Zorro: A Distributed Thread-Pool with Stealing Proposal

Yukun Jiang
Carnegie Mellon University
Language Technologies Institute
yukunj@andrew.cmu.edu

Leo Guo
Carnegie Mellon University
Language Technologies Institute
jiongtig@andrew.cmu.edu

1 SUMMARY

We are going to implement a thread-pool framework called Zorro (https://github.com/YukunJ/Zorro) that allows users to submit tasks to be executed in parallel without explicit management. This thread pool features a distributed local work queue and enables various work-stealing policies to enhance work balance.

2 BACKGROUND

This framework aims to separate the task definition and parallel task execution. It should ease users' experience of executing multiple tasks in parallel by automatically managing the scheduling and work balancing while squeezing all the parallel performance out of the underlying hardware.

3 Challenge

One major challenge is load balancing. Without knowing the execution time of individual tasks, it is impossible to come up with a static scheduling policy achieving load balancing. Thus, we choose to implement a distributed work queue system with a stealing policy in the hope to cope with the insufficiency of a static scheduling policy. However, even if any thread can steal tasks from other threads, the problem is still challenging in the sense that it is nontrivial to determine when to steal and whom to steal. Thus, one focus would be on proposing and analyzing different stealing policies.

The other difficulty is synchronization. It would be very challenging to collect results from spawned tasks based on what we learned from the lecture during the recursion step without blocking. The other challenging part is that it is nontrivial to synchronize the portion of the code executed in parallel and get back to the sequential execution portion. A waiting mechanism needs to be implemented carefully.

4 RESOURCES

We plan to implement everything from scratch. We might check existing implementation on thread-pool, but we do not have anything specific in mind, and we could not provide a reference now.

For benchmark purposes, we will use the GHC machine cluster and PSC supercomputers. We need multi-core machines to test out the parallelism of our implementation.

5 GOALS AND DELIVERABLES

PLAN TO ACHIEVE: We plan to finish a workable implementation of the thread pool framework with a global work queue, a distributed local work queue system with different stealing policies, and various benchmark tasks to evaluate the result. We hope that we can achieve a close-linear speedup even if the execution time of tasks is highly variable. If things really go worse, we at least hope to get a working version of the distributed thread pool regardless of performance.

HOPE TO ACHIEVE: Right now, we do not plan to support recursion that returns any value, and we only plan to support recursion that modifies data in place to avoid the difficulty to collect results from finished tasks. If the project goes really well, we might choose to support recursion with return values.

6 PLATFORM CHOICE

We will be implementing this project on Linux OS using C++17. As thread-pool is a relatively low-level infrastructure, it makes sense to use compiled language that yields high performance and gives us more control over details.

7 Schedule

Week1 (Apr 3 - Apr 9): Read related literature and setup the repo build and base interface

Week2 (Apr 10 - Apr 16): Build the global work queue version of threadpool

Week3 (Apr 17 - Apr 23): Build the distributed local work queue version of the threadpool

Week4 (Apr 24 - Apr 30): Enable work-stealing policy to enhance work balance

Week5 (May 1 - Final): Wrap up the project and prepare report poster.