Asynchronous coordinate update methods

Juan C. Osorio

School of Mechanical Engineering
Purdue University
Email: osorio2@purdue.edu

Chapter 6: Large-Scale Convex Optimization
Algorithms & Analysis via Monotone Operators

Ernest K.RYU & Wotao Yin

PURDUE UNIVERSITY

Problem Set-up

Let $\mathbb{T}: \mathbb{R}^n \to \mathbb{R}^n$ be θ-average with $\mathbb{T} = \mathbb{I} - \theta S$.

Partition $x = (x_1, \dots, x_m) \in \mathbb{R}^n$ and:

$$\mathbb{T}(x) = \begin{bmatrix} (\mathbb{T}(x))_1 \\ \vdots \\ (\mathbb{T}(x))_i \\ \vdots \\ (\mathbb{T}(x))_m \end{bmatrix} \quad \mathbb{T}_i(x) = \begin{bmatrix} x_1 \\ \vdots \\ x_{i-1} \\ (\mathbb{T}(x))_i \\ x_{i+1} \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{S}_i(x) = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ (\mathbf{S}(x))_i \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

We can perform a standard synchronous or asynchronous implementation of

$$x^{k+1} = \mathbb{T}(x^k) \qquad\qquad x^{k+1} = x^k - \eta \mathbf{S} x^k,$$

On multiple agents or CPUS.

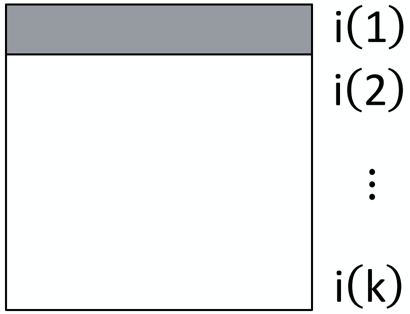
Coordinate Update Methods

Our main goal is to **decompose a large problem into smaller subproblems** and are thus useful for solving large-sized problems.

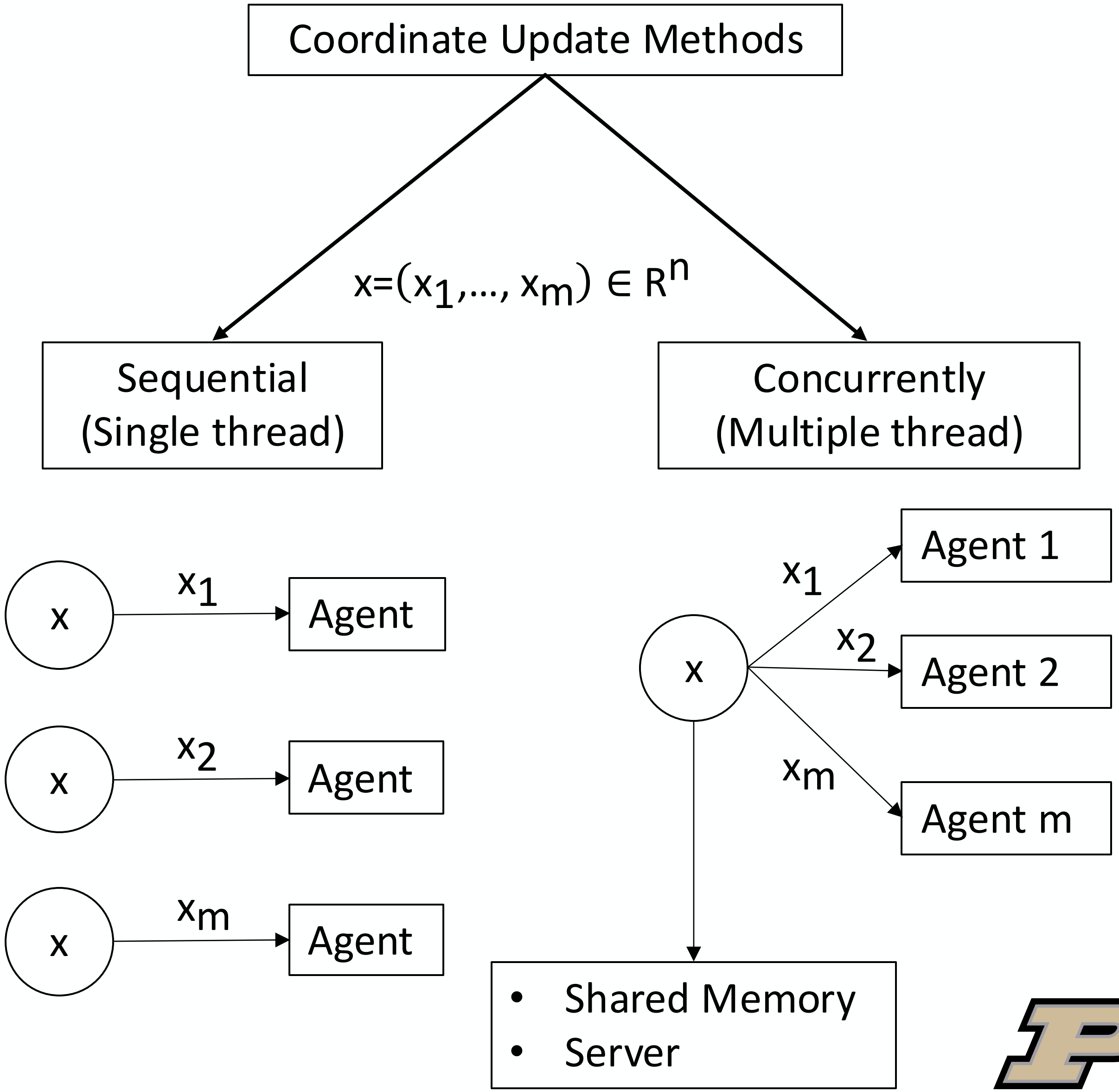
$$x^{k+1} = x^k - \eta \nabla f(x_k)_{i(k)}$$

