[[1]](#footnote-1)

The preliminary research of Adaptivity in Stateful Parallel Pattern

Tianbai Peng[[2]](#footnote-2)

*Abstract*

Nowadays, parallelism has become a new trend in software and architecture design. And with cloud and edge deployments of parallel/distributed applications, the need for adaptivity in such applications has become important. However, the stateful pattern deployed in a parallel pattern which shows discontinuous parallel and therefore low efficiency. To address this problem, it’s necessary to measure how the ratio of sequence computing and parallel computing influence the running efficiency in a parallel pattern, and need to think about a new methodology or develop a new feasibility technique. In this preliminary research article, a c++ pattern-based parallel programming framework - Fastflow would be used, it provides developers with a set of parallel programming pattern, such as Map, Task-Farm, and Pipeline. In the next few months, I expect the experimental results that relative to the stateful pattern implementation built on Fastflow(Danelutto and Torquati, 2015) would be present, and demonstrate the feasibility and efficiency of a stateful pattern.

*Keywords*

Parallel patterns, skeletons, stateful computations

# INTRODUCTION

S

tate access pattern is a software design pattern which allow an object to change its behavior when its internal state changes, and this pattern models the most conservative situation of computations with ordered access to a partitionable/nonpartitionable state when we have no further information about the properties of the sequential computation and the input items.[1] In most of parallel pattern framework ,people usually implement the code as the stateless patterns which means in concurrent programming the processes do not support the behavior changes when its internal state change. Many programmers may be try to implements state pattern on the parallel framework but cannot get a good efficiency which may just do the sequential computation or get many error. So that it’s a high time to let the state pattern taken into account in different parallel pattern and get the experimental result of stateful computation to find a better way to solve this problem.

In this work, I am focusing on implementing the state access pattern on Fastflow and try to get some experimental results. According to the paper “state access pattern in stream parallel computations”, Fastflow parallel framework was introduced, I had installed the framework on my local machine and I will mostly be using the Fastflow to solve the problem thorough the year. At the beginning I met some problem that I running the code on this framework but cannot get the parallel benefit. So, I Analysis the reason of failure that running in parallel is taken much time than running in series. That’s because that time all the code running on first stage and just pass to the end so that it actually didn't do the parallel work. Then I think about how to break up an algorithm to many parts and write them to each stage. Now any codes can be run on Fastflow by parallel but the problem is the state rather than stateless. At the moment, the problem is implementing the state pattern on the Fastflow, and want to get expected results in future.

The expected result is, they (Marco Danelutto, Peter Kilpatrick, Gabriele Mencagli and Massimo Torquati) found a key point the fact that there exists intermediate cases where there are clearly defined state updates [2]. That’s what I need to research in the future. The contribution of this preliminaty research article consists the steps to finally find the intermediate case, and along with some preliminary experimentalt result about the FastFlow implement of some patterns(pipeline and Task-Farm).

# literature review

During this period of time I have read 3 very good articles which closely relevant to the problem which I am researching, the most important one is “state access pattern in stream parallel computations” which is written by Marco Danelutto, Peter Kilpatrick, Gabriele Mencagli and Massimo Torquati. Actually, this article is very importance to me and I could get lots of useful information on it so that I fully follow the idea and guidance from this article to implement my Fastflow code. What’s more, their experimental result gives me lots of inspiration as well. But as space is limited, I cannot write more. For the rest 2 articles, one is “finding parallel patterns through static analysis in c++ applications”, another is “parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern”. Both of these 2 articles are talking about around parallel pattern which is worthy of referencing in my research Project.

In article “finding parallel patterns through static analysis in c++ applications”, this article provide support for identifying some Fastflow parallel patterns such as Map, Task-Farm, and Pipeline and evaluate the quality of the detection for a set of different C++ applications and structured as follows. Section 2 reviews the state of the art of existing parallel refactoring and parallel pattern detection tools. Section 3 describes the parallel patterns supported in the detection process. Section 4 explains the Parallel Pattern Analyser Tool in detail, along with the Pipeline, Farm, and Map detection modules. Section 5 addresses the experimental analysis and evaluation of the tool. Finally, Section 6 lists some concluding remarks and future works. It’s very good for my research because the basic theory of the parallel programming framework - Fastflow that I am using, is detailed described on this article. When I normally do the research by using the Fastflow, I only know how to call those function to finish the work, but don’t understand the math knowledge and underlying principle of this framework. For example, in the most basic pipeline parallel pattern, I may know that the input data is start from stage1 , then thorough state i … then finally terminate at stage n. But after read the article, it totally changes my thinking on this parallel pattern and makes me know more about it. Because if a Pipeline has n stages, the ith stage computes the function fi: a -> b.[3] Then, for each item x appearing in the input stream, the functions of the Pipeline stages (f1, . . . , fn) are applied consecutively to produce elements in the output stream, so the final output from stage n is fn(f[n-1]( . . . f1(x) . . . )) ! That what I totally cannot think about. This article is very good and easy for me to know more about the FastFlow underlying principle and give me more idea about doing current project. But it also have some weakness. and one gap is that all the c++ code used on their project is writing as the stateless pattern rather than state pattern so that in my future research I need to think about a new way for the state and can’t totally follow all the thing written in their article. Only in this way that I could deal with this difficult problem on state pattern and finish the research successfully.

