Asynchronous Optimization Algorithms with GPUs

Shuyao Li
Supervisor: Professor Ernest Ryu

CSST Peer Seminar
Sept. 4, 2019

Optimization with coordinate descent

Problem:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}} \;\; f(x).$$

Solve with stochastic coordinate descent (SCD):

Write $\nabla_i f(x) = \frac{\partial f}{\partial x_i}(x)$.

---

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

Optimization with Finito

Problem:

$$\underset{x \in \mathbb{R}^d}{\text{minimize}}\ f(x), \text{ where } f = \frac{1}{n}\sum_{i=1}^{n} f_i$$

Solve with Finito:

Let $z_1, ..., z_n \in \mathbb{R}^d, x = \frac{1}{n}\sum_{i=1}^{n} z_i$.

---

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

$z_i \leftarrow z_i + w$

$x \leftarrow x + \frac{1}{n} w$

end

Asynchronous optimization algorithms

- Introduced in 2011 as *Hogwild!* (Async SGD[1])
- Explored in *e.g. AsySCD*[2], *AsySPCD*[3], *ARock*[4], *ASAGA*[5], *etc.*
- Those paper experiment only with CPUs
  - ~10 cores in multi-core CPUs
  - Communication through shared memory — simpler memory model

---

[1] Niu, Recht, Ré, and Wright; NIPS 2011
[2] Liu, Wright, Ré, Bittorf, and Sridhar; JMLR 2013
[3] Liu, and Wright; SIOPT 2015
[4] Peng, Xu, Yan, and Yin; SISC 2016
[5] Leblond, Pedregosa, and Lacoste-Julien; AISTATS 2017

Graphic Processing Units (GPU)

- ~1000 slower processors (compared to CPUs)
- Usage
  - Originally designed for generating output images to a display device
    e.g. graphics and gaming
  - Has been used recently for medium & large-scale computation
    e.g. deep learning

- Difficulties of GPU computing in optimization:
  - Limited capability to communicate and coordinate (*e.g.* synchronize)
    Not all parallel things are GPU parallelizable
  - Small local memory

Parallelism in scientific computing

- Past:
  - Thought at a high level in an abstract way (*i.e.* simpler model)
  - Implementation: multi-core CPU / supercomputer
- Present: GPU computing becomes popular with deep learning boom
  e.g. GPUs are common platform for training neural network
- But GPUs are not widely used in optimization
- My research question:
  Can asynchronous optimization algorithms implemented with GPUs provide substantial speed-up?

Experiments

- Experiment Details:
  - Cost function: Binary regularized logistic regression
  - Datasets: Synthetic datasets and CIFAR-10 (cat & dog classes)

- Primary results:
  - GPUs give 10–30x speed-up over CPU counterparts
  - Asynchronous GPU algorithms give 1.7x speed-up over synchronous GPU algorithms

---

CPU models: Intel(R) Core(TM) i9-9940X CPU @ 3.30GHz (28 virtual cores)

GPU models: GeForce RTX 2080 Ti (4352 CUDA Cores)

Convergence plot

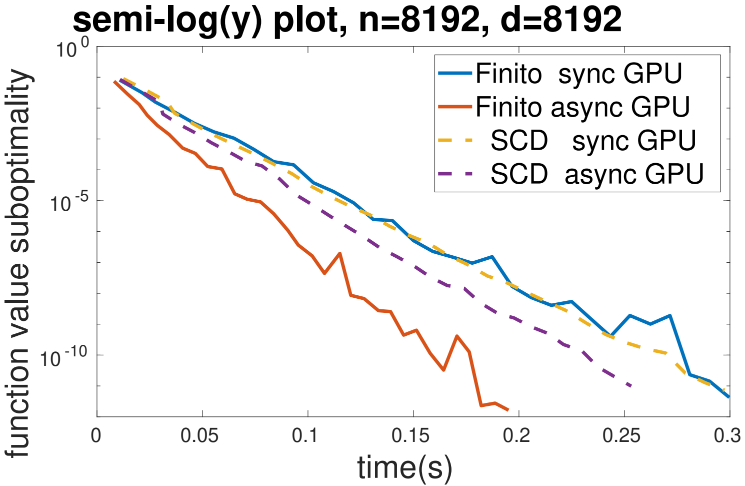

Figure: Convergence plot for SCD/ Finito implemented with GPU

Insights

- Couple a warp (32 threads) into a single agent
- Device memory access in GPU is expensive — try to reduce it
  e.g. Recall in Finito, $x, z_i \in \mathbb{R}^d$,

  while *not converged, every agent* **do**

      // Synchronize

      select $i(k) \in \{1, \ldots, n\}$ randomly

  $$\left.\begin{array}{l} w \leftarrow x - z_i - \alpha \nabla f_{i(k)}(x) \\ \text{// Synchronize} \\ z_i \leftarrow z_i + w \\ x \leftarrow x + \frac{1}{n} w \end{array}\right\} \text{Loops can be combined after removing synchronization barrier}$$

  end

Novel observations and future work

- Much speed-up of asynchronous GPU algorithms comes from reducing memory access
  $\rightarrow$ Can **multi-GPU** bring further speed-ups?
- Sometimes synchronous algorithms catch up with asynchronous counterparts
  $\rightarrow$ Find out when this happens and explore the theory if possible
- Robustness of asynchronous algorithms —
  Sometimes they converge but their synchronous counterparts diverge
  $\rightarrow$ How this holds generally is useful for laborious parameter tuning.
- Trade-off for increasing number of agents —
  Faster iteration & worse convergence behavior
  $\rightarrow$ Explore

Q & A