Partition $x=(x_1,\ldots, x_m) \in R^n$

$$\nabla f(x_k)_1 =$$



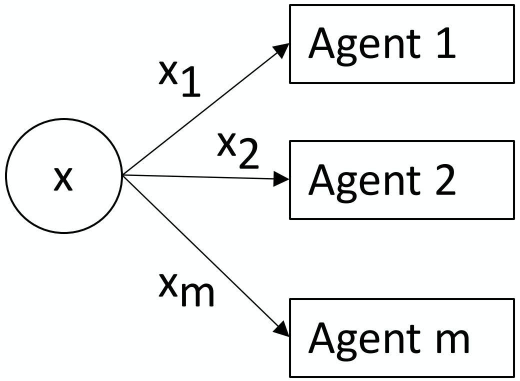
Variables can be updated in cyclic, **random** or greedy orders.



Parallel Coordinate Update Methods

We can use multiple computational agents to speed up the algorithm

Concurrently
(Multiple thread)



$$x^{k+1} = x^k - \eta \mathbf{S} x^k,$$

Synchronous parallel FPI

```
while not converged do
    while not all indices processed do
        Claim index i not yet claimed
        Read x
        Write s[i] = ηS[i](x)
    end while
    Synchronize: wait for all agents
    while not all indices processed do
        Claim index i not yet claimed
        Write x[i] = x[i] − s[i]
    end while
    Synchronize: wait for all agents
end while
```

Asynchronous FPI

An algorithm is asynchronous parallel if it avoids synchronization barriers.

```
while not converged do
    Select i from Uniform{1, 2, ..., m}
    Read x
    Compute s[i] = ηS[i](x)
    Write x[i] = x[i] − s[i]
end while
```

Now, agents run completely uncoordinated, and the cost of synchronization is eliminated.

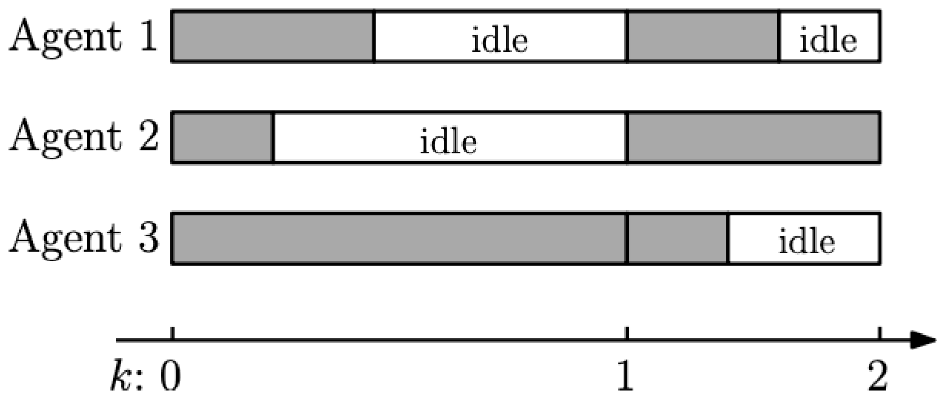
Synchronous vs Asynchronous - Cost

sync-parallel computing

As the number of computing agents grows, the cost of synchrony becomes significant.

Faster agents must wait idly for the slower ones.

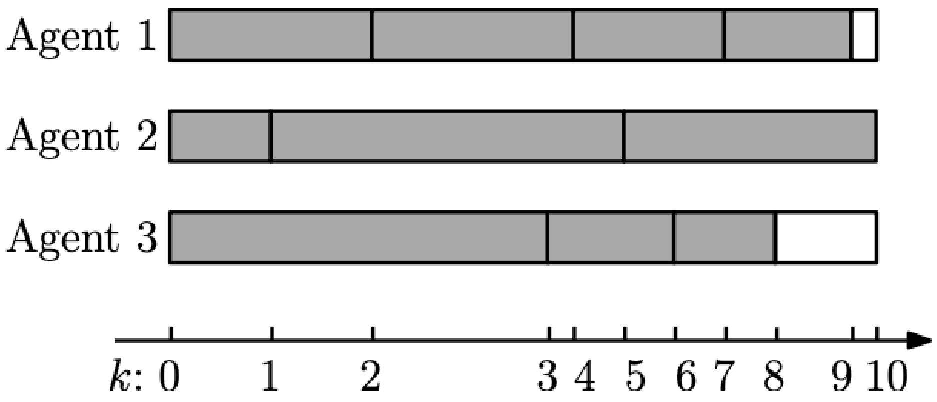
The synchronization barrier is itself an algorithm with a cost



