ECE 901: Large-scale Machine Learning and Optimization **Fall 2016**

Lecture 17 — 11/17

Lecturer: Dimitris Papailiopoulos *Scribe: Akshay Sood, Zhenyu Zhang*

Note: These lecture notes are still rough, and have only have been mildly proofread.

17.1 Introduction

This lecture discusses serially equivalent and scalable parallel machine learning. Consider a single-machine, multi-core setup (Figure 17.1) where each of P processors talks to the same shared memory in an asynchronous manner (Figure 17.2).

Figure 17.1. Single multi-core machine

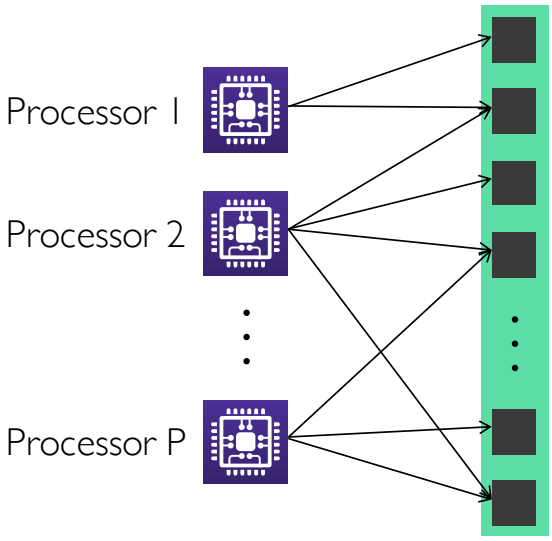


Figure 17.2. Each processor talks to the same shared memory in a maximally asynchronous manner

Then, for serial equivalence is defined as:

$$A_{serial}(S, \pi) = A_{parallel}(S, \pi)$$

i.e. the output of the serial and parallel versions of algorithm A is the same for all data sets S and data order π, where data points can be arbitrarily repeated.

Main advantages:

- We only need to "prove" speedups
- Convergence proofs are inherited directly from the serial case

Main issues:

- Serial equivalence is too strict
- We cannot guarantee speedups in the general case

17.2 Stochastic Updates Meta-algorithm

Consider Figure 17.3. The stochastic updates meta-algorithm is given as:

1. **Input:** $x; f_1 \dots f_n; u_1 \dots u_n; \mathcal{D}; T$
2. **for** $t = 1 : T$ **do**
3. sample $i \sim \mathcal{D}$
4. $x_{S_i} = u_i(x_{S_i}, f_i)$ // update global model on S_i
5. **Output:** x

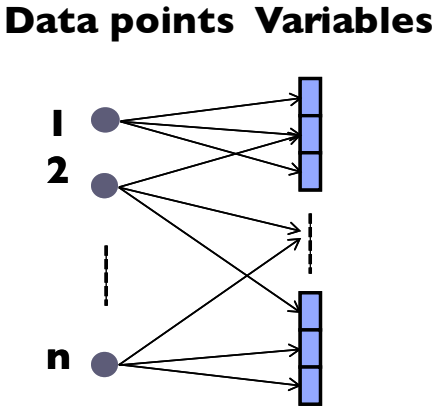


Figure 17.3. Setup for stochastic update meta-algorithm

A large family of machine learning algorithms with sparse access patterns belong to this class of stochastic update algorithms. These include SGD, SVRG/SAGA, matrix factorization, word2vec, K-means, stochastic PCA and graph clustering. Thus the goal will be to show serial equivalence for a parallelized version of the stochastic update meta-algorithm.

17.2.1 Update Conflict Graph

The update conflict graph can be constructed from the original graph mapping loss functions to parameters they depend on (Figure 17.4). The nodes of the conflict graph represent updates for each loss function/data point, and an edge between two updates represents a non-zero overlap in the parameters they depend on.

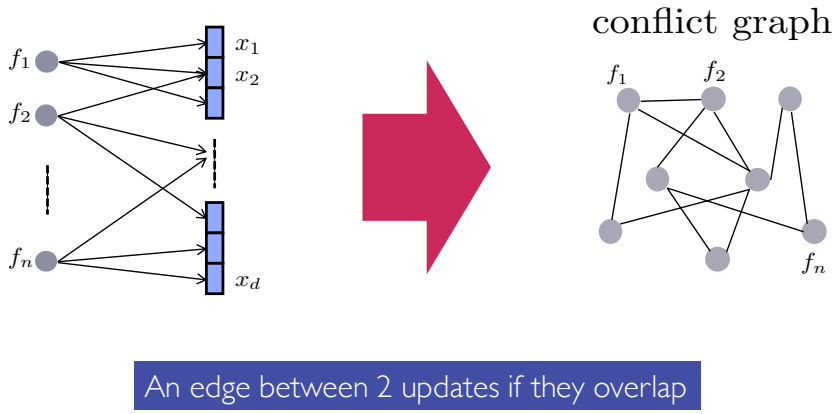


Figure 17.4. Update Conflict Graph, with an edge between any 2 updates that overlap

In order to achieve serial equivalence, we leverage the following theorem by [1]:

Theorem 17.1. *Let G be a graph on n vertices, with maximum vertex degree Δ. Let us sample each vertex independently with probability $p = \frac{1-\epsilon}{\Delta}$ and define as G' the induced subgraph on the sampled vertices. Then, the largest connected component of G' has size at most $\frac{4}{\epsilon^2}\log n$, with high probability.*

See Figure 17.5 for an illustration.

