Parallel and Distributed Sparse Optimization

Zhimin Peng Ming Yan Wotao Yin

Computational and Applied Mathematics

Rice University

May 2013

outline

1. sparse optimization background

2. model and data distribution

3. parallel and distributed algorithms

   parallel/distributed prox-linear iteration

   GRock: greedy coordinate-block descent

   parallel/distributed ADMM

4. numerical experiments

   Rice cluster STIC

   Amazon EC2

5. conclusions

sparse optimization

processes "large" data sets for "structured" solutions

key points:

- finding the sparsest solution is generally NP-hard
- under some conditions (e.g., enough equations, dual certificate, incoherence, RIP, etc.), minimal ℓ_1 solution is the sparsest solution [1]
- most time, ℓ_1 minimization in $\mathbb{R}^n$ is quicker than solving an $n \times n$ dense system
- has many interesting generalizations in signal/image processing, machine learning, data mining, etc.

[1] Donoho [2006], Candès, Romberg, and Tao [2006]

current challenges

the size, complexity, and diversity of sparse optimization grow significantly

examples:

- video, 4D CT images, dynamical system, higher-dimensional tensors
- (the Internet) distributed data, streaming data, cloud computing

single-threaded approaches for sparse optimization

off-the-shelf: subgradient descent, LP/SOCP/SDP

smoothing: approximate ℓ_1 by

- $\ell_{1+\epsilon}$-norm
- $\sum_i \sqrt{x_i^2 + \epsilon}$
- Huber-norm

and apply gradient-based and quasi-Newton methods.

splitting: split smooth and nonsmooth terms into simple subproblems

- operator splitting: GPSR/FPC/SpaRSA/FISTA/SPGL1/...[2]
- dual operator splitting: Bregman/ADMM/split Bregman/... [3]

all operations have forms: $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$

greedy approaches: GRock, OMP, CoSaMP ...[4]

[2] Hale, Yin, and Zhang [2008], Beck and Teboulle [2009], van den Berg and Friedlander [2008]

[3] Yin [2010], Yang and Zhang [2011]

[4] Tropp and Gilbert [2007], Needell and Tropp [2009]

parallel/distributed sparse optimization

should be capable of handling instances of very large scales

this set of slides covers

- parallel/distrbuted alternating direction method of multipliers (ADMM)
- parallel/distrbuted prox-linear approaches
- (coming ...) parallel implementation of linearize Bregman
- GRock: parallel greedy coordinate-block descent

going down the list, algorithms assume more structures but are also faster

Outline

1. sparse optimization background

2. model and data distribution

3. parallel and distributed algorithms

    parallel/distributed prox-linear iteration

    GRock: greedy coordinate-block descent

    parallel/distributed ADMM

4. numerical experiments

    Rice cluster STIC

    Amazon EC2

5. conclusions

general model

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \;\; \mathcal{F}(\mathbf{x}) = \lambda \cdot \mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$$

- $\mathcal{R}(\mathbf{x})$ is a regularizer such as $\|\mathbf{x}\|_1$, $\|\Psi^T\mathbf{x}\|_1$, $\|\mathbf{x}\|_{2,1}$, etc.
- $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ is a fitting measure such as squared loss, logistic loss, etc.

- we say $f(\mathbf{x})$ is separable if

$$f(\mathbf{x}) = \sum_{s=1}^{S} f_s(\mathbf{x}_s)$$

distributing big data $\mathbf{A}$

reasons:

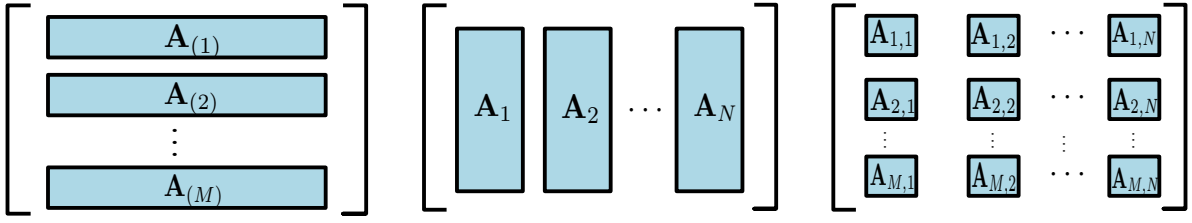
- $\mathbf{A}$ is too large to store centrally
- $\mathbf{A}$ is collected in a distributed manner

we consider three distribution schemes:

a. row blocks: stacking row-blocks $\mathbf{A}_{(1)}, \mathbf{A}_{(2)}, \ldots, \mathbf{A}_{(M)}$ forms $\mathbf{A}$

b. columns blocks $\mathbf{A} = [\mathbf{A}_1 \ \mathbf{A}_2 \ \cdots \ \mathbf{A}_N]$

c. $\mathbf{A}$ is partitioned into MN sub-blocks. The (i,j)th block is $\mathbf{A}_{i,j}$.



a b c

assume each computing node stores a block

computing $\mathbf{A}\mathbf{x}$ and $\mathbf{A}^T\mathbf{y}$: broadcast then parallel

computing

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} \mathbf{A}_{(1)}\mathbf{x} \\ \mathbf{A}_{(2)}\mathbf{x} \\ \cdots \\ \mathbf{A}_{(M)}\mathbf{x} \end{bmatrix} \quad \text{in scenario a}$$

and

$$\mathbf{A}^T\mathbf{y} = \begin{bmatrix} \mathbf{A}_1^T\mathbf{y} \\ \mathbf{A}_2^T\mathbf{y} \\ \cdots \\ \mathbf{A}_N^T\mathbf{y} \end{bmatrix} \quad \text{in scenario b}$$

each takes two steps

1. $\mathbf{x}$ and $\mathbf{y}$ are *broadcasted to* or *synchronized among* all the nodes
2. every node *independently computes* $\mathbf{A}_{(i)}\mathbf{x}$ and $\mathbf{A}_j^T\mathbf{y}$

computing $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$: parallel then reduce

computing

$$\mathbf{A}^T\mathbf{y} = \sum_{i=1}^{M} \mathbf{A}_{(i)}^T \mathbf{y}_i \quad \text{in scenario a}$$

and

$$\mathbf{Ax} = \sum_{i=1}^{N} \mathbf{A}_j \mathbf{x}_j \quad \text{in scenario b}$$

each takes two steps

1. every node computes $\mathbf{A}_{(i)}^T \mathbf{y}_i$ and $\mathbf{A}_j \mathbf{x}_j$ independently
2. a *reduce* operation computes the sum
   (*allreduce* also returns the sum to every node)

broadcast, parallel, then reduce

in scenario c, either one of $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$ requires both broadcast and reduce

example of $\mathbf{Ax}$:

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{A}_{(1)}\mathbf{x} \\ \mathbf{A}_{(2)}\mathbf{x} \\ \cdots \\ \mathbf{A}_{(M)}\mathbf{x} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{N} \mathbf{A}_{1,j}\mathbf{x}_j \\ \sum_{j=1}^{N} \mathbf{A}_{2,j}\mathbf{x}_j \\ \cdots \\ \sum_{j=1}^{N} \mathbf{A}_{M,j}\mathbf{x}_j \end{bmatrix}$$

1. for every j, $\mathbf{x}_j$ is *broadcasted* to nodes $(1,j),(2,j),\ldots,(M,j)$
2. compute $\mathbf{A}_{i,j}\mathbf{x}_j$ in parallel over all i,j
3. for every i, *reduce* over nodes $(i,1),(i,2),\ldots,(i,N)$ to compute $\mathbf{A}_{(i)}\mathbf{x} = \sum_{j=1}^{N} \mathbf{A}_{i,j}\mathbf{x}_j$

node speed and block size

- most clusters have **identical nodes**
- but clouds can have **different nodes**
- most parallel algorithms **require synchronized computation**
  faster (or less busy) nodes wait for slower (or more busy) nodes
- equal block size + different nodes $\Longrightarrow$ faster nodes wait for slower ones

a better solution: larger blocks assigned to faster (or less busy) nodes

Outline

1. sparse optimization background

2. model and data distribution

3. parallel and distributed algorithms

   parallel/distributed prox-linear iteration

   GRock: greedy coordinate-block descent

   parallel/distributed ADMM

4. numerical experiments

   Rice cluster STIC

   Amazon EC2

5. conclusions

background: prox-linear iteration

original model:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ \mathcal{F}(\mathbf{x}) = \lambda \cdot \mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$$

linearize $\mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b})$ at $\mathbf{x}^k$:

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \lambda \cdot \mathcal{R}(\mathbf{x}) + \langle \mathbf{x}, \mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}) \rangle + \frac{1}{2\delta_k} \|\mathbf{x} - \mathbf{x}^k\|_2^2,$$

advantages: well understood and its solution

$$\mathbf{x}^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{x}^k - \delta_k \mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})),$$

is often in **closed form**

example: if $\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_1$, $\mathbf{prox}_{\lambda\mathcal{R}}$ is known as soft-thresholding

algorithms: FPC/SpaSRA/ISTA/FISTA ...

scenario a

assume separability $\mathcal{L}(\mathbf{z}) = \sum_i \mathcal{L}_i(\mathbf{z}_i)$

storage: node i keeps row block $\mathbf{A}_{(i)}$, subvector $\mathbf{b}_i$, and current $\mathbf{x}^k$

each iteration: compute

$$\mathbf{x}^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{x}^k - \delta_k \mathbf{A}^T \nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})) \tag{1}$$

in three steps, noticing

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

1. parallel: every node i computes $\mathbf{A}_{(i)}^T \nabla\mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}^k, \mathbf{b}_i)$
2. *allreduce*: $\mathbf{A}^T \nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}) = \sum_{i=1}^{M} \mathbf{A}_{(i)}^T \nabla\mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}^k; \mathbf{b}_i)$ for all nodes
3. parallel: every node gets identical $\mathbf{x}^{k+1}$ following (1)

scenario b

assume separability $\mathcal{R}(\mathbf{x}) = \sum_j \mathcal{R}_j(\mathbf{x}_j)$

consequently:

storage: node i keeps column block $\mathbf{A}_j$, entire $\mathbf{b}$, and subvector $\mathbf{x}_j^k$

each iteration: computes

$$\mathbf{x}^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}}(\mathbf{x}^k - \delta_k \mathbf{A}^T \nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}))$$

$$\Rightarrow \quad \text{for each } j, \quad \mathbf{x}_j^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}_j}(\mathbf{x}_j^k - \delta_k \mathbf{A}_j^T \nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})) \qquad (2)$$

in three steps

- parallel: every node i computes $\mathbf{A}_j\mathbf{x}_j^k$
- *allreduce*: $\mathbf{A}\mathbf{x}^k = \sum_{j=1}^N \mathbf{A}_j\mathbf{x}_j^k$ for all nodes
- parallel: ever node computes $\mathbf{A}_j^T \nabla\mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b})$ and then $\mathbf{x}_j^{k+1}$ following (2)

scenario c

assume separability $\mathcal{R}(\mathbf{x}) = \sum_j \mathcal{R}(\mathbf{x}_j)$ **and** $\mathcal{L}(\mathbf{z}) = \sum_i \mathcal{L}(\mathbf{z}_i)$

two *allreduce* operations at each iteration, remaining operations are *parallel*

(details left to the reader)

Outline

1. sparse optimization background

2. model and data distribution

3. parallel and distributed algorithms

   parallel/distributed prox-linear iteration

   GRock: greedy coordinate-block descent

   parallel/distributed ADMM

4. numerical experiments

   Rice cluster STIC

   Amazon EC2

5. conclusions

GRock: greedy coordinate-block descent

general messages about coordinate (block) descent:

- update one coordinate (block) at a time
- coordinate selection: cycle (Gauss-Seidel), random, greedy, mixtures
- advantage: update is easy
- disadvantage: (usually but not always) more iterations, may fail on non-smooth and non-convex problems (stuck at non-stationary points)

but **GRock converges after fewer iterations for sparse optimization with $\mathbf{A}$ formed of nearly orthogonal columns**

GRock iteration for scenario b

1. assign a merit value for each coordinate i

$$d_i = \arg\min_d \ \lambda \cdot r(x_i + d) + g_i d + \frac{1}{2} d^2$$

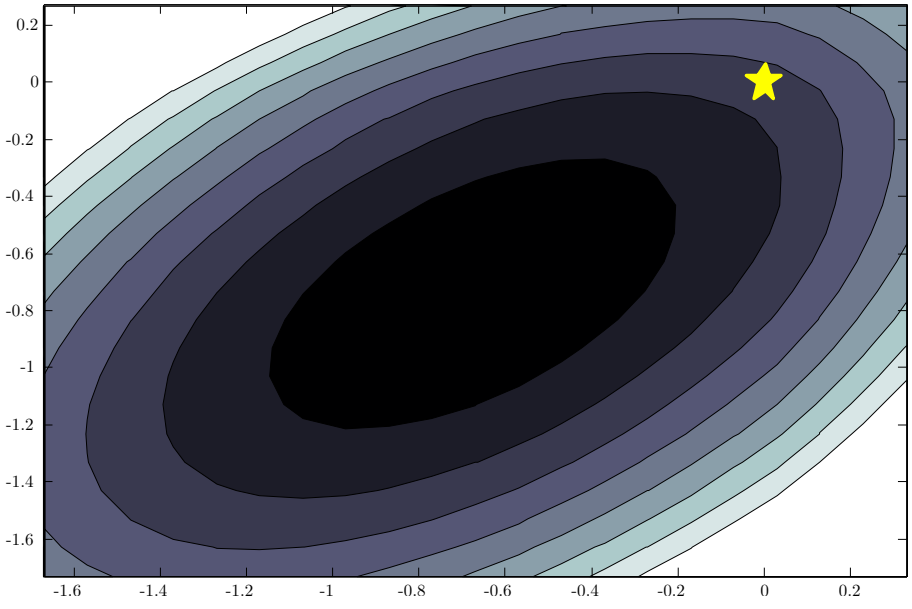
   - g_i is the ith component of $\mathbf{g}(\mathbf{x}) = \mathbf{A}_{(i)}^T \nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}; \mathbf{b}_i)$
   - larger d_i means x_i is more willing to reduce energy

