# Parallel Coordinate Optimization

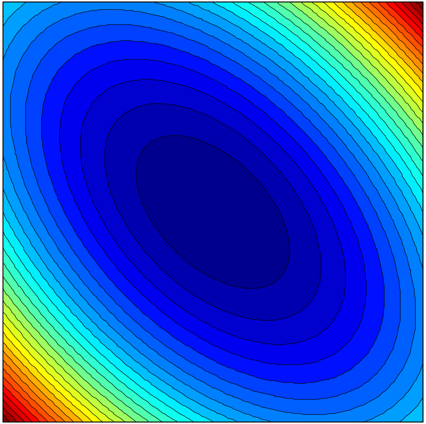
Julie Nutini

MLRG - Spring Term

March $6^{\text{th}}$, 2018

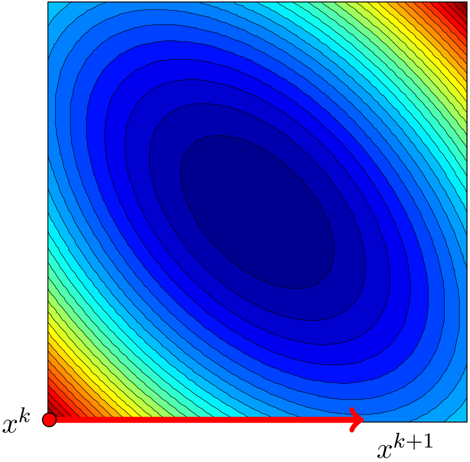
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



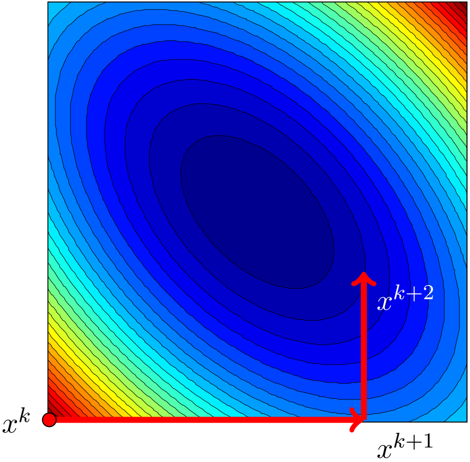
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



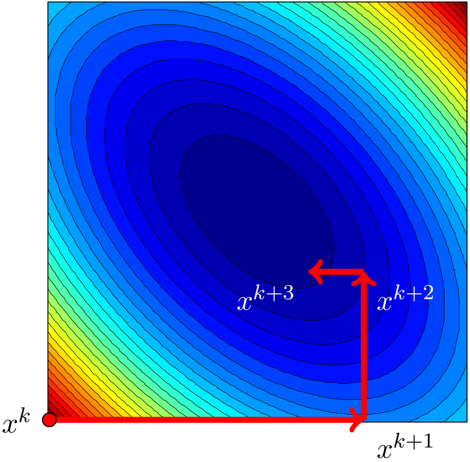
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



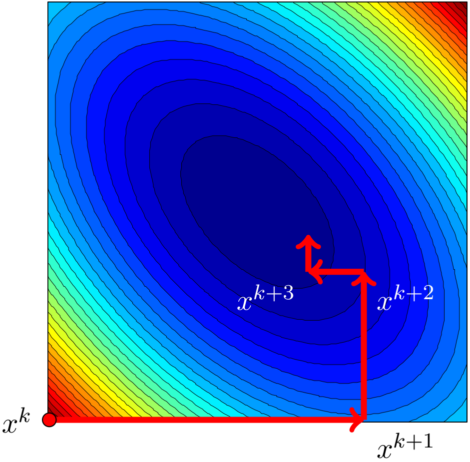
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



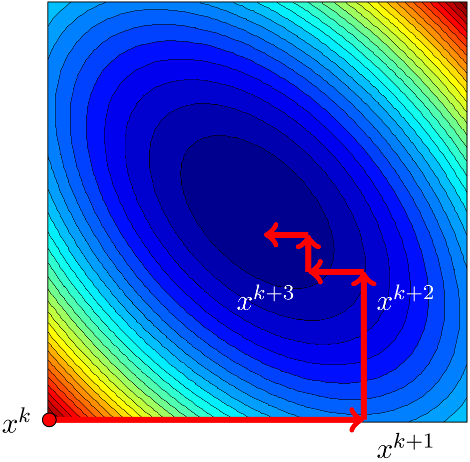
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



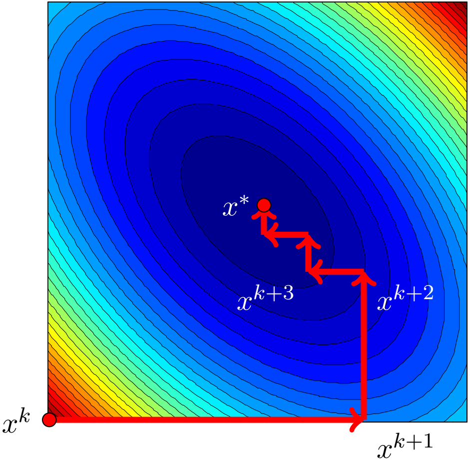
# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



# Coordinate Descent in 2D

- Contours of a function $F : \mathbb{R}^2 \to \mathbb{R}$.
- **Goal**: Find the minimizer of $F$.



# Coordinate Descent

- Update a single coordinate at each iteration,

$$x_i^{k+1} \leftarrow x_i^k - \alpha_k \nabla f_i(x^k)$$

- Easy to implement, low memory requirements, cheap iteration costs.
- Suitable for large-scale optimization (dimension $n$ is large):
  - Certain smooth (unconstrained) problems.
  - Non-smooth problems with *separable* constraints/regularizers.
    - e.g., $\ell_1$-regularization, bound constraints

✶ Faster than gradient descent if iterations $n$ times cheaper.

→ Adaptable to distributed settings.

→ For truly huge-scale problems, it is absolutely necessary to parallelize.

