Distributed Sparse Optimization

Wotao Yin

UCLA Math Dept.

2013/9/18 – Peking University

Zhimin Peng, Ming Yan, Wotao Yin. *Parallel and Distributed Sparse Optimization*, 2013

Optimization meets big data: Background

- Tech. advances in data gathering

$\Longrightarrow$ rapid proliferation of massive data in many areas:

  - Internet (Google, Baidu, Facebook)
  - business (Walmart, SuNing, JingDong)
  - signal processing
  - climate, cosmology
  - medicine, medical imaging

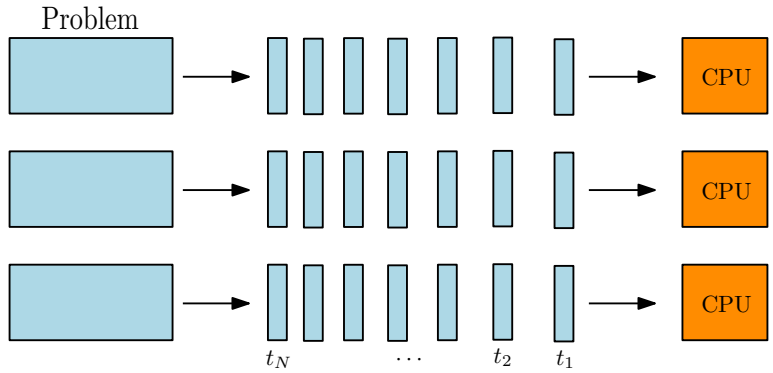
- To make sense of the data, new computational methods must be introduced

- Structured solutions (e.g., sparse vector, low-rank matrix, ...) have grown enormously important in many areas: statistics (LASSO, sparse logistic regression), machine learning (sparse SVM and PCA), image processing (total variation), seismic imaging, compressive sensing, natural language processing, bioinformatics, and many more...

Massive data $\Rightarrow$ parallel & distributed computing

- Many modern applications of sparse optimization has big data
- Many problems cannot be solved on a single workstation

$\Rightarrow$ Moving to parallel/distributed/cloud computing is a viable option



- This talk introduces parallel sparse optimization algorithms running on clusters or cloud computers

The basic idea

To explore two useful structures in the model and data:

1. **Separability**
   - (block)-separable: $f(\mathbf{x}) = \sum_{s=1}^{S} f_s(\mathbf{x}_s)$
   - partial (block)-separable: $f(\mathbf{x}) = \sum_{s=1}^{S} f_s(\mathbf{x})$
   - common, e.g., ℓ_1, $\ell_{1,2}$, Huber, elastic net, square/logistic/hinge loss

   $\Rightarrow$ enables computation *divide and conquer*, well known and widely used
2. **Data is "nearly orthogonal" and solution is sparse**

   $\Rightarrow$ enables only updating a few coordinates each time

   $\Rightarrow$ enables these coordinates are greedily selected
   (seemingly awkward for parallel computing, *but not really*)

   $\Rightarrow$ intermediate points are sparse, cheaper updates, fewer total iterations

Related prior art

- **ADMM** (alternating direction method of multipliers)
  - dates back to the 50', recently becomes popular
  - based on operator-splitting and variable-splitting
  - solves many convex signal processing problems
  - its distributed versions[1] *scale poorly* for big data (#.itr $\sim$ #.splitting)
- **Parallel coordinate descent**[2]
  - update multiple (blocks of) coordinates each time
  - (typically) select coordinates cyclically or at random
  - not tailored for nearly-orthogonal data and sparse solutions

This talk: parallelizes existing 1st-order algorithms and introduce an algorithm

Solve LASSO with 170GB data in 1.7 minutes to 5-digit accuracy under $1 cost

---

[1] Boyd, Parikh, Chu, Peleato, and Eckstein [2011], Mota, Xavier, Aguiar, and Puschel [2012]

[2] Bradley, Kyrola, Bickson, Guestrin, and Guestrin [2011], Scherrer, Halappanavar, Tewari, and Haglin [2012a], Scherrer, Tewari, Halappanavar, and Haglin [2012b]

Model

Many sparse optimization model has the form

$$\min_{\mathbf{x}} \ \lambda \mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$$

- $\mathcal{R}(\mathbf{x})$ encourages $\mathbf{x}^*$ to have structure, often non-smooth but simple
- $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ penalizes data fidelity loss, often differentiable

