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

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

Parallel Coordinate Descent

1: **Input:** Training data matrix $A \in \mathbb{R}^{d \times n}$
   Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
2: **for** $t = 1, 2, \ldots$ **do**
3: **parfor** $j \in \text{RandomPermutation}(n)$ **do**
4: Find δ 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**

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

- 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

Input: Training data matrix $A \in \mathbb{R}^{d\times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
for $t = 1, 2, \ldots$ **do**
  parfor $j \in \textsc{RandomPermutation}(n)$ **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

- 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.,

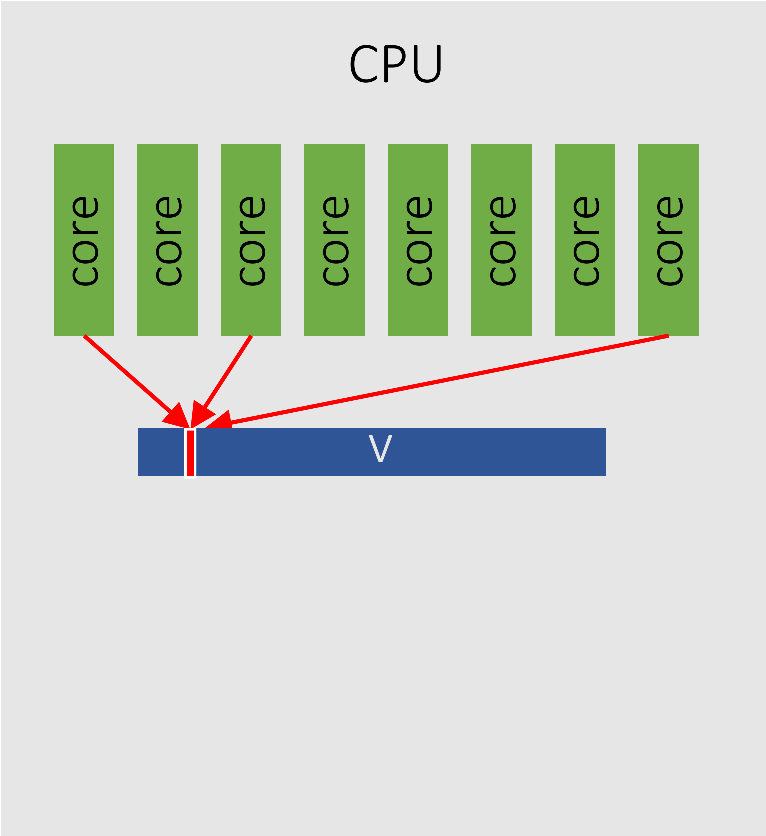
  1. Inefficient cache accesses
  2. Write-contention on **v**
  3. Scalability across NUMA nodes

→ SySCD addresses all of these

Parallel Coordinate Descent

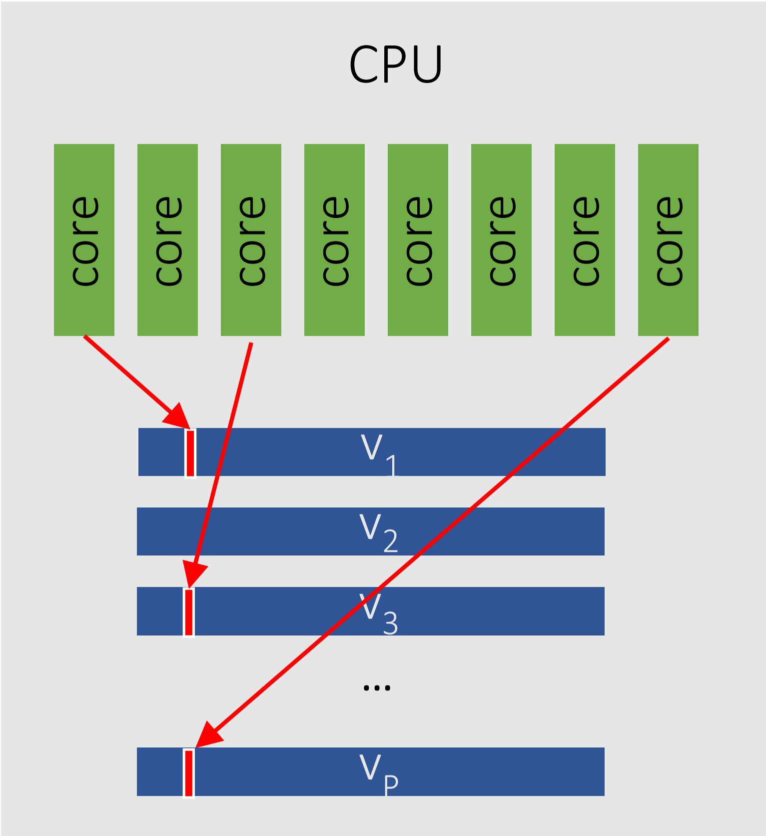
Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
for $t = 1, 2, \ldots$ **do**
  parfor $j \in \textsc{RandomPermutation}(n)$ **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Resolving write-contention on **v**