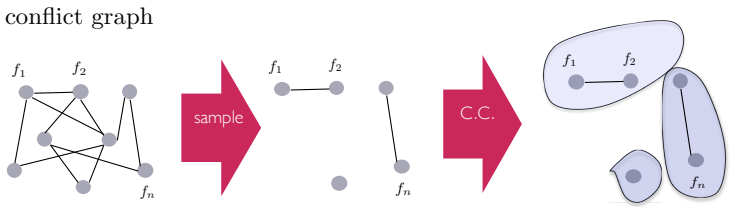


Figure 17.5. Sample conflict graph and identify connected components

17.3 Parallelization framework

Theorem 17.1 may be used to build a conflict-free asynchronous parallelization framework. After a sampling the conflict graph, identifying the connected components allows us to form groups of updates with no conflicts across them. Thus we can run the stochastic updates meta-algorithm on each of them in parallel. The overall procedure is as follows:

1. Sample a batch, identify connected components
2. Allocate each group to p different cores
3. Compute stochastic updates in an asynchronous and lock-free manner, then go back to step 1

See Figure 17.6 for an illustration. Thus we have the CYCLADES algorithm [2]:

1. **Input:** G_u, T, B
2. Sample $n_b = \frac{T}{B}$ subgraphs $G_u^1 \dots G_u^{n_b}$ from G_u
3. Cores compute in parallel CCs for sampled subgraphs
4. **for** batch $i = 1 : n_b$ **do**
5. Allocation of $C_1^i \dots C_{m_i}^i$ to P cores
6. **for** each core in parallel **do**
7. **for** each allocated component $\mathcal{C}$ **do**
8. **for** each update j (in order) from $\mathcal{C}$ **do**
9. $x_{S_j} = u_j(x_{S_j}, f_j)$
10. **Output:** x

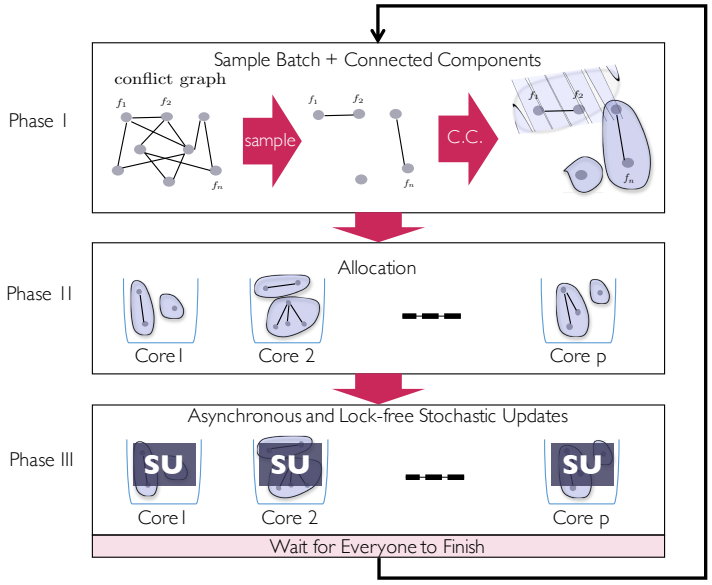


Figure 17.6. CYCLADES: Asynchronous lock-free parallelization framework

17.4 Main Theorem

Theorem 17.2. *Let us assume any given update-variable graph G_u with average, and max left degree $\overline{\Delta}_L$ and Δ_L, such that $\frac{\Delta_L}{\overline{\Delta}_L} \leq \sqrt{n}$, and with induced max conflict degree Δ. Then, CYCLADES on $P = O(\frac{n}{\Delta \cdot \Delta_L})$ cores, with batch sizes $B = (1-\epsilon)\frac{n}{\Delta}$ can execute $T = c \cdot n$ updates, for any constant $c \geq 1$, select uniformly at random with replacement, in time*

$$\mathcal{O}(\frac{E_u \cdot \kappa}{P} \cdot \log^2 n) \tag{17.1}$$

with high probability.

