Outline

- Multi-core v.s. multi-processor
- Parallel Gradient Descent
- Parallel Stochastic Gradient
- Parallel Coordinate Descent
Parallel Programming

Parallel algorithms can be different for the following two cases:

- **Shared Memory Model (Multiple cores/multiple processors)**
  - Independent L1 cache
  - Shared/independent L2 cache
  - Shared memory

- **Distributed Memory Model (Multi computers)**
  - Multiple computers
Shared Memory Model (Multiple cores)

- Shared memory model: each CPU can access the same memory space
- Programming tools:
  - C/C++: openMP, C++ thread, pthread, intel TBB, ...
  - Python: thread, ...
  - Matlab: parfor, ...

![Diagram showing shared memory model](image)
Parallel for loop in OpenMP (*)

```c
#pragma omp parallel for private(i)
for (i=0; i<w_size; i++)
g[i] = w[i] + g[i];
```
Shared Memory Model (*)
Shared Memory Model

- Two types of shared memory model:
  1. Uniform Memory Access (UMA)
  2. Non-Uniform Memory Access (NUMA)
Distributed Memory Model

- Programming tools: MPI, Hadoop, Spark, …

(Figure from http://web.sfc.keio.ac.jp/ rdv/keio/sfc/teaching/architecture/computer-architecture-2013/lec09-smp.html)
Parallel Gradient Descent
Parallel Gradient Descent

- Gradient descent:
  \[ \mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x}) \]

- Gradient computation is usually embarrassingly parallel

- Example: empirical risk minimization can be written as
  \[
  \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w})
  \]

- Partition the dataset into \( k \) subsets \( S_1, \ldots, S_k \)
- Each machine or CPU computes \( \sum_{i \in S_i} \nabla f_i(\mathbf{w}) \)
- Aggregated local gradients to get the global gradient (communication)

  \[
  \nabla f(\mathbf{w}) = \frac{1}{n} \left( \sum_{i \in S_1} \nabla f_i(\mathbf{w}) + \cdots + \sum_{i \in S_k} \nabla f_i(\mathbf{w}) \right)
  \]
Parallel Stochastic Gradient
Stochastic Gradient (SG):

For $t = 1, 2, \ldots i$

Randomly pick an index $i$

$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta^t \nabla f_i(\mathbf{w}^t)$

Computation of $\nabla f_i(\mathbf{w}^t)$ only depends on the $i$-th sample—usually cannot be parallelized.

Parallelizing SG is a hard research problem.
Mini-batch SG

- Mini-batch SG with batch size $b$:

  For $t = 1, 2, \ldots$
  
  Randomly pick a subset $S \subseteq \{1, \ldots, n\}$ with size $b$
  
  $$w^{t+1} \leftarrow w^t - \eta^t \frac{1}{b} \sum_{i \in S} \nabla f_i(w^t)$$

- Equivalent to gradient descent when $b = n$
- Equivalent to stochastic gradient when $b = 1$
Mini-batch SG

- Mini-batch SG with batch size $b$:

  For $t = 1, 2, \ldots$
  
  Randomly pick a subset $S \subseteq \{1, \ldots, n\}$ with size $b$
  
  $$w^{t+1} \leftarrow w^t - \eta^t \frac{1}{b} \sum_{i \in S} \nabla f_i(w^t)$$

- Equivalent to gradient descent when $b = n$
- Equivalent to stochastic gradient when $b = 1$
- Parallelization with $k$ processors:
  
  Let $S = S_1 \cup S_2 \cup \cdots \cup S_k$
  
  $$\sum_{i \in S} \nabla f_i(w^t) = \sum_{i \in S_1} \nabla f_i(w^t) + \sum_{i \in S_2} \nabla f_i(w^t) + \cdots + \sum_{i \in S_k} f_i(w^t)$$
  
  can be computed in parallel

- Other versions: divide-and-average (Mann et al., 2009; Zinkevich et al., 2010)
Mini-batch SG

- How to choose batch size $b$?

  - smaller $b$ (batch size) larger
  - faster convergence slower
  - more Communication time less
Can we avoid wasting communication time?