2. assign a merit value for each block

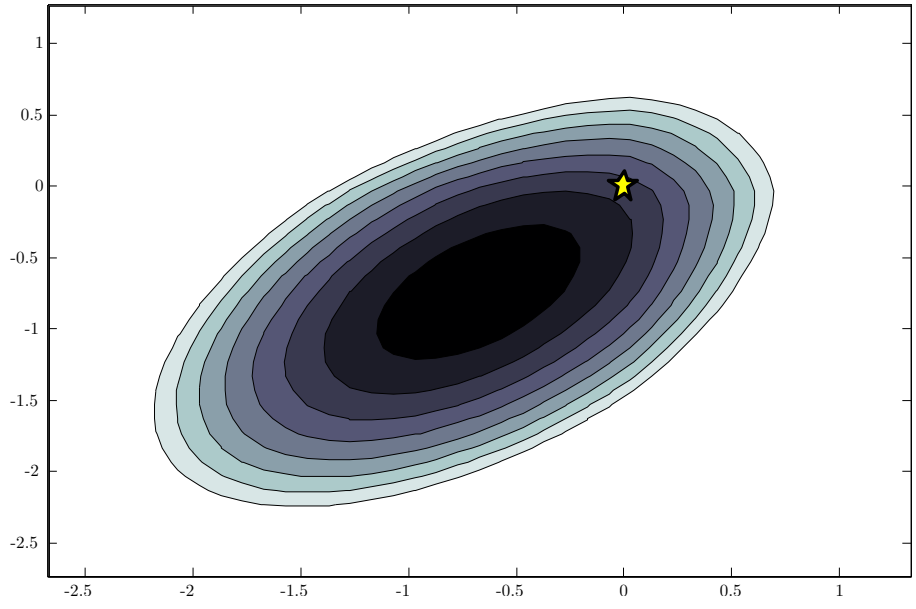
$$m_j = \max\{|d| : d \text{ is an element of } \mathbf{d}_j\}, \quad m_j = d_{s_j}$$

3. $\mathcal{P} \leftarrow$ select P blocks with largest m_j, $2 \leq P \leq N$
4. *parallel* update $x_{s_j} \leftarrow x_{s_j} + d_{s_j}$ for all $j \in \mathcal{P}$

