# ECS289: Scalable Machine Learning

Cho-Jui Hsieh
UC Davis

Oct 4, 2016

# Outline

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

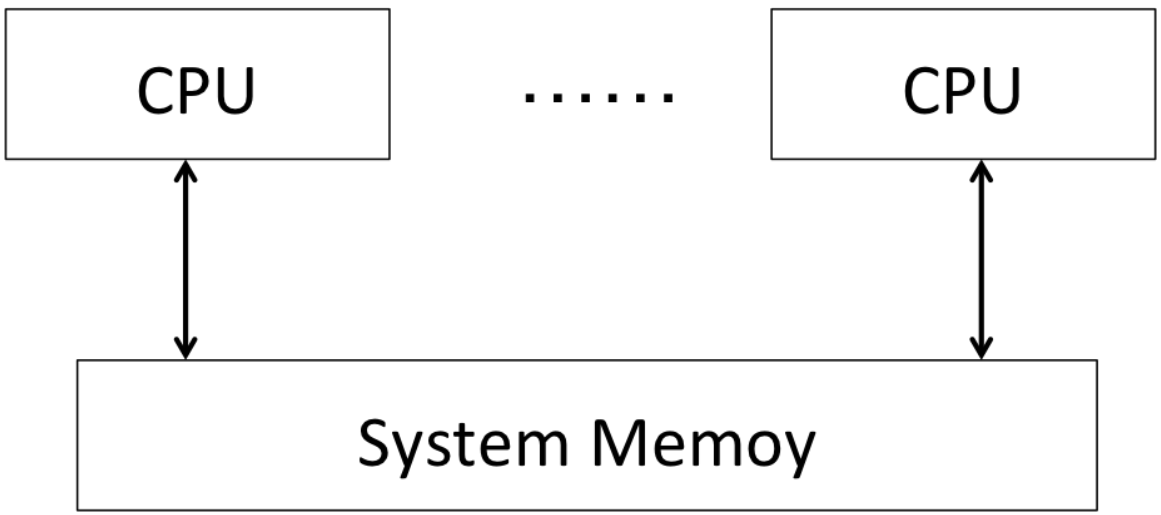
# Parallel Programming

Parallel algorithms can be different in the following two cases:

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

# Shared Memory Model (Multiple cores)

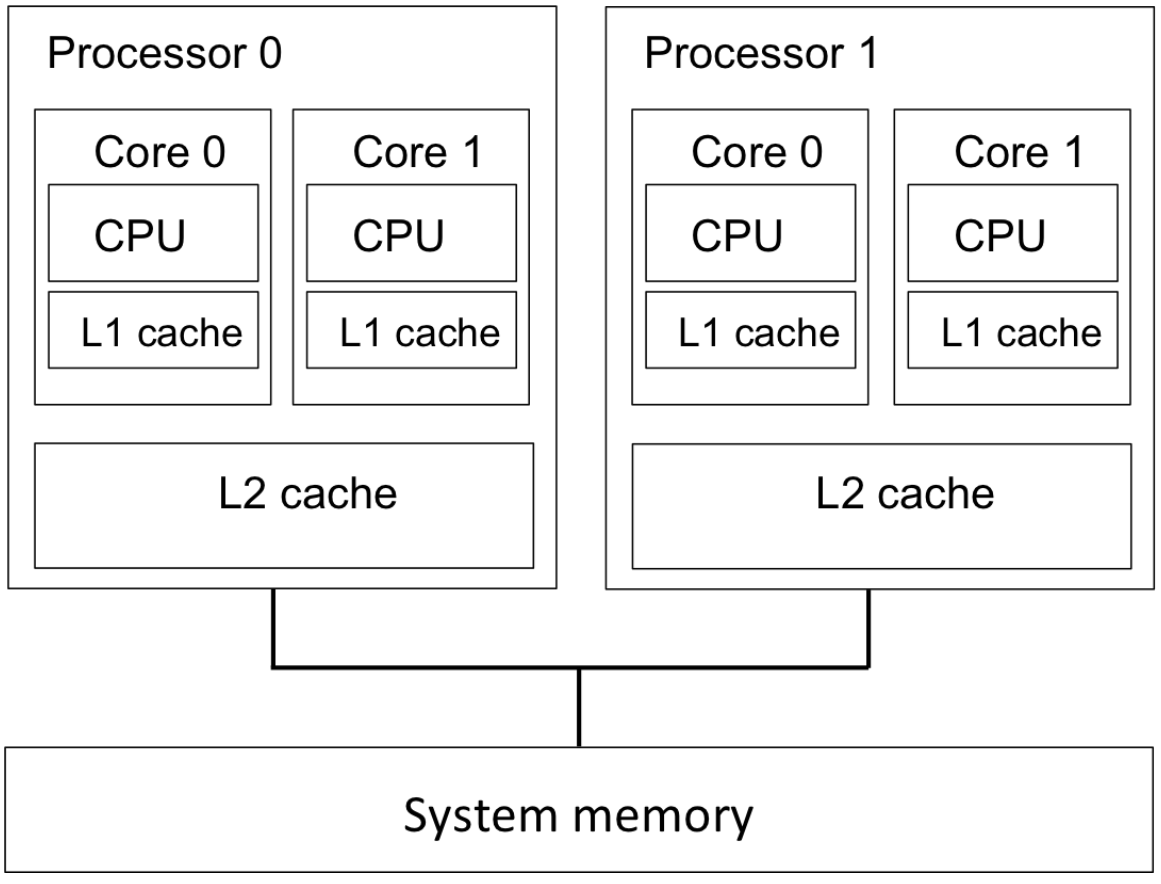
- Shared memory model: each CPU can access the same memory space
- Programming tools:
  - C/C++: openMP, C++ thread, pthread, intel TBB, ...
  - Python: thread, ...
  - Matlab: parfor, ...