Resolving write-contention on **v**

→ replicate **v** across threads



Resolving write-contention on v

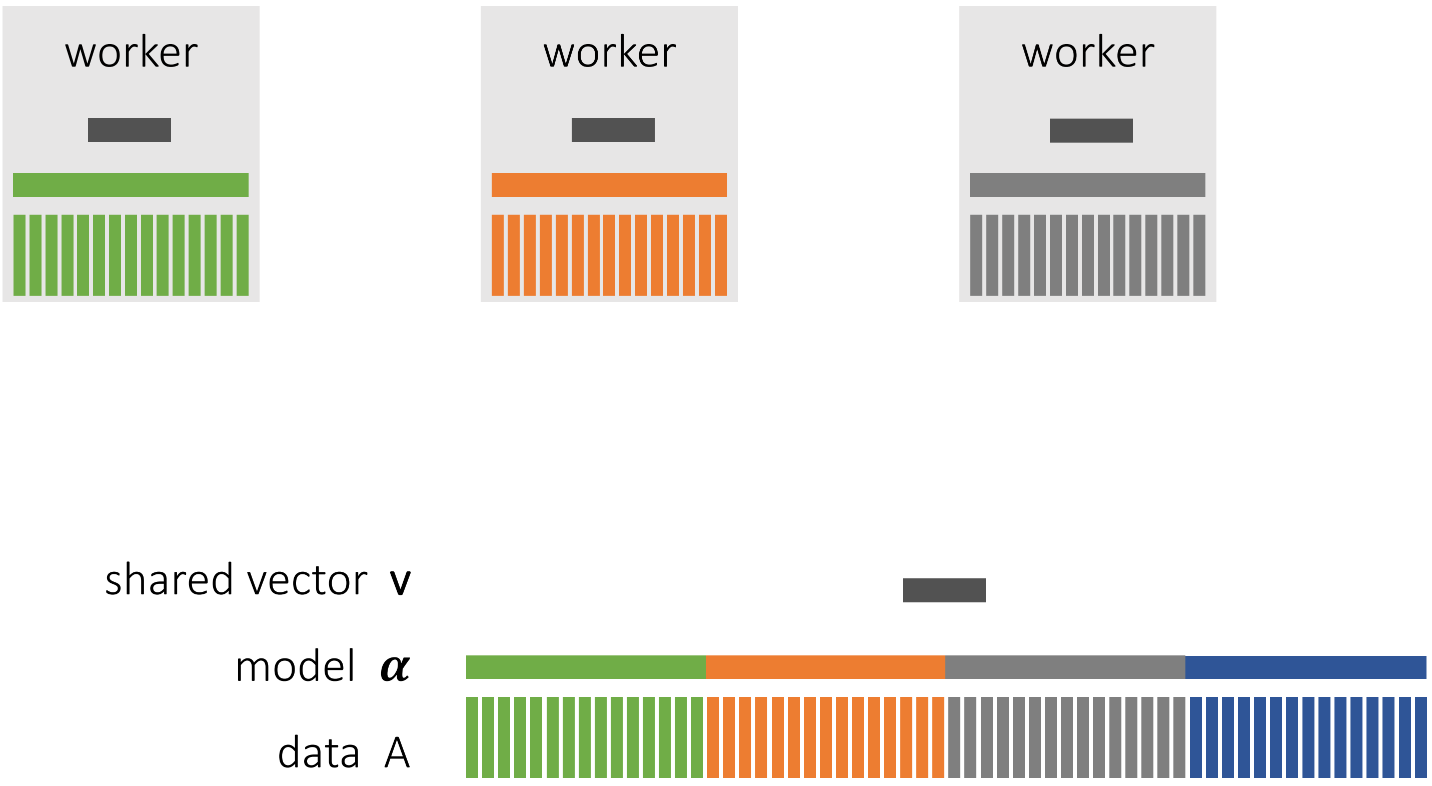
→ replicate **v** across threads

Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
for $t = 1, 2, \ldots$ **do**
$\quad \mathbf{v}_p \leftarrow \mathbf{v} \quad \forall p \in [P]$ — # threads
$\quad$ **parfor** $j \in \textsc{RandomPermutation}(n)$ **do**
$\quad\quad$ Find δ minimizing $\hat{f}(\mathbf{v}_p, A_{:,j}, \alpha_j) + g_j(\alpha_j + \delta)$
$\quad\quad \alpha_j \leftarrow \alpha_j + \delta$
$\quad\quad \mathbf{v}_p \leftarrow \mathbf{v}_p + \delta A_{:,j}$
$\quad$ **end parfor**
$\quad \mathbf{v} \leftarrow \sum_p \mathbf{v}_p$
end for