Consider

- very large $\mathbf{A}$
- one, or both, of $\mathcal{R}$ and $\mathcal{L}$ is (partially) separable

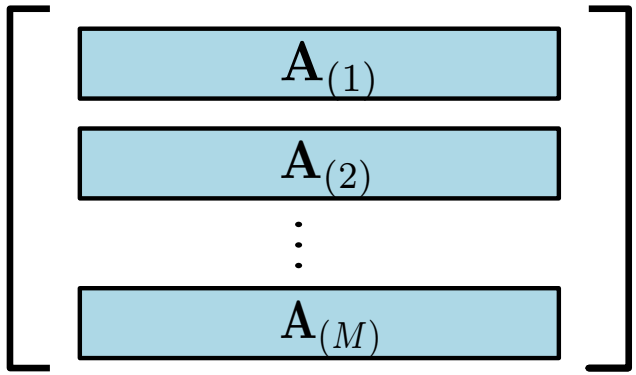
First-order approaches:

- prox-linear: $\mathbf{x}^{k+1} \leftarrow \min \lambda \mathcal{R}(\mathbf{x}) + \langle \mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}), \mathbf{x} \rangle + \frac{1}{2\delta_k} \|\mathbf{x} - \mathbf{x}^k\|^2$
- accelerated prox-linear
- dual / Bregman / linearized Bregman / ...
- .....

Common bottleneck: $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$

Row partition of $\mathbf{A}$:

- let $\mathbf{A}_{(i)}$ be the ith row block of $\mathbf{A}$



- compute $\mathbf{A}\mathbf{x}$: *broadcast* $\mathbf{x}$, if needed, and *parallel* $\mathbf{A}_{(i)}\mathbf{x}$
- compute $\mathbf{A}^T\mathbf{y}$: *parallel* $\mathbf{A}_{(i)}^T\mathbf{y}_{(i)}$ and *reduce* $\mathbf{A}^T\mathbf{y} = \sum_{i=1}^M \mathbf{A}_{(i)}^T\mathbf{y}_{(i)}$

$\Rightarrow$ compute $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b}) = \sum_{i=1}^M \mathbf{A}_{(i)}^T \nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}, \mathbf{b}_i)$:

(1) *broadcast* $\mathbf{x}$, if needed;

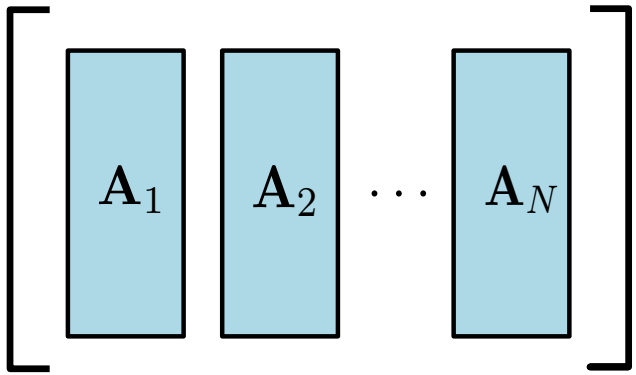
(2) *parallel* $\mathbf{A}_{(i)}^T \nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}, \mathbf{b}_i)$;

(3) *reduce* $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$.

Bottleneck: $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$

Column partition of $\mathbf{A}$:

- let $\mathbf{A}_j$ be the jth column block of $\mathbf{A}$



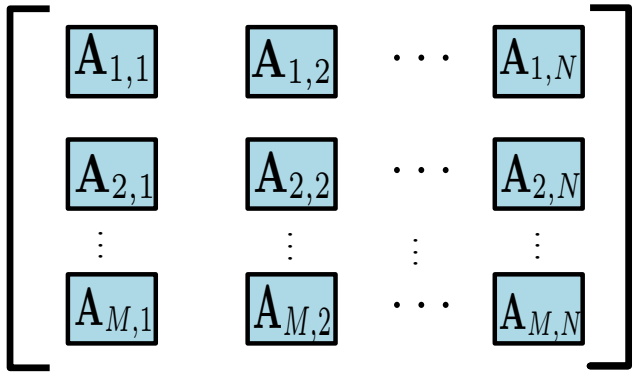
- compute $\mathbf{A}\mathbf{x}$: *parallel* $\mathbf{A}_j\mathbf{x}_j$ and *reduce* $\sum_{j=1}^{N} \mathbf{A}_j\mathbf{x}_j$
- compute $\mathbf{A}^T\mathbf{y}$: *broadcast* $\mathbf{y}$, if needed, and *parallel* $\mathbf{A}_j^T\mathbf{y}$

