Parallel and Distributed Programming
Uppsala University – Spring 2020
Report for Individual Project: Simulation for Malaria Epidemic

Li Ju

9th June 2020

I Introduction

1 Background

Malaria:
Malaria is an ancient disease having a huge social, economic, and health burden. Even though the disease has been investigated for hundreds of years, it still remains a major public health problem with 109 countries declared as endemic to the disease. Mathematics and mathematical models play an important role of providing an explicit framework for understanding the disease transmission dynamics within and between hosts and parasites. In a mathematical expression or a model, several known clinical and biological information are included in a simplified form by selecting features that seem to be important to the question being investigated in disease progression and dynamics. Therefore, a model is an ”approximation” of the complex reality, and its structure depends upon the processes being studied and aimed for extrapolation[1].

Stochastic Simulation:
A stochastic simulation is a simulation of a system that has variables that can change stochastically (randomly) with individual probabilities. Realizations of these random variables are generated and inserted into a model of the system. Outputs of the model are recorded, and then the process is repeated with a new set of random values. These steps are repeated until a sufficient amount of data is gathered. In the end, the distribution of the outputs shows the most probable estimates as well as a frame of expectations regarding what ranges of values the variables are more or less likely to fall in[2].

Monte Carlo Simulation:
Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other approaches. Monte Carlo methods are mainly used in three

problem classes: optimization, numerical integration, and generating draws from a probability distribution[3].

2 Problem Description

Here, the project is aimed to simulate the development of malaria epidemic with Monte Carlo method, combined with the Stochastic Simulation Algorithm. The program will run on n processors parallelly and for each processor, p individual Stochastic simulations will be done. All $n \times p$ test results will be collected and all data for attribute ”susceptible humans” will be saved. Further, statistics-based visualization and analysis will be done on the as-obtained data.

The performance of the program will also be evaluated, including running time analysis, strong scalability and weak scalability.

3 Error Control

In the simulation program, all integers are stored and calculated as ”int” and all real numbers are stored and calculated as ”double” in C.

4 Computer details

For the individual project, all tests are conducted on UPPMAX system Rackham. The detailed information of Rackham cluster is listed below:

a) Operating system: CentOS Linux 7 (Core)

b) Interconnection: Infiniband FDR, with a theoretical bandwidth of 56Gb/s and a latency of 0.7 microseconds

c) CPU model: Intel(R) Xeon(R) Xeon V4

d) Number of nodes: 486

e) Compiler: gcc (GCC) version 10.1.0

II Solution

1 Algorithm design

In this project, to do a Monte Carlo simulation for malaria epidemic combined with the Stochastic simulation algorithm, N parallel simulation should be done and collected. For each individual simulation, Gillespie’s direct method is used. Therefore, following algorithm is designed.
To parallelize the computation, MPI library is used. As the computation in each for iteration is independent, the whole computation is perfectly parallelizable. Therefore, for N experiments,

```
Algorithm Gillespie's direct method-based MC simulation
Require: The number of MC experiments $N$
  for i in 0: $N$ do
    Set a final simulation time $T$, current time $t = 0$ and initial state $x = x_0$
    while $t < T$ do
      Compute $\boldsymbol{w} = \text{prob}(\boldsymbol{x})$
      Compute $a_0 = \sum_{i=1}^{R} (i)$
      Generate two uniform random numbers $u_1$ and $u_0$ between 0 and 1
      Set $\tau = -\ln u_1 / a_0$
      Find $r$ that $\sum_{k=1}^{r-1} \boldsymbol{w}(k) < a_0 u_2 \leq \sum_{k=1}^{r} \boldsymbol{w}(k)$
      Update the state vector $\boldsymbol{x} = \boldsymbol{x} + P(r, :)$
      Update time $t = t + \tau$
    end while
  end for
  Collect the first elements of all final state vectors and save them
```

if n nodes are given, each node will perform $p = N/n$ experiments and the final data will be collected in one node and saved by this node.

As the computation cost for each simulation is constant C, time complexity for all N simulations is linear to the number, $\mathcal{O}(N)$. If n processors are given, time complexity for the computation is $\mathcal{O}(N/n)$.

