Homework 1

School of Data and Computer Science
17341015 Hongzheng Chen

Problem 1. Write an essay describing a research problem in your major that would benefit from the use of parallel computing. Provide a rough outline of how parallelism would be used. Would you use task- or data-parallelism?

Answer.
Since my major is Big Data and Artificial Intelligence, in the following, I will briefly introduce **graph computing** and describe the parallelism in it.

Nowadays, graph-based applications are ubiquitous, such as internetwork[1], social network[2], recommendation system[3], scientific computing[4], and machine learning[5]. The graphs in these applications are so large that billions or even trillions of vertices are considered[6], which makes efficient processing challenging.

To handle with this kind of big data, parallelism must be considered. Parallelism may greatly accelerate the applications running on the graphs and can achieves several magnitudes of speedup.

Expert engineers may easily accelerate one graph application by adding some parallelism patterns. However, as new applications emerge everyday, it is impossible for experts to tune performance for all applications and related data. Thus, graph processing system are built to make graph programming much easier. These systems hinder the underlayer parallel infrastructure, but provide high-level abstractions for programmers.

Shared-memory systems like Ligra[7], GraphMat[8] and Galois[9], consider the parallelism in a single machine, while distributed systems like Pregel[10], Giraph[11] and GraphLab[12], tackle graph problems in distributed settings.

The core of these systems are the push-based engine and the pull-based engine shown below.

Push-based engine

```
parallel_for (vSrc : vertices) {
  if (!vSrc in frontier)
    continue;
  for (vDst : vSrc.outngh) {
    if (cond(vDst))
      atomicUpdate(vDst);
  }
}
```

Pull-based engine

```
parallel_for (vDst : vertices) {
  if (!cond(vDst))
    continue;
  for (vSrc : vDst.inngh) {
    if (vSrc in frontier)
      update(vDst);
  }
}
```

This is the fine-grained task parallelism in vertex level. Users only need to define the `update` and `cond` function, and the graph applications will automatically execute in parallel.

Above is the overview of graph computing. Recently, I also conduct a project on graph computing, so I will briefly introduce my work here.

As you can see, many of graph frameworks and systems have been built recent years, but they are all for single graph task[1]. I wonder if multiple tasks are running on the same graph, what will happen? Concisely, how to process these concurrent tasks efficiently?

Definitely, parallelism is also needed to be considered, and this is the most coarse-grained task parallelism. We propose some techniques to enable this parallelism, and the brief contributions of our work are shown below:

- We use graph kernel fusion to reduce redundant data access in concurrent graph tasks.
- We leverage AVX[2] instruction set to increase compute to global memory access (CGMA) ratio.
- We propose a semi-synchronous scheduler to maximumly leverage memory bandwidth.
- Finally, we propose a Fetcher-Scheduler-Executor-Profiler (FSEP) model and combine them altogether into a system named Krill.

This work, named *Krill: Processing Concurrent Graph Task with Kernel Fusion in a Semi-Synchronous Way*, will be submitted to *ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'19)*, and I will be the first author.

A supplementary `pdf` file is attached in the document, which is the outline of my work. After the work is finished (maybe after Apr. 10), I will resubmit it to the class system.

Problem 2. Assume that you have the following mix of instructions with average CPIs:

	% of Mix	Average CPI
ALU	47%	6.7
Load	19%	7.9
Branch	20%	5.0
Store	14%	7.1

The clock rate for this machine is 1GHz.
You want to improve the performance of this machine, and are considering redesigning your

[1]A task or a job can be any graph applications like BFS, SSSP, PageRank, etc.
[2]AVX, Advanced Vector Extension, is the hardware support in Intel's CPU for SIMD (Single Instruction Multiple Data) computing. And SIMD can be viewed as data parallelism.

multiplier to reduce the average CPI of multiply instructions. (*Digress – why do multiplies take longer than adds?*)
If you make this change, the CPI of multiply instructions would drop 6 (from 8). The percentage of ALU instructions that are multiply instructions is 23%. How much will performance improve by?

