Full Name: Pengyun Zhao & Zian Ke
Andrew Id: pengyunz & ziank

15-418/618 Spring 2020
Project Proposal Checkpoint

Assigned:	Wed., Mar. 27
Due:	Sun., Apr. 5, 11:00 pm

Currently we have three ideas for the project. We'll decide which one to work on next week. Your feedbacks and suggestions will be helpful to us.

1 Parallel linear programming algorithms

The idea will be to implement a parallelized version of Linear Programming algorithms (Simplex/Primal-Dual/Dantzig-Wolfe/Benders/Augmented Lagrangian) in parallel programming languages such as CUDA, OpenMP or MPI and benchmarking the performance of the algorithm using a set of LP problems of varying sizes of variables and constraints according to the algorithms given in the paper **Decomposition and Parallelization of Linear Programming Algorithms** by Andrzej Karbowski in 2015.

The idea interests us because one of our team number has taken several Algorithm classes before (and he is also taking 15-651 right now) and Linear Programming has always been a very important part of the Theoretical field of Computer Science. Although a variety of algorithms exist that can solve the problem in polynomial time, parallelism has hardly been introduced to the field in spite of its long time of existence to provide further improvement to the algorithms' actual running time as well as their scalability for large-scaled LP problems.

And for benchmarking, the implementation can simply be benchmarked on GHC and Lateday machines for CUDA, OpenMP or MPI implementations.

We have consulted the following papers, with each one providing us with different help:

- **N. Alon and N. Megiddo, "Parallel linear programming in fixed dimension almost surely in constant time," Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 574-582 vol.2.** This paper sets the motivation for this idea, proving that parallelism will provide a large performance speedup to the traditional algorithms.
- **Karbowski, Andrzej. (2015) Decomposition and Parallelization of Linear Programming Algorithms. Advances in Intelligent Systems and Computing. 350. 113-126. 10.1007/978-3-319-15796-2_12.** This paper sets the implementation basis of the idea, providing parallelized versions of the most common algorithms used in solving LPs, this will be where our implementation will be based on.

- **Lubin, Miles & Hall, Julian & Petra, Cosmin & Anitescu, Mihai. (2013) Parallel distributed-memory simplex for large-scale stochastic LP problems. Computational Optimization and Applications. 55. 10.1007/s10589-013-9542-y.** This paper provides a sample implementation of the Simplex algorithm, which traditionally solves LPs in expected polynomial running time (exponential in the worst case), that utilizes shared memory. It also provides us with sample codes that uses OpenMP, which could be a reference for us in the actual implementation.

And some things that need to be resolved before choosing this problem:

- Fully understand the algorithms, especially the ones in the Karbowski's paper. This problem is very mathematics-focused and requires deep understanding of LPs and relevant algorithms.
- Establishing benchmark standards and gather enough testcases, especially those with large scales.

2 Optimization of network infrastructure with parallel computing

The idea will be to combine Parallel Computing with network infrastructure components such as network coding and network protocol stacks, specifically implementint one of the 2 ideas presented in past papers (we haven't decided which to choose). The idea interests us because by introducing parallelism to those low-level components we will decrease the processing delay of the packets and further increase the utilization of internet resources and the QoS, laying a solid foundation for technologies and applications that critically relies on network speed, such as cloud services and DFSs. Some papers we consulted:

- **Sizemore, Charles. (2011) Parallel network protocol stacks using replication.** This paper presents a very interesting parallelized network processing stack protocol design for common network protocols such as TCP and UDP without the need for fine-grained locks or explicit synchronization. This will be one of our options for implementing and benchmarking.
- **Willman, Paul & Rixner, Scott & Cox, Alan. (2006) An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems.. 91-96.** The paper describes and evaluates several network stack parallization strategies, the best part of the paper is its analysis about overheads of locks, caches and schedulers and we should take that into consideration in our own implementation.
- **Nahum, Erich & Yates, David & Kurose, James & Towsley, Don. (1994) Performance Issues in Parallelized Network Protocols.** The best part of the paper is that it thoroughly defines the environement as well as the standard to benmark parallelized network algorithms and it will be very useful for benchmarking our implementation. It also gives several advice and detailed analysis on approaches commonly used in parallelizing network protocols and those advice will be very helpful for our design.
- **Shojania, Hassan & Li, Baochun. (2007) Parallelized Progressive Network Coding With Hardware Acceleration. IEEE International Workshop on Quality of Service, IWQoS. 47-55. 10.1109/IWQOS.2007.376547.** Different from the previous 2 papers, this paper actually gives a detailed description of a new approach to speedup network coding with SIMD instructions. It is theoretically

based on Gauss-Jordan elimination and is thus very mathematics-oriented. But the paper gives a very clear explanation of the algorithm as well lots of implementation details and bechmarking results, so this can also be one of our options to implement.

And as for our codebase we have (or need):

- The $x-$kernel framework at `https://www2.cs.arizona.edu/projects/xkernel/`, it will be used for simulating and benchmarking network protocols.
- The Scout operating system used for testing the replication approach, as stated in the paper.
- The Dominoes framework, which is used to adopt Scout so that network protocols can be tested.

And there are lot of challenges for this idea:

- Amount of work is huge, we are not sure if we can complete the project in one month since it involves a lot of frameworks and low-level kernel programming.
- A lot of the resources needed is not open-source or is unavailable, it may also be hard to find appropriate machines to run the simulations.
- Problem with different OS platforms. For the fourth paper, the alignment of memory locations seems to be different between Windows, Linux and MacOS, along with a lot of other differences. So we have to think of a way to deal with those differences.
- Understand the mathematical reasoning and foundations presented in the papers.

3 Parallel implementation of zlib compression & decompression

The idea will be to implement a parallelized version of zlib, a library used for data compression and decompression. The library and its underlying algorithms have broad applications such as gzip file compression format and HTTP content encoding. When it comes to compressing large amount of data, it would be helpful to utilize multiple processors to achieve speedup. One of the authors of zlib, Mark Adler, has completed a parallel implementation of gzip in C, named pigz. Though the implementation is for gzip, a large portion of its algorithms can also be applied to zlib. We have also noticed that many programming languages, like Java and Go, provide a native implementation of zlib as part of their standard libraries. We think it would be useful to make the parallelized version a third-party package that extends the standard library.

Here are several resources we have consulted:

- **Feldspar, Antaeus. (1997) An Explanation of the Deflate Algorithm. `https://zlib.net/feldspar.html`.** This article explains the Deflate algorithm used in zlib. The underlying algorithms include Huffman coding and LZ77 compression.
- The official website of pigz at `https://zlib.net/pigz`, where source code can be downloaded. We'll take the implementation as reference.

- An example of zlib implementation in another language (Go), at `https://github.com/golang/go/tree/master/src/compress`. We are planning to implement the parallel zlib in some language other than C as a third-party package.

One of the main challenges of this idea is that we need to make our implementation fully compatible with the zlib standard. In other words, the data compressed by our parallel zlib can be decompressed by the standard zlib, and vice versa. Therefore, we cannot simply divide the data into chunks, and have one thread compress each chunk. Instead, we need to find a way to exchange data, e.g. the dictionary used for compression, among threads.

To accomplish this idea, there are several steps we need to take:

- Fully understand the algorithm and implementation of zlib.
- Complete a parallel implementation of zlib using the language we choose.
- Compare the performance of our parallel zlib and the zlib in the standard library, as well as the performance of pigz.
- Demonstrate the application of parallel zlib, e.g. speedup in compressing and decompressing large amount of data and whether it can improve the performance of file transport such as SFTP.