# Problem

- Consider the optimization problem

$$\min_{x \in \mathbb{R}^n} F(x) := f(x) + g(x),$$

where

  - $f$ is loss function – convex (smooth or nonsmooth)
  - $g$ is regularizer – convex (smooth or nonsmooth), separable

# Regularizer Examples

$$g(x) = \sum_{i=1}^{n} g_i(x_i), \quad x = (x_1, x_2, \ldots, x_n)^T$$

- **No regularizer**: $g_i(x_i) \equiv 0$
- **Weighted L1-norm**: $g_i(x_i) = \lambda_i |x_i| \quad (\lambda_i > 0)$ $\leftarrow$ e.g., LASSO
- **Weighted L2-norm**: $g_i(x_i) = \lambda_i (x_i)^2 \quad (\lambda_i > 0)$
- **Box constraints**: $g_i(x_i) = \begin{cases} 0, & x_i \in X_i, \\ +\infty, & \text{otherwise.} \end{cases}$ $\leftarrow$ e.g., SVM dual

# Loss Examples

| Name | $f(x)$ | References |
|---|---|---|
| Quadratic loss | $\frac{1}{2}\|\|Ax-y\|\|_2^2 = \frac{1}{2}\sum_{j=1}^{m}(A_{j:}x-y_j)^2$ | Bradley et al., 2011, |
| Logistic loss | $\sum_{j=1}^{m}\log(1+\exp(-y_jA_{j:}x))$ | Richtárik & Takáč, 2011b, 2013a, |
| Square hinge loss | $\frac{1}{2}(\max\{0,1-y_jA_{j:}x\})^2)$ | Takáč et al., 2013 |
| | | |
| L-infinity | $\\|Ax-y\\|_\infty = \max_{1\le j\le m}\|A_{j:}x-y_j\|$ | |
| L1-regression | $\\|Ax-y\\|_1 = \sum_{j=1}^{m}\|A_{j:}x-y_j\|$ | Fercoq & Richtárik, 2013 |
| Exponential loss | $\log\left(\frac{1}{m}\sum_{j=1}^{m}\exp(y_jA_{j:}x)\right)$ | |

# Parallel Coordinate Descent

- Embarrassingly parallel if objective is separable.
  - $\rightarrow$ Speedup equal to number of processors, $\tau$.
- For partially-separable objectives:
  - Assign $i^{\text{th}}$ processor task of updating $i^{\text{th}}$ component of $x$.
  - Each processor communicates respective $x_i^+$ to processors that require it.
  - The $i^{\text{th}}$ processor needs current value of $x_j$ only if $\nabla_i f$ or $\nabla_{ii}^2 f$ depends on $x_j$.

$\rightarrow$ Parallel implementations suitable when dependency graph is sparse.

# Dependency Graph

- Given a fixed **serial** ordering of updates, those in red can be done in parallel.

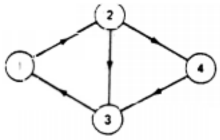



Fig. 1. A dependency graph.

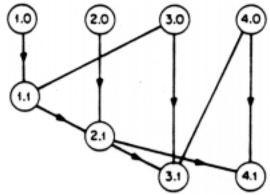



Fig. 2. The data dependencies in a Gauss–Seidel iteration.

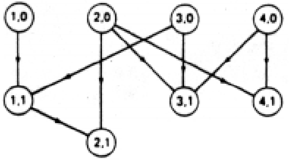



Fig. 3. The data dependencies in a Gauss–Seidel iteration for a different updating order.

$\rightarrow$ $(i,j)$ is an arc of the dependency graph iff update function $h_j$ depends on $x_i$

**Update order**: $\{1,2,3,4\}$

$$x_1^{k+1} = h_1(x_1^k, x_3^k)$$
$$x_2^{k+1} = h_2(x_1^{k+1}, x_2^k)$$
$$x_3^{k+1} = h_3(x_2^{k+1}, x_3^k, x_4^k)$$
$$x_4^{k+1} = h_4(x_2^{k+1}, x_4^k)$$

**Better update order**: $\{1,3,4,2\}$

$$x_1^{k+1} = h_1(x_1^k, x_3^k)$$
$$x_3^{k+1} = h_3(x_2^k, x_3^k, x_4^k)$$
$$x_4^{k+1} = h_4(x_2^k, x_4^k)$$
$$x_2^{k+1} = h_2(x_1^{k+1}, x_2^k)$$

# Parallel Coordinate Descent

- **Synchronous parallelism**:
  - Divide iterate updates between processors, followed by synchronization step.
  - Very slow for large-scale problems (wait for slowest processor).
- **Asynchronous parallelism**:
  - Each processor has access to $x$, chooses index $i$, loads components of $x$ that are needed to compute the gradient component $\nabla_i f(x)$, then updates the $i$th component $x_i$.
    - No attempt to coordinate or synchronize with other processors.
    - Always using 'stale' $x$: convergence results restrict how stale.

$\rightarrow$ Many numerical results actually use asynchronous implementation, ignore synchronization step required by theory.

# Totally Asynchronous Algorithm

**Definition**: An algorithm is *totally asynchronous* if

1. each index $i \in \{1, 2, \dots, n\}$ of $x$ is updated at infinitely many iterations, and
2. if $\nu_j^k$ denotes the iteration at which component $j$ of the vector $\hat{x}^k$ was last updated, then $\nu_j^k \to \infty$ as $k \to \infty$ for all $j = 1, 2, \dots, n$.

**Algorithm 7** Asynchronous Coordinate Descent for (1)

Set $k \leftarrow 0$ and choose $x^0 \in \mathbb{R}^n$;
**repeat**
  Choose index $i_k \in \{1, 2, \dots, n\}$;
  $x^{k+1} \leftarrow x^k - \alpha_k [\nabla f(\hat{x}^k)]_{i_k} e_{i_k}$ for some $\alpha_k > 0$;
  $k \leftarrow k + 1$;
**until** termination test satisfied;

$\rightarrow$ No condition on how stale $\hat{x}_j$ is, just requires that it **will** be updated eventually.

## Theorem (Bertsekas and Tsitsiklis, 1989)

*Suppose a mapping* $T(x) := x - \alpha \nabla f(x)$ *for some* $\alpha > 0$ *satisfies*

$$\|T(x) - x^*\|_\infty \leq \eta \|x - x^*\|_\infty, \quad \textit{for some } \eta \in (0, 1).$$

*Then if we set* $\alpha_k \equiv \alpha$ *in Algorithm 7, the sequence* $\{x^k\}$ *converges to* $x^*$.

