SySCD
A System-Aware Parallel Coordinate Descent Algorithm

Nikolas Ioannou*
IBM Research

Celestine Mendler-Dünner*
UC Berkeley

Thomas Parnell
IBM Research

*equal contribution
Parallel Coordinate Descent

\[
\min_{\alpha} f(A\alpha) + \sum_i g_i(\alpha_i)
\]

Parallel Coordinate Descent

1: **Input**: Training data matrix \( A \in \mathbb{R}^{d \times n} \)

   Initial model \( \alpha = 0, \, v = 0 \)

2: **for** \( t = 1, 2, \ldots \) **do**

3: \hspace{1em} **parfor** \( j \in \text{RANDOM\_PERMUTATION}(n) \) **do**

4: \hspace{2em} Find \( \delta \) minimizing \( f(v + A_{:,j} \delta) + g_j(\alpha_j + \delta) \)

5: \hspace{2em} \( \alpha_j \leftarrow \alpha_j + \delta \)

6: \hspace{2em} \( v \leftarrow v + \delta A_{:,j} \)

7: \hspace{1em} **end parfor**

8: **end for**
Parallel Coordinate Descent

\[
\min_{\alpha} f(A\alpha) + \sum_{i} g_i(\alpha_i)
\]

- Multi-core parallelism has previously been taken into account, e.g. [Hsieh’15]
- There are many more system bottlenecks one could incorporate into the algorithm design, i.e.,
  1. Inefficient cache accesses
  2. Write-contention on \( v \)
  3. Scalability across NUMA nodes

→ SySCD addresses all of these

### Parallel Coordinate Descent

1. **Input:** Training data matrix \( A \in \mathbb{R}^{d \times n} \)
   Initial model \( \alpha = 0, \ v = 0 \)

2. **for** \( t = 1, 2, \ldots \ **do**

3. **parfor** \( j \in \text{RANDOMPERMUTATION}(n) \ **do**

4. Find \( \delta \) minimizing \( f(v + A_{:,j} \delta) + g_j(\alpha_j + \delta) \)

5. \( \alpha_j \leftarrow \alpha_j + \delta \)

6. \( v \leftarrow v + \delta A_{:,j} \)

7. **end parfor**

8. **end for**
Parallel Coordinate Descent

\[
\min_{\alpha} f(A\alpha) + \sum_i g_i(\alpha_i)
\]

- Multi-core parallelism has previously been taken into account, e.g. [Hsieh’15]
- There are many more system bottlenecks one could incorporate into the algorithm design, i.e.,
  1. Inefficient cache accesses
  2. Write-contention on \( \mathbf{v} \)
  3. Scalability across NUMA nodes

→ SySCD addresses all of these

Parallel Coordinate Descent

1: **Input:** Training data matrix \( A \in \mathbb{R}^{d \times n} \)
   Initial model \( \alpha = 0, \mathbf{v} = 0 \)

2: **for** \( t = 1, 2, \ldots \) **do**

3: \[ \text{parfor } j \in \text{RANDOMPERMUTATION}(n) \] **do**

4: Find \( \delta \) minimizing \( f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta) \)

5: \( \alpha_j \leftarrow \alpha_j + \delta \)

6: \( \mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j} \)

7: **end parfor**

8: **end for**
Resolving write-contention on v
Resolving write-contention on \( \mathbf{v} \)

\[\text{replicate } \mathbf{v} \text{ across threads}\]
Resolving write-contention on $v$ → replicate $v$ across threads

**Parallel Coordinate Descent**

1: **Input:** Training data matrix $A \in \mathbb{R}^{d \times n}$
   
   Initial model $\alpha = 0$, $v = 0$

2: **for** $t = 1, 2, \ldots$ **do**
   
   # threads

3: \hspace{1cm} $v_p \leftarrow v \quad \forall p \in [P]$

4: \hspace{1cm} **parfor** $j \in \text{RANDOM\_PERMUTATION}(n)$ **do**

5: \hspace{2cm} Find $\delta$ minimizing $\hat{f}(v_{p}, A_{\cdot,j}, \alpha_{j}) + g_{j}(\alpha_{j} + \delta)$

6: \hspace{2cm} $\alpha_{j} \leftarrow \alpha_{j} + \delta$

7: \hspace{2cm} $v_{p} \leftarrow v_{p} + \delta A_{\cdot,j}$

8: \hspace{1cm} **end parfor**

9: \hspace{1cm} $v \leftarrow \sum_{p} v_{p}$

10: **end for**

Auxiliary model inspired by CoCoA [Smith’18]
Connection to Distributed Methods

- **shared vector** $\mathbf{v}$
- **model** $\mathbf{\alpha}$
- **data** $\mathbf{A}$

- Data is partitioned across workers.
- Each worker has its own replica of the shared vector which is synchronized periodically.
Connection to Distributed Methods

- In the parallel setting, a worker corresponds to a core.
- Motivation of data separability is not communication but implementation-efficiency.
Repartitioning

- In the parallel setting we can afford to repartition the data in each round
- Improved convergence behavior
System-Aware Coordinate Descent (SySCD)

- Combination of distributed methods with repartitioning
  - high implementation efficiency
  - theoretically sound parallel method
  - scales to large degrees of parallelism

- Additional optimizations (not covered in this talk)
  - NUMA - affinity
  - alignment with cache access pattern

> 10x faster than state-of-the-art asynchronous CD methods