CS267 Homework #0

Frank Austin Nothaft

1 About Me

I am a second year PhD student in the Computer Science division of the Department of Electrical Engineering and Computer Sciences. My research is at the intersection of computational biology and computer systems. Specifically, I work on `ADAM` [3], a highly scalable system for processing genomic data in a distributed or cloud setting. In a prior life, I worked in semiconductor design and computer architecture, and have done a lot of work in the multi/many-core setting. My interest in this class is to learn more about high performance computing (HPC) techniques. Specifically, I'm interested in applying a critical eye to techniques that are used in HPC environments, to see if they can be applied "elastically" in a cloud architecture. I'm also interested in *user-friendly parallelism*; or techniques that make it simple for users to implement code on parallel systems.

2 Application

I decided to take a look at the recent implementation of `Meraculous`, a *de Bruijn* graph based genome assembler using Unified Parallel C (UPC) and MPI on a Cray XC30 [1]. This application takes short read genomic data as an input and attempts to produce the highest quality "assembly" from this dataset. "Short read" implies that the input dataset consists of fragments composed of two non-overlapping "paired" reads, which are typically shorter than 300 base pairs (bp). The quality of an assembly is judged by several factors, including the N50 score which is the size of the contiguous sequence that more than 50% of the genome can be assembled into, as well as the number of *misassemblies* which are errors in the genome assembly.

Here, the authors have targeted a shared memory programming model on a supercomputer. The authors use the Berkeley UPC language [2] and MPI on "Edison", a Cray XC30 supercomputer. Edison ranks 24th on the November 2014 edition of the Top 500 list [4]. The authors parallelize various steps of the assembly process, with emphasis on the k-mer analysis and "sub-contig" generation phases. The authors use the following approaches:

- **k-mer analysis:** The authors parallelize k-mer counting by defining a deterministic mapping between k-mers $\leftrightarrow$ processors. Reads are cut into k-mers and then sent to their processor for processing via MPI's Alltoallv primitive. Erroneous k-mers are filtered out by a multi pass algorithm that uses Bloom filters. In the first pass, k-mers are kept if they are already "present" in the Bloom filter. In the second pass, the k-mers are counted.
- **Graph Construction:** The authors use a high performance distributed hash table (DHT) to construct the graph. Specifically, they apply a communication optimization where they only make a request to a remote machine when the request buffer for that machine is full. This reduces the number of messages sent, and improves overhead.

- **Graph Traversal:** Each processor chooses a random k-mer from the distributed hash table, and starts walking the graph from there by looking into the DHT. This runs until they cannot find a new k-mer to continue the sub-contig. Coordination is needed if two processors pick k-mers that both belong to the same sub-contig. In this case, the two processors synchronize via a finite state machine described in the paper.

The authors achieve a near linear speedup on up to 15,360 cores, which is impressive. As they point out, `Meraculous` is a highly accurate assembler, so these performance improvements are important. While this work is good, I don't particularly like the performance evaluation stage. Specifically, since they have changed the input format (which alters the I/O characteristics of the entire pipeline), many of the comparisons seem to be slightly apples-to-oranges. Specifically, since the k-mer preprocessing stage takes the most time of any stage in the pipeline, and the I/O is a contributor to this stage, it seems like it would be impossible to fairly evaluate each pipeline without compensating for the I/O differences. Similarly, the impact of this paper is a bit overstated. While it is nice to think that accelerating the `Meraculous` assembler is useful for cancer analytics, additional improvements would need to be made to support variant analysis [5].

References

[1] Georganas, E., Buluç, A., Chapman, J., Oliker, L., Rokhsar, D., and Yelick, K. Parallel de bruijn graph construction and traversal for de novo genome assembly. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for* (2014), IEEE, pp. 437–448.

[2] Husbands, P., Iancu, C., and Yelick, K. A performance analysis of the berkeley UPC compiler. In *Proceedings of the 17th annual international conference on Supercomputing* (2003), ACM, pp. 63–73.

[3] Massie, M., Nothaft, F., Hartl, C., Kozanitis, C., Schumacher, A., Joseph, A. D., and Patterson, D. A. Adam: Genomics formats and processing patterns for cloud scale computing. Tech. rep., Technical Report UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[4] Top 500. Edison—Cray XC30. `http://www.top500.org/system/178443`.

[5] Weisenfeld, N. I., Yin, S., Sharpe, T., Lau, B., Hegarty, R., Holmes, L., Sogoloff, B., Tabbaa, D., Williams, L., Russ, C., et al. Comprehensive variation discovery in single human genomes. *Nature genetics* (2014).