how large P can be

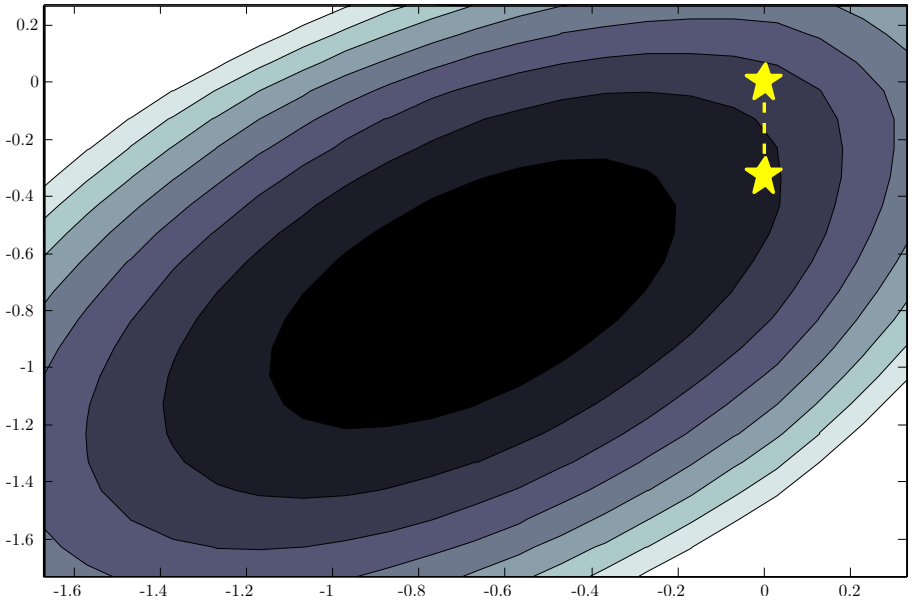


$P = 1$

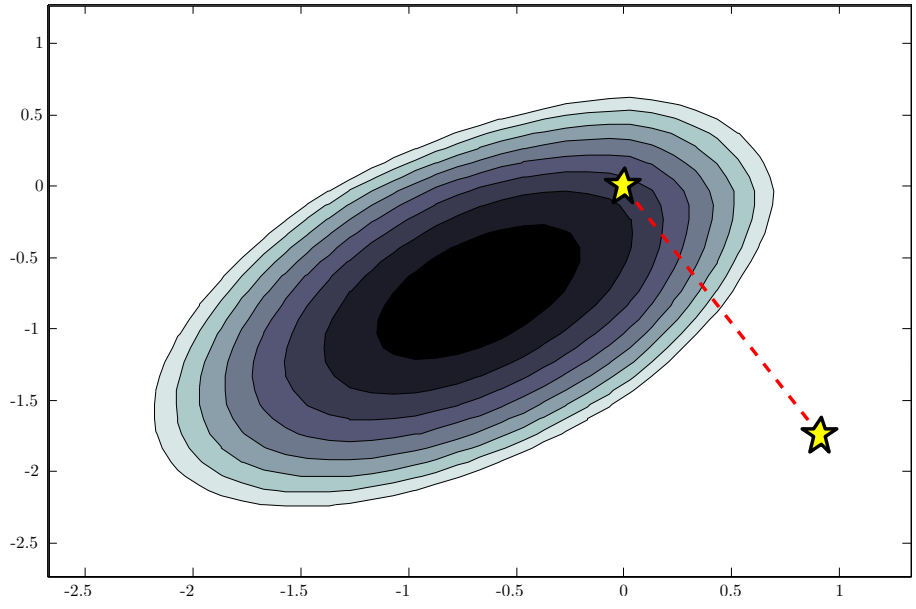


$P = 3$

how large P can be

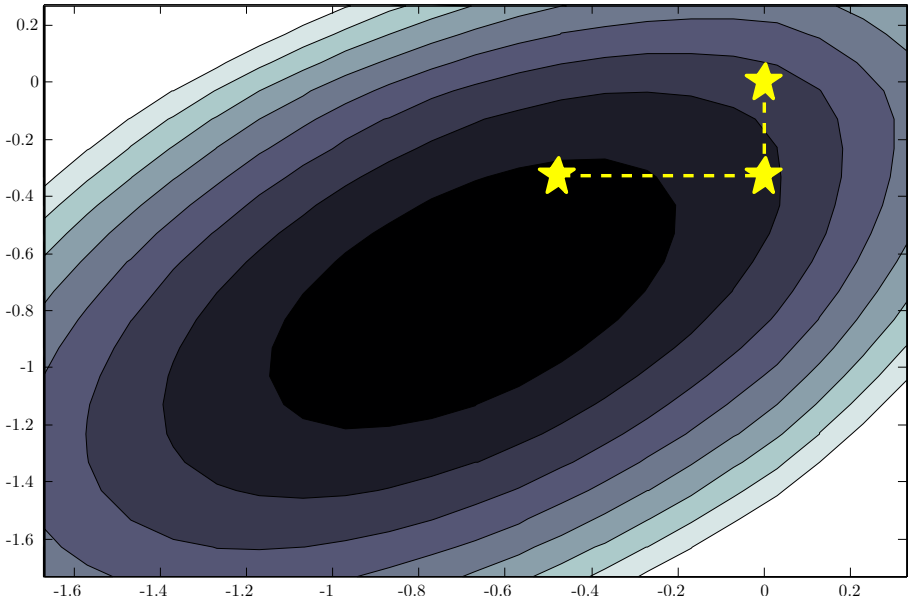


$P = 1$

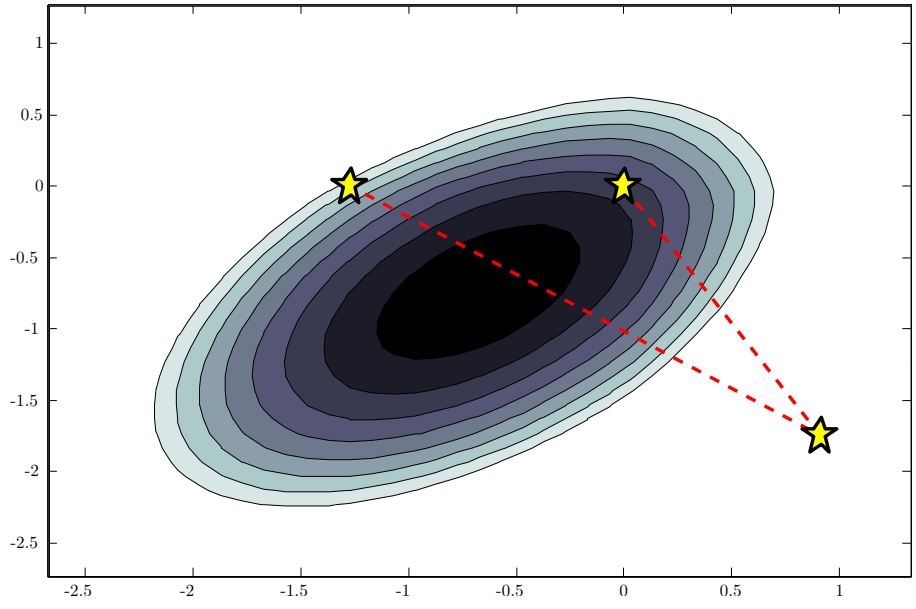


$P = 3$

how large P can be

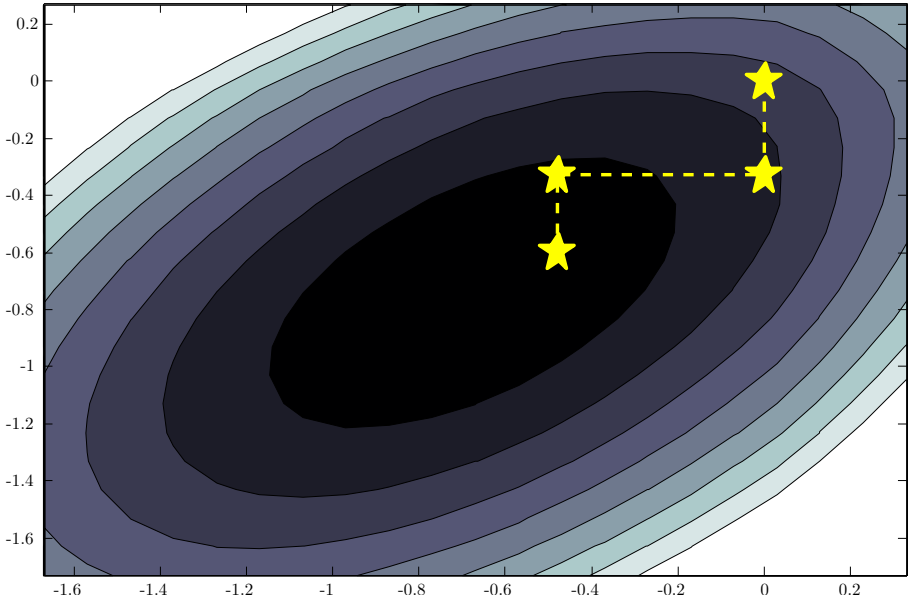


$P = 1$

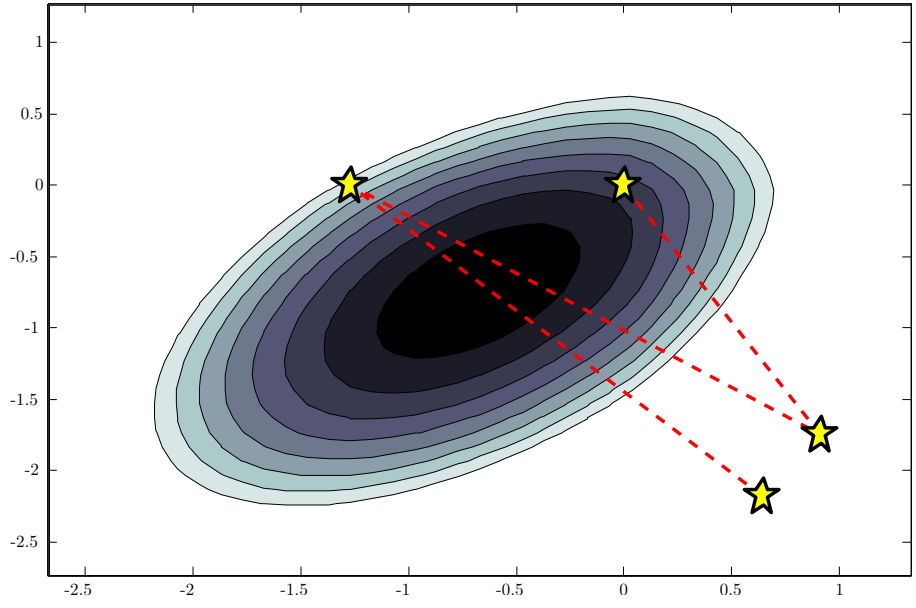