# Partly Asynchronous Algorithm

- No convergence rate for totally asynchronous, given weak assumptions on $\hat{x}^k$.
- $\ell_\infty$ contraction assumption on mapping $T$ is quite strong.

- Liu et al. (2015) assume no component of $\hat{x}^k$ older than nonnegative integer $\tilde{\tau}$ (maximum delay) at any $k$.
  - $\tilde{\tau}$ related to number of processors $\tau$ (indicator of potential parallelism)
  - If all processors complete updates at approx same rate, $\tilde{\tau} \approx c\tau$ for some positive integer $c$.

$\rightarrow$ Linear convergence if "essential strong convexity" holds.

$\rightarrow$ Sublinear convergence for general convex functions.

$\rightarrow$ Near-linear speedup if number of processors is:

- $O(n^{1/2})$ in unconstrained optimization.
- $O(n^{1/4})$ in the separable-constrained case.

# Question

Under what structural assumptions does parallelization lead to acceleration?

# Convergence of Randomized Coordinate Descent

- In $\mathbb{R}^n$, randomized coordinate descent with uniform selection requires:

$$O(n \times \xi(\epsilon)) \quad \text{iterations}$$

- **Strong convex** $F$: $\xi(\epsilon) = \log\left(\frac{1}{\epsilon}\right)$
- **Smooth, or simple nonsmooth** $F$: $\xi(\epsilon) = \frac{1}{\epsilon}$
- **'Difficult' nonsmooth** $F$: $\xi(\epsilon) = \frac{1}{\epsilon^2}$

→ When dealing with big data, we only care about $n$.

# The Parallelization Dream

| **Serial** | | **Parallel** |
|---|---|---|
| (1 coordinate per iteration) | | ($\tau$ coordinates per iteration) |
| $O(n \times \xi(\epsilon))$ iterations | $\Rightarrow$ | $O\left(\frac{n}{\tau} \times \xi(\epsilon)\right)$ iterations |

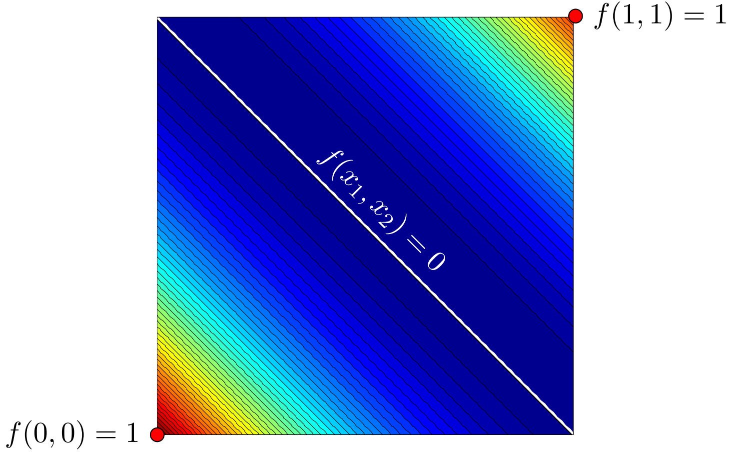
- What do we actually get?

$$O\left(\frac{n\beta}{\tau} \times \xi(\epsilon)\right)$$

- Want $\beta = O(1)$.
  - $\rightarrow$ Depends on extent to which we can add up individual updates.
  - $\rightarrow$ Properties of $F$, select of coordinates at each iteration.

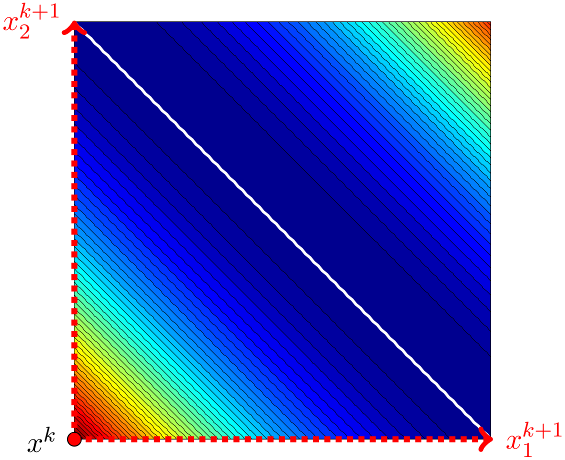
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.



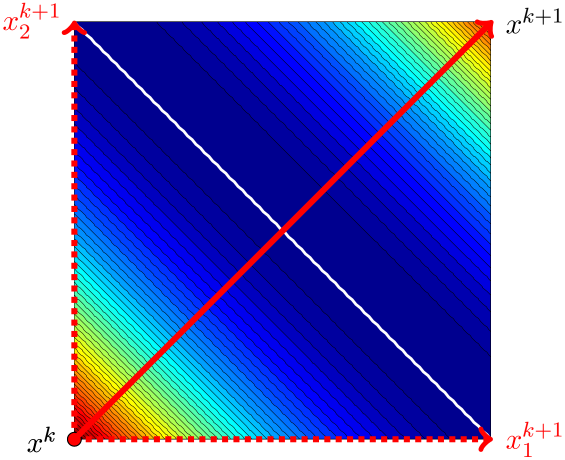
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.



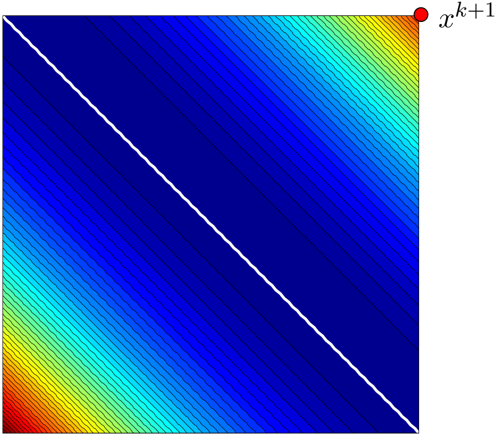
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.