2 Performance experiments

For performance evaluation, time analysis is introduced. In our performance test, the counted time included the time cost on message passing and local calculation in each processor (data writing not included). The maximum time cost of all processes is used to represent the time cost of the parallelized program. All the tests are with optimization flag O3 of GCC.

In case of strong scalability, the number of processors is increased while the problem size remains constant. This results in a reduced workload per processor. In our experiments, multiple cases are tested, and the problem size is fixed with a total $N = 10^5$. Each test case is done 5 times and the minimum time cost is used to present the time cost. Numbers of processors are chosen as 1, 2, 4, 5, 8, 10 and 16. Time cost in the experiments include local calculation and message passing (data collection in this case) and time cost on writing results to disk is not included. The detailed information about the strong scalability test is shown in Performance analysis part[5].

In case of weak scalability, both the number of processors and the problem size are increased. This results in a constant workload per processor. In our experiments, we fix the workload for each process as 10000 simulations. Same as strong scalability tests, each test case is also done 5 times and the minimum time cost is used to present the time cost of the test case. The detailed test cases are listed in performance analysis part.

III Performance analysis

1 Analysis for the simulation results

3 different N are chosen to do the simulation, which are $10^6, 1.2 \times 10^6$ and 1.5×10^6 and further analysis are performed based on the results. Basic statistics are done and the results are shown in Table(1)

N	Mean	Standard Deviation
1000000	606.8289	58.64262
1200000	606.8673	58.71637
1500000	606.7498	58.59967

Table 1: Statistics of 3 simulation experiments

Also, to check the distribution of the as-obtained data, histograms are also made and shown as Fig(1), Fig(2) and Fig(3). As we can see from the histograms, the distributions of data are almost normal. This is also confirmed by the instruction of the project.

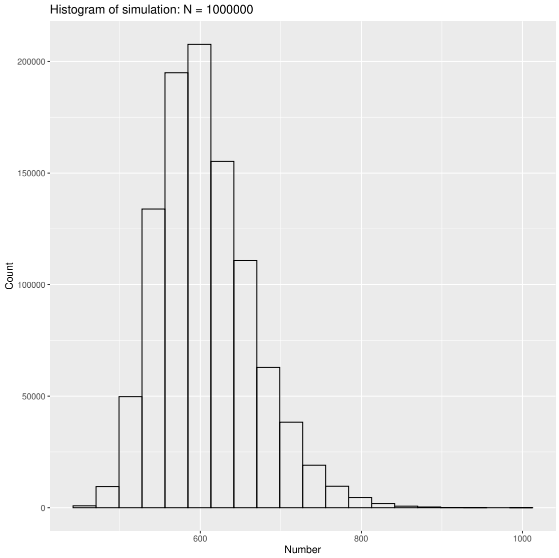

Figure 1: Histogram of data: N = 1000000

2 Strong scalability analysis

The results for strong scalability test is shown as Table(2). Also, the speedup versus number of processors are also plotted as Figure(4) shows.

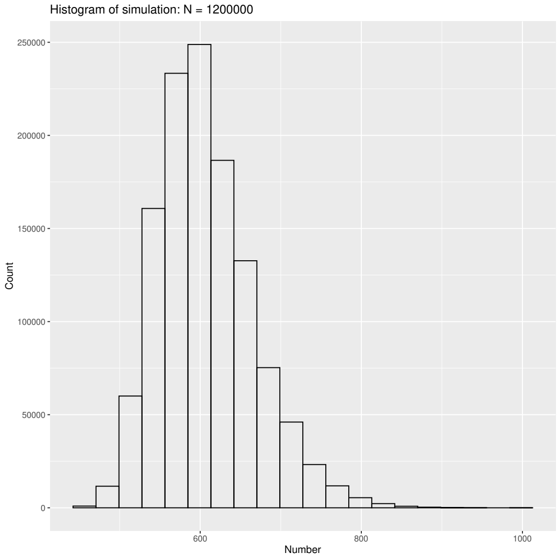

Figure 2: Histogram of data: N = 1200000

Total N	Processors	Local computation n	Time cost/s	Speedup
$N = 10^5$	1	100000	131.329901	1
$N = 10^5$	2	50000	67.073093	1.958012
$N = 10^5$	4	25000	34.903835	3.762621
$N = 10^5$	5	20000	27.695708	4.741886
$N = 10^5$	8	12500	17.512676	7.499134
$N = 10^5$	10	10000	14.017911	9.368721
$N = 10^5$	16	6250	8.900410	14.755489

Table 2: strong scalability analysis results

As we can see from the figure, when the amount of task is fixed, as the number of processors increases, the speedup of the program increases correspondingly. Due to the time cost on communication and system, it is not possible to reach the ideal speedup. However, the real speedup is very close to the ideal speedup line, which indicates that the program have good scalability for fix-sized problem.