async-parallel computing

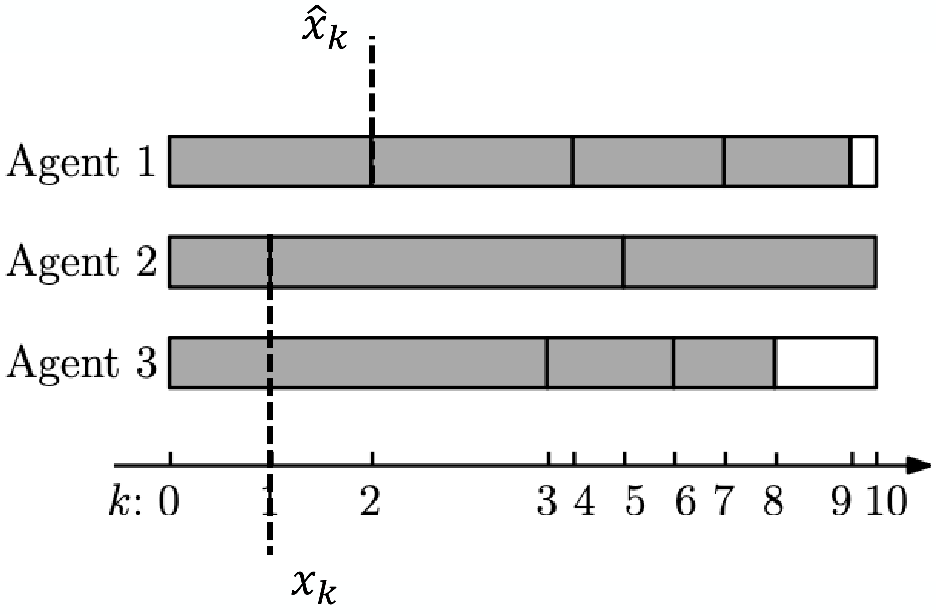
Requires exclusive memory access to avoid race condition.

Communication congestion can become a factor.



Asynchronous Parallelism


```
1: while not converged do
2:     Select i from Uniform{1, 2, ..., m}
3:     Read x
4:     Compute s[i] = ηS[i](x)
5:     Write x[i] = x[i] − s[i]
6: end while
```



When an agent performs Step 5, other agents may have updated x, rendering x used to compute Step 4 outdated. In this case, we say x is stale