Another article “parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern” is much different from the first research article which I read. In this article, they focus on how to use the Fastflow framework to deal with the image restoration problem. Because the performance reason, they decide to use a parallel framework, but moving to frameworks not specifically designed for targeting GPGPU offloading, MPI is often considered as the mainstream solution for writing efficient parallel applications for heterogeneous architectures. The low-level approach advocated by MPI falls short in supporting performance portability, especially when hundreds or thousands of concurrent activities are involved and hybrid solutions have to be adapted. Applications must often be re-designed or manually tuned for each new platform by an expert parallel programmer. [4] So finally they choose the Fastflow programming framework because its environment was originally designed to support efficient streaming on cache coherent multicore platforms. [5] It is realized as a C++ pattern-based parallel programming framework aimed at simplifying the development of applications for (shared-memory) multicore and GPGPUs platforms. Then their experimental results prove that the Fastflow framework does increase the performance of the image restoration process. From the application of the Fastflow framework by scientists to the complete use of this framework to solve the problem of image restoration, I thoroughly realized that learning a framework should not only be limited to the content of the tutorial, but should be further practiced from the actual case of others and gain experience to perfect my project. They done a prefect work on image restoration on a good performance but as we know the process of image restoration is a stateless rather than stateful. So that make stateful pattern implementation built on Fastflow as better as possible is the thing I need to consider.

# Preliminary experiment

Although I am in the early stage of the development, I got some preliminary experimentation carried out on the Fastflow.

To measure the parallel performance on pipeline and task-farm pattern on Fastflow, I wrote some code to do the same work running on the serial and parallel and record their running time, and calculate the parallel speedup and parallel efficiency. And I done this by using 5 cores on my visual machine.

Pipeline:

I used 5 stages and input 10 tasks to Pipeline:

Serial Elapsed wall clock time:122

Parallel Elapsed wall clock time:40

Parallel speedup = 122/40 = 3.05

**Parallel efficiency = 3.05/5 = 0.61**

Task-Farm:

I used one emitter, one collector and 5 workers to task-farm, but the emitter and collector didn’t do any calculate

Serial Elapsed wall clock time:251

Parallel Elapsed wall clock time:62

Parallel speedup = 251/62 = 4.05

**Parallel efficiency = 4.05/5 = 0.81**

According to the data we can see that parallel of task-farm pattern is better than pipeline because the mechanism of pipeline may cause some processes idle.

# Conclusion

In this preliminary research article, I have presented some tools and learn the approach from other research article. As we have seen the Fastflow framework that really could get a very good performance on parallel and there is still have lots of development space on stateful access pattern. To this end, as part of the future work, I will continue to implement the stateful pattern on Fastflow, know the Fastflow framework more, even if try to investigate the framework and improve its performance when using stateful access pattern on Fastflow.

# Reference

[1] Marco Danelutto, Peter Kilpatrick, Gabriele Mencagli

and Massimo Torquati

The International Journal of High Performance Computing Applications2018, Vol. 32(6) 807–818

State access patterns in stream parallel computations page 780-781

[2] Marco Danelutto, Peter Kilpatrick, Gabriele Mencagli

and Massimo Torquati

The International Journal of High Performance Computing Applications2018, Vol. 32(6) 807–818

State access patterns in stream parallel computations page 780

[3] David del Rio Astorga, Manuel F Dolz, Luis Miguel Sa′nchez, J Daniel Garcı′a1, Marco Danelutto and Massimo Torquati

The International Journal of High Performance Computing Applications 2018, Vol. 32(6) 779–788

Finding parallel patterns through static analysis in C++applications page 780-782

[4] Marco Aldinucci, Guilherme Peretti Pezzi, Maurizio Drocco, Concetto Spampinato and Massimo Torquati

The International Journal of High Performance Computing Applications 2015, Vol. 29(4) 461–472

Parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern page 464

[5] Marco Aldinucci, Guilherme Peretti Pezzi, Maurizio Drocco, Concetto Spampinato and Massimo Torquati

The International Journal of High Performance Computing Applications 2015, Vol. 29(4) 461–472

Parallel visual data restoration on multi-GPGPUs using stencil-reduce pattern page 465

1. 1. School of Electronics, Electrical Engineering and Computer Science, Queen’s University Belfast, UK (e-mail: tpeng01@qub.ac.uk). [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)