Big Data Optimization:
Randomized lock-free methods for minimizing partially separable convex functions

Peter Richtárik

School of Mathematics
The University of Edinburgh

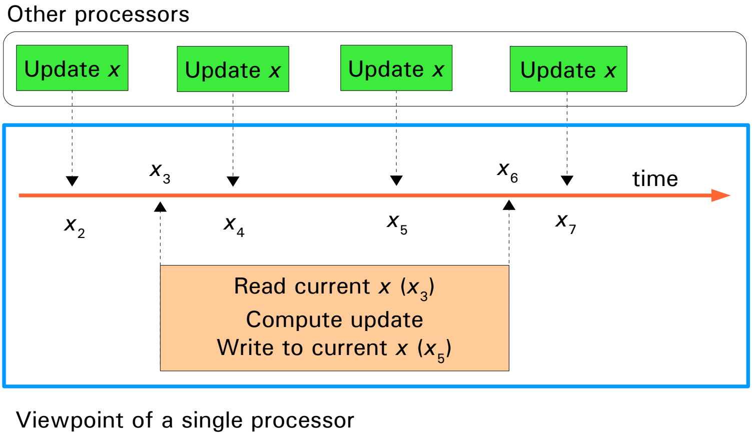
Joint work with Martin Takáč (Edinburgh)

Les Houches ◇ January 11, 2013

Lock-Free (Asynchronous) Updates

Between the time when x is read by any given processor and an update is computed and applied to x by it, other processors apply their updates.

$$x_6 \leftarrow x_5 + update(x_3)$$



Generic Parallel Lock-Free Algorithm

In general:

$$x_{j+1} = x_j + update(x_{r(j)})$$

- $r(j)$ = index of iterate current at reading time
- j = index of iterate current at writing time

Assumption:

$$j - r(j) \leq \tau$$

$$\tau + 1 \approx \# \text{ processors}$$

The Problem and Its Structure

$$\text{minimize}_{x \in \mathbf{R}^{|V|}} \quad [f(x) \equiv \sum_{e \in E} f_e(x)] \qquad (OPT)$$

- Set of vertices/coordinates: V $(x = (x_v,\ v \in V),\ \dim x = |V|)$
- Set of edges: $E \subset 2^V$
- Set of blocks: B (a collection of sets forming a partition of V)
- Assumption: f_e depends on x_v, $v \in e$, only

Example (convex $f : \mathbf{R}^5 \to \mathbf{R}$):

$$f(x) = \underbrace{7(x_1 + x_3)^2}_{f_{e_1}(x)} + \underbrace{5(x_2 - x_3 + x_4)^2}_{f_{e_2}(x)} + \underbrace{(x_4 - x_5)^2}_{f_{e_3}(x)}$$

$$V = \{1,2,3,4,5\}, \quad |V| = 5, \quad e_1 = \{1,3\}, \quad e_2 = \{2,3,4\}, \quad e_3 = \{4,5\}$$

Applications

- structured stochastic optimization (via Sample Average Approximation)
- learning
- sparse least-squares
- sparse SVMs, matrix completion, graph cuts (see Niu-Recht-Ré-Wright (2011))
- truss topology design
- optimal statistical designs

PART 1:

LOCK-FREE HYBRID SGD/RCD METHODS

based on:

P. R. and M. Takáč, Lock-free randomized first order methods, manuscript, 2013.

Problem-Specific Constants

function	definition	average	maximum
Edge-Vertex Degree (# vertices incident with an edge) (relevant if $\lvert B\rvert = \lvert V\rvert$)	$\omega_e = \lvert e\rvert = \lvert\{v \in V : v \in e\}\rvert$	$\bar{\omega}$	ω'
Edge-Block Degree (# blocks incident with an edge) (relevant if $\lvert B\rvert > 1$)	$\sigma_e = \lvert\{b \in B : b \cap e \neq \emptyset\}\rvert$	$\bar{\sigma}$	σ'
Vertex-Edge Degree (# edges incident with a vertex) (not needed!)	$\delta_v = \lvert\{e \in E : v \in e\}\rvert$	$\bar{\delta}$	δ'
Edge-Edge Degree (# edges incident with an edge) (relevant if $\lvert E\rvert > 1$)	$\rho_e = \lvert\{e' \in E : e' \cap e \neq \emptyset\}$	$\bar{\rho}$	ρ'