$$x^{k+1} = x^k - \eta \mathbf{S}_{i(k)} \hat{x}^k$$

where $\hat{x}_k$ contains information older than x_k

Asynchronous Method

Asynchronous fixed-point iteration

Convergence of Asynchronous Methods

Exclusive Memory Access

Methods

Asynchronous Method

Asynchronous fixed-point iteration

Convergence of Asynchronous Methods

Exclusive Memory Access

Methods

Asynchronous fixed-point iteration

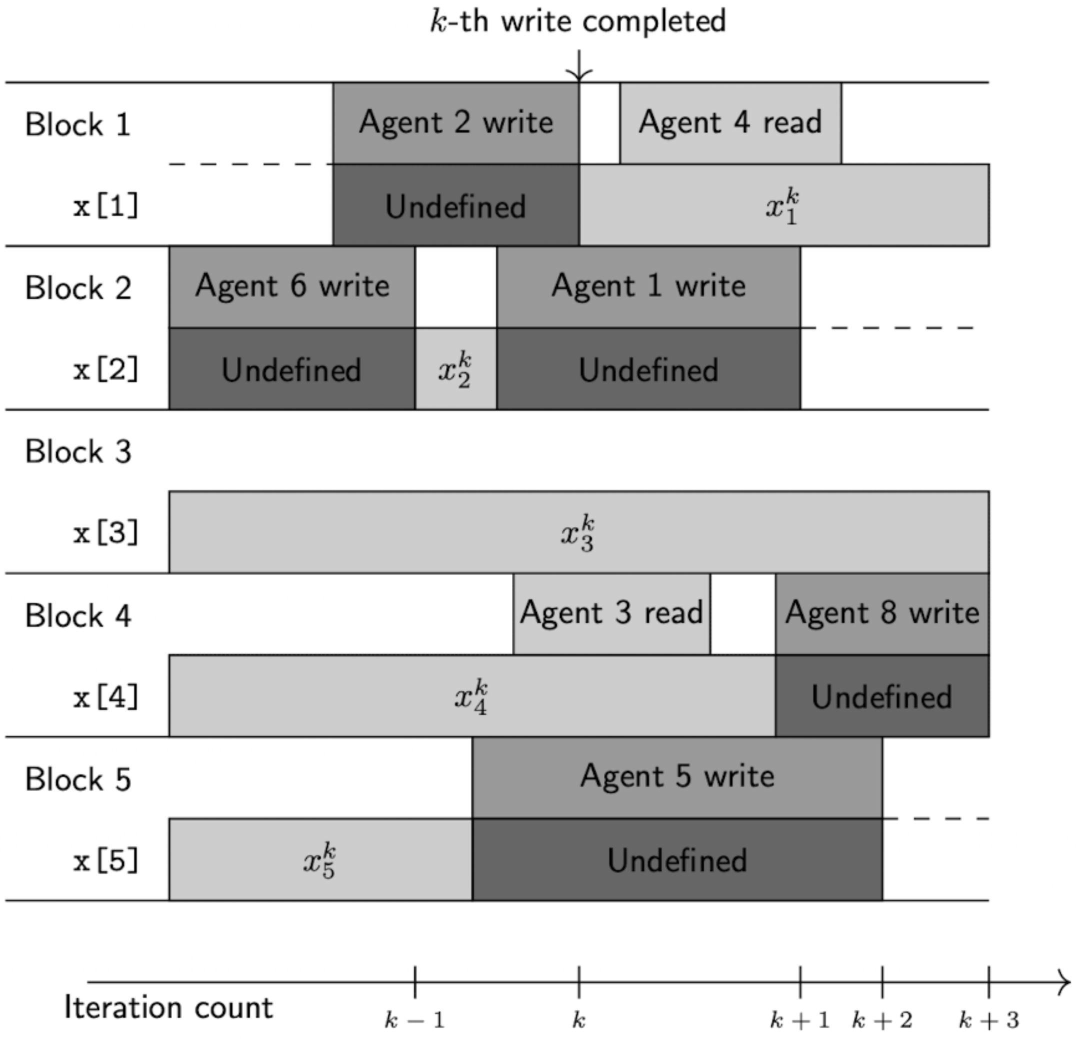
We can account for staleness if we enforce exclusive access in Step 5. An agent has exclusive access to x in **shared memory** if no other agent can read from or write to x simultaneously

1: **while** not converged **do**
2: Select i from Uniform$\{1, 2, \ldots, m\}$
3: **Read** x
4: Compute $s[i] = \eta S[i](x)$
5: Exclusively **Read** $x[i]$
6: Exclusively **Write** $x[i] = x[i] - s[i]$
7: **end while**

- AC-FPI removes explicit synchronization barriers, though it still requires exclusive access in writing to x[i].
- When there are much more blocks than agents, i.e., $p \ll m$, it is rare for an agent to wait for the release of a block's exclusive access; hence, most, albeit not all, idle time is eliminated.

Asynchronous fixed-point iteration - iterate

- x^0 is the state of x before the start of the algorithm.
- The kth interate is $x^k = (x_1^k, \ldots, x_m^k)$
- **Iteration count increment**: The iteration counter increases by 1 whenever an agent completes an update of x in the global memory.
- **State of $x_k[j]$ (no updates in progress)**: When the iteration counter reaches k, if no agent is writing to x[j], then $x_k[j]$ reflects the current state of x[j] at that moment.
- **State of $x_k[j]$ (updates in progress)**: When the iteration counter reaches k, if an agent is actively updating x[j], then $x_k[j]$ represents the state of x[j] right before the agent began its current update.



Asynchronous fixed-point iteration - Delays

Delay notation of staleness

Write i(k) for the index of the kth update, consider a coordinate-by-coordinate notion of staleness:

$$\hat{x}^k = \left(x_1^{k-d_1(k)}, \ldots, x_m^{k-d_m(k)}\right) \qquad \boldsymbol{d}(k) = (d_1(k), \ldots, d_m(k)) \in \mathbb{N}_+^m$$

Mathematical definition of the AC-FPI:

$$x^{k+1} = x^k - \eta \mathbf{S}_{i(k)} \hat{x}^k \longrightarrow x^{k+1} = x^k - \eta \mathbf{S}_{i(k)} x^{k-\boldsymbol{d}(k)}$$

AC-FPI is a stochastic algorithm realized by the random variables i(0), i(1), . . . and d(0), d(1), ...

