Asynchronous Optimization Algorithms with GPUs

Shuyao Li
Supervisor: Professor Ernest K. Ryu

Introduction

As the number of computational cores in CPUs and GPUs increases, so does the cost of synchronization in parallel computing, and the need for asynchronous algorithms grows. This project studies asynchronous optimizations algorithms on GPUs.

We extended the optimization algorithms stochastic coordinate descent (SCD) [1] and Finito [2] to the asynchronous setup and implemented them on CPUs and GPUs. We observed that the asynchronous GPU algorithms were 20–30x faster than the asynchronous CPU algorithms, which were 3.6–4x faster than the synchronous parallel CPU algorithms.

Graphic Processing Units (GPU)

GPUs were traditionally used for graphics and games but in the past decade, people have started to use them for large-scale parallel computing.

GPUs have many more cores than CPUs do. However, the cores individually run slower, have small local memory, and are limited in their capability to communicate and coordinate. Therefore, reducing the use of local memory and the amount of communication and coordination is essential for designing an asynchronous algorithm efficient on GPUs.

Figure 1: Nvidia GPU Telsa M60

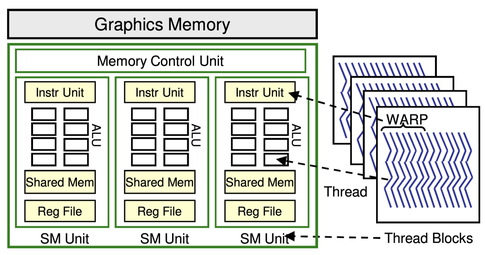

Figure 2: GPU architecture and thread execution model (Hong, 2011)

Asynchronous SCD

Minimize$_{x\in\mathbb{R}^d}$ $f(x)$, where $f:\mathbb{R}^d\to\mathbb{R}$ is convex:

$$x_{i(k)} \leftarrow x_{i(k)} - \alpha\nabla_{i(k)}f(x),$$

where $i(k)\in\{1,...,d\}$ is drawn randomly, $\nabla_i f(x) = \frac{\partial f}{\partial x_i}(x)$.

while *not converged, every agent* **do**
  // Synchronize
  select $i(k)\in\{1,...,d\}$ randomly
  $w \leftarrow \alpha\nabla_{i(k)}f(x)$
  // Synchronize
  $x_{i(k)} \leftarrow x_{i(k)} - w$
end

Asynchronous Finito

Minimize$_{x\in\mathbb{R}^d}$ $\sum_{i=1}^n f_i(x)$, where $\forall i$ $f_i:\mathbb{R}^d\to\mathbb{R}$ is convex:
Let $i(k)\in\{1,...,n\}$ be drawn randomly, z_1, ..., $z_n\in\mathbb{R}^d$,

$$x \leftarrow \frac{1}{n}\sum_{i=1}^n z_i$$
$$z_{i(k)} \leftarrow x - \alpha\nabla f_{i(k)}(x)$$

while *not converged, every agent* **do**
  // Synchronize
  select $i(k)\in\{1,...,n\}$ randomly
  $w \leftarrow x - z_i - \alpha\nabla f_{i(k)}(x)$
  // Synchronize
  $z_i \leftarrow z_i + w$
  update x
end

Looping through coordinates can be merged to reduce local memory access after removing synchronization barrier

Experiment Details and Results

The experiments were based on binary regularized logistic regression for n data-points of dim dimensions, with f as the cost function and f_i as the cost contributed by each data-point, where $i\in\{1...n\}$. Cat and dog classes from CIFAR-10 ($n=10000, dim=3072$) are used to validate the convergence behavior in a practical setting. Synthetic datasets $(n, dim)\in\{(8192,8192),(2048,16384),(16384,2048)\}$ give us flexibility to test our algorithms under different combinations of n and dim.

Below are convergence plots for both synchronous and asynchronous algorithms implemented on GPUs. For each curve, related hyperparameters are chosen to minimize the time it takes to reach 10^{-6} precision.