Remarks:

- **Our results depend on:** $\bar{\sigma}$ (avg Edge-Block degree) and $\bar{\rho}$ (avg Edge-Edge degree)
- First and second row are identical if $|B| = |V|$ (blocks correspond to vertices/coordinates)

Example

$$A = \begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \\ A_4^T \end{bmatrix} = \begin{pmatrix} 5 & 0 & -3 \\ 1.5 & 2.1 & 0 \\ 0 & 0 & 6 \\ .4 & 0 & 0 \end{pmatrix} \in \mathbf{R}^{4\times 3}$$

$$f(x) = \frac{1}{2}\|Ax\|_2^2 = \frac{1}{2}\sum_{i=1}^{4}(A_i^T x)^2, \quad |E| = 4, \quad |V| = 3$$

Example

$$A = \begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \\ A_4^T \end{bmatrix} = \begin{pmatrix} 5 & 0 & -3 \\ 1.5 & 2.1 & 0 \\ 0 & 0 & 6 \\ .4 & 0 & 0 \end{pmatrix} \in \mathbf{R}^{4\times 3}$$

$$f(x) = \frac{1}{2}\|Ax\|_2^2 = \frac{1}{2}\sum_{i=1}^{4}(A_i^T x)^2, \quad |E| = 4, \quad |V| = 3$$

Computation of $\bar{\omega}$ and $\bar{\rho}$:

	v_1	v_2	v_3	ω_{e_i}	ρ_{e_i}
e_1	×		×	2	4
e_2	×	×		2	3
e_3			×	1	2
e_4	×			1	3
δ_{v_j}	3	1	2	$\bar{\omega} = \frac{2+2+1+1}{4} = 1.5,$	$\bar{\rho} = \frac{4+3+2+3}{4} = 3$

$$\omega_e = |e|, \quad \rho_e = |\{e' \in E \; : \; e' \cap e \neq \emptyset\}|, \quad \delta_v = |\{e \in E \; : \; v \in e\}|$$

Algorithm

Iteration $j+1$ looks as follows:

$$\boxed{x_{j+1} = x_j - \gamma|E|\sigma_e \nabla_b f_e(x_{r(j)})}$$

Viewpoint of the processor performing this iteration:

- Pick edge $e \in E$, uniformly at random
- Pick block b intersecting edge e, uniformly at random
- Read current x (enough to read x_v for $v \in e$)
- Compute $\nabla_b f_e(x)$
- Apply update: $x \leftarrow x - \alpha \nabla_b f_e(x)$ with $\alpha = \gamma|E|\sigma_e$ and $\gamma > 0$
- Do not wait (no synchronization!) and start again!

Easy to show that

$$\mathbf{E}[|E|\sigma_e \nabla_b f_e(x)] = \nabla f(x)$$

Main Result

Setup:

- $c =$ strong convexity parameter of f
- $L =$ Lipschitz constant of ∇f
- $\|\nabla f(x)\|_2 \leq M$ for x visited by the method
- Starting point: $x_0 \in \mathbf{R}^{|V|}$
- $0 < \epsilon < \frac{L}{2}\|x_0 - x_*\|_2^2$
- constant stepsize: $\gamma := \frac{c\epsilon}{(\bar{\sigma}+2\tau\bar{\rho}/|E|)L^2M^2}$

Main Result

Setup:

- $c =$ strong convexity parameter of f
- $L =$ Lipschitz constant of ∇f
- $\|\nabla f(x)\|_2 \leq M$ for x visited by the method
- Starting point: $x_0 \in \mathbf{R}^{|V|}$
- $0 < \epsilon < \frac{L}{2}\|x_0 - x_*\|_2^2$
- constant stepsize: $\gamma := \frac{c\epsilon}{(\bar{\sigma}+2\tau\bar{\rho}/|E|)L^2M^2}$

Result: Under the above assumptions, for

$$k \geq \left(\bar{\sigma} + \frac{2\tau\bar{\rho}}{|E|}\right)\frac{LM^2}{c^2\epsilon}\log\left(\frac{L\|x_0 - x_*\|_2^2}{\epsilon} - 1\right),$$

we have

$$\min_{0\leq j\leq k} \mathbf{E}\{f(x_j) - f_*\} \leq \epsilon.$$

Special Cases

General result: $$\underbrace{\left(\bar{\sigma} + \frac{2\tau\bar{\rho}}{|E|}\right)}_{\Lambda} \underbrace{\frac{LM^2}{c^2\epsilon} \log\left(\frac{2L\|x_0 - x_*\|_2}{\epsilon} - 1\right)}_{\text{common to all special cases}}$$

special case	lock-free parallel version of ...	Λ
$\|E\| = 1$	**Randomized Block Coordinate Descent**	$\|B\| + \frac{2\tau}{\|E\|}$
$\|B\| = 1$	**Incremental Gradient Descent** (Hogwild! as implemented)	$1 + \frac{2\tau\bar{\rho}}{\|E\|}$
$\|B\| = \|V\|$	**RAINCODE: RAndomized INcremental COordinate DEscent** (Hogwild! as analyzed)	$\bar{\omega} + \frac{2\tau\bar{\rho}}{\|E\|}$
$\|E\| = \|B\| = 1$	**Gradient Descent**	$1 + 2\tau$

Analysis via a New Recurrence

Let $a_j = \frac{1}{2}\mathbf{E}[\|x_j - x_*\|^2]$

Nemirovski-Juditsky-Lan-Shapiro:

$$a_{j+1} \le (1 - 2c\gamma_j)a_j + \frac{1}{2}\gamma_j^2 M^2$$

Niu-Recht-Ré-Wright (Hogwild!):

$$a_{j+1} \le (1 - c\gamma)a_j + \gamma^2(\sqrt{2}c\omega' M\tau(\delta')^{1/2})a_j^{1/2} + \frac{1}{2}\gamma^2 M^2 Q,$$

where $Q = \omega' + 2\tau\frac{\rho'}{|E|} + 4\omega'\frac{\rho'}{|E|}\tau + 2\tau^2(\omega')^2(\delta')^{1/2}$

R.-Takáč:

$$\boxed{a_{j+1} \le (1 - 2c\gamma)a_j + \frac{1}{2}\gamma^2(\bar{\sigma} + 2\tau\frac{\bar{\rho}}{|E|})M^2}$$

Parallelization Speedup Factor

$$\text{PSF} = \frac{\Lambda \text{ of serial version}}{(\Lambda \text{ of parallel version})/\tau} = \frac{\bar{\sigma}}{(\bar{\sigma} + 2\tau\frac{\bar{\rho}}{|E|})/\tau} = \boxed{\frac{1}{\frac{1}{\tau} + \frac{2\bar{\rho}}{\bar{\sigma}|E|}}}$$

Parallelization Speedup Factor

$$\text{PSF} = \frac{\Lambda \text{ of serial version}}{(\Lambda \text{ of parallel version})/\tau} = \frac{\bar{\sigma}}{(\bar{\sigma} + 2\tau\frac{\bar{\rho}}{|E|})/\tau} = \boxed{\frac{1}{\frac{1}{\tau} + \frac{2\bar{\rho}}{\bar{\sigma}|E|}}}$$

Three modes:

- Brute force (many processors; τ very large):

$$\text{PSF} \approx \frac{\bar{\sigma}|E|}{2\bar{\rho}}$$

- Favorable structure ($\frac{\bar{\rho}}{\bar{\sigma}|E|} \ll \frac{1}{\tau}$; fixed τ):

$$\text{PSF} \approx \tau$$

- Special τ ($\tau = \frac{|E|}{\bar{\rho}}$):

$$\text{PSF} = \frac{|E|}{\bar{\rho}} \frac{\bar{\sigma}}{\bar{\sigma} + 2} \approx \tau$$

Improvements vs Hogwild!