Asynchronous Method

Asynchronous fixed-point iteration

Convergence of Asynchronous Methods

Exclusive Memory Access

Methods

Convergence of AC-FPI – ARock assumptions

The AC-FPI update $x^{k+1} = x^k - \eta S_{i(k)} x^{k-d(k)}$ models can be analyzed with the **ARock** assumptions:

- $i(0), i(1), \ldots$ are IID with uniform probability.
- $i(k)$ and $d(\ell)$ are independent for $k = 0, 1, \ldots$ and $\ell \leq k$
- $d(0), d(1), \ldots$ is a stochastic process with nonincreasing $Q_0, Q_1, \ldots \in [0, 1]$ such that for every $k = 0, 1, \ldots$,

$$\text{Prob}\left[\max_{i=1,\ldots,m} d_i(k) \geq \ell \mid \boldsymbol{d}(k-1), \ldots, \boldsymbol{d}(0), i(k-1), \ldots, i(0)\right] \leq Q_\ell,$$

$$\sum_{\ell=1}^{\infty} \ell \left(Q_\ell\right)^{1/2} < \infty$$

Convergence of AC-FPI

Theorem

Assume S: $\mathbb{R}^n \to \mathbb{R}^n$ is (1/2)-cocoersive or, equivalently, that $\mathbb{T} = \mathbb{I} - \theta S$ is θ-averaged with $\theta \in (0,1)$. Assume Fix $\mathbb{T} \neq \emptyset$. Under the **ARock** assumptions, the AC-FPI with any starting point $x^0 \in \mathbb{R}^n$ and step size η obeying

$$0 < \eta < \left(1 + \frac{2}{\sqrt{m}} \sum_{\ell=1}^{\infty} Q_\ell^{1/2}\right)^{-1}$$

Converges to one fixed point with probability 1

$$x^k \to x^\star \qquad \text{for some } x^\star \in \operatorname{Fix} \mathbb{T}$$

Furthermore, with probability 1,

$$\operatorname{dist}\left(x^k,\ \operatorname{Fix}\ \mathbb{T}\right) \to 0$$

Discussion of assumptions

Exclusive Access: Step 4 requires exclusive access. Otherwise, we would not be able to use the notation

$$\boldsymbol{d}(k) = (d_1(k), \ldots, d_m(k)) \in \mathbb{N}_+^m$$

Independence:

- i(k) and d(k) are independent for k= 0,1,.... This is realistic if the computational costs of the blocks are uniform.

- Sequence i(0),i(1),... Is assumed to be IID.
  - If the blocks have non-uniform computational costs, the choice of index affects the iteration count the update is assigned to and the IID assumption is violated.

Dependence allowed:

- Independence of d(0), d(1) … is not assumed.
  - It is likely that $\hat{x}^k$ and $\hat{x}^{k+1}$ are read at close points in time, and this makes d(k) and d(k+1) highly correlated.