# Parallel for loop in OpenMP

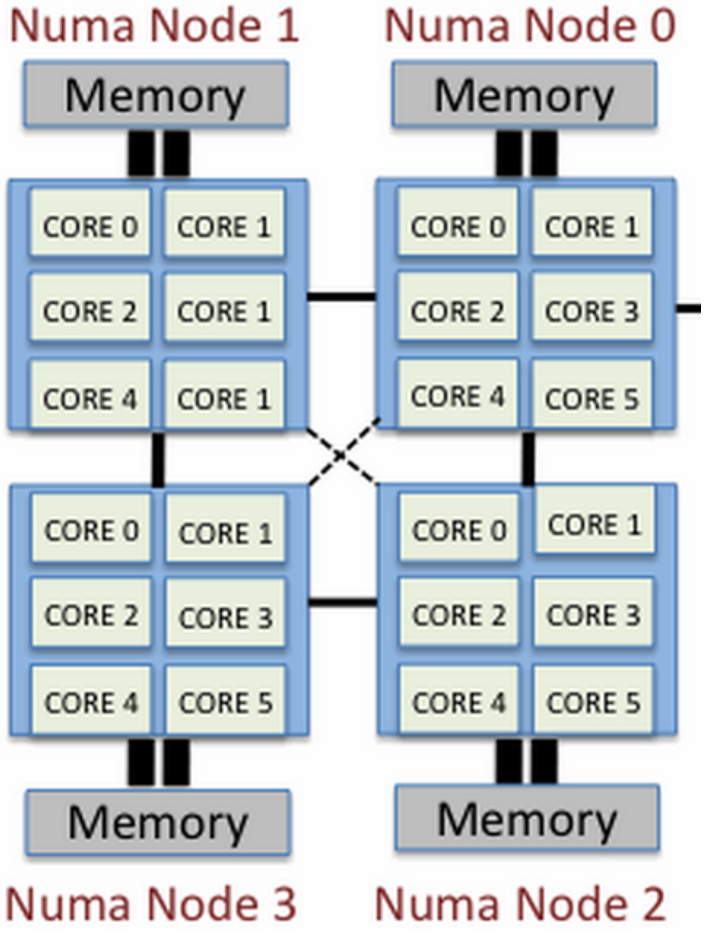
```
#pragma omp parallel for private(i)
    for(i=0;i<w_size;i++)
        g[i] = w[i] + g[i];
```