If $|B| = |V|$ (blocks = coordinates), then **our method coincides with Hogwild!** (as analyzed in Niu et al), up to stepsize choice:

$$\boxed{x_{j+1} = x_j - \gamma |E| \omega_e \nabla_v f_e(x_{r(j)})}$$

Improvements vs Hogwild!

If $|B| = |V|$ (blocks = coordinates), then **our method coincides with Hogwild!** (as analyzed in Niu et al), up to stepsize choice:

$$\boxed{x_{j+1} = x_j - \gamma |E| \omega_e \nabla_v f_e(x_{r(j)})}$$

Niu-Recht-Ré-Wright (Hogwild!, 2011):

$$\Lambda = 4\omega' + 24\tau \frac{\rho'}{|E|} + 24\tau^2 \omega' (\delta')^{1/2}$$

R.-Takáč:

$$\Lambda = \bar{\omega} + 2\tau \frac{\bar{\rho}}{|E|}$$

Improvements vs Hogwild!

If $|B| = |V|$ (blocks = coordinates), then **our method coincides with Hogwild!** (as analyzed in Niu et al), up to stepsize choice:

$$\boxed{x_{j+1} = x_j - \gamma |E| \omega_e \nabla_v f_e(x_{r(j)})}$$

Niu-Recht-Ré-Wright (Hogwild!, 2011):

$$\Lambda = 4\omega' + 24\tau \frac{\rho'}{|E|} + 24\tau^2 \omega' (\delta')^{1/2}$$

R.-Takáč:

$$\Lambda = \bar{\omega} + 2\tau \frac{\bar{\rho}}{|E|}$$

Advantages of our approach:

- Dependence on averages and not maxima! ($\omega' \to \bar{\omega}$, $\rho' \to \bar{\rho}$)
- Better constants ($4 \to 1$, $24 \to 2$)
- The third **large term** is **not present** (no dependence on τ^2 and δ')
- Introduction of blocks ($\Rightarrow$ cover also block coordinate descent, gradient descent, SGD)
- Simpler analysis

Modified Algorithm: Global Reads and Local Writes*

Partition vertices (coordinates) into $\tau + 1$ blocks

$$V = b_1 \cup b_2 \cup \cdots \cup b_{\tau+1}$$

and assign block b_i to processor i, $i = 1, 2, \ldots, \tau + 1$.

Processor i will (asynchronously) do:

- Pick edge $e \in \{e' \in E \ : \ e' \cap b_i \neq \emptyset\}$, uniformly at random (edge intersecting with block owned by processor i)
- Update:

$$\boxed{x_{j+1} = x_j - \alpha \nabla_{b_i} f_e(x_{r(j)})}$$

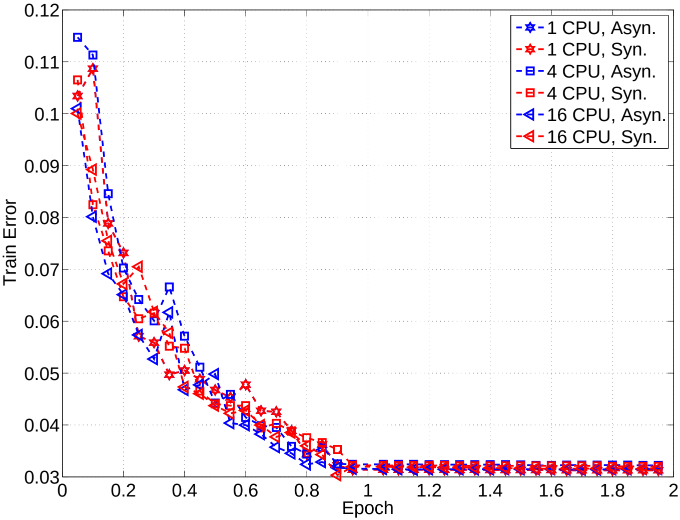
Pros and cons:

- $+$ good if global reads and local writes are cheap, but global writes are expensive (NUMA = Non Uniform Memory Access)
- - do not have an analysis

* Idea proposed by Ben Recht.

