C Library for fork-join style parallelism

Ayush Agrawal (ayushagr)

https://www.andrew.cmu.edu/user/ayushagr/

All the code for the project is hosted at https://github.com/A-star-ayush/silk. This is my private repository. I was not sure whether I could make it public or not. I will be meeting someone from the course staff once the exam on Wednesday (Nov 20, 2019) is concluded to share my progress and get some suggestions on the ease of use of the library being implemented. If there is a requirement that the source code be available public at all times, kindly let me know so that I can make the repository public.

(1) Detailed description of schedule so far and the revised schedule for the time remaining is posted on the main project page (https://www.andrew.cmu.edu/user/ayushagr/).

I am running a little late on the schedule I proposed. This is mainly due to the fact that from 11 Nov to 17 Nov I was in California giving multiple on-site interviews. This was actually unforeseen. I though the interviews would be more scattered and most of them would be during the first week of December. I have revised my schedule which can be seen on the project page.

(2) To summarize the work done so far, I have read the two research papers I proposed to read in the project proposal. But while going through them, I found others which I will read:

“The implementation of Cilk-5 multi-threaded language”
“Tapir: Embedding Fork-Join Parallelism into LLVM's Intermediate Representation”

I am done implementing the single work queue solution. I wouldn’t say it’s completely stable right now. If I use my library with an array of more than 100 million entries, I am currently crashing. I am investigating into the issue.

I have also written an implementation of quick sort that uses my library to test my implementation so far.

I spent quite a lot of time thinking of various ways to make the library easier to use. At the end, couldn’t find a workaround for mandating the work function to be of a certain prototype. But in the process, came to appreciate the advantages that come from implementing thread fork
parallelism as an extension to the language rather than a separate library which I was trying to achieve.

(3) As mentioned earlier, I am running a little late. But I think I will be able to deliver what I mentioned in the project proposal. Goals that I plan to hit for the poster session are as follows:

**Plan to Achieve:** Have a working library that can be used to model fork-join pattern for any underlying problem. Having implemented both types of stealing, come up with a comparison of the two across 3 different algorithms. Be able to quantitatively talk about the overhead of maintaining distributed work queues and what all optimizations went into implementing the system. I anticipate that the average work queue length will be smaller in case of continuation stealing and moving from a single work queue to a distributed work queue should improve the performance significantly.

This seems a lot like the plan I had presented in the project proposal. To add to this, while I was doing the project, I came up with a new idea:

“why can't we limit the queue length of continuation first, i.e, block if trying to add work to the queue and the queue is full. I would like to implement this idea and see how does it affect the overall performance - I suspect it to increase the synchronization overhead but at the same time knowing the queue size in advance can help improve performance in other areas.”

I plan to implement this new idea as well and present it during the poster session.

(4) For the poster session, I plan to show graphics depicting the comparison between child first and continuation first (and also the idea I mentioned above). There will also be a quick demo of how can I one easily integrate the library I am developing into their project to express parallelism. More than anything else, I feel my poster session will be more of a story - all the little problems and solutions I came across while developing this project. I have come to understand one thing that parallelizing recursive routines is not trivial. I will present why I think so as well.

(5) The issue that is currently concerning me the most is how to implement continuation first. The idea I have right now involves setjmp and longjump but before I actually go on to implement it, I can't say for sure whether it will work or not. Other than that, I think I am doing fine.