Zorro: A Distributed Thread-Pool with Stealing Milestone

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

So far we have laid out the project scope and started implementation as scheduled. We first designed a test suite consisting of various kinds of workloads to comprehensively measure the performance of different implementations of **Zorro**, including correctness test, lightweight test, computationally-intensive test, imbalanced test, and recursion test. And we have implemented the global work queue version, and are in the middle of implementing the thread-local work queue version.

2 PROGRESS

Right now, we are on track with respect to the goals and deliverables stated in our proposal. We believe that we are able to produce all deliverables stated in the proposal. Given the current progress, it is still hard to complete the ”nice to haves” stated in the proposal. It needs to fundamentally change our framework’s interface to receive return value from the spawn tasks. It is possible to get the return value from spawned tasks via input parameters, but the type of spawned task must be void.

We have already finished a workable implementation of the thread pool framework with a global work queue and various test cases. For the next step, we plan to finish implementing a distributed local work queue system with different stealing policies. We also want to implement more computationally heavy and parallelizable test cases to further test the best possible speedup provided by our framework and how our framework resolves workload imbalance and contention. Ideally, the framework should have a close-linear speedup under such cases. Further, to support stealing policies, we also plan to implement a dequeue with fine-grained locking.

3 POSTER SESSION PLAN

We plan to show graphs demonstrating the speedup on various test cases by the different implementations of our thread-pool framework. It is not really possible for us to make a demo since there is no graphical output being generated, and we only care about the speedup and synchronization overhead of our framework.

4 PRELIMINARY RESULT

Here is our preliminary result comparing the global work queue version Versus the dummy serial version of threadpool on our test suite.

Thread = 8	**Dummy**	**Global Queue**
Correctness	0	501 (**0.000**)
Light	1	473 (**0.002**)
Computation	5260	686 (**7.668**)
Imbalanced	2187	328 (**6.668**)
Recursion (Divide)	106	27 (**3.926**)
Recursion (Divide+Merge)	1269	888 (**1.429**)

5 CONCERN

Right now we are having a mediocre speedup with the global queue version implementation. Ideally, we are envisioning that with the thread-local queue version and work stealing, the speedup will improve non-trivially for test cases with low computation and high contention. We are not sure if this is going to be the case or not.

6 SCHEDULE

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

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

Week3 (Apr 17 - Apr 23): Build the distributed local work queue version of the threadpool [**IN PROGRESS**]

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.