Experiment 1: rcv

size = 1.2 GB, features = $|V|$ = 47,236, training: $|E|$ = 677,399, testing: 20,242



Experiment 2

Artificial problem instance:

$$\text{minimize} \quad f(x) = \tfrac{1}{2}\|Ax\|^2 = \sum_{i=1}^{m} \tfrac{1}{2}(A_i^T x)^2.$$

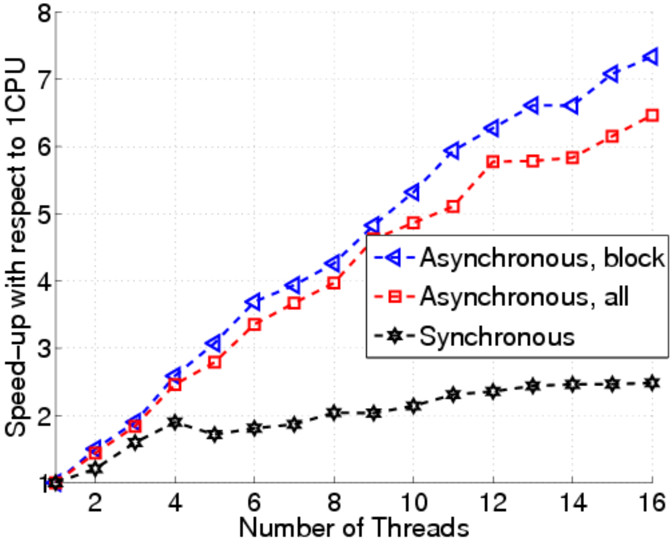
$$A \in \mathbf{R}^{m\times n}; \qquad m = |E| = 500,000; \qquad n = |V| = 50,000$$

Three methods:

- **Synchronous, all** = parallel synchronous method with $|B| = 1$
- **Asynchronous, all** = parallel asynchronous method with $|B| = 1$
- **Asynchronous, block** = parallel asynchronous method with $|B| = \tau$
(no need for atomic operations $\Rightarrow$ additional speedup)

We measure elapsed time needed to perform $20m$ iterations (20 epochs)

Uniform instance: $|e| = 10$ for all edges



PART 2:

PARALLEL BLOCK COORDINATE DESCENT

based on:

P. R. and M. Takáč, Parallel coordinate descent methods for big data optimization, arXiv:1212:0873, 2012.

Overview

- A rich family of synchronous parallel block coordinate descent methods

Overview

- A rich family of synchronous parallel block coordinate descent methods
- Theory and algorithms work for convex composite functions with block-separable regularizer:

$$\text{minimize:} \;\; F(x) \equiv \underbrace{\sum_{e \in E} f_e(x)}_{f} + \lambda \underbrace{\sum_{b \in B} \Psi_b(x)}_{\Psi} .$$

  - Decomposition $f = \sum_{e \in E} f_e$ does not need to be known!
  - f: convex or strongly convex (complexity for both)

Overview

- A rich family of synchronous parallel block coordinate descent methods
- Theory and algorithms work for convex composite functions with block-separable regularizer:

$$\text{minimize:} \quad F(x) \equiv \underbrace{\sum_{e \in E} f_e(x)}_{f} + \lambda \underbrace{\sum_{b \in B} \Psi_b(x)}_{\Psi}.$$

  - Decomposition $f = \sum_{e \in E} f_e$ does not need to be known!
  - f: convex or strongly convex (complexity for both)
- All parameters for running the method according to theory are easy to compute:
  - block Lipschitz constants $L_1, \dots, L_{|B|}$
  - ω'

ACDC: Lock-Free Parallel Coordinate Descent C++ code

http://code.google.com/p/ac-dc/

Can solve a LASSO problem with

- $|V| = 10^9$,
- $|E| = 2 \times 10^9$,
- $\omega' = 35$,
- on a machine with $\tau = 24$ processors,
- to $\epsilon = 10^{-14}$ accuracy,
- in 2 hours,
- starting with initial gap $\approx 10^{22}$.