$\Rightarrow$ compute $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ ..... (left to the reader)

Bottleneck: $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}, \mathbf{b})$

Two-way block partition of $\mathbf{A}$:

- let $\mathbf{A}_{i,j}$ be the (i,j)th block of $\mathbf{A}$



- compute $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$: *mixed use of broadcast, parallel, and reduce*

$\Rightarrow$ compute $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{Ax}, \mathbf{b})$ ..... (left to the reader)

Load balancing:

- blocks can have different sizes
- assign larger blocks to faster nodes for better load balance

Example: parallel/distributed ISTA for LASSO

LASSO

$$\min \ f(\mathbf{x}) = \lambda \|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

ISTA algorithm

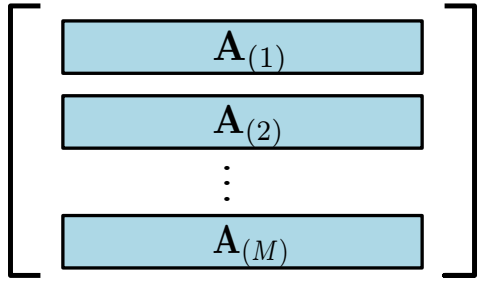
$$\mathbf{x}^{k+1} = \text{shrink}\left(\mathbf{x}^k - \delta_k \mathbf{A}^T\mathbf{A}\mathbf{x}^k + \delta_k \mathbf{A}^T\mathbf{b},\ \lambda\delta_k\right)$$

Serial code

1: initialize $\mathbf{x} = \mathbf{0}$ and δ;
2: pre-compute $\mathbf{A}^T\mathbf{b}$
3: **while** not converged **do**
4: $\quad \mathbf{g} = \mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}$
5: $\quad \mathbf{x} = \text{shrink}\,(\mathbf{x} - \delta\mathbf{g},\ \lambda\delta)$;
6: **end while**

Example: parallel/distributed ISTA for LASSO

distribute blocks of rows to M nodes

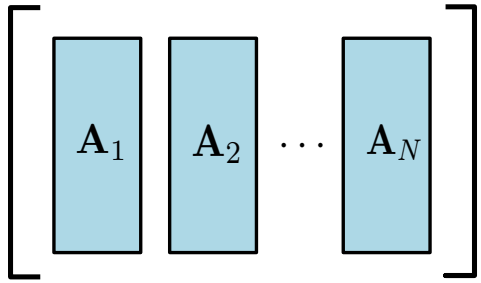


1: initialize $\mathbf{x} = \mathbf{0}$ and δ;
2: i = find_my_processor_id
3: processor i loads $\mathbf{A}_{(i)}$
4: pre-compute $\delta \mathbf{A}^T \mathbf{b}$
5: **while** not converged **do**
6: processor i computes $\mathbf{c}_i = \mathbf{A}_{(i)}^T \mathbf{A}_{(i)} \mathbf{x}$
7: $\mathbf{c}$ = allreduce($\mathbf{c}_i$, SUM)
8: $\mathbf{x} = \text{shrink}\left(\mathbf{x}^k - \delta \mathbf{c} + \delta \mathbf{A}^T \mathbf{b}, \lambda \delta\right)$;
9: **end while**

- assume $\mathbf{A} \in \mathbb{R}^{m \times n}$
- one allreduce per iteration
- speedup
  $\approx \frac{1}{\rho/M + (1-\rho) + O(\log(M))}$
- ρ is close to 1
- requires synchronization