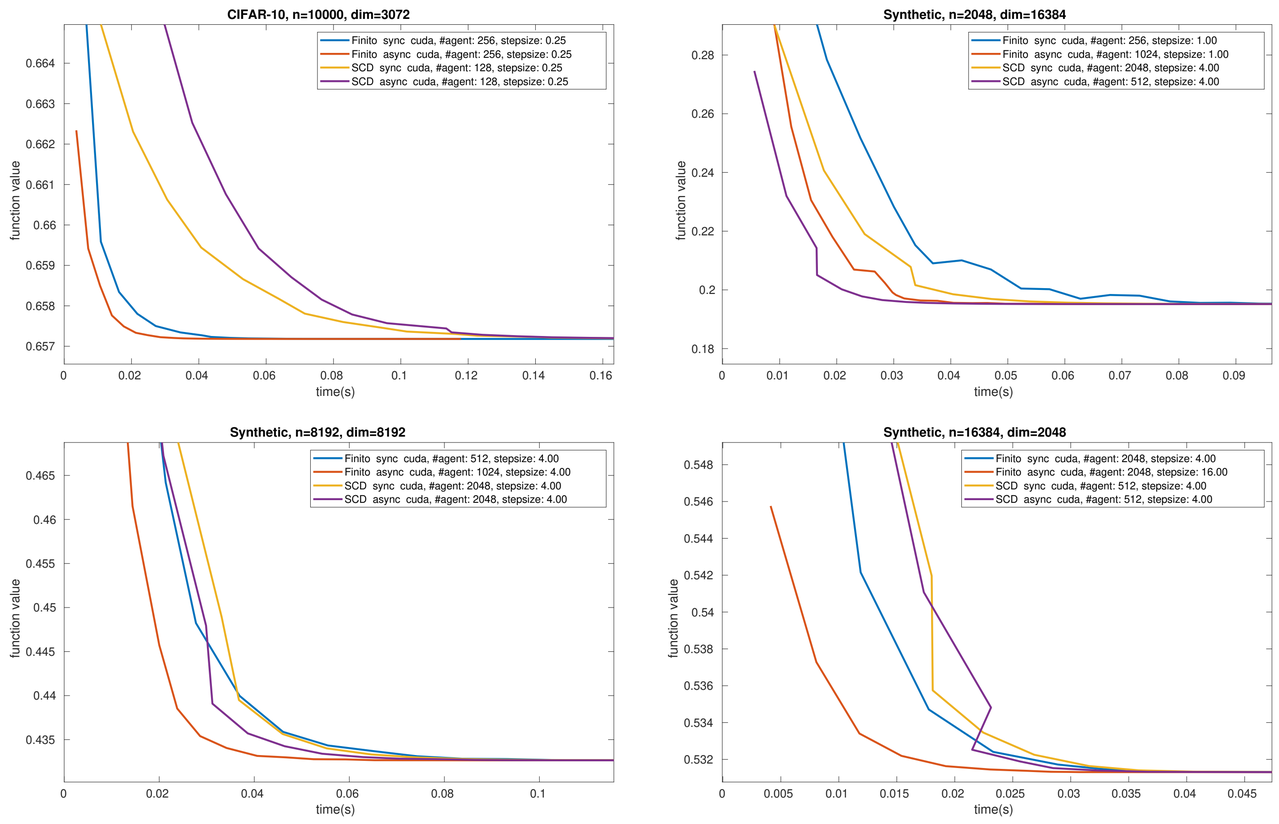

Figure 3: Convergence plots of GPU implementations

- Key details:
  - One insight that gives GPU algorithms much speed-up is to couple a warp (32 threads) into a single agent. (Reason: memory coalescing)
  - On single GPU implementation, it's critical for asynchronous Finito to merge loops as shown in the left pannel. (Reason: limited local memory)
- Primary results:
  - 1.7x speed-up of asynchronous algorithms over synchronous algorithms in GPUs.
  - For generality, similar results are obtained in both SCD and Finito algorithms, with SCD favoring $n\ll dim$ and Finito favoring $n\gg dim$.

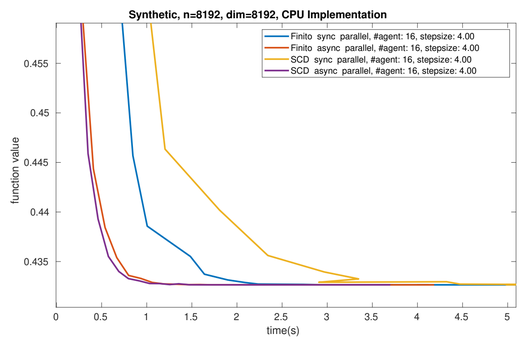

Figure 4: Reference convergence plot of CPU algorithms

- Other remarks:
  - GPU algorithms typically run 20-30x faster than their CPU counterparts
  - Since algorithms in GPUs are parallelized by much more agents than those in CPUs, it poses larger challenge to ensure convergence.

Novel Observations and Future Work

Most speed-up of asynchronous algorithms in GPUs comes from the fact that asynchronous algorithms require less memory access than their synchronous counterparts. We will further investigate multi-GPU settings to see whether removing synchronization barrier can in itself bring speed-ups for asynchronous algorithms in multi-GPU.

Asynchronous algorithms seem to be more robust in that they manage to converge in some parameter combinations under which their synchronous counterparts diverge. Whether and how this holds generally is useful to know to benefit laborious parameter tuning.

When the number of agents increases, asynchronous algorithms seem to enjoy a more rapid speed-up than their synchronous counterparts. We would like to validate this result in more general setups, and explore the theory behind it if possible.

References

[1] Liu et al. "An asynchronous parallel stochastic coordinate descent algorithm" JMLR (2015).

[2] Defazio et al. "Finito: A faster, permutable incremental gradient method for big data problems" ICML (2014).

Acknowledgments

Special thanks to Professor Ernest Ryu for supervison, support, and generous help.

Big thanks to CSST Program for scholarship.