Asynchronous Method

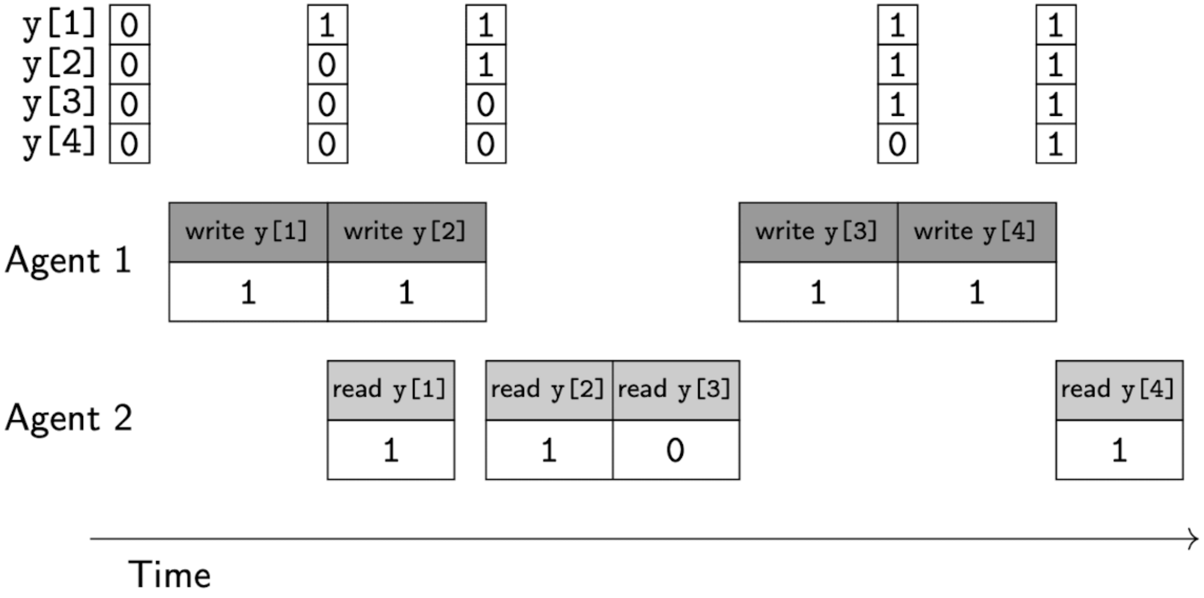
Asynchronous fixed-point iteration

Convergence of Asynchronous Methods

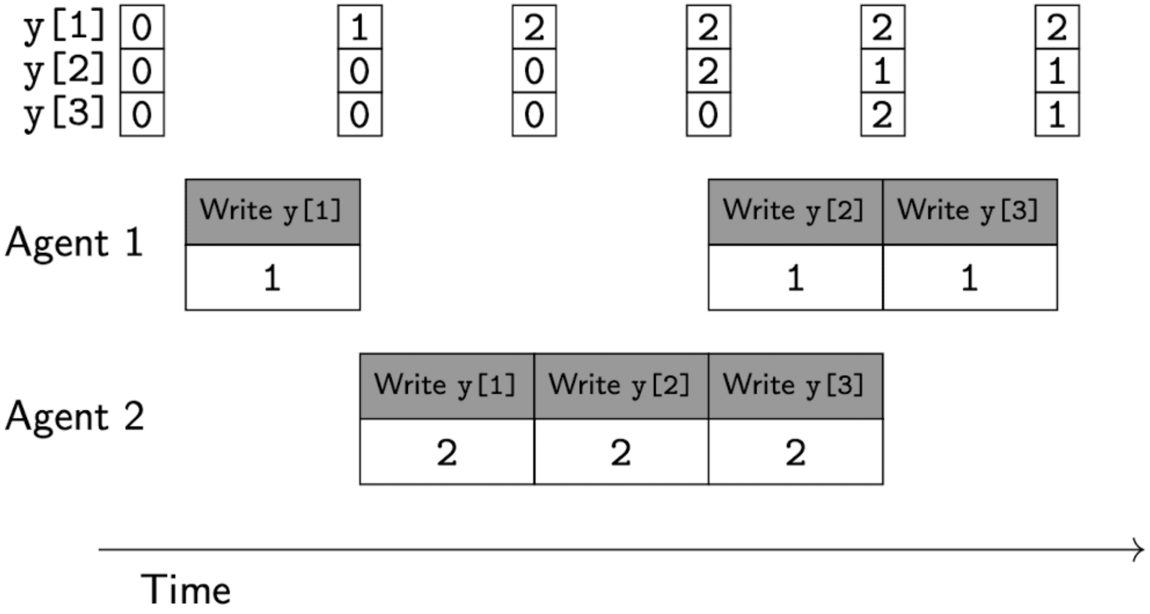
Exclusive Memory Access

Methods

Exclusive Memory Access



inconsistent read



inconsistent write

Exclusive access can be implemented with standard parallel computing techniques such as:

- Atomic operations
- Mutexes
- semaphore

Exclusive Memory Access – Atomic Operations

An operation of a computational agent is atomic if:

- The whole operation is guaranteed to complete without interruption from other agents.
- An atomic operation consists of multiple steps, other agents will not observe intermediate results

```matlab
% Create a DataQueue for communication
dq = parallel.pool.DataQueue;

% Define a callback to update the shared vector
afterEach(dq, @(data) assignin('base', 'sharedVector', ...
    evalin('base', 'sharedVector') + data));

% Use parallel processing
parfor i = 1:numIterations
    % Create a vector where only the i-th position is updated
    localUpdate = zeros(numIterations, 1);
    localUpdate(i) = 1; % Assign exclusive value for worker i

    % Send the local update to the DataQueue
    send(dq, localUpdate);
end
```

Asynchronous Method

Asynchronous fixed-point iteration

Convergence of Asynchronous Methods

Exclusive Memory Access

Methods

Methods – Asynchronous coordinate gradient descent

Consider the problem

$$\min_{x\in\mathbb{R}^n} f\left(\sum_{i=1}^{m} A_{:,i}x_i - b\right) + \sum_{i=1}^{m} g_i\left(x_i\right), \qquad A = \begin{bmatrix} A_{:,1} & A_{:,2} & \cdots & A_{:,m}\end{bmatrix}$$

Every i_{th} agent has access to x_i^k, $Prox_{\alpha g_i}, A_{:,i}$ and ∇f for $i = 1, \ldots, m$. The RC-FPI with the FBS operator is:

$$\begin{aligned} x_{i(k)}^{k+1} &= \mathrm{Prox}_{\alpha g_{i(k)}}\left(x_{i(k)}^{k} - \alpha A_{:,i(k)}^{\top}\nabla f\left(y^k\right)\right) \\ y^{k+1} &= y^k + A_{:,i(k)}\left(x_{i(k)}^{k+1} - x_{i(k)}^{k}\right) \end{aligned}$$

where we initialize $y^0 = Ax^0 - b$. The corresponding AC-FPI is

$$\begin{aligned} s_{i(k)}^{k} &= \eta\left(\hat{x}_{i(k)}^{k} - \mathrm{Prox}_{\alpha g_{i(k)}}\left(\hat{x}_{i(k)}^{k} - \alpha A_{:,i(k)}^{\top}\nabla f\left(\hat{y}^k\right)\right)\right) \\ x_{i(k)}^{k+1} &= x_{i(k)}^{k} - s_{i(k)}^{k} \\ y^{k+1} &= y^k - A_{:,i(k)} s_{i(k)}^{k} \end{aligned}$$

