Table of Contents

Abstract. The implementation of parallel computing on the integral of a function by Monte Carlo method is described in this assignment. Particularly, the assignment exploits the FastFlow parallel framework and C++ threads for building stream parallelism including pipeline and farm. The experiments illustrate the comparison of time consuming of these ways with different interval numbers, a set of functions and a various threads.

1 Introduction

With the development of hardware technologies and a plenty of data, the requirement of time consuming to process or compute some tasks have been noticed, which leads to the growth of parallelism. The parallelism tasks are divided into some parts such as: stream parallelism, data parallelism, task parallelism and so on. This assignment is about exploiting the stream parallelism on computing integral of a given function with Monte Carlo method. Because of a stream of interval numbers from the re- quirement of the project, a stream parallelism with pipeline and farm is utilized to parallel computing the Monte Carlo method for each interval number. To construct a stream par- allelism structure, the FastFlow (FF) framework and C++ threads are mentioned in this assignment.
The remaining of the assignment is constructed: the Section 2 describes about the implementation of computing the integral of a given function with Monte Carlo. The results and experiments using FF and C++ threads are represented in Section 3, while Section 4 concludes the assignment.

2 Implementation

In this part of the report, the general idea of the Monte Carlo in computation of a integral function is designed by FF framework and C++ threads. The basic idea is about forming an interval number object with basic fields (a range of the interval number, a random number (N), list of N random numbers, Monte Carlo number of this interval and an ID for sorting) and methods. Hence, each interval number goes through each stage which executes specific tasks in stream parallelism built by FF and C++ threads.

2.1 FastFlow

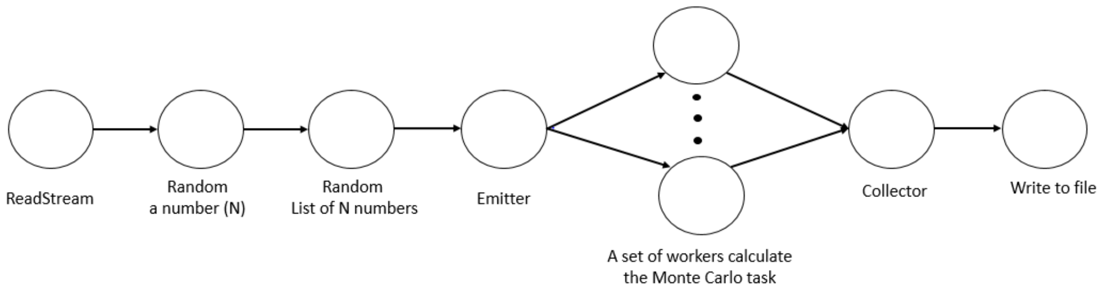


Fig. 1: The diagram of FF structure

To build a parallelism, the FF a structured parallel programming framework [2] is utilized because of plenty of advantages such as highly efficient stream parallel and a set of ready-touse for programmers. In addition, FF can be flexible for creation complex parallelism with a lot of different nodes. With the Monte Carlo task, the FF framework is used to build a stream parallelism with pipeline and farm. Particularly, from the Fig 1, it can be clear that the structure is built based on a set of stages including: reading a set of interval numbers, choosing a random number (N), choosing a list of N random numbers, setting Monte Carlo number calculations to threads and writing to a file. Generally, each stage has a specific task to resolve. Because of the convenience of FF, each of stage or task or node is implemented as a function. However, if the a stage has more than one input or output, it can be redefined. In more detail, the tasks of stages is described:

- readStream: takes a couple numbers at each time from a stream of file.
- random a number (N): randomly selects a number.
- random a list of N numbers: is generated from this stages.
- emitter: split interval numbers to free workers.
- a set of workers: compute Monte Carlo method for set of interval numbers.
- collector: takes bunch of interval numbers and send to the next stage.
- write to file: write the Monte Carlo to a file.