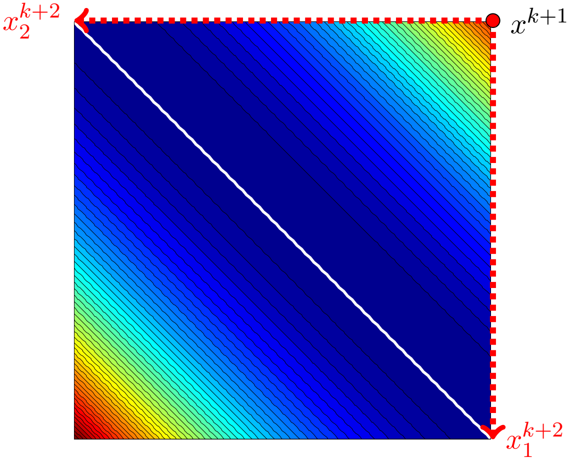
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.



# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.

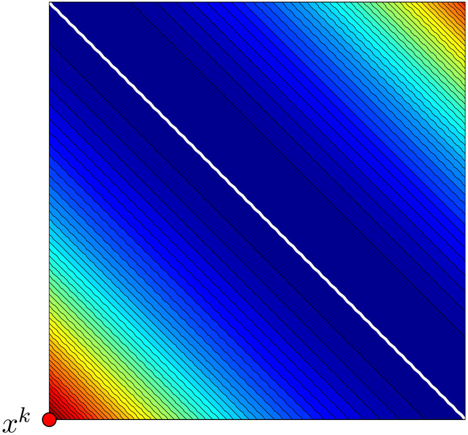


# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- Just compute for more/all coordinates and then add up the updates.

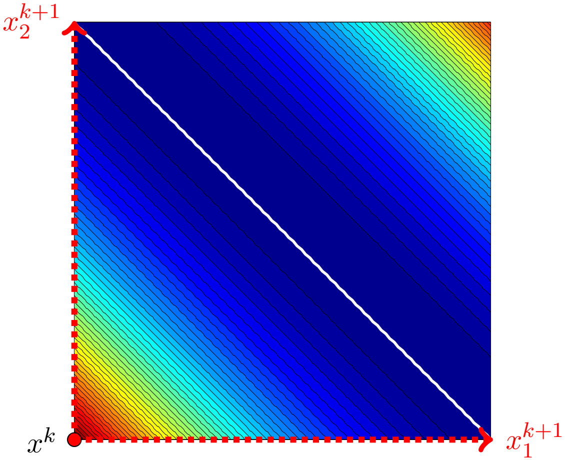
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- What about averaging the updates??



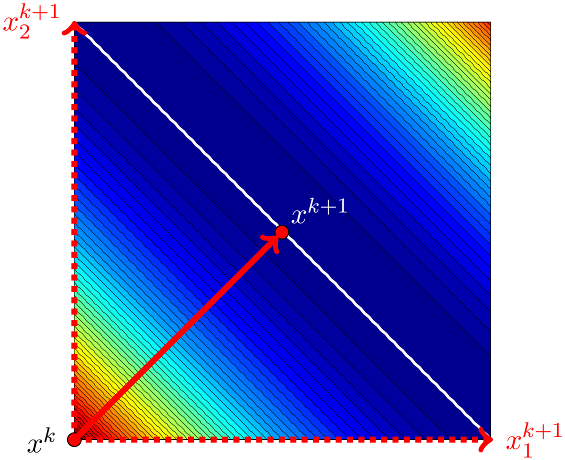
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- What about averaging the updates??



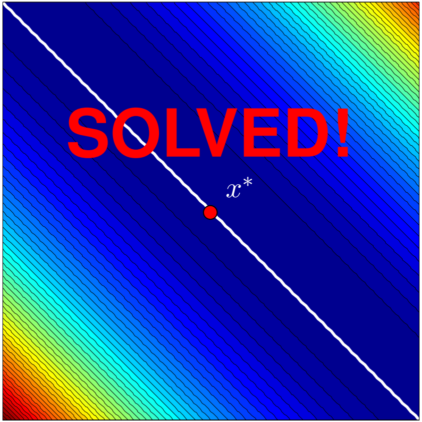
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- What about averaging the updates??



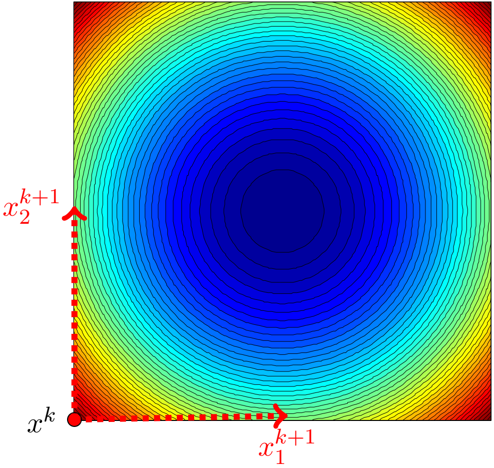
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 + x_2 - 1)^2$
- What about averaging the updates??



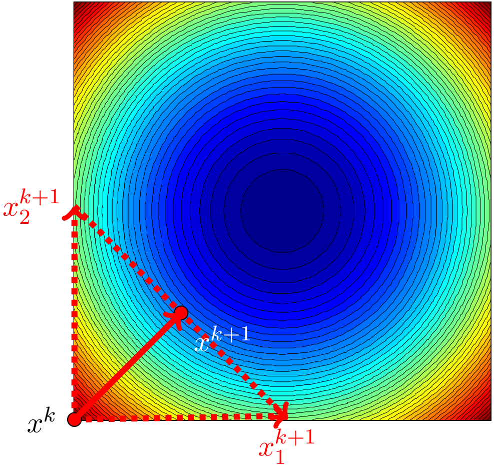
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$
- What about averaging the updates?? COULD BE TOO CONSERVATIVE...



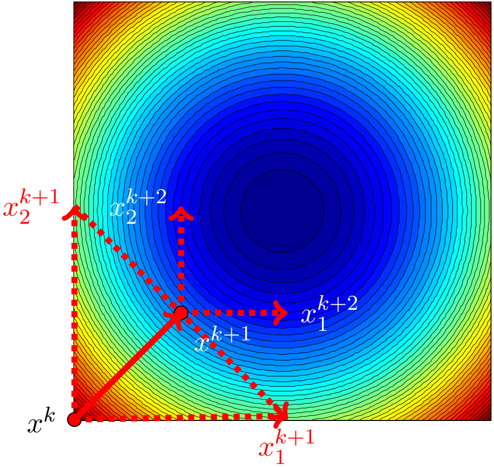
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$
- What about averaging the updates?? COULD BE TOO CONSERVATIVE...