Example: parallel ISTA for LASSO

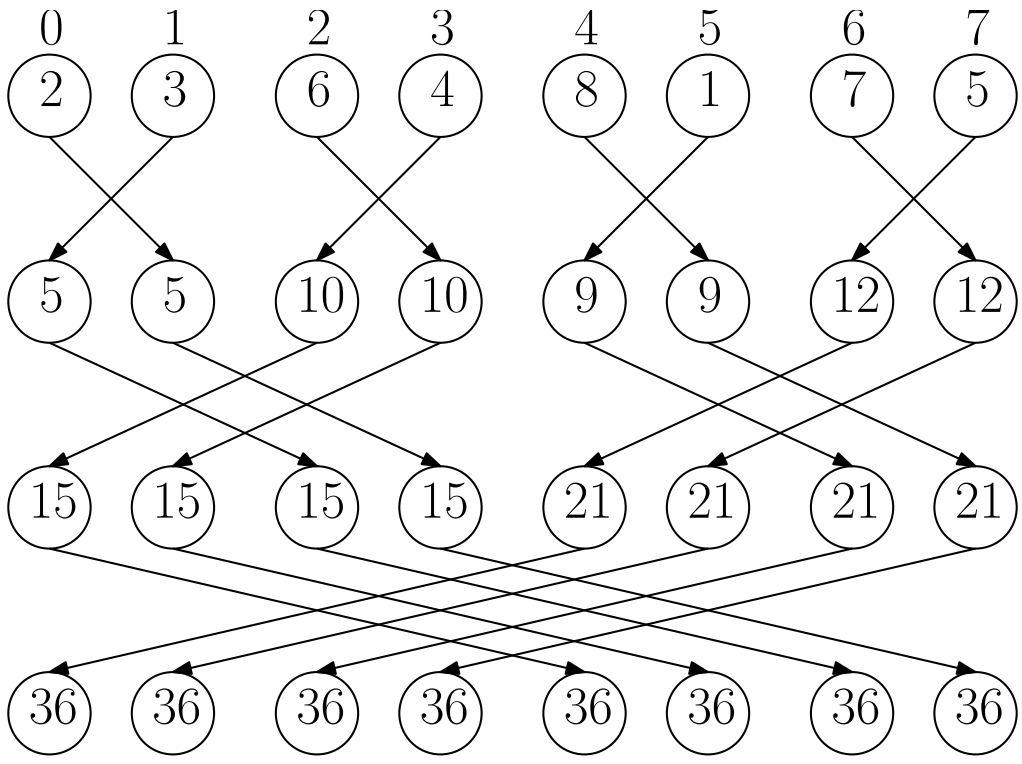
distribute blocks of columns to N nodes



1: initialize $\mathbf{x} = \mathbf{0}$ and δ;
2: i = find_my_processor_id
3: processor i loads $\mathbf{A}_i$
4: pre-compute $\delta \mathbf{A}_i^T \mathbf{b}$
5: **while** not converged **do**
6: processor i computes $\mathbf{y}_i = \mathbf{A}_i \mathbf{x}_i$
7: $\mathbf{y}$ = allreduce($\mathbf{y}_i$, SUM)
8: processor i computes $\mathbf{z}_i = \mathbf{A}_i^T \mathbf{y}$
9: $\mathbf{x}_i^{k+1} = \text{shrink}\left(\mathbf{x}_i^k - \delta \mathbf{z}_i + \delta \mathbf{A}_i^T \mathbf{b}, \lambda\delta\right)$;
10: **end while**

- one allreduce per iteration
- speedup $\approx \frac{1}{\rho/N + (1-\rho) + O(\frac{m}{n}\log(N))}$
- ρ is close to 1
- requires synchronization
- if $m \ll n$, this approach is faster

Example of allreduce SUM



$\log(N)$ layers, N parallel communications per layer

Example: parallel FISTA[3]

Algorithm P-FISTA

1: node j keeps $\mathbf{A}_j$, $\mathbf{b}$, initializes $\mathbf{x}_j^0 = \mathbf{x}_j^1 = 0$;
2: **for** $k = 1, 2, \ldots, K$ **do**
3: $\quad \bar{\mathbf{x}}_j \leftarrow \mathbf{x}_j^k + \frac{k-2}{k+1}\left(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}\right)$;
4: $\quad \mathbf{w} \leftarrow \sum_{j=1}^{N} \mathbf{A}_j \bar{\mathbf{x}}_j$ by Allreduce;
5: $\quad \mathbf{y} \leftarrow \nabla \mathcal{L}(\mathbf{w}; \mathbf{b})$;
6: $\quad \mathbf{g}_j \leftarrow \mathbf{A}_j^T \mathbf{y}$;
7: $\quad \mathbf{x}_j^{k+1} \leftarrow \mathbf{prox}_{\lambda\|\cdot\|_1}(\bar{\mathbf{x}}_j - \delta_k \mathbf{g}_j)$;
8: **end for**

Also: parallel algorithms for sparse logistic regression, sparse SVM, ...