Use **non-blocking** network IO:
Keep computing updates while aggregating the gradient

See (Dekel et al., “Optimal Distributed Online Prediction Using Mini-Batches”. In JMLR 2012)
Asynchronous Stochastic Gradient

- Synchronized algorithms: all the machine has to stop and synchronize at some points
  ⇒ longer waiting time
Asynchronous Stochastic Gradient (shared memory)

- The original SG:

For $t = 1, 2, \ldots$

Randomly pick an index $i$

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f_i(\mathbf{w})$
Asynchronous Stochastic Gradient (shared memory)

The asynchronous parallel SG:

Each thread repeatedly performs the following updates:
For \( t = 1, 2, \ldots \)

Randomly pick an index \( i \)

\[
\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f_i(\mathbf{w})
\]
Asynchronous Stochastic Gradient (shared memory)

- The asynchronous parallel SG:

  Each thread repeatedly performs the following updates:
  
  For $t = 1, 2, \ldots$
  
  Randomly pick an index $i$
  
  $w \leftarrow w - \eta \nabla f_i(w)$

- Main trick: in shared memory systems, every threads can access the same parameter $w$

- First discussed in (Langford et al., “Slow learners are fast”. In NIPS 2009)

For convex function, converges to the global optimum under certain conditions:
(1) bounded delay, (2) small confliction rate

For non-convex function, the convergence property is proved in:
“Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization”. In NIPS 2015
Asynchronous Stochastic Gradient (distributed memory)

- Use a parameter server to update the parameters

\[ w' = w - \eta \Delta w \]

See Dean et al., “Large Scale Distributed Deep Networks”, in NIPS 2012
Parallel Coordinate Descent
(Stochastic) Coordinate Descent (CD):

For $t = 1, 2, \ldots$

Randomly pick an index $i$

$$w_{i}^{t+1} \leftarrow w_{i}^{t} - \left( \arg\min_{\delta} f(w^{t} - \delta e_{i}) \right)$$

A simplified version: each coordinate is updated by a gradient step

For $t = 1, 2, \ldots$

Randomly pick an index $i$

$$w_{i}^{t+1} \leftarrow w_{i}^{t} - \eta \nabla_{i} f(w^{t})$$

How to parallelize it?
Synchronized Parallel Coordinate Descent

- Synchronized Parallel Coordinate Descent:

For $t = 1, 2, \ldots$

Randomly pick a subset $S \subset \{1, \ldots, n\}$ with size $b$

$w_i^{t+1} \leftarrow w_i^t - \eta \nabla_i f(w^t)$ for all $i \in S$
Synchronized Parallel Coordinate Descent:

For $t = 1, 2, \ldots$
- Randomly pick a subset $S \subset \{1, \ldots, n\}$ with size $b$
- $w_i^{t+1} \leftarrow w_i^t - \eta \nabla_i f(w^t)$ for all $i \in S$

Parallelization: let $S = S_1 \cup S_2 \cup \cdots \cup S_k$,
- $j$-th machine updates the variables in $S_j$

Will it converge?
- Yes, if $\eta$ is small enough

First discussed in Bradley et al., “Parallel coordinate descent for $\ell_1$-regularized loss minimization”. In ICML 2011
Asynchronous Parallel Coordinate Descent

- The asynchronous parallel coordinate descent:

  Each thread repeatedly performs the following updates:
  For \( t = 1, 2, \ldots \)
  
  Randomly pick an index \( i \)
  
  \[ w \leftarrow w - \eta \nabla f_i(w) \]
Asynchronous Parallel Coordinate Descent

The asynchronous parallel coordinate descent:

Each thread repeatedly performs the following updates:
For $t = 1, 2, \ldots$
    Randomly pick an index $i$
    \[ \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f_i(\mathbf{w}) \]

Main trick: in shared memory systems, every thread can access the same parameter $\mathbf{w}$

First implemented in (Bradley et al., “Parallel coordinate descent for $\ell_1$-regularized loss minimization”. In ICML 2011)

Coming up

- Next class: Support Vector Machines (SVM)

Questions?