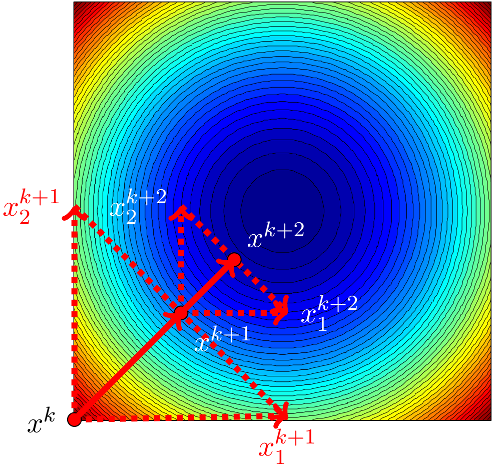
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$
- What about averaging the updates?? COULD BE TOO CONSERVATIVE...



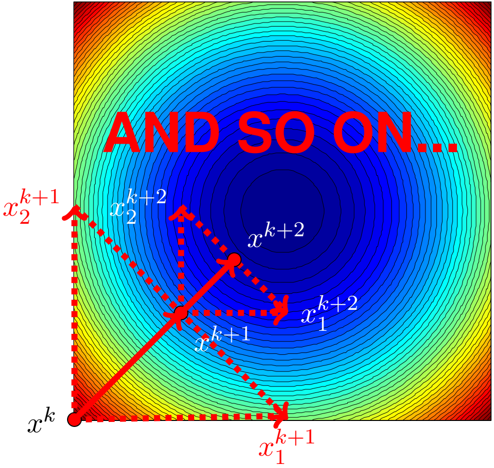
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$
- What about averaging the updates?? COULD BE TOO CONSERVATIVE...



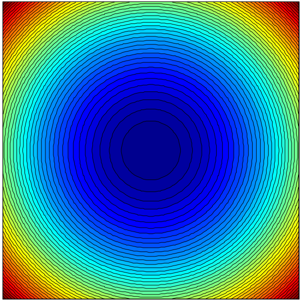
# Naive Parallelization

- Consider the function $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2$
- What about averaging the updates?? COULD BE TOO CONSERVATIVE...



# Averaging may be too conservative...

- Consider the function $f(x) = (x_1 - 1)^2 + (x_2 - 1)^2 + \cdots + (x_n - 1)^2$.
- Evaluate at $x_0 = 0$, $f(x_0) = n$.



- We want

$$f(x^k) = n\left(1 - \frac{1}{n}\right)^{2k} \leq \epsilon.$$

- With averaging, we get

$$k \geq \frac{n}{2}\log\left(\frac{n}{\epsilon}\right) \rightarrow \text{Factor of } n \text{ is bad!}$$

- We wanted $O\left(\frac{n\beta}{\tau} \times \xi(\epsilon)\right)$

# What to do?

- We can write the coordinate descent update as follows,

$$x^+ \leftarrow x + \frac{1}{\beta} \sum_{i=1}^{n} h_i e_i,$$

  where

  - $h_i$ is the update to coordinate $i$
  - $e_i$ is the $i$th unit coordinate vector

- **Averaging**: $\beta = n$
- **Summation**: $\beta = 1$

$\rightarrow$ When can we safely use $\beta \approx 1$?

# When can we use small $\beta$?

- Three models for $f$ with small $\beta$:
  1. Smooth partially separable $f$ [Richtárik & Takáč, 2011b]

$$f(x + te_i) \le f(x) + \nabla f(x)^T (te_i) + \frac{L_i}{2} t^2$$

$$f(x) = \sum_{J \in \mathcal{J}} f_J(x), \quad f_J \text{ depends on } x_i \text{ for } i \in J \text{ only}$$

$\omega := \max_{J \in \mathcal{J}} |J|$

  2. Nonsmooth max-type $f$ [Fercoq & Richtárik, 2013]

$$f(x) = \max_{z \in Q} \{z^T A x - g(z)\}$$

$\omega := \max_{1 \le j \le m} |\{i : A_{ji} \neq 0\}|$

  3. $f$ with 'bounded Hessian' [Bradley et al., 2011, Richtárik & Takáč, 2013a]

$$f(x + h) \le f(x) + \nabla f(x)^T h + \frac{1}{2} h^T A^T A h$$

$L = \mathbf{diag}(A^T A)$

$\sigma := \lambda_{max}(L^{-1/2} A^T A L^{-1/2})$

$\rightarrow$ $\omega$ is the degree of partial separability, $\sigma$ is spectral radius.

# Parallel Coordinate Descent Method

**Algorithm 1** Parallel Coordinate Descent Method 1 (PCDM1)

Choose initial point $x_0 \in \mathbf{R}^N$<br>
**for** $k = 0, 1, 2, \dots$ **do**<br>
  Randomly generate a set of blocks $S_k \subseteq \{1, 2, \dots, n\}$<br>
  $x_{k+1} \leftarrow x_k + (h(x_k))_{[S_k]}$<br>
**end for**

- At iteration $k$, select a random set $S_k$.
- $S_k$ is a realization of a random set-valued mapping (or sampling) $\hat{S}$.
- Update $h_i$ depends on $F$, $x$ and on law describing $\hat{S}$.

$\rightarrow$ Continuously interpolates between serial coordinate descent and gradient.

- Manipulates $n$ and $\mathbb{E}[|\hat{S}|]$.

# ESO: Expected Separable Overapproximation

- We say that $f$ admits a $(\beta, \omega)$-ESO with respect to (uniform) sampling $\hat{S}$ if for all $x, h \in \mathbb{R}^n$

$$(f, \hat{S}) \sim ESO(\beta, w) \iff \mathbb{E}\left[f(x + h_{[\hat{S}]})\right] \leq f(x) + \frac{\mathbb{E}[|\hat{S}|]}{n}\left(\nabla f(x)^T h + \frac{\beta}{2}||h||_w^2\right)$$

  where
  - $h_{[\hat{S}]} = \sum_{i \in \hat{S}} h_i e_i$, and $\|h\|_w^2 := \sum_{i=1}^n w_i (h_i)^2$
- We note that $\nabla f(x)^T h + \frac{\beta}{2}||h||_w^2$ is separable in $h$.
  - Minimize with respect to $h$ in parallel $\to$ yields update

$$x^+ \to x + \frac{1}{\beta} \sum_{i \in \hat{S}} \frac{1}{w_i} \nabla_i f(x) e_i$$

  - Compute updates for $i \in \hat{S}$ only.