[3] Beck and Teboulle [2009]

Block-coordinate descent

Definition: update one block of variables each time, keeping others fixed

(a better name would be *block coordinate update*)

Advantage: each update is simple

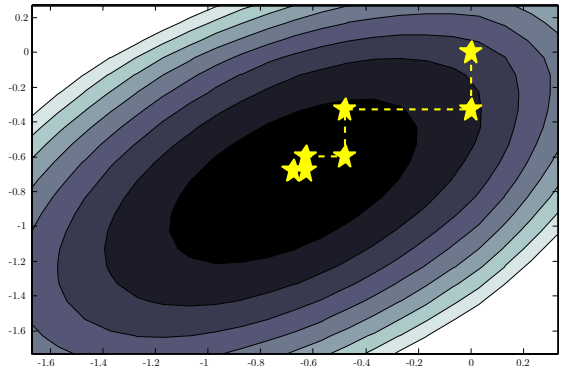
Disadvantages:

- more iterations (with exceptions)
- may stuck at non-stationary points if problem is non-convex and/or non-smooth (with exceptions)

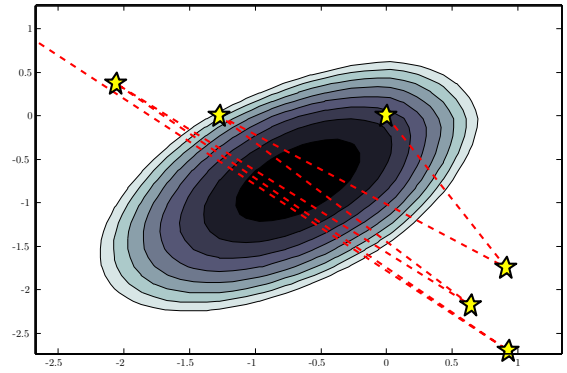
Selection rules: cycle (Gauss-Seidel), parallel (Jacobi), random, greedy

How many coordinates can be updated non-cooperatively?

- **Example 1**: (figures show 2D projection of a 3D problem)

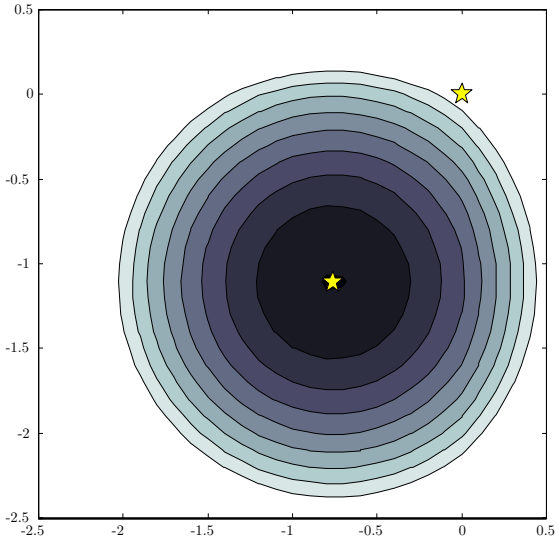


1 coordinate update



3 coordinate update

- **Example 2**:



3 coordinate update

How many coordinates can be updated non-cooperatively?

P (#. non-cooperative coordinate updated) depends on **block spectral radius**

$$\rho_P = \max_{\mathbf{M} \in \mathcal{M}} \rho(\mathbf{M}),$$

where $\mathcal{M}$ is the set of all $P \times P$ submatrices that we can obtain from $\mathbf{A}^T\mathbf{A}$ corresponding to selecting exactly one column from each of the P blocks

Theorem

Assume each column of $\mathbf{A}$ *has unit 2-norm. If* $\rho_P < 2$*, then any* P *parallel (prox-linear) updates per iteration gives*

$$\mathcal{F}(\mathbf{x} + \mathbf{d}) - \mathcal{F}(\mathbf{x}) \leq \frac{\rho_P - 2}{2}\beta\|\mathbf{d}\|_2^2$$

moreover,

$$\mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*) \;\leq\; \frac{2C^2\left(2L + \beta\sqrt{\frac{N}{P}}\right)^2}{(2-\rho_P)\beta} \cdot \frac{1}{k}.$$

(see our paper for parameter definitions and proof; partial credits to Scherrer, Tewari, Halappanavar, and Haglin [2012b])

Greedy coordinate selection

Greed is good for sparse optimization!

