because of **Speed-up = P/ (f \* (P-1) +1)**, where

**P**: parallelization factor

The speed-ups can be calculated like the table below according to this law.

In this test of performance, the amounts of random number is set 1010 .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Total time of serial execution** | **The number of Node** | **The number of Processor** | **parallelization factor** | **The execution time after parallelizing** | **Speed-up** | **f** |
| 90.12 | 1 | 1 | 1 | 67.700719 | 1.3312 | 0 |
| 90.12 | 1 | 2 | 2 | 34.892877 | 2.5828 | -0.2256 |
| 90.12 | 1 | 4 | 4 | 16.945040 | 5.3184 | -0.0826 |
| 90.12 | 1 | 8 | 8 | 8.473347 | 10.6357 | -0.0354 |
| 90.12 | 2 | 8 | 16 | 8.474091 | 10.6348 | 0.0336 |
| 90.12 | 3 | 8 | 24 | 8.482555 | 10.6242 | 0.0547 |
| 90.12 | 4 | 8 | 32 | 8.475114 | 10.6335 | 0.0648 |

md

1. **Conclusion**

As I expected before, when the numbers of processor increases, the running time of the program decreases, and the speed-up approximates the same multiple with the numbers of the processors, but when the number of processors reach certain a value, the time of processing will not change any more, which tends to be stable.

1. **File description**

|  |  |
| --- | --- |
| **Filename** | **Description** |
| CalcPiInParallel.cpp | Implementing to calculate the pi in parallel environment |
| CalcPiInSequence.cpp | Implementing to calculate the pi sequentially |
| CalcPI.pbs | The configuration for parallelizing |

Student ID: 16242950

Student Name: Hongyu Zhou (Oscar)