$P = 3$

how large P can be

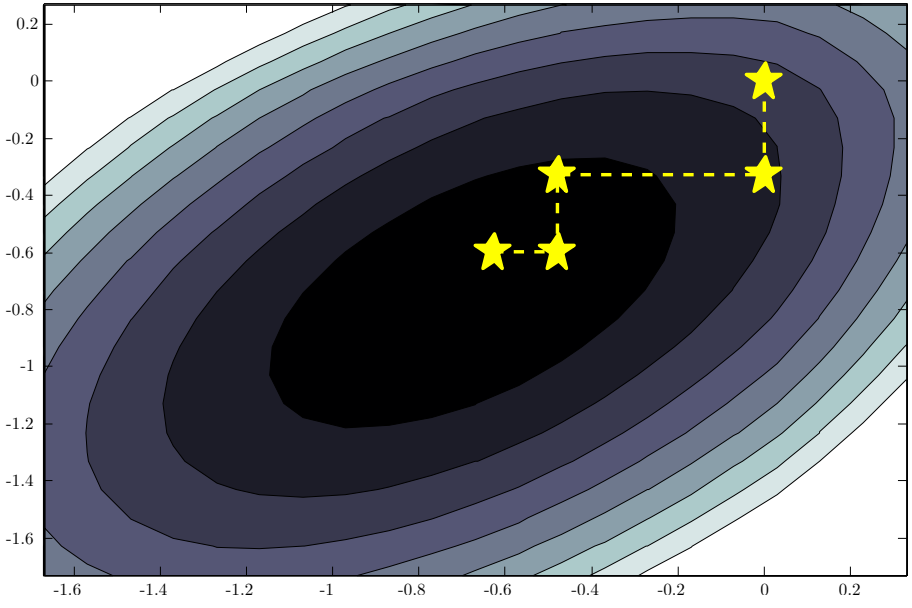


$P = 1$

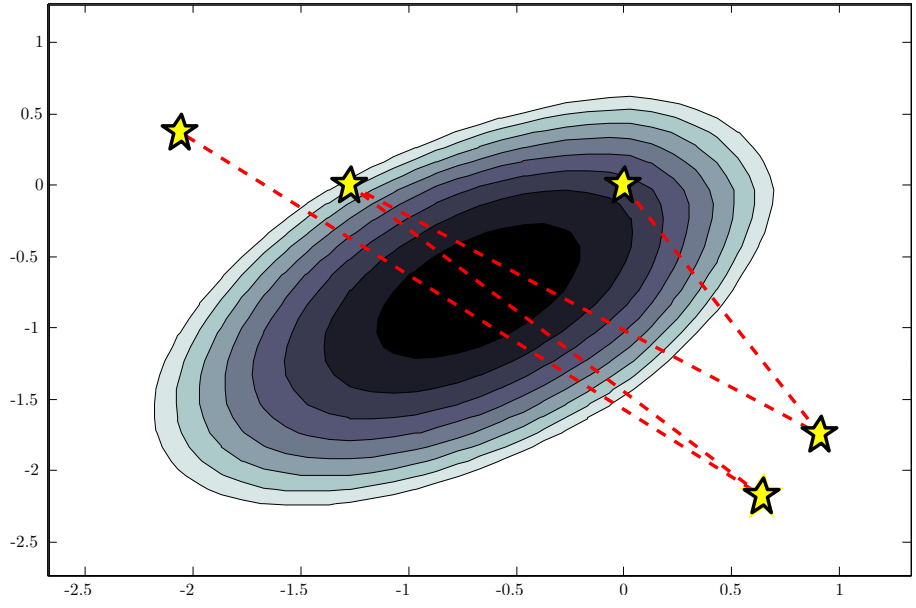


$P = 3$

how large P can be

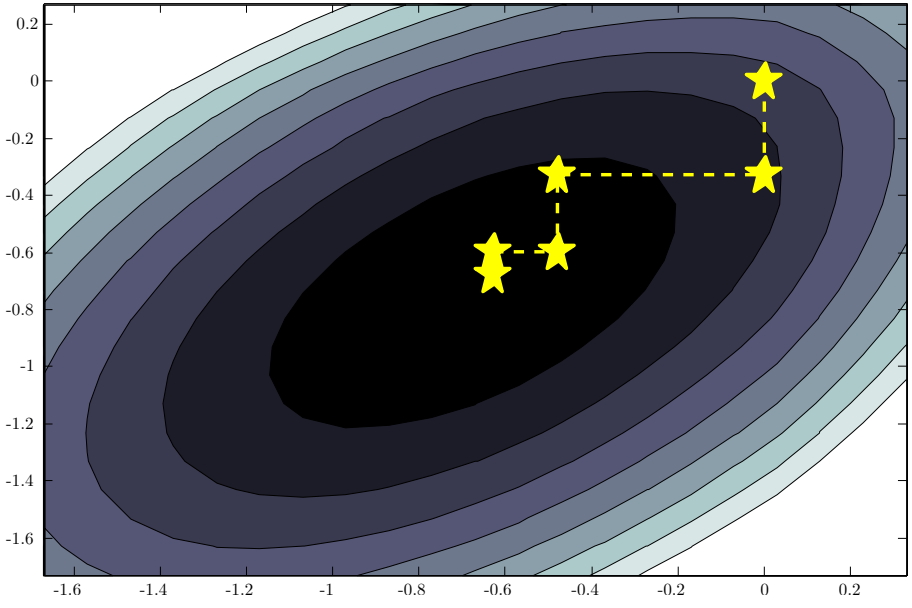


$P = 1$

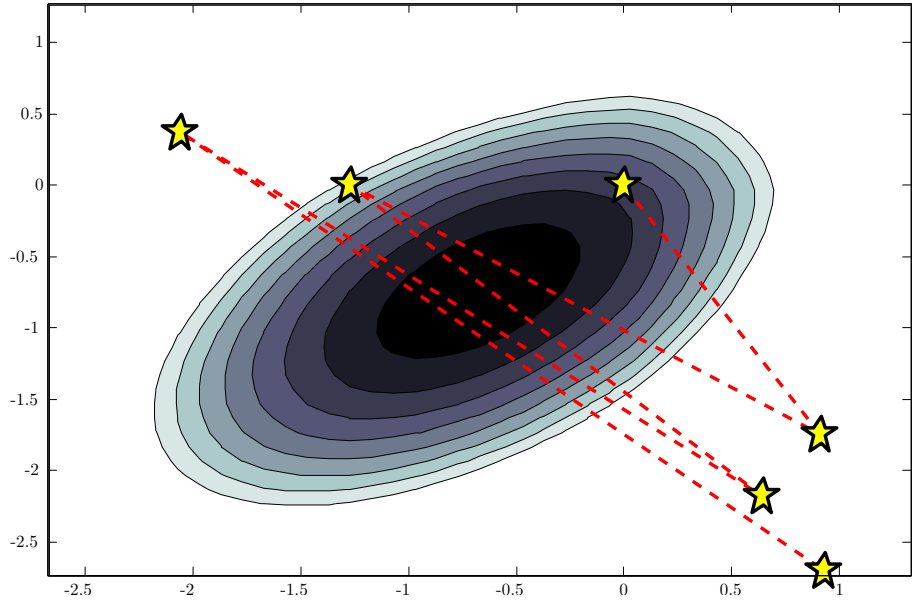


$P = 3$

how large P can be

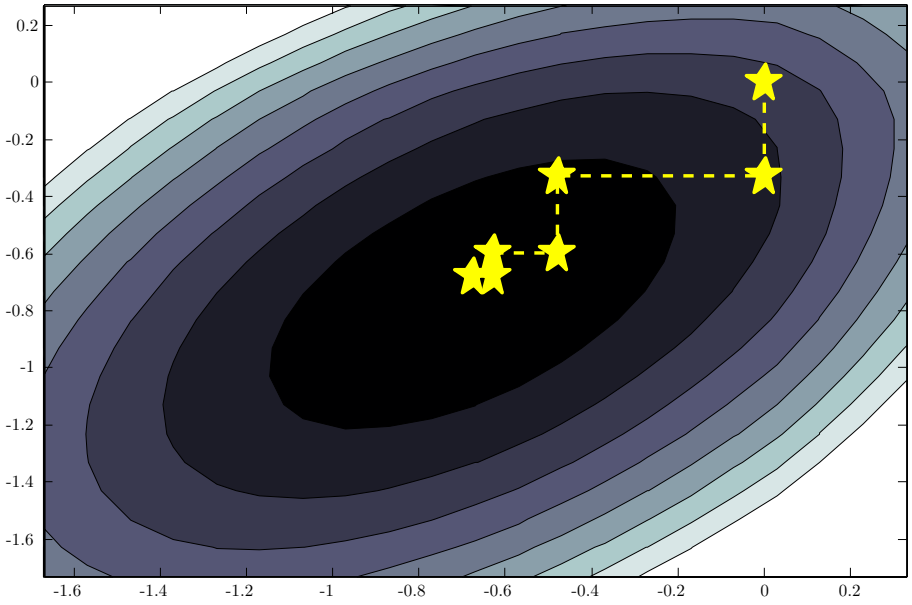


$P = 1$