Complexity Results

First complexity analysis of parallel coordinate descent:

$$\mathbf{P}(F(x_k) - F^* \leq \epsilon) \geq 1 - p$$

- Convex functions:

$$k \geq \left(\frac{2\beta}{\alpha}\right)\frac{\|x_0 - x_*\|_L^2}{\epsilon} \log \frac{F(x_0) - F^*}{\epsilon p}$$

- Strongly convex functions (with parameters μ_f and μ_Ψ):

$$k \geq \frac{\beta + \mu_\Psi}{\alpha(\mu_f + \mu_\Psi)} \log \frac{F(x_0) - F^*}{\epsilon p}$$

- Leading constants matter!

Parallelization Speedup Factors

Closed-form formulas for parallelization speedup factors (PSFs):

- PSFs are functions of ω', τ and $|B|$, and depend on sampling
- Example 1: fully parallel sampling (all blocks are updated, i.e., $\tau = |B|$):

$$PSF = \frac{|B|}{\omega'}.$$

- Example 2: τ-nice sampling (*all subsets* of τ blocks are chosen with the same probability):

$$PSF = \frac{\tau}{1 + \frac{(\omega'-1)(\tau-1)}{|B|-1}}.$$

A Problem with Billion Variables

LASSO problem:

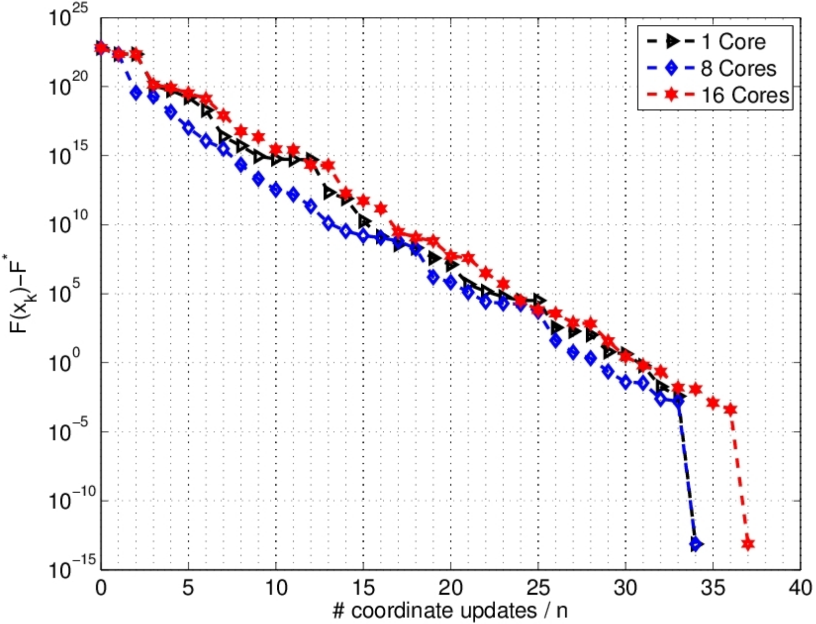
$$F(x) = \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1$$

The instance:

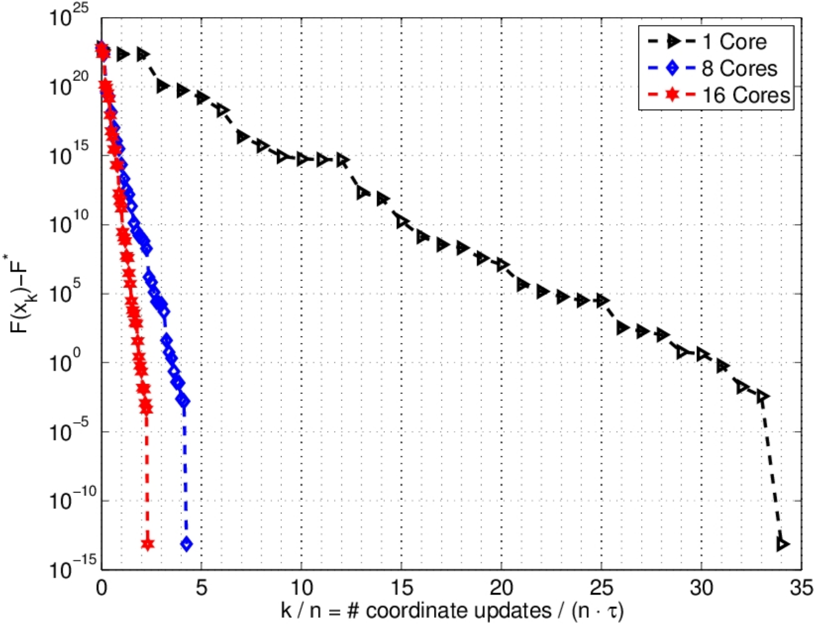
- A has
  - $|E| = m = 2 \times 10^9$ rows
  - $|V| = n = 10^9$ columns (= # of variables)
  - exactly 20 nonzeros in each column
  - on average 10 and at most 35 nonzeros in each row ($\omega' = 35$)
- optimal solution x^* has 10^5 nonzeros
- $\lambda = 1$

Solver: Asynchronous parallel coordinate descent method with independent nice sampling and $\tau = 1, 8, 16$ cores

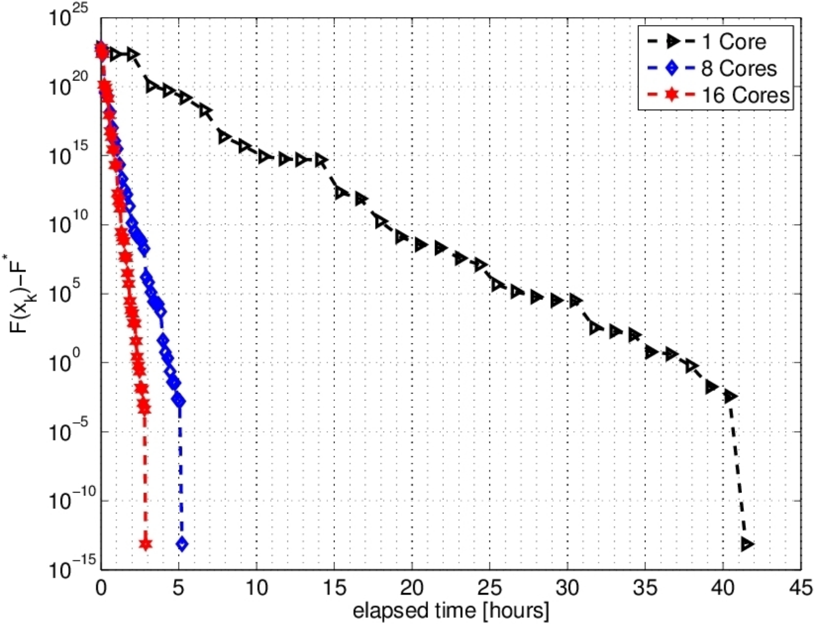
Coordinate Updates / n



Iterations / n



Wall Time



10^{25}
10^{20}
10^{15}
10^{10}
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
$F(x_k)-F^*$
0
5
10
15
20
25
30
35
40
45
elapsed time [hours]
1 Core
8 Cores
16 Cores