2.2 C++ Thread

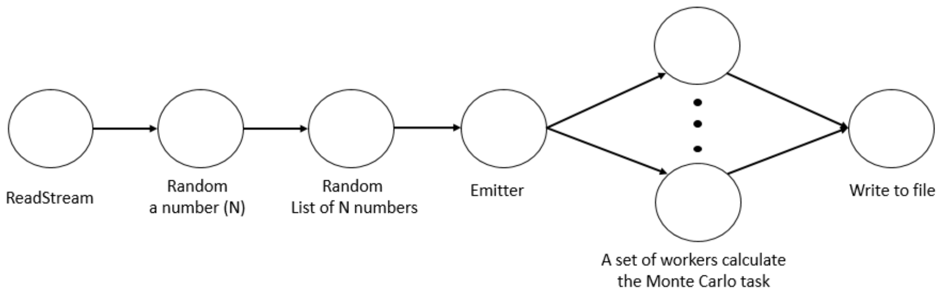
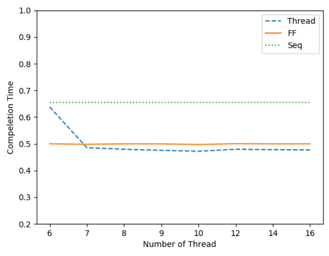
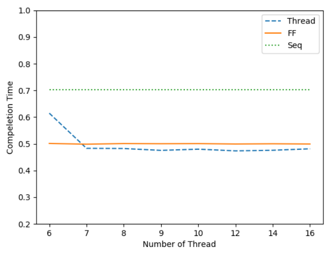


Fig. 2: The diagram of C++ threads structure

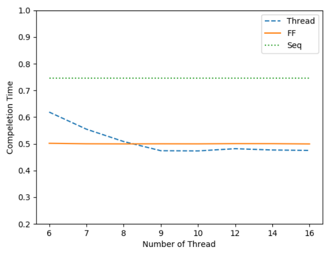
In case of using C++ threads, a similar construction is built with pipeline and farm on stream parallelism (in the Fig 2). However, without supporting from FF, the communication between stages is executed through queues and combination of mutex and condition variable in C++ [1]. In more detail, the queues run a role like containers which hold interval numbers executed from previous stage and waiting the next call of the next stage. Meanwhile, the mutex and condition variable are similar to doors and bells, respectively. Particularly, to start a stage from a thread, the mutex is used to lock the thread. After a stage has completed its task, the thread is unlocked by the mutex and sends a notify to next threads waiting for their turns. Based on some conditions, the notified threads can be waken up for their tasks or still sleep. In addition, each stage in case of C++ threads is implemented by a thread with a specific task like FF construction. The general idea of stream parallelism in C++ threads construction is quite similar the way of FF construction. Nevertheless, at the last stage, the C++ threads construction uses a queue to contain results of workers, in stead of using a collector to gather output of workers from farm. Then the final stage pops each result from queue to a file. Before writing the result to the file, the Monte Carlo numbers are sorted based on the ID of interval numbers.



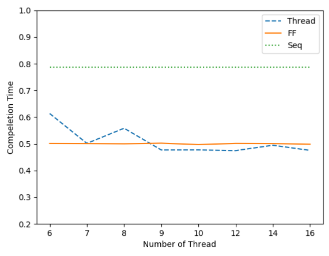
(a) Power function: 5



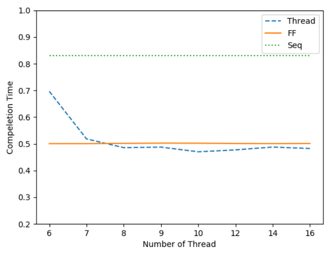
(b) Power function: 6



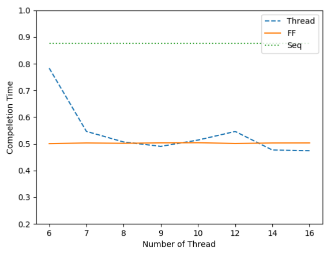
(c) Power function: 7



(d) Power function: 8



(e) Power function: 9



(f) Power function: 10

Fig. 3: Graph of completion time in different functions with 100 interval numbers

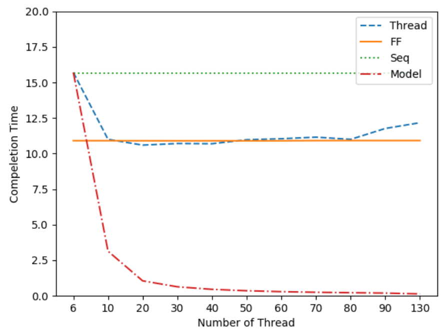
3 Experiments

The experiments of the Monte Carlo task with FF framework and C++ threads are mentioned in this section. Particularly, a set of different functions in Table 1 (with powers from 5 to 10) tried, while streams of interval numbers are generated randomly including 100 and 2000 interval numbers with a in [1, 1000] and b in [a, 1500]. Additionally, the assignment considers many different threads from 6 to 130 in experiments. About the complexity of construction between FF framework and C++ threads, with the FF, programmers need to define tasks for every stage of construction and correctly select classes or structures to build the application, whereas in case of using C++ threads, the programmers is required to clearly understand the idea of communication between threads.