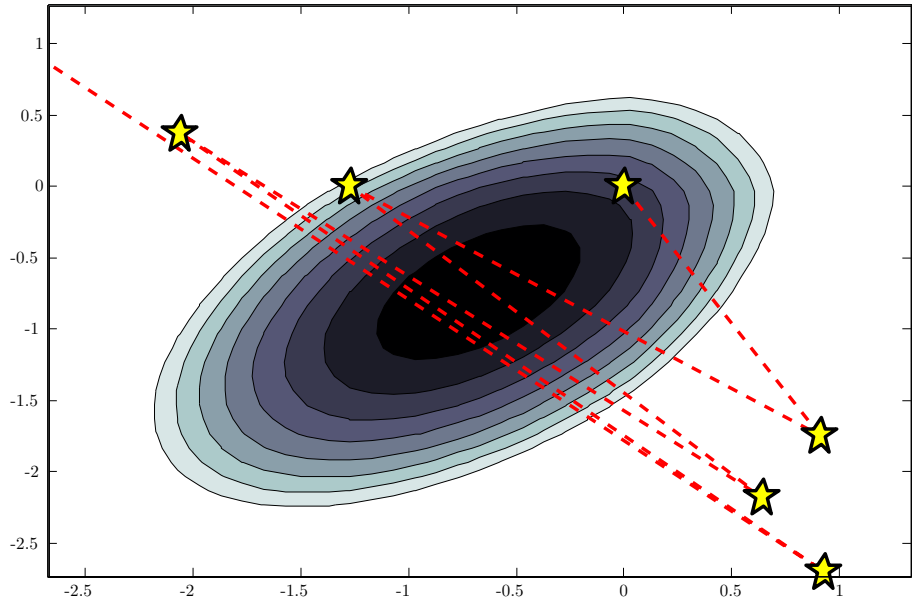


$P = 3$

how large P can be



$P = 1$ converge



$P = 3$ diverge

near orthogonality enables $P \geq 2$

assume each column of $\mathbf{A}$ has unit 2-norm

define 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

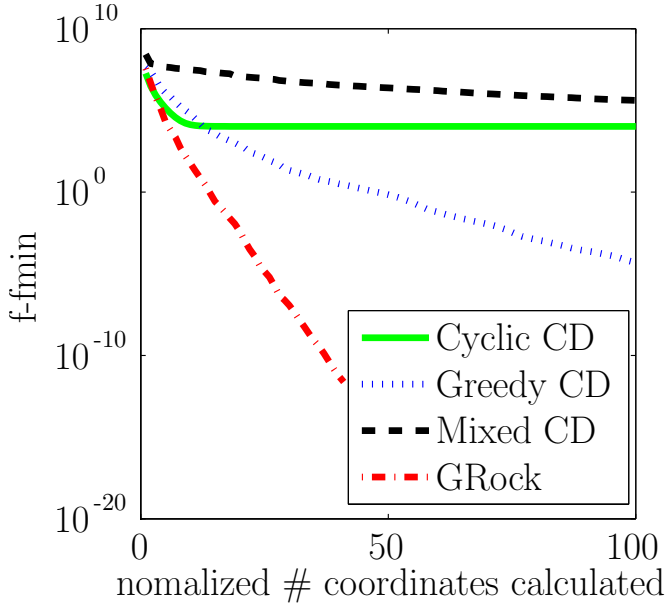
if $\rho_P < 2$, then GRock with P parallel updates per iteration gives