Billion Variables — 1 Core

k/n	$F(x_k) - F^*$	$\|x_k\|_0$	time [hours]
0	$< 10^{23}$	0	0.00
3	$< 10^{21}$	451,016,082	3.20
4	$< 10^{20}$	583,761,145	4.28
6	$< 10^{19}$	537,858,203	6.64
7	$< 10^{17}$	439,384,488	7.87
8	$< 10^{16}$	329,550,078	9.15
9	$< 10^{15}$	229,280,404	10.43
13	$< 10^{13}$	30,256,388	15.35
14	$< 10^{12}$	16,496,768	16.65
15	$< 10^{11}$	8,781,813	17.94
16	$< 10^{10}$	4,580,981	19.23
17	$< 10^{9}$	2,353,277	20.49
19	$< 10^{8}$	627,157	23.06
21	$< 10^{6}$	215,478	25.42
23	$< 10^{5}$	123,788	27.92
26	$< 10^{3}$	102,181	31.71
29	$< 10^{1}$	100,202	35.31
31	$< 10^{0}$	100,032	37.90
32	$< 10^{-1}$	100,010	39.17
33	$< 10^{-2}$	100,002	40.39
34	$< 10^{-13}$	100,000	41.47

Billion Variables — 1, 8 and 16 Cores

	$F(x_k) - F^*$			Elapsed Time		
$(k \cdot \tau)/n$	1 core	8 cores	16 cores	1 core	8 cores	16 cores
0	6.27e+22	6.27e+22	6.27e+22	0.00	0.00	0.00
1	2.24e+22	2.24e+22	2.24e+22	0.89	0.11	0.06
2	2.25e+22	3.64e+19	2.24e+22	1.97	0.27	0.14
3	1.15e+20	1.94e+19	1.37e+20	3.20	0.43	0.21
4	5.25e+19	1.42e+18	8.19e+19	4.28	0.58	0.29
5	1.59e+19	1.05e+17	3.37e+19	5.37	0.73	0.37
6	1.97e+18	1.17e+16	1.33e+19	6.64	0.89	0.45
7	2.40e+16	3.18e+15	8.39e+17	7.87	1.04	0.53
⋮	⋮	⋮	⋮	⋮	⋮	⋮
26	3.49e+02	4.11e+01	3.68e+03	31.71	3.99	2.02
27	1.92e+02	5.70e+00	7.77e+02	33.00	4.14	2.10
28	1.07e+02	2.14e+00	6.69e+02	34.23	4.30	2.17
29	6.18e+00	2.35e-01	3.64e+01	35.31	4.45	2.25
30	4.31e+00	4.03e-02	2.74e+00	36.60	4.60	2.33
31	6.17e-01	3.50e-02	6.20e-01	37.90	4.75	2.41
32	1.83e-02	2.41e-03	2.34e-01	39.17	4.91	2.48
33	3.80e-03	1.63e-03	1.57e-02	40.39	5.06	2.56
34	7.28e-14	7.46e-14	1.20e-02	41.47	5.21	2.64
35	-	-	1.23e-03	-	-	2.72
36	-	-	3.99e-04	-	-	2.80
37	-	-	7.46e-14	-	-	2.87

References

P. R. and M. Takáč, Lock-free randomized first order methods, arXiv:1301:xxxx.

P. R. and M. Takáč, Parallel coordinate descent methods for big data optimization, arXiv:1212:0873, 2012.

P. R. and M. Takáč, Iteration complexity of block coordinate descent methods for minimizing a composite function, Mathematical Programming, Series A, 2013.

P. R. and M. Takáč, Efficient serial and parallel coordinate descent methods for huge-scale truss topology design, Operations Research Proceedings, 2012.

F. Niu, B. Recht, C. Ré, and S. Wright, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, NIPS 2011.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, Robust stochastic approximation approach to stochastic programming, SIAM J. Opt., (4):1574–1609, 2009.

M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent, NIPS 2010.

Yu. Nesterov, Efficiency of coordinate descent methods on huge-scale optimization problems, SIAM J. Opt. 22(2):341–362, 2012.