3 Weak scalability analysis

The results for strong scalability test is shown as Table(3). Also, the efficiency versus number of processors are also plotted as Figure(5) shows.
As we can see from the figure, when the computation amount for each processor is fixed, as the number of processors increases, though the efficiency of the program decrease slightly, generally speaking the efficiency keeps to be stable and very close to 1. This is because the problem we are facing is perfectly parallelizable: each individual simulation is absolutely independent and even the communication between processors is required only once: only if the simulation is finished and data is collected once. This is the main reason why the program is of very good weak scalability.

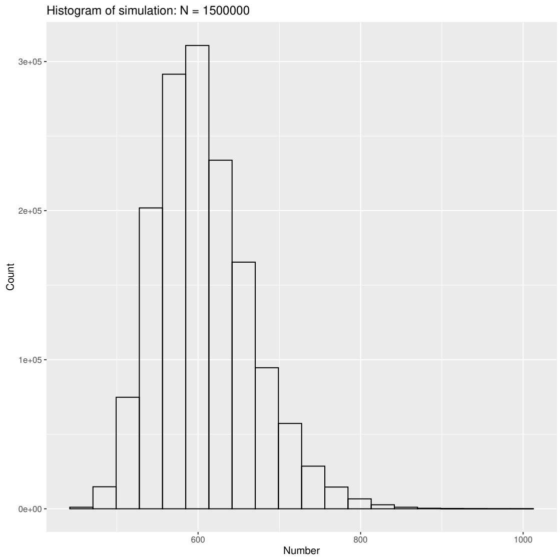

Figure 3: Histogram of data: N = 1500000

Local computation n	Processors	N	Time cost/s	Efficiency
10^4	1	10000	12.084322	1
10^4	2	20000	12.471116	0.9689848
10^4	4	40000	12.804184	0.9437792
10^4	5	50000	13.276616	0.9101959
10^4	8	80000	14.176783	0.8524023
10^4	10	100000	14.185153	0.8518993
10^4	16	160000	14.202948	0.8508320

Table 3: Weak scalability analysis results

4 Communication analysis

In the program, inter-process communication is required in two parts: 1. after calculation for local maximum and minimum values in each process, comminucation is required to let every process get the global maximum and minimum value; 2. after local statistics are done, collection for local data is required.

For the first communication, MPI_Allreduce function is used, reducing all local maximum and minimum value to global maximum and minimum value and saving them in each process. Another approach is to use Send/Recv pair to collect all data in one process, calculate global minimum and maximum locally and broadcast them. One more approach is to gather all maximum and minimum values with allgather function and calculate global ones locally. Comparing with these two methods, allreduce function is the easiest way taking advantage of its built-in MPI_MAX and MPI_MIN operation and can be finished in one line. Also it is the most efficient way: only one system communication call is required.

For the second communication, after local statistics are done and stored in an array, these data need to be collected to a single process and saved to output file. Here in our implementation MPI_Reduce function is used, again, taking advantage of MPI_SUM operation. Another approach is to use gather function and save all arrays in a longer array in one process. For each