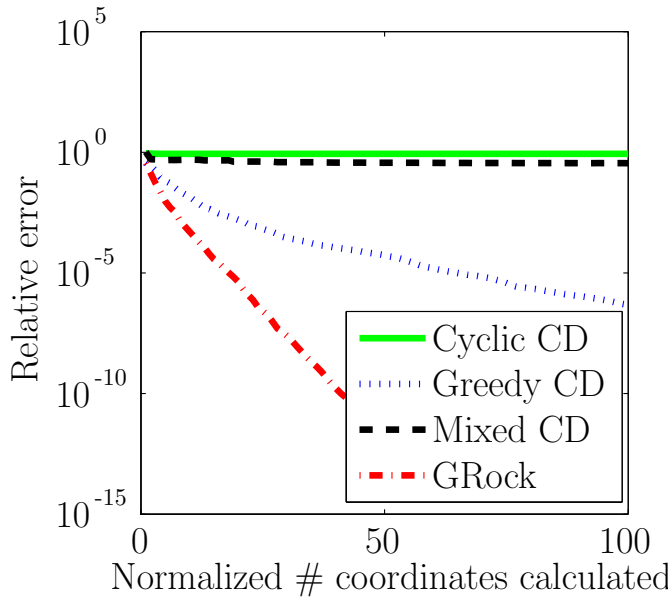
$$\mathcal{F}(\mathbf{x} + \mathbf{d}) - \mathcal{F}(\mathbf{x}) \leq \frac{\rho_P - 2}{2} \beta \|\mathbf{d}\|_2^2$$

moreover,

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

compare different block selection rules





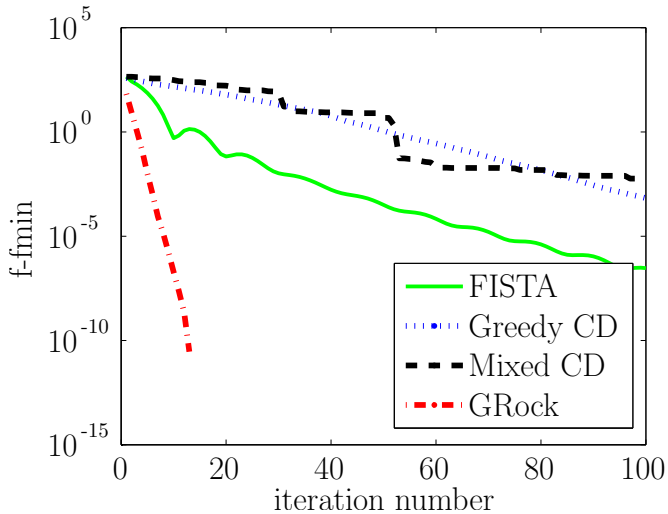
- LASSO test with $\mathbf{A} \in \mathbb{R}^{512\times 1024}$, $N = 64$ column blocks
- GRock uses $P = 8$ updates each iteration
- greedy CD [5] uses $P = 1$
- mixed CD [6] selects $P = 8$ random blocks and best coordinate in each iteration

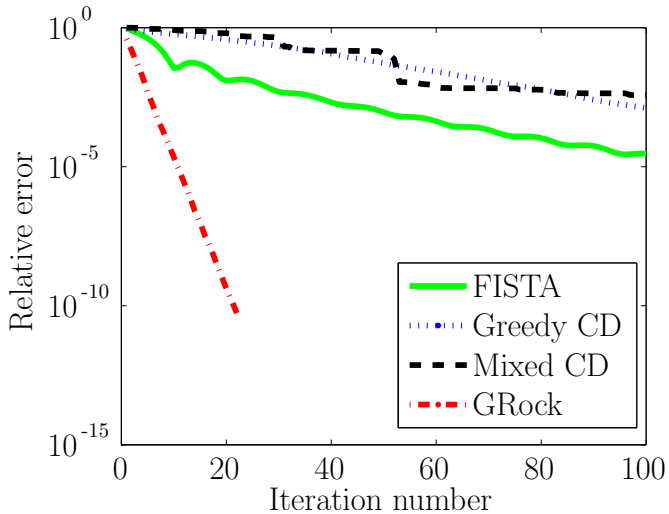
[5] Li and Osher [2009]

[6] Scherrer, Tewari, Halappanavar, and Haglin [2012]

real CMU data [7]

$\mathbf{A}$ has dimension [6205, 24820] with about 0.4% nonzero entries





GRock is often (but not always) quicker than FISTA

update of g in GRock is about 50% cheaper than in FISTA

GRock requires near orthogonality to work but FISTA does not

[7] Bradley, Kyrola, Bickson, Guestrin, and Guestrin [2011]

Why GRock rocks:

- coordinate selections often occur on those corresponding to nonzero entries in the solution
- such selection keeps most entries in $\mathbf{x}$ as zero throughout the iterations
- hence, the problem dimension is effectively reduced
- sparse update also makes gradient easier to update

prox-linear iterations typically have dense intermediate solutions at the beginning and can take many iterations before the intermediate solutions become finally sparse

Outline

1. sparse optimization background
2. model and data distribution
3. parallel and distributed algorithms
   - parallel/distributed prox-linear iteration
   - GRock: greedy coordinate-block descent
   - parallel/distributed ADMM
4. numerical experiments
   - Rice cluster STIC
   - Amazon EC2
5. conclusions

alternating direction method of multipliers (ADMM)

step 1: turn problem into the form of

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{b}. \end{aligned}$$

f and g are **convex**, maybe **nonsmooth**, can **incorporate constraints**

step 2: iterate

1. $\mathbf{x}^{k+1} \leftarrow \min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{y}^k) + \frac{\beta}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y}^k - \mathbf{b} - \mathbf{z}^k\|_2^2$,
2. $\mathbf{y}^{k+1} \leftarrow \min_{\mathbf{y}} f(\mathbf{x}^{k+1}) + g(\mathbf{y}) + \frac{\beta}{2}\|\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y} - \mathbf{b} - \mathbf{z}^k\|_2^2$,
3. $\mathbf{z}^{k+1} \leftarrow \mathbf{z}^k - (\mathbf{A}\mathbf{x}^{k+1} + \mathbf{B}\mathbf{y}^{k+1} - \mathbf{b})$.

history: dates back to 60s, formalized 80s, parallel versions appeared late 90s, revived very recently, convergence results enriched recently

parallel/distributed ADMM[8]

idea: form *separable* $f = \sum_i f_i(\mathbf{x}_i)$, *block diagonal* $\mathbf{A}$, and *simple* g and $\mathbf{B}$

$$\begin{array}{ll} \min_{\mathbf{x},\mathbf{y}} & \sum_i f_i(\mathbf{x}_i) + g(\mathbf{y}) \\ \text{s.t.} & \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_M \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_M \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_M \end{bmatrix} \mathbf{y} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_M \end{bmatrix} \end{array}$$

consequently, computing $\mathbf{x}^{k+1}$ reduces to parallel subproblems

$$\mathbf{x}_i^{k+1} \leftarrow \min_{\mathbf{x}} f_i(\mathbf{x}_i) + \frac{\beta}{2} \|\mathbf{A}_i \mathbf{x}_i + \mathbf{B}_i \mathbf{y}^k - \mathbf{b}_i - \mathbf{z}_i^k\|_2^2, \quad i = 1, \ldots, M$$