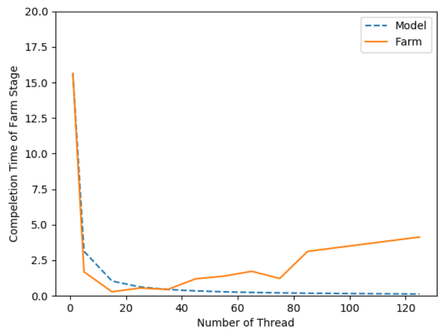
Table 1: List of functions

Power	Set of Function
5	x^5 + $2x^4$ + $3x^3$ + $4x^2$ + 5x + 6
6	x^6 + $2x^5$ + $3x^4$ + $4x^3$ + $5x^2$ + 6x + 7
7	x^7 + $2x^6$ + $3x^5$ + $4x^4$ + $5x^3$ + $6x^2$ + 7x + 8
8	x^8 + $2x^7$ + 3xx^6 + $4x^5$ + $5x^4$ + $6x^3$ + $7x^2$ + 8x + 9
9	x^9 + $2x^8$ + $3x^7$ + $4x^6$ + $5x^5$ + $6x^4$ + $7x^3$ + 8xˆ2 + 9x + 10
10	x^{10} + $2x^9$ + $3x^8$ + $4x^7$ + $5x^6$ + $6x^5$ + $7x^4$ + $8x^3$ + $9x^2$ + 10x + 11

The first experiment on Fig 3 illustrates the difference of functions on the Table 1. The vertical and horizontal axes are the completion time and the number of threads, respectively. From



(a) Completion time of application



(b) Completion time of Farm stage

Fig. 4: Graph of completion time in case of whole application and farm stage with 2000 interval numbers and power of function being 7

this experiment, it can be seen that there are not too much different from these functions. Therefore, the next experiment choose a fixed function. In this case, the function "x^7 + $2x^6$ + $3x^5$ + $4x^4$ + $5x^3$ + $6x^2$ + 7x + 8" is selected.
In the Fig 4, the experiment is displayed through the vertical and horizontal axes with completion time and the number of threads, respectively. In particular, with one worker in a farm stage, the completion time of C++ threads quite similar to the completion time of sequentiality because it reduces some time with pipeline and introduces some overhead from communication between stages, creating and termination threads. Then, when the number of workers increases from two to twenty, the completion time of C++ threads drops, but the level of decrease cannot be exactly equal to the predictive version due to the overhead of other stages. Meanwhile, in the farm stage in the Fig 4b, the latency of C++ threads is better than the predictive because with a processor, the data which fit to this process's cache is quite small, need some time for read and write to the main memory, whereas in case of more processor (more cache) the time for reading and writing from main memory can be eliminated or reduced. After falling, the completion time of C++ thread reaches the plateau when the number of workers rises until under forty, before slowly rising and soaring. Due to the increase of latency of farm stage, the completion time of the C++ threads goes up with the number of worker is greater than around 25.

4 Conclusion

The computation of integral of a given function with Monte Carlo method from a stream of interval numbers is implemented with stream parallelism. In more detail, the implementation is built based on the combination of pipeline and farm with two different technologies such as FF framework and C++ threads. The comparison of these techniques can be clear that in case of time performance, the C++ threads is better than FF framework. Nevertheless, the construction of FF framework is less complex than using C++ threads. are

5 Ackowledgements

I would like to show my gratitude to Prof. Marco Danelutto and his Assistant Dr. Massimo Torquati about lessons in Distributed systems: paradigms and models course at University of Pisa. Additionally, I say thank to my friends who support and encourage me in this course.

References

1. C++ threads. http://en.cppreference.com/w/cpp/thread. [accessed 05 Feb 2018].
2. Fastflow framework tutorial. http://calvados.di.unipi.it/storage/tutorial/fftutorial.pdf. [accessed 05 Feb 2018].

6 Compiling and Runing

To compile the source code of these implementations on the server, the Cmake is utilized as below.

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

To running these implementations, the same number of parameters are used.

```
./ff <NoThread><IntervalNumberFile><PowerFunc><ElementFunction>
./threads <NoThread><IntervalNumberFile><PowerFunc><ElementFunction>
./seq <IntervalNumberFile><PowerFunc><ElementFunction>
```

For example, the FF framework and C++ threads are executed with 10 threads, a path of Interval Number File "../input/Input.txt", and a given function $x^3 + 5x^2 + 2x + 1$.

```
./ff 10 ../input/Input.txt 3 1 5 2 1
./threads 10 ../input/Input.txt 3 1 5 2 1
./seq ../input/Input.txt 3 1 5 2 1
```