$\rightarrow$ Separable quadratic overapproximation of $\mathbb{E}[f]$ evaluated at update.

# Convergence Rate for Convex $f$

- If $(f, \hat{S}) \sim ESO(\beta, w)$, then [Richtárik & Takáč, 2011b]

$$k \geq \left(\frac{\beta n}{\mathbb{E}[|\hat{S}|]}\right)\left(\frac{2R_w^2(x^0, x^*)}{\epsilon}\right)\log\left(\frac{F(x^0) - F^*}{\epsilon\rho}\right),$$

which implies that

$$P(F(x^k) - F^* \leq \epsilon) \geq 1 - \rho.$$

- $\epsilon$: error tolerance
- $n$: # coordinates
- $\mathbb{E}[|\hat{S}|]$: average # updated coordinates per iteration
- $\beta$: step size parameter

# Convergence Rate for Strongly Convex $f$

- If $(f, \hat{S}) \sim ESO(\beta, w)$, then [Richtárik & Takáč, 2011b]

$$k \geq \left(\frac{n}{\mathbb{E}[|\hat{S}|]}\right)\left(\frac{\beta + \mu_g(w)}{\mu_f(w) + \mu_g(w)}\right) \log\left(\frac{F(x^0) - F^*}{\epsilon\rho}\right),$$

which implies that

$$P(F(x^k) - F^* \leq \epsilon) \geq 1 - \rho.$$

  - $\mu_f(w)$: strong convexity constant of loss $f$
  - $\mu_g(w)$: strong convexity constant of regularizer $g$

$\rightarrow$ If $\mu_g(w)$ is large, then the slowdown effect of $\beta$ is eliminated.

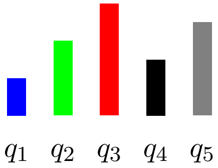
# What if problem is only partially separable?

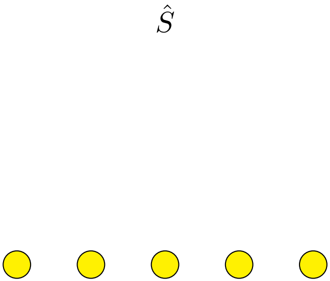
- **Uniform sampling**: $P(\hat{S} = \{i\}) = \frac{1}{n}$
- **$\tau$-nice sampling**: $P(\hat{S} = S) = \begin{cases} \frac{1}{\binom{n}{\tau}}, & |S| = \tau \\ 0, & \text{otherwise} \end{cases}$ $\leftarrow$ for shared memory systems
  - At each iteration:
    - $\rightarrow$ Choose set of $i$, each subset of $\tau$ coordinates chosen with the same probability.
    - $\rightarrow$ Assign each $i$ to a dedicated processor.
    - $\rightarrow$ Compute and apply the update.

$\rightarrow$ All blocks are the same size $\tau$ (otherwise, probability is 0).

# What if problem is only partially separable?

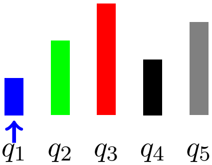
- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.

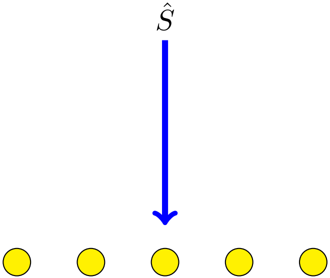


# What if problem is only partially separable?

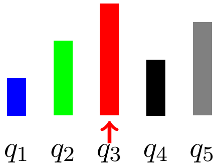
- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.

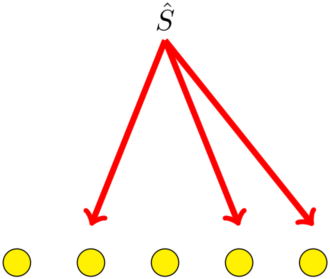


# What if problem is only partially separable?

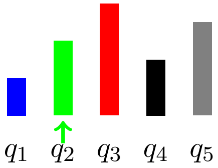
- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.

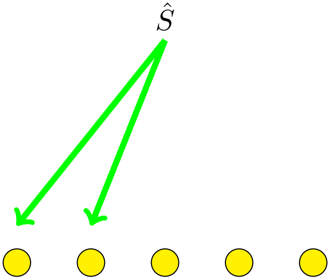


# What if problem is only partially separable?

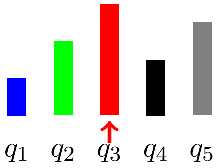
- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.

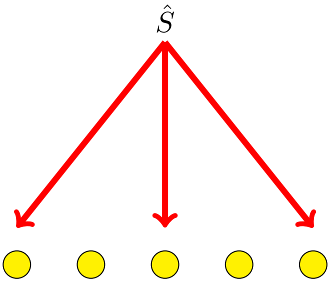


# What if problem is only partially separable?

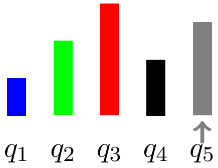
- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.

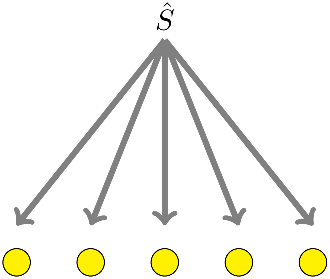


# What if problem is only partially separable?

- **Doubly uniform (DU) sampling**: $P(\hat{S} = S) = \frac{q_{|S|}}{\binom{n}{|S|}}$
  - Generates all sets of equal cardinality with equal probability.
  - Can model unreliable processors/machines.
  - Let $q_\tau = P(|\hat{S}| = \tau)$, with $n = 5$ coordinates.



# What if problem is only partially separable?

- **Binomial sampling**: consider independent equally unreliable processors.
  - Each of $\tau$ processors available with probability $p_b$, busy with probability $1 - p_b$.
  - $\#$ available processors (number of blocks that can be updated in parallel) at each iteration is a binomial random variable with parameters $\tau$ and $p_b$.
  - Use explicit or implicit selection.