Methods – Asynchronous coordinate gradient descent

Where:

$$\hat{x}^{k}_{i(k)} = x^{k-d_{i(k)}(k)}_{i(k)}, \quad \hat{y}^{k} = A_{:,1}x_1^{k-d_1(k)} + \cdots + A_{:,m}x_m^{k-d_m(k)}$$

In a shared memory system, we can implement AC-FPI with:

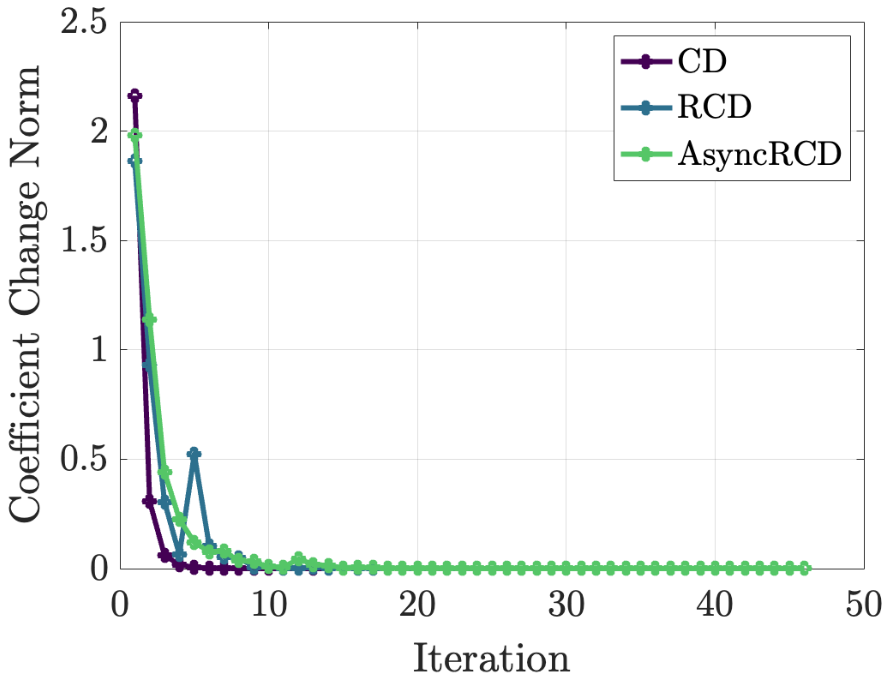
Algorithm ACGD

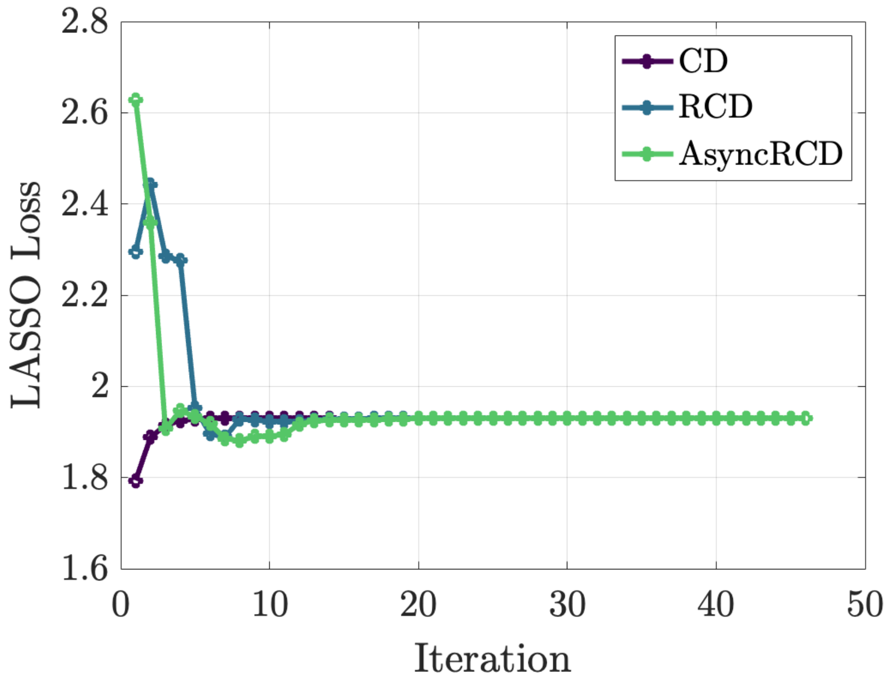
Initialize $x = 0$, $y = -b$
$Pr_i = \text{prox}_{\alpha g_i}$, $G_f =$ gradient of f
while not converged **do**
  Read y
  $s[i] = \eta \cdot (x[i] - Pr_i(x[i] - \alpha \cdot A[:,i]' \cdot G_f(y)))$
  $\text{del}[i] = -A[:,i] \cdot s[i]$
  Acquire exclusive access to y
  $y = y + \text{del}[i]$
  Release exclusive access to y
  $x[i] = x[i] - s[i]$
end while