# Shared Memory Model



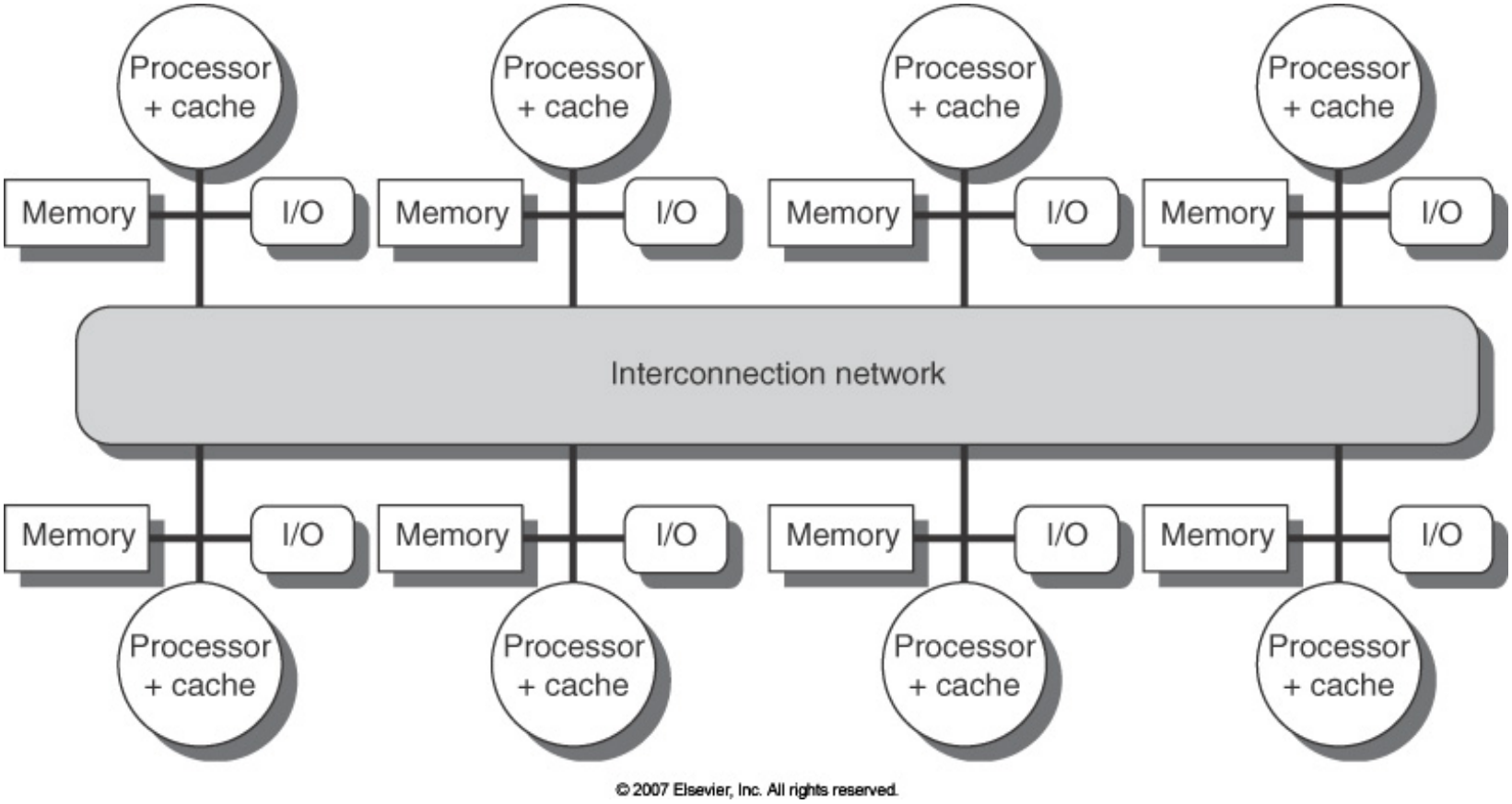
# Shared Memory Model

- 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:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$$

- Gradient computation is usually embarrassingly parallel
- Example: empirical risk minimization can be written as

$$\operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{w})$$

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

$$\nabla f(\boldsymbol{w}) = \frac{1}{n} \left( \sum_{i \in S_1} \nabla f_i(\boldsymbol{w}) + \cdots + \sum_{i \in S_k} \nabla f_i(\boldsymbol{w}) \right)$$

# Parallel Stochastic Gradient

# Parallel Stochastic Gradient in Shared Memory Systems

- Stochastic Gradient (SG):

For $t = 1, 2, \ldots$ i
    Randomly pick an index $i$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta^t \nabla f_i(\boldsymbol{w}^t)$

- Computation of $\nabla f_i(\boldsymbol{w}^t)$ 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 $b$

  $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta^t \frac{1}{b} \sum_{i \in S} \nabla f_i(\boldsymbol{w}^t)$

- 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 $b$
  $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta^t \frac{1}{b} \sum_{i \in S} \nabla f_i(\boldsymbol{w}^t)$

- Equivalent to gradient descent when $b = n$
- Equivalent to stochastic gradient when $b = 1$
- Parallelization with $k$ processors:

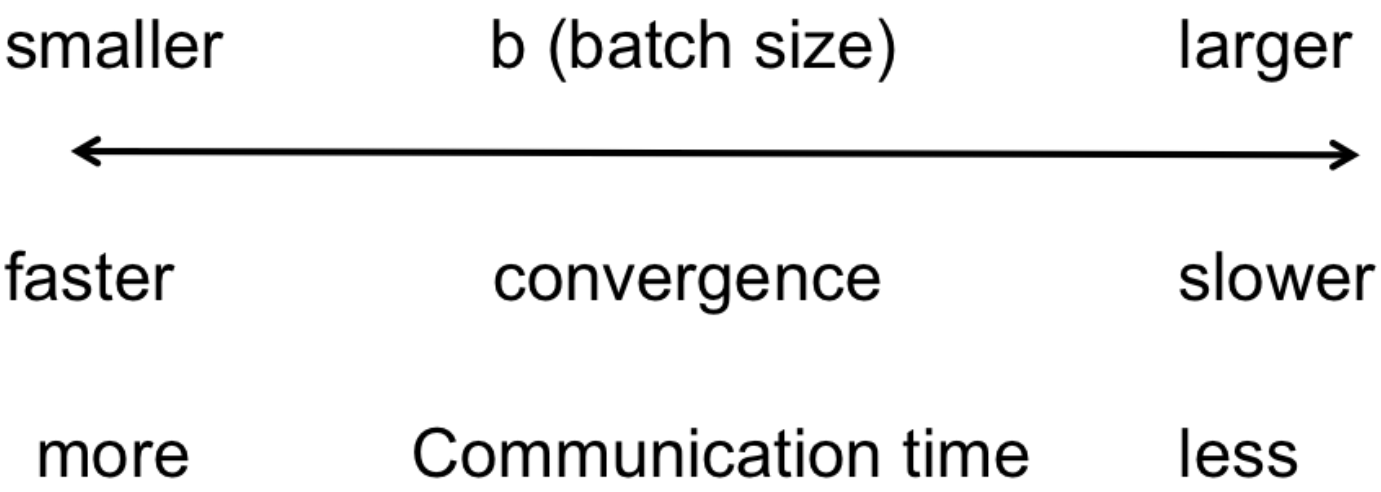
  Let $S = S_1 \cup S_2 \cup \cdots \cup S_k$

  $\sum_{i \in S} \nabla f_i(\boldsymbol{w}^t) = \sum_{i \in S_1} \nabla f_i(\boldsymbol{w}^t) + \sum_{i \in S_2} \nabla f_i(\boldsymbol{w}^t) + \cdots + \sum_{i \in S_k} f_i(\boldsymbol{w}^t)$

  can be computed in parallel
- Other versions: divide-and-average (Mann et al., 2009; Zinkevich et al., 2010)

# Mini-batch SG

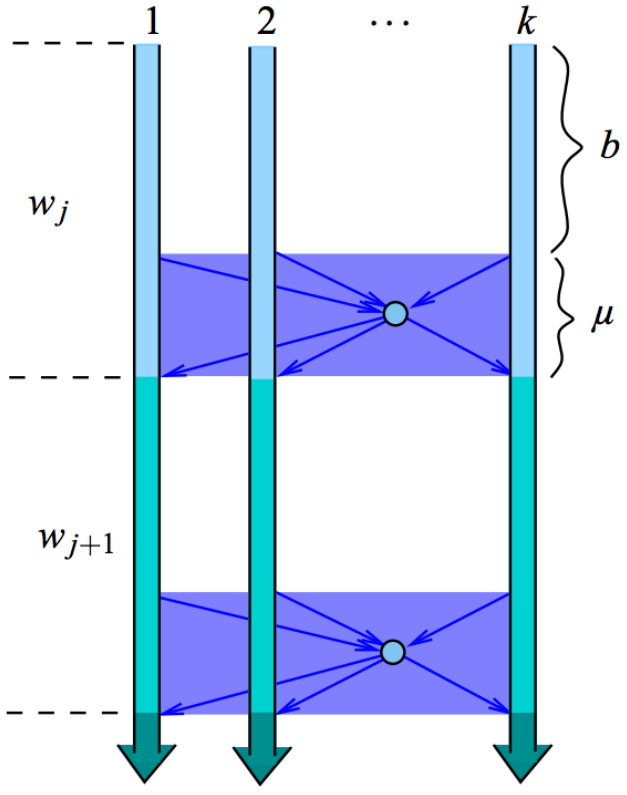
- How to choose batch size $b$?