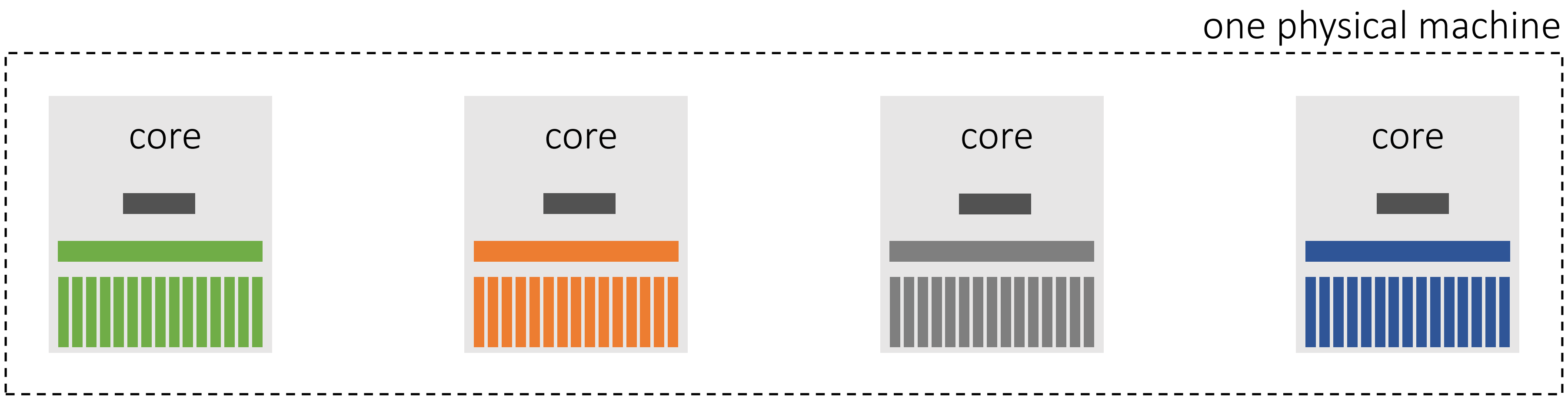
auxiliary model inspired by CoCoA [Smith'18]

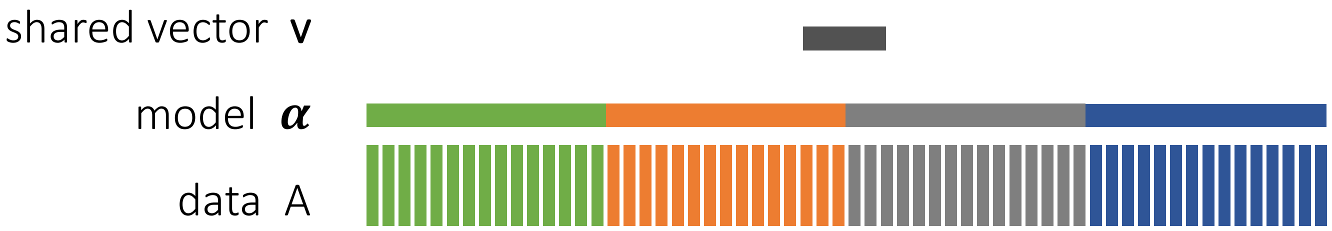
Connection to Distributed Methods



- data is partitioned across workers
- each worker has its own replica of the shared vector which is synchronized periodically