Examples – Lasso Regression

Now consider the lasso problem

$$\min_{\beta} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

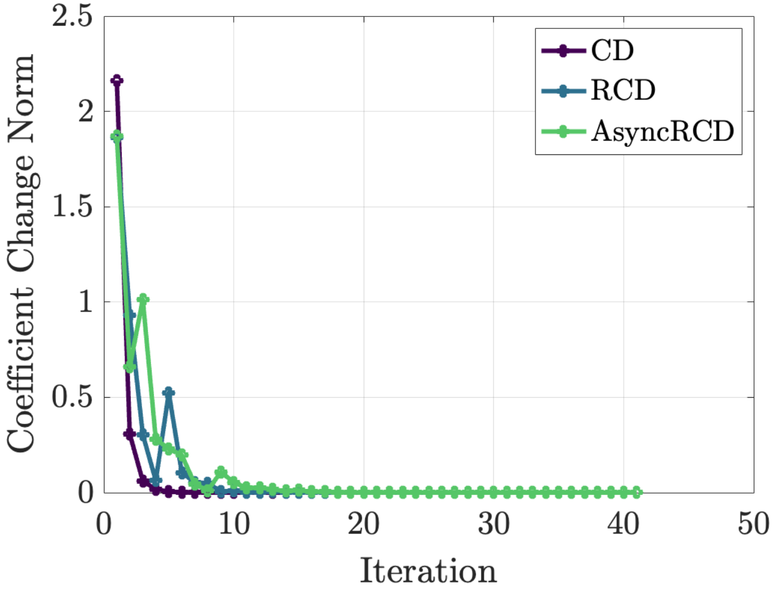




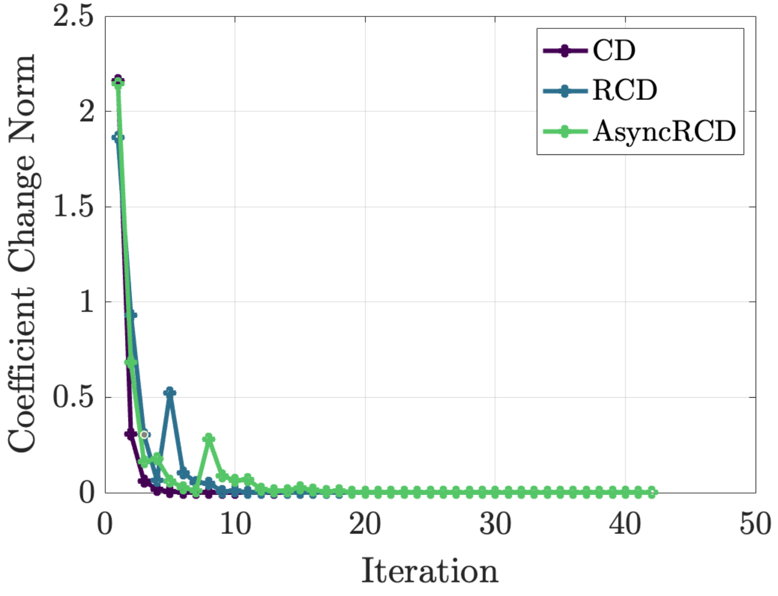
Method	RMSE
MATLAB Lasso	0.61
CD	0.59
RCD	0.59
AsyncRCD	0.59

Examples – Lasso Regression

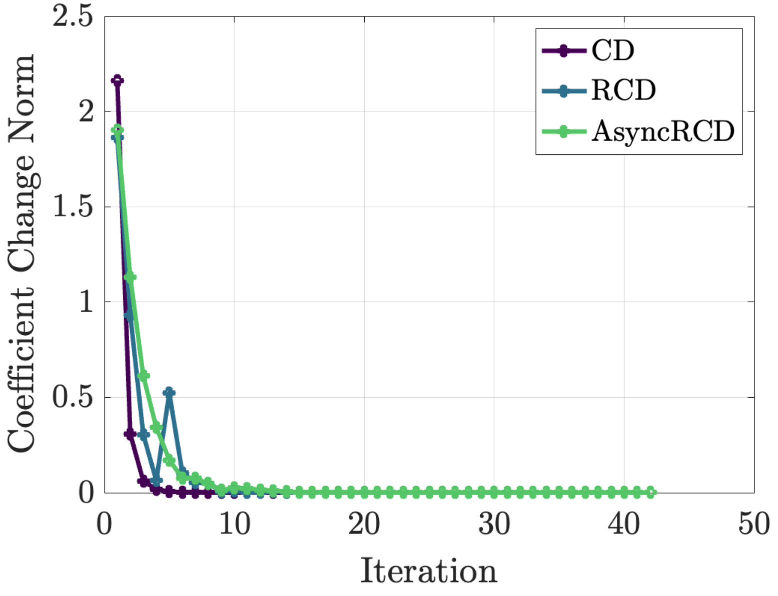
2 Workers



4 Workers



8 Workers



Thank You!

Any Questions?

PURDUE UNIVERSITY®