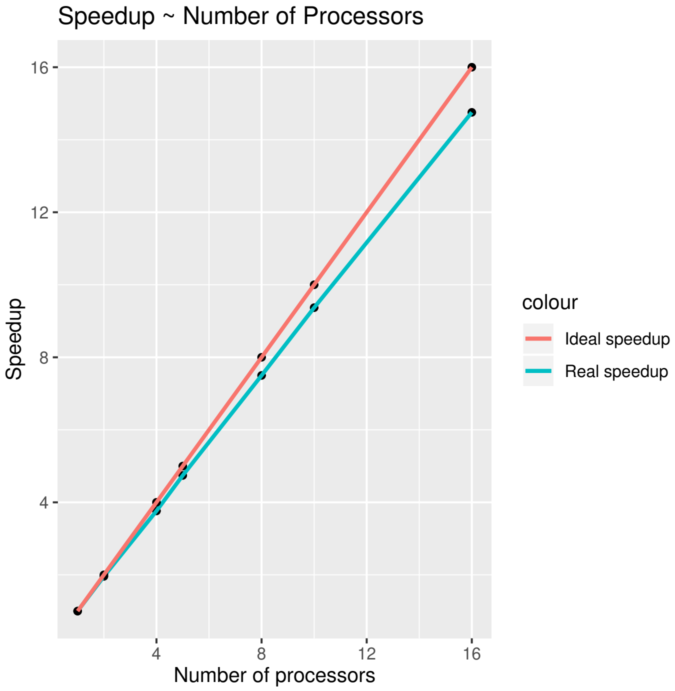

Figure 4: Speedup versus number of processors: strong scalability analysis

interval, data from each process need to be summed and stored in another array. Comparing with this, MPI_Reduce is much more easier and can collect and sum data in one function call.

Therefore, for this program, MPI_Allreduce and MPI_Reduce function are used to collect data, which are the most efficient and easiest way for data communication.

IV Conclusion

In the project, simulation for the development of malaria epidemic with Monte Carlo method, combined with the Stochastic Simulation Algorithm is implemented parallel using MPI libraries. Analysis for the results and for the program performance are conducted. Based on the analysis, it can be concluded that the program is able to produce reliable simulation for malaria epidemic, and the program is perfectly parallel and scalable. Further optimization still can be done, like thread-level optimization using pthread or OpenMP library to parallelize the program further, or using in-line functions to replace numerous function calling.

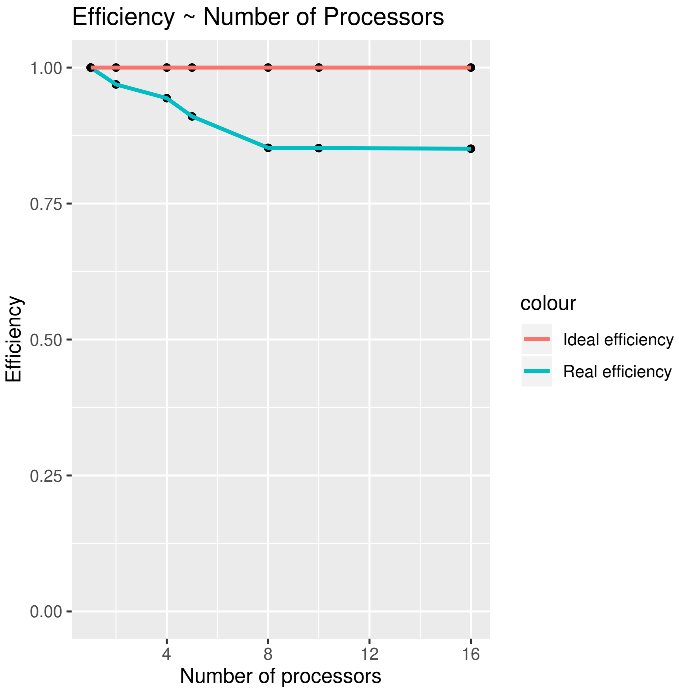

Figure 5: Efficiency versus number of processors: weak scalability analysis

References

[1] Mandal, S., Sarkar, R.R. & Sinha, S. Mathematical models of malaria - a review. Malar J 10, 202 (2011). https://doi.org/10.1186/1475-2875-10-202

[2] Wikipedia contributors. "Stochastic simulation." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 18 Feb. 2020. Web. 22 May. 2020.

[3] Wikipedia contributors. "Monte Carlo method." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 20 May. 2020. Web. 22 May. 2020.

[4] James W Longley (1981) Modified gram-schmidt process vs. classical gram-schmidt, Communications in Statistics - Simulation and Computation, 10:5, 517-527, DOI: 10.1080/03610918108812227

[5] Xin Li,Scalability: strong and weak scaling, https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/