- select P coordinates based on some merit (e.g., most descent, largest step)
- widely used in *matching pursuit* (build solution by coordinate)
- applied to sparse optimization with $P = 1$ by Li and Osher [2009]
- we extend it to $P \geq 2$ (or dynamic P) in parallel
- if solution is sparse, then exceptionally effective!
  - ▶ coordinates in $\mathrm{supp}(\mathbf{x}^*)$ are selected and updated
  - ▶ most zero variables stay zero *the whole time*
  - ▶ since $\mathbf{x}^k - \mathbf{x}^*$ is sparse, objective is effectively strongly convex
  - ▶ "problem dimension" is reduced $\Rightarrow$ #.iterations is reduced
  - ▶ *caution*: selection requires "sorting", which can be expensive

GRock: greedy coordinate-block descent

Consider:

$$\min \lambda\|\mathbf{x}\|_1 + f(\mathbf{A}\mathbf{x} - \mathbf{b})$$

- decompose $\mathbf{A}\mathbf{x} = \sum_j \mathbf{A}_i\mathbf{x}_j$; block $\mathbf{A}_j$ and $\mathbf{x}_j$ are kept on node j

Parallel GRock:

1. (*parallel*) compute coordinate-merit, for each coordinate i

$$d_i = \arg\min_d \ \lambda \cdot r(x_i + d) + g_i d + \frac{1}{2}d^2, \quad \text{where } g_i = \mathbf{A}_{(i)}^T \nabla \mathbf{f}(\mathbf{A}\mathbf{x} - \mathbf{b})$$

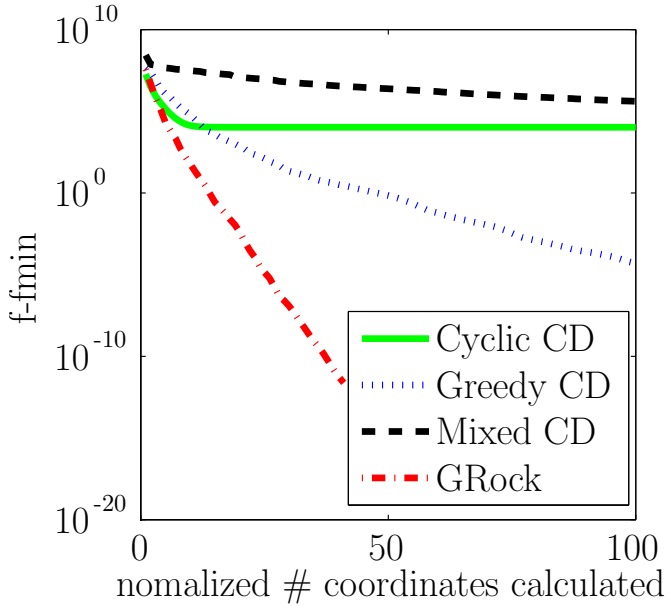
2. (*parallel*) compute block-merit, for each block j

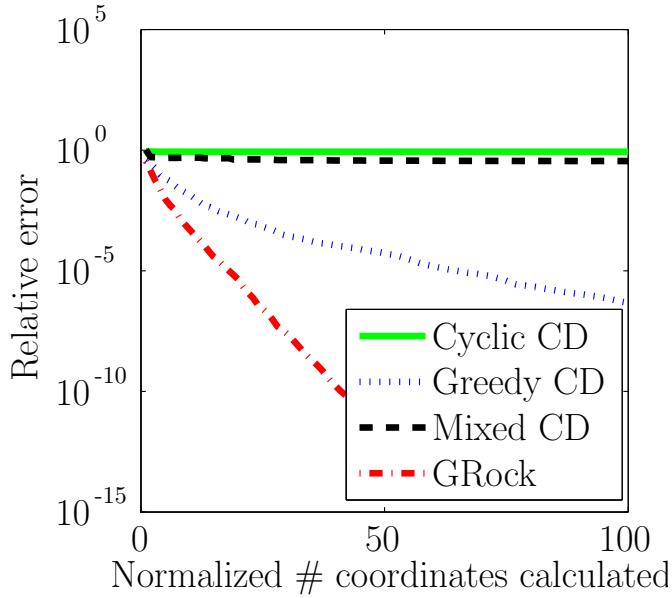
$$m_j = \max\{|d| : d \text{ is an element of } \mathbf{d}_j\}$$

   s_j denotes the index of the maximal coordinate in block j

3. (*allreduce*) $\mathcal{P} \leftarrow$ select P blocks with largest m_j, $2 \le P \le N$
4. (*parallel*) update $x_{s_j} \leftarrow x_{s_j} + d_{s_j}$ for all $j \in \mathcal{P}$
5. (*allreduce*) update $\mathbf{A}\mathbf{x}$