Assumptions of the theorem:
1. Not too large max degree.
2. Not too many cores.
3. Sampling according to the Graph Theorem

Note:
Serial Cost $= E_u \cdot \kappa$, κ is the cost of every coordinate update, comparing with the theorem we have:

$$Speedup = \frac{P}{\log^2 n} \tag{17.2}$$

17.5 Fast Connected Components

If we already have the conflict graph, it is easy to get connected components by sampling edges. But building the conflict graph requires $O(n^2)$ time, it is not an efficient way.

Instead of sampling on the Conflict Graph, we sample on the Bipartite. The algorithm of Simple Message Passing Idea is:

1. Gradient function send their IDs to each coordinate.
2. Each coordinate may receive several IDs, they compute the minimum of IDs and send back to gradient functions.
3. Each Gradient functions may receive values from several coordinate, it computes the min of all received values and send back to the coordinates.
4. If not done, return step 1.

The cost of the message passing algorithm will be:

$$Cost = O(\frac{E_u log^2 n}{P})$$

under the same assumptions as 17.2

Lemma 17.3. *Active each vertex with probability*

$$p = (1 - \epsilon)/\delta$$

Then, the induced subgraph shatters, and the largest connected component has size

$$\frac{4}{\epsilon^2} \cdot \log n$$

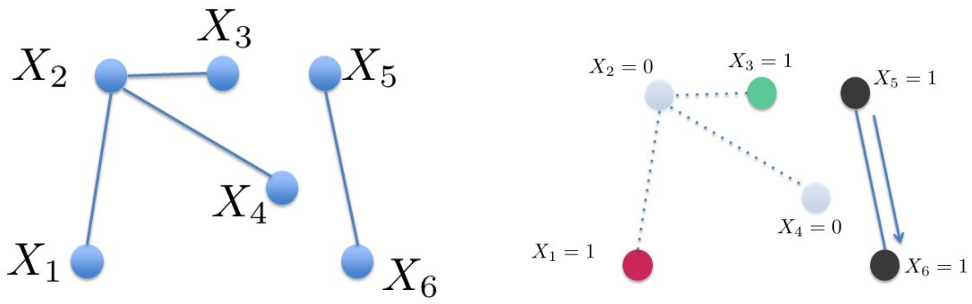
17.5.1 Deep First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

17.5.2 Probabilistic DFS

Algorithm:

- For each vertex DFS is visiting, flip a coin (generate a number either 0 or 1).
- 1 indicates visiting that vertex, 0 indicates don't visit that vertex and delete with its edges.



(a) figure before visited. (b) figure after visited.

Figure 17.7. DFS with random coins

Suppose we have a connected component of size k, and let the number of coins flipped associated with that component $\leq k \cdot \Delta$. Since we have a size k component, it means we had the event "At least k coins are 'head' in a set of $k \cdot \Delta$ coins".

17.6 Robustness against High-degree Outliers

Lemma 17.4. *Let us assume that there are $O(n)$ outlier vertices in the original conflict graph G with degree at most Δ_o, and let the remaining vertices have degree (induced on the remaining graph) at most Δ. Let the induced update variable graph on these low degree vertices abide to the same graph assumptions as those of 17.2. Moreover, let the batch size be bounded as*

$$B \leq \min\{(1-\epsilon)\frac{n - O(n^\delta)}{\Delta}, O(\frac{n^{1-\delta}}{P})\}$$

Then, the expected runtime of CYCLADES will be $O(\frac{E_u \cdot \kappa}{P} \cdot \log^2 n)$

Proof: Let w_s^i denote the total work required for batch i if that batch contains no outlier notes, and w_o^i otherwise. We can see that $w_s = \sum_i w_s^i = O(\frac{E_u \cdot \kappa}{P} \cdot \log^2 n)$ and $w_o = \sum_i w_s^i = O(E_u \cdot \kappa \cdot \log^2 n)$. Hence, the expected computational effort by CYCLADES will be

$$w_s \cdot Pr\{abatchcontainsnooutliers\} + w_o \cdot Pr\{abatchcontainsoutliers\}$$

where

$$Pr\{abatchcontainsnooutliers\} = \Omega((1 - \frac{1}{n^{1-\delta}})^B) \geq 1 - O(\frac{B}{n^{1-\delta}})$$

Hence the expected running time will be proportional to $O(\frac{E_u \cdot \kappa}{P} \cdot \log^2 n)$ which holds when $B = O(\frac{n^{1-\delta}}{P})$

□

References

[1] Michael Krivelevich. The phase transition in site percolation on pseudo-random graphs. *arXiv preprint arXiv:1404.5731*, 2014.

[2] Xinghao Pan, Maximilian Lam, Stephen Tu, Dimitris Papailiopoulos, Ce Zhang, Michael I Jordan, Kannan Ramchandran, Chris Re, and Benjamin Recht. Cyclades: Conflict-free asynchronous machine learning. *arXiv preprint arXiv:1605.09721*, 2016.