# ESO Theory

1. Smooth partially separable $f$ [Richtárik & Takáč, 2011b]

$$f(x + te_i) \leq f(x) + \nabla f(x)^T (te_i) + \frac{L_i}{2} t^2$$

$$f(x) = \sum_{J \in \mathcal{J}} f_J(x), \quad f_J \text{ depends on } x_i \text{ for } i \in J \text{ only}$$

$\omega := \max_{J \in \mathcal{J}} |J|$

**Theorem**: If $\hat{S}$ is **doubly uniform**, then

$$\mathbf{E}\left[f(x + h_{[\hat{S}]})\right] \leq f(x) + \frac{\mathbf{E}[|\hat{S}|]}{n} \left(\nabla f(x)^T h + \frac{\beta}{2} ||h||_w^2\right),$$

where

$$\beta = 1 + \frac{(\omega - 1)\left(\frac{\mathbf{E}[|\hat{S}|^2]}{\mathbf{E}[|\hat{S}|]} - 1\right)}{n - 1}, \quad w_i = L_i. \quad i = 1, 2, \ldots, n$$

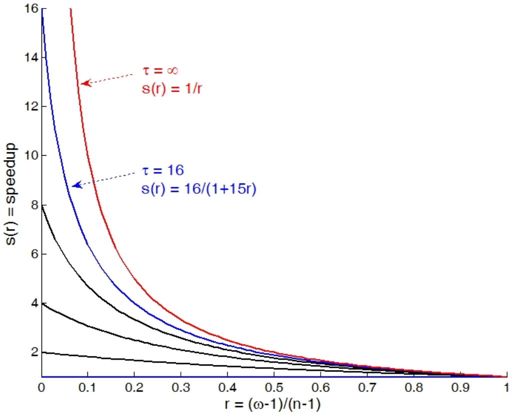
$\rightarrow$ $\beta$ is small if $\omega$ is small (i.e., more separable)

# ESO Theory

| sampling $\hat{S}$ | $\mathbf{E}[\|\hat{S}\|]$ | $\beta$ | $w$ | ESO monotonic? | Follows from |
|---|---|---|---|---|---|
| uniform | $\mathbf{E}[\|\hat{S}\|]$ | $1$ | $\nu \odot L$ | No | Thm 12 |
| nonoverlapping uniform | $\frac{n}{l}$ | $1$ | $\gamma \odot L$ | Yes | Thm 13 |
| doubly uniform | $\mathbf{E}[\|\hat{S}\|]$ | $1 + \frac{(\omega-1)\left(\frac{\mathbf{E}[\|\hat{S}\|^2]}{\mathbf{E}[\|\hat{S}\|]}-1\right)}{\max(1,n-1)}$ | $L$ | No | Thm 15 |
| $\tau$-uniform | $\tau$ | $\min\{\omega,\tau\}$ | $L$ | Yes | Thm 12 |
| $\tau$-nice | $\tau$ | $1 + \frac{(\omega-1)(\tau-1)}{\max(1,n-1)}$ | $L$ | No | Thm 14/15 |
| $(\tau, p_b)$-binomial | $\tau p_b$ | $1 + \frac{p_b(\omega-1)(\tau-1)}{\max(1,n-1)}$ | $L$ | No | Thm 15 |
| serial | $1$ | $1$ | $L$ | Yes | Thm 13/14/15 |
| fully parallel | $n$ | $\omega$ | $L$ | Yes | Thm 13/14/15 |

$\longrightarrow$ (Richtárik & Takáč, 2013) “Parallel Coordinate Descent Methods for Big Data Optimization”.

# Theoretical Speedup



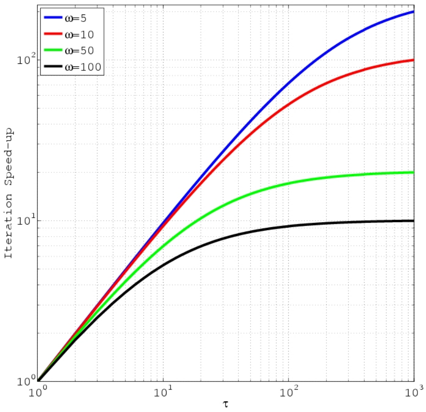

$$s(r) = \frac{\tau}{1 + r(\tau - 1)},$$

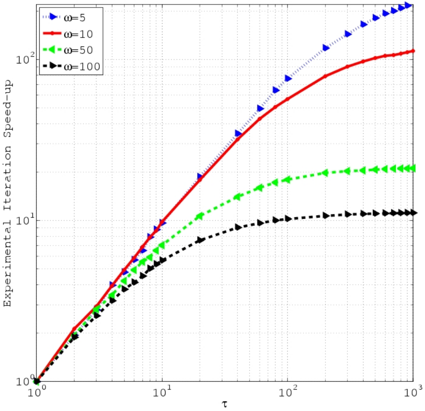
$$r = (\omega - 1)/(n - 1)$$

- $\omega$ often a constant that depends on $n$.
- $r$ is a measure of 'density'.

  $\rightarrow$ MUCH OF BIG DATA IS HERE!

# Theory vs Practice




- $\tau = \#$ processors vs. theoretical (left) and experimental (right) speed-up for $n = 1000$ coordinates

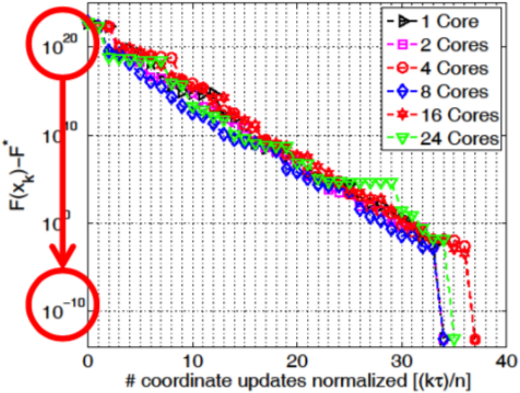
# Experiment

- 1 billion-by-2 billion LASSO problem [Richtárik & Takáč, 2012]