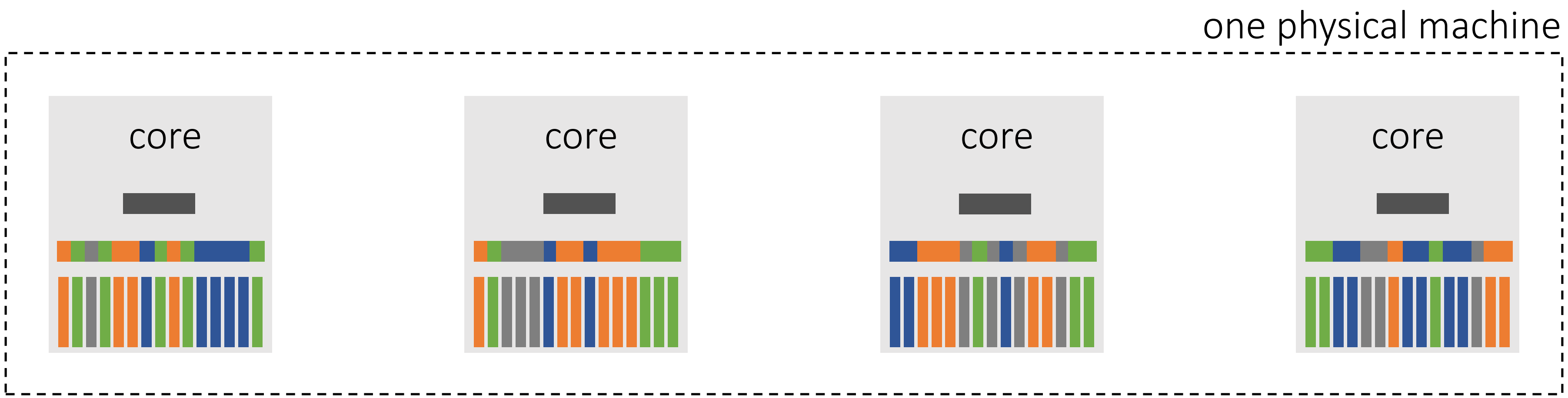
Connection to Distributed Methods

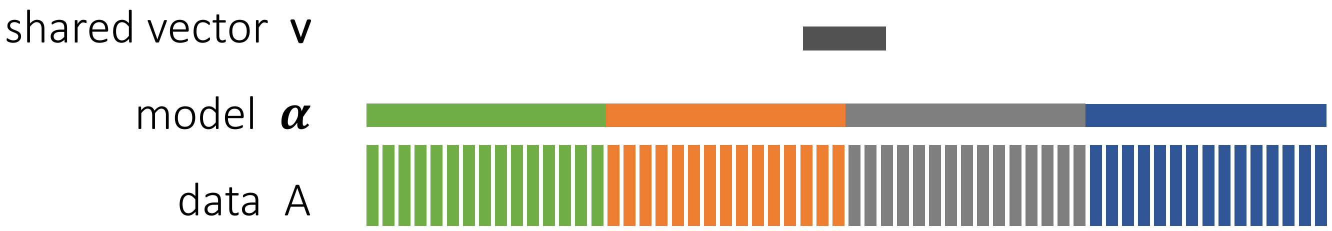




- In the parallel setting a worker corresponds to a core
- Motivation of data separability is not communication- but implementation-efficiency

Repartitioning



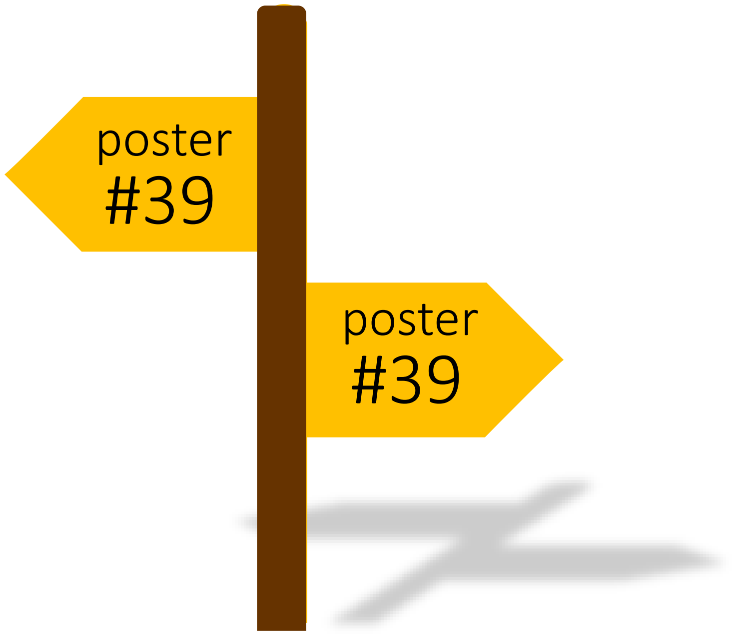


- In the parallel setting we can afford to repartition the data in each round

✓ 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 sate-of-the-art asynchronous CD methods