Compare different selection rules





- LASSO test with $\mathbf{A} \in \mathbb{R}^{512\times 1024}$, $N = 64$ column blocks
- GRock uses $P = 8$ updates each iteration
- greedy CD[4] uses $P = 1$
- mixed CD[5] selects $P = 8$ random blocks and best coordinate

[4] Li and Osher [2009]

[5] Scherrer, Tewari, Halappanavar, and Haglin [2012b]

Numerical comparison

Compared algorithms: flops+communication, per iteration, N parallel nodes

- parallel FISTA[6]: $O(\frac{mn}{N}) + O(n \log N)$
- parallel dual ADMM: $O(\frac{mn}{N} + 2m^2) + O(mN \log N)$, matrix factor cache
- GRock: $O(\frac{mn}{N} + Pm) + O(n \log N + N \log N)$

At each iteration

- parallel FISTA and GRock have comparable per-iteration cost
- parallel dual-ADMM needs more communication

[6] Beck and Teboulle [2009]

Test on cluster STIC @ Rice U.

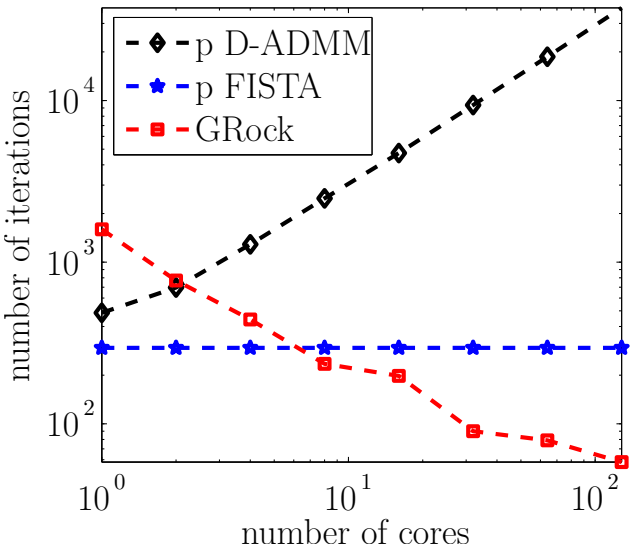
Hardware:

- 170 Appro Greenblade E5530 nodes each with two quad-core 2.4GHz Xeon (Nahalem) CPUs
- each node has 12GB of memory shared by all 8 cores
- # of processes used on each node = 8

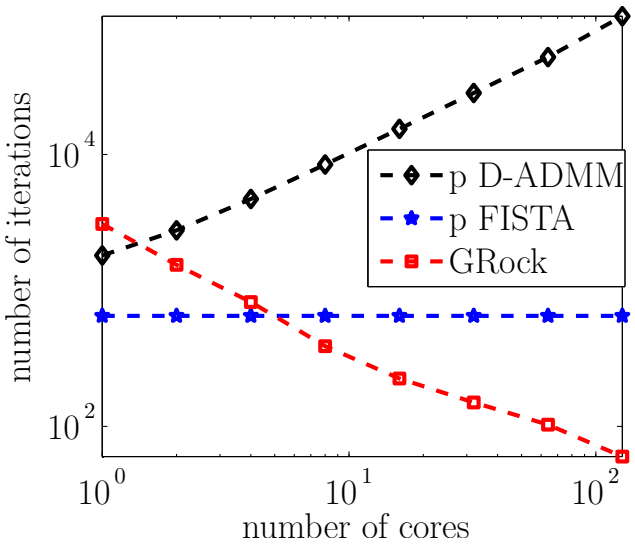
Test dataset:

	$\mathbf{A}$ type	$\mathbf{A}$ size	λ	sparsity of $\mathbf{x}^*$
dataset I	Gaussian	1024×2048	0.1	100
dataset II	Gaussian	2048×4096	0.01	200