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2, \quad g(x) = \|x\|_1$$

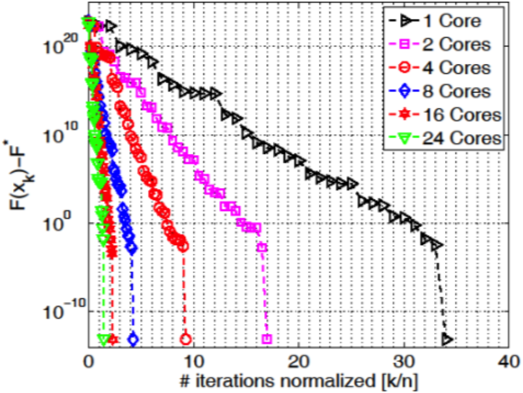
- $A$ has $2 \times 10^9$ rows and $n = 10^9$ columns.
- $\|x^*\|_0 = 10^5$
- $\|A_{:,i}\|_0 = 20$ (column)
- $\max_j \|A_{j,:}\|_0 = 35$ (row) $\Rightarrow \omega = 35$ (degree of partial separability of $f$).
- Used approximation of $\tau$-nice sampling $\hat{S}$ (independent sampling, $\tau << n$).
- Asynchronous implementation.
  - $\rightarrow$ Older information used to update coordinates, but observed slow down is limited.

# Experiment: Coordinate Updates



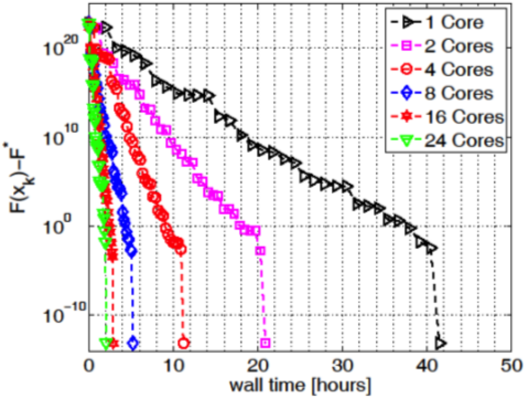

- For each $\tau$, serial and parallel CD need approximately same number of coordinate updates.
- Method identifies active set.

# Experiment: Iterations




- Doubling $\tau$ roughly translates to halving the number of iterations.

# Experiment: Wall Time




- Doubling $\tau$ roughly translates to halving the wall time.

| Citation | Algorithm | Paper |
|---|---|---|
| (Bradley et al, 2011) | Shotgun | **Parallel coordinate descent for $L1$-regularized loss minimization**. *ICML*, 2011 (arXiv: 1105.5379) |
| (Richtárik & Takáč, 2011) | SCD | **Efficient serial and parallel coordinate descent methods for huge-scale truss topology design**. *Operations Research Proceedings*, 27-32, 2012 (Opt Online 08/2011) |
| (Richtárik & Takáč, 2012) | PCDM | **Parallel coordinate descent methods for big data optimization**. *Mathematical Programming*, 2015 (arXiv:1212.0873) |
| (Fercoq & Richtárik, 2013) | SPCDM | **Smooth minimization of nonsmooth functions with parallel coordinate descent method**. 2013 (arXiv:1309.5885) |
| (Richtárik & Takáč, 2013a) | HYDRA | **Distributed coordinate descent method for learning with big data**. 2013 (arXiv:1310.2059) |
| (Liu et al., 2013) | AsySCD | **An asynchronous parallel stochastic coordinate descent algorithm**. *ICML* 2014 (arXiv: 1311.1873) |
| (Fercoq & Richtárik, 2013) | APPROX | **Accelerated, parallel and proximal coordinate descent**. 2013 (arXiv:1312.5799) |
| (Yang, 2013) | DisDCA | **Trading computation for communication: distributed stochastic dual coordinate ascent**. *NIPS* 2013 |
| (Bian et al, 2013) | PCDN | **Parallel coordinate descent Newton method for efficient $\ell_1$-regularized minimization**. 2013 (arXiv:1306.4080) |
| (Liu & Wright, 2014) | AsySPCD | **Asynchronous stochastic coordinate descent: parallelism and convergence properties**. *SIAM J. Optim.* 25(1), 351376, 2015 (arXiv:1403.3862) |
| (Mahajan et al, 2014) | DBCD | **A distributed block coordinate descent method for training $\ell_1$-regularized linear classifiers**. arXiv:1405.4544, 2014 |

| Citation | Algorithm | Paper |
|---|---|---|
| (Fercoq et al, 2014) | Hydra2 | **Fast distributed coordinate descent for non-strongly convex losses**. *MLSP* 2014 (arXiv:1405.5300) |
| (Mareček, Richtárik and Takáč, 2014) | DBCD | **Distributed block coordinate descent for minimizing partially separable functions**. *Numerical Analysis and Opt.*, Springer Proc. in Math. and Stat. (arXiv:1406.0238) |
| (Jaggi, Smith, Takáč et al, 2014) | CoCoA | **Communication-efficient distributed dual coordinate ascent**. *NIPS* 2014 (arXiv: 1409.1458) |
| (Qu, Richtárik & Zhang, 2014) | QUARTZ | **Randomized dual coordinate ascent with arbitrary sampling**. arXiv:1411.5873, 2014 |
| (Ma, Smith, Jaggi et al, 2015) | CoCoA+ | **Adding vs. averaging in distributed primal-dual optimization**. *ICML* 2015 |
| (Tappenden, Takáč & Richtárik, 2015) | PCDM | **On the complexity of parallel coordinate descent**. arXiv:1503.03033, 2015 |
| (Hsieh, Yu & Dhillon, 2015) | PASSCoDe | **PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent**. *ICML*, 2015 |
| (Peng et al, 2016) | ARock | **ARock: an algorithmic framework for asynchronous parallel coordinate updates**. *SIAM J. Sci. Comput.*, 2016 |
| (You et al, 2016) | Asy-GCD | **Asynchronous parallel greedy coordinate descent**. *NIPS*, 2016 |
| ⋮ | ⋮ | ⋮ |

# Conclusions

- Coordinate descent scales very well to big data problems of special structure.
  - $\rightarrow$ Requires (partial) separability/sparse dependency graph.
- Care is needed when combining updates (add them up? average?)
- Sampling strategies that take into account unreliable processors.