# Mini-batch SG on distributed systems

- Can we avoid wasting communication time?
- Use non-blocking network IO:

  Keep computing updates while aggregating the gradient

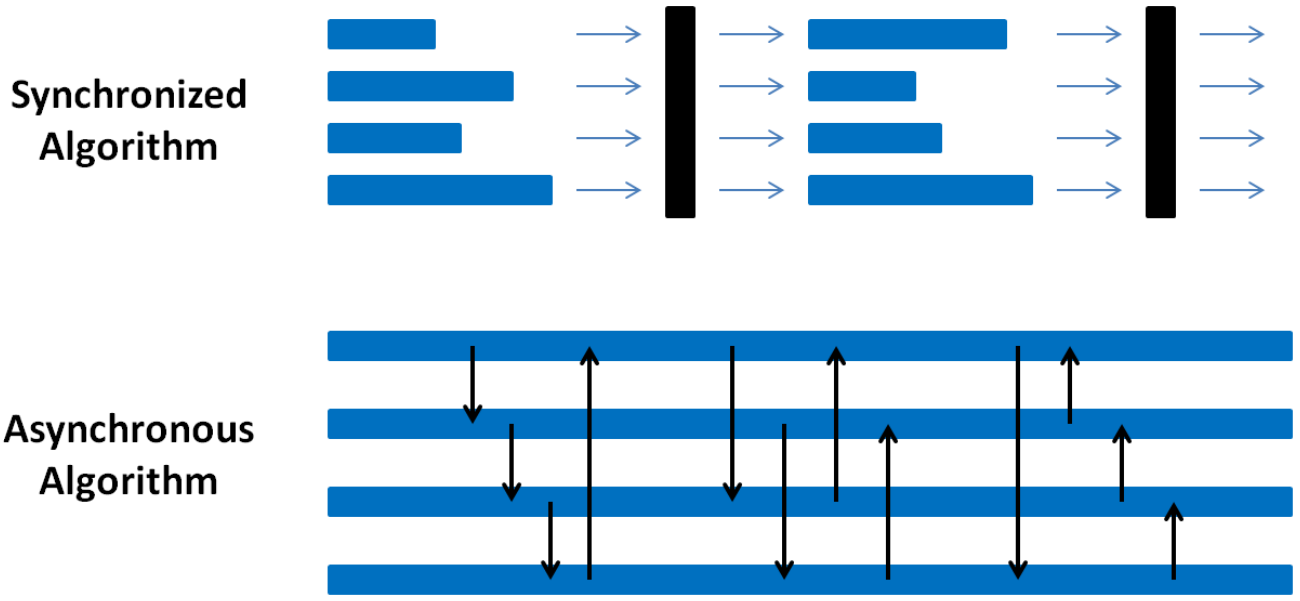



See (Dekel et al., "Optimal Distributed Online Prediction Using Mini-Batches". In JMLR 2012)

# Asynchronous Stochastic Gradient

- Synchronized algorithms: all the machine has to stop and synchronize at some points

  $\Rightarrow$ longer waiting time



# Asynchronous Stochastic Gradient (shared memory)

- The original SG:

For $t = 1, 2, \ldots$
    Randomly pick an index $i$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$

# 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 $i$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$

# 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 $i$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$

- Main trick: in shared memory systems, every threads can access the same parameter $\boldsymbol{w}$
- First discussed in (Langford et al., "Slow learners are fast". In NIPS 2009)
- Proposed in "Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent", NIPS 2011.

# Asynchronous Stochastic Gradient (shared memory)

- For convex function, converges to the global optimum under certain conditions:

  (1) bounded delay, (2) small confliction rate

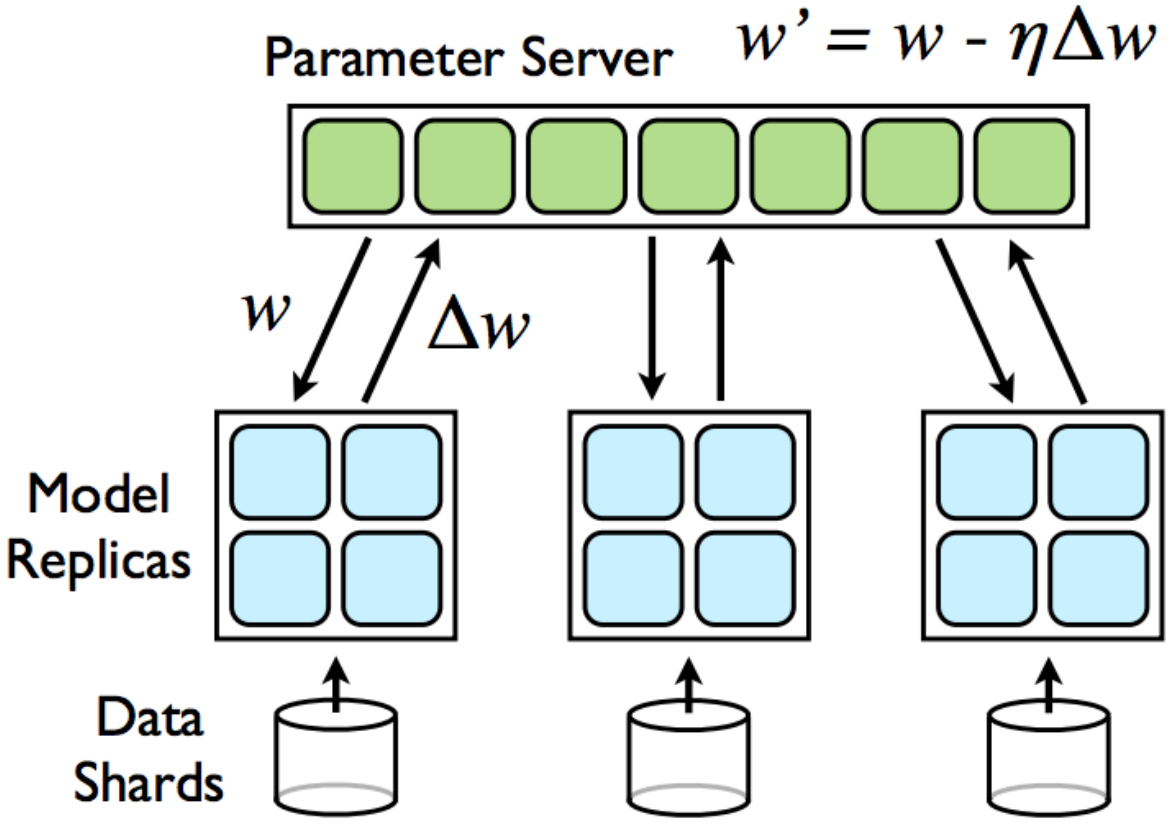
- A general framework proving the convergence rate of asynchronous SGD and coordinate descent converge to stationary points:

  Liang et al., "A Comprehensive Linear Speedup Analysis for Asynchronous Stochastic Parallel Optimization from Zeroth-Order to First-Order", NIPS 2016.

  (Prove the linear speedup for asynchronous algorithms).

# Asynchronous Stochastic Gradient (distributed memory)

- Use a parameter server to update the parameters




See Dean et al., "Large Scale Distributed Deep Networks", in NIPS 2012

# Parallel Coordinate Descent

# Parallel Coordinate Descent

- (Stochastic) Coordinate Descent (CD):

  For $t = 1, 2, \ldots$
  Randomly pick an index $i$
  $w_i^{t+1} \leftarrow w_i^t - \left(\text{argmin}_\delta f(\boldsymbol{w}^t - \delta \boldsymbol{e}_i)\right)$

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

  For $t = 1, 2, \ldots$
  Randomly pick an index $i$
  $w_i^{t+1} \leftarrow w_i^t - \eta \nabla_i f(\boldsymbol{w}^t)$

- 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 $b$

$w_i^{t+1} \leftarrow w_i^t - \eta \nabla_i f(\boldsymbol{w}^t)$ for all $i \in S$

# Synchronized Parallel Coordinate Descent

- Synchronized Parallel Coordinate Descent:

  For $t = 1, 2, \ldots$

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

  $w_i^{t+1} \leftarrow w_i^t - \eta \nabla_i f(\boldsymbol{w}^t)$ 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

Theoretical guarantee in Richtarik & Takac, "Parallel coordinate descent methods for big data optimization". Mathematical Programming, 2012.

# 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 $i$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$

# 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 $i$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla f_i(\boldsymbol{w})$

- Main trick: in shared memory systems, every thread can access the same parameter $\boldsymbol{w}$
- First implemented in (Bradley et al., "Parallel coordinate descent for $\ell_1$-regularized loss minimization". In ICML 2011)
- Officially discussed in Liu & Wright"An Asynchronous Parallel Stochastic Coordinate Descent Algorithm", ICML 2014.

# Coming up

- Next class: Support Vector Machines (SVM)

Questions?