Answer.

- Actually, one multiplication is made up of several additions. No matter which method is used, roughly $O(n)$ times of additions are needed, where n is the number of digits in the multiplier or multiplicand.
- We abbreviate multiplication as "mul" and other ALU functions as "add".
  The original CPI is

$$47\% \times 6.7 + 19\% \times 7.9 + 20\% \times 5.0 + 14\% \times 7.1 = 6.644\,.$$

  Since "The percentage of ALU instructions that are multiply instructions is 23%" is ambiguous, I will discuss the two understandings in the following.

  – In ALU instructions, the percentage of multiply instructions is 23%.
    23% of mul means 1-23%=77% of add in ALU.
    The original CPI of mul is 8. Thus,

$$23\% CPI_{mul} + 77\% CPI_{add} = CPI_{ALU}\,.$$

    We obtain $CPI_{add} = 6.312$. (Well, the new add CPI is bigger than mul.)
    Then, the new CPI is

$$(23\% \times 6 + 77\% \times 6.312) \times 47\% + 19\% \times 7.9 + 20\% \times 5.0 + 14\% \times 7.1 = 6.428\,.$$

    The performance is improved by

$$\left(\frac{6.644}{6.428} - 1\right) \times 100\% = 3.36\%\,.$$

  – In all instructions, the percentage of multiply instructions is 23%.
    23% of mul means 47%-23%=24% of add.
    The original CPI of mul is 8. Thus,

$$\frac{23\%}{47\%} CPI_{mul} + \frac{24\%}{47\%} CPI_{add} = CPI_{ALU}\,.$$

    We obtain $CPI_{add} = 5.45$.

Then, the new CPI is

$$23\% \times 6 + 24\% \times 5.45 + 19\% \times 7.9 + 20\% \times 5.0 + 14\% \times 7.1 = 6.183\,.$$

Then, the performance is improved by

$$\left(\frac{6.644}{6.183} - 1\right) \times 100\% = 7.46\%\,.$$

References

[1] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Stanford Tech Report, 1999

[2] Kwak Haewoon, Lee Changhyun, Park Hosung, and Moon Sue, *What is Twitter, a Social Network or a News Media?*, Proceedings of the 19th International Conference on World Wide Web, 2010

[3] Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl, *Item-based Collaborative Filtering Recommendation Algorithms*, Proceedings of the 10th international conference on World Wide Web (WWW), 2001

[4] Andrea Apolloni, Chiara Poletto, Jose J. Ramasco, Pablo Jensen, Vittoria Colizza, *Metapopulation epidemic models with heterogeneous mixing and travel behaviour*, BMC Theoretical Biology and Medical Modelling, 2014

[5] Peter W.Battaglia et al., *Relational inductive biases, deep learning, and graph networks*, Relational inductive biases, deep learning, and graph networks, arXiv:1806.01261

[6] Graph 500, `https://graph500.org/`

[7] Shun Julian, and Blelloch Guy E., *Ligra: A Lightweight Graph Processing Framework for Shared Memory*, Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 2013

[8] Sundaram Narayanan, Satish Nadathur, et al., *GraphMat: High Performance Graph Analytics Made Productive*, Proceedings of Very Large Data Bases (VLDB), 2015

[9] Nguyen Donald, Lenharth Andrew, and Pingali Keshav, *A lightweight infrastructure for graph analytics*, Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), 2013

[10] Malewicz Grzegorz, Austern Matthew H., et al., *Pregel: A System for Large-scale Graph Processing*, Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 2010

[11] Ching Avery, Edunov Sergey, et al., *One Trillion Edges: Graph Processing at Facebook-scale*, Proceedings of Very Large Data Bases (VLDB), 2015

[12] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin and Joseph M. Hellerstein , *Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud*, Proceedings of Very Large Data Bases (VLDB), 2012