iterations vs cores



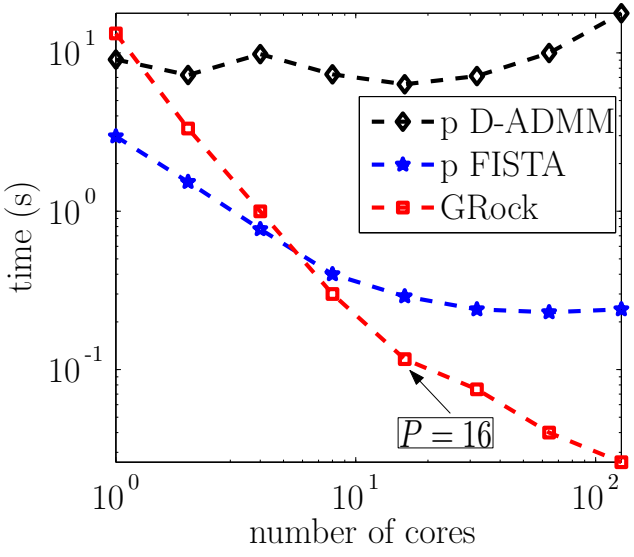
dataset I



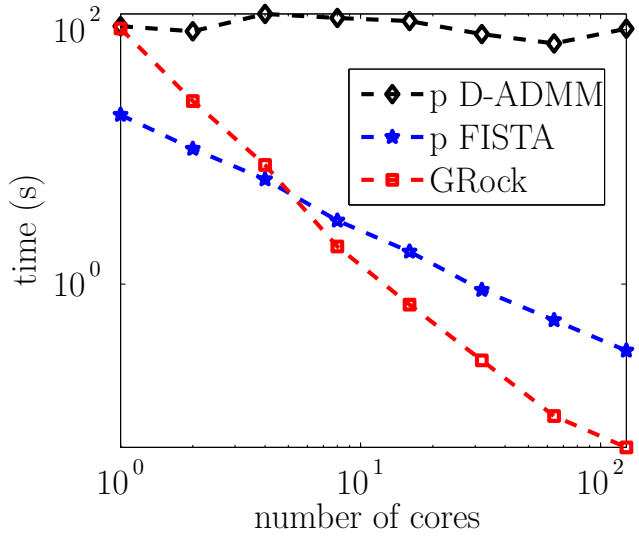
dataset II

Note: we set $P =$ number of cores

time vs cores



dataset I



dataset II

Note: we set $P =$ number of cores

Big data LASSO on Amazon EC2

Amazon EC2 is an elastic, pay-as-you-use cluster

Advantage: no hardware investment, everyone can have an account

Test dataset

- $\mathbf{A}$ dense matrix, 20 billion entries, and **170GB** size
- $\mathbf{x}$: 200K entries, 4K nonzeros, Gaussian values

Requested system from Amazon

- 20 "high-memory quadruple extra-large instances"
- each instance has 8 cores and 60GB memory

Code

- written in C using GSL (for matrix-vector multiplication) and MPI

parallel versions of dual-ADMM, FISTA, GRock

	p D-ADMM	p FISTA	GRock
estimate stepsize (min.)	n/a	1.6	n/a
matrix factorization (min.)	51	n/a	n/a
iteration time (min.)	105	40	1.7
number of iterations	2500	2500	104
communication time (min.)	30.7	9.5	0.5
stopping relative error	**1E-1**	**1E-3**	**1E-5**
total time (min.)	**156**	**41.6**	**1.7**
Amazon charge	**$85**	**$22.6**	**$0.93**

- ADMM performance is sensitive to penalty parameter β, which we picked as the best out of only a few trials (we cannot afford more)
- parallel dual-ADMM and FISTA were capped at 2500 iterations
- GRock used adaptive P and stopped at relative error 1E-5

Summary

- Big data applications requires parallel and distributed computing
- Separability structure enables parallel algorithms
- Greed is good (for sparse solutions) and greedy algorithms are parallelizable
- Algorithms are rather simple and must be simple to parallelize

Overheads of parallelism

- **computing overhead:** *start up time, synchronization wait time, data communication time, termination (data collection time)*
- **I/O overhead:** slow read and write of *non-local* data
- **algorithm overhead:** *extra variables, data duplication*
- **coding overhead:** language, library, operating system, debug, maintenance

Software codes can be found in the author's website.

Funding agencies: NSF, DoD

Acknowledgements: Qing Ling (USTC) and Zaiwen Wen (PKU)

References:

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel. Distributed basis pursuit. *Signal Processing, IEEE Transactions on*, 60(4):1942–1956, 2012.

Joseph K. Bradley, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Carlos Guestrin. Parallel coordinate descent for ℓ_1-regularized loss minimization. In *ICML*, pages 321–328, 2011.

Chad Scherrer, Mahantesh Halappanavar, Ambuj Tewari, and David J Haglin. Scaling up coordinate descent algorithms for large ℓ_1 regularization problems. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), 2012a.

C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. Feature clustering for accelerating parallel coordinate descent. In *NIPS*, pages 28–36, 2012b.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Y. Li and S. Osher. Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing: a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.