SySCD
A System-Aware Parallel Coordinate Descent Algorithm

Nikolas Ioannou*
IBM Research

Celestine Mendler-Dünner*
UC Berkeley

Thomas Parnell
IBM Research

*equal contribution
standard CPU model for algorithm development
standard CPU model for algorithm development

better
... going one step further
Bottlenecks

- Non-uniform memory access
Bottlenecks

- Non-uniform memory access
Bottlenecks

- Non-uniform memory access
Bottlenecks

- ✔ Non-uniform memory access
- ❑ Access to model vector due little cache reuse
Bottlenecks

- Non-uniform memory access
- Access to model vector due little cache reuse
Bottlenecks

- Non-uniform memory access
- Access to model vector due little cache reuse
Bottlenecks

- Non-uniform memory access

- Access to model vector due to little cache reuse
Bottlenecks

- Non-uniform memory access
- Access to model vector due little cache reuse
- Access to shared vector
Bottlenecks

- Non-uniform memory access
- Access to model vector due little cache reuse
- Access to shared vector
Bottlenecks

- ✔ Non-uniform memory access
- ✔ Access to model vector due little cache reuse
- ❐ Access to shared vector
Bottlenecks

- ✔️ Non-uniform memory access
- ✔️ Access to model vector due little cache reuse
- ✔️ Access to shared vector

![NUMA NODE 0 diagram]
Bottlenecks

- ✔ Non-uniform memory access
- ✔ Access to model vector due little cache reuse
- ✔ Access to shared vector
- ❑ Convergence
Bottlenecks

- Non-uniform memory access
- Access to model vector due to little cache reuse
- Access to shared vector
- Convergence
Bottlenecks

- Non-uniform memory access
- Access to model vector due to little cache reuse
- Access to shared vector
- Convergence
SySCD

Fast and scalable parallel CPU solver

✓ convergence guarantees
✓ scalable
✓ fast

Poster #XXX

Algorithm 1 SySCD for minimizing (1)

1: **Input:** Training data matrix \( A = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n} \)
2: Initialize model \( \alpha \) and shared vector \( \mathbf{v} = \sum_{i=1}^{n} \alpha_i x_i \).
3: Partition coordinates into buckets of size \( B \).
4: Partition buckets across numa nodes according to \( \{ P_k \}_{k=1}^{K} \).
5: for \( t = 1, 2, \ldots, T_1 \) do
6: \hspace{1em} \textbf{parfor} \( k = 1, 2, \ldots, K \) across numa nodes do
7: \hspace{2em} \( \mathbf{v}_k = \mathbf{v} \)
8: \hspace{2em} for \( t = 1, 2, \ldots, T_2 \) do
9: \hspace{3em} create random partitioning of local buckets across threads \( \{ P_{k,p} \}_{p=1}^{P} \)
10: \hspace{2em} \textbf{parfor} \( p = 1, 2, \ldots, P \) across threads do
11: \hspace{3em} \( \mathbf{v}_p = \mathbf{v}_k \)
12: \hspace{3em} for \( j = 1, 2, \ldots, T_3 \) do
13: \hspace{4em} randomly select a bucket \( B \in P_{k,p} \)
14: \hspace{4em} for \( i = 1, 2, \ldots, T_4 \) do
15: \hspace{5em} randomly sample a coordinate \( j \) in bucket \( B \)
16: \hspace{5em} \( \delta = \arg \min_{\delta \in \mathbb{R}} f(\mathbf{v}_p + x_j \delta) + g_j(\alpha_j + \delta) \)
17: \hspace{5em} \( \alpha_j = \alpha_j + \delta \)
18: \hspace{5em} \( \mathbf{v}_p = \mathbf{v}_p + \delta x_j \)
19: \hspace{4em} end for
20: \hspace{3em} end for
21: \hspace{2em} end parfor
22: \( \mathbf{v}_k = \mathbf{v}_k + \sum_p (\mathbf{v}_p - \mathbf{v}_k) \)
23: \hspace{1em} end for
24: \hspace{1em} end parfor
25: \( \mathbf{v} = \mathbf{v} + \sum_k (\mathbf{v}_k - \mathbf{v}) \)
26: end for