[8] Boyd, Parikh, Chu, Peleato, and Eckstein [2011]

scenario a: primal ADMM

assume separability $\mathcal{L}(\mathbf{z}) = \sum_i \mathcal{L}_i(\mathbf{z}_i)$

reformulate

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

into

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \textstyle\sum_i \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}_{(i)}, \mathbf{b}_i) \qquad + \mathcal{R}(\mathbf{x}) \\ \text{s.t.} \quad & \begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}_{(M)} \end{bmatrix} - \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} \mathbf{x} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned}$$

note: $\mathbf{x}_{(i)}$ are *not* subvectors of $\mathbf{x}$ but *local copies* of $\mathbf{x}$

scenario a: primal ADMM

storage: node i keeps row block $\mathbf{A}_{(i)}$, subvector $\mathbf{b}_i$, and current $\mathbf{x}_{(i)}^k$, $\mathbf{x}^k$, $\mathbf{z}^k$
each iteration:

1. *parallel*: $\mathbf{x}_{(i)}^{k+1} \leftarrow \min_{\mathbf{x}_{(i)}} \mathcal{L}_i(\mathbf{A}_i\mathbf{x}_{(i)}, \mathbf{b}_i) + \frac{\beta}{2}\|\mathbf{x}_{(i)} - \mathbf{x}^k - \mathbf{z}_i^k\|_2^2$, for every i
2. *allreduce*: $\frac{1}{M}\sum_i(\mathbf{x}_{(i)}^{k+1} - \mathbf{z}_i^k)$ for all nodes
3. *parallel*: $\mathbf{x}^{k+1} \leftarrow \mathcal{R}(\mathbf{x}) + \frac{\beta M}{2}\left\|\mathbf{x} - \frac{1}{M}\sum_i(\mathbf{x}_{(i)}^{k+1} - \mathbf{z}_i^k)\right\|_2^2$
4. *parallel*: $\mathbf{z}_i^{k+1} \leftarrow \mathbf{z}_i^k - (\mathbf{x}_{(i)}^{k+1} - \mathbf{x}^{k+1})$ for every i

note: steps 2 & 3 are due to

$$\frac{\beta}{2}\left\|\begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix}\begin{bmatrix} \mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}_{(M)} \end{bmatrix} - \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix}\mathbf{x} - \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_M \end{bmatrix}\right\|_2^2 = \frac{\beta M}{2}\left\|\mathbf{x} - \frac{1}{M}\sum_i(\mathbf{x}_{(i)} - \mathbf{z}_i)\right\|_2^2 + C$$

where C is independent of $\mathbf{x}$

dual ADMM

reformulate

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

into

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & \lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y}, \mathbf{b}) \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} - \mathbf{y} = 0 \end{aligned}$$

dualization:

$$\begin{aligned} \Rightarrow \quad & \min_{\mathbf{x},\mathbf{y}} \quad \max_{\mathbf{z}}\{\lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y}, \mathbf{b}) + \mathbf{z}^T(\mathbf{A}\mathbf{x} - \mathbf{y})\} \\ \Rightarrow \quad & \max_{\mathbf{z}} \quad \min_{\mathbf{x},\mathbf{y}} \{\lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{y}, \mathbf{b}) + \mathbf{z}^T(\mathbf{A}\mathbf{x} - \mathbf{y})\} \\ \Rightarrow \quad & \max_{\mathbf{z}} \quad \min_{\mathbf{x}}\{\lambda\mathcal{R}(\mathbf{x}) + \mathbf{z}^T\mathbf{A}\mathbf{x}\} + \min_{\mathbf{y}}\{\mathcal{L}(\mathbf{y}, \mathbf{b}) - \mathbf{z}^T\mathbf{y}\} \\ \Rightarrow \quad & \max_{\mathbf{z}} \quad \mathcal{G}(\mathbf{A}^T\mathbf{z}) + \mathcal{H}(\mathbf{z}) \end{aligned}$$

where $\mathcal{G}(\mathbf{z}) := \min_{\mathbf{x}}\{\lambda\mathcal{R}(\mathbf{x}) + \mathbf{z}^T\mathbf{x}\}$ and $\mathcal{H}(\mathbf{z}) := \min_{\mathbf{y}}\{\mathcal{L}(\mathbf{y}, \mathbf{b}) - \mathbf{z}^T\mathbf{y}\}$

scenario b: dual ADMM

assume separability $\mathcal{R}(\mathbf{x}) = \sum_j \mathcal{R}_j(\mathbf{x}_j)$, then $\mathcal{G}$ is also separable

$$\mathcal{G}(\mathbf{A}^T\mathbf{z}) = \sum_j \mathcal{G}_j(\mathbf{A}_j^T\mathbf{z})$$

reformulate the dual problem as:

$$\min_{\mathbf{z}} \quad -\tfrac{1}{N}\textstyle\sum_j \mathcal{H}(\mathbf{z}_{(j)}) - \sum_j \mathcal{G}_j(\mathbf{y}_j)$$

$$\text{s.t.} \quad \begin{bmatrix} \mathbf{A}_1^T & & \\ & \ddots & \\ & & \mathbf{A}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{z}_{(1)} \\ \vdots \\ \mathbf{z}_{(N)} \end{bmatrix} - \begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{I} & & \\ & \ddots & \\ & & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{z}_{(1)} \\ \vdots \\ \mathbf{z}_{(N)} \end{bmatrix} - \begin{bmatrix} \mathbf{I} \\ \vdots \\ \mathbf{I} \end{bmatrix} \mathbf{z} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

note: $\mathbf{z}_{(j)}$ are *not* subvectors of $\mathbf{z}$ but *local copies* of $\mathbf{z}$

scenario b: dual ADMM

storage: node j keeps column block $\mathbf{A}_j$, vector $\mathbf{b}$, and current $\mathbf{y}_j^k$, $\mathbf{z}^k$, $\mathbf{z}_{(j)}^k$, and dual variables $\mathbf{x}_j^k$, $\mathbf{p}_i^k$ of the dual problem

each iteration:

1. *parallel*: $\mathbf{y}_i^{k+1} \leftarrow \arg\min_{\mathbf{y}_i} -\mathcal{G}_i(\mathbf{y}_i) + \frac{\alpha}{2}\|\mathbf{A}_i\mathbf{z}_{(i)}^k - \mathbf{y}_i - \mathbf{x}_i^k\|_2^2$, for every i
2. *parallel*: $\mathbf{z}_{(i)}^{k+1} \leftarrow \arg\min_{\mathbf{z}_{(i)}} -\frac{1}{N}\mathcal{H}(\mathbf{z}_{(i)}) + \frac{\alpha}{2}\|\mathbf{A}_i\mathbf{z}_{(i)} - \mathbf{y}_i^{k+1} - \mathbf{x}_i^k\|_2^2 + \frac{\beta}{2}\|\mathbf{z}_{(i)} - \mathbf{z}^k - \mathbf{p}_i^k\|_2^2$, for every i
3. *allreduce*: $\mathbf{z} \leftarrow \frac{1}{N}\sum_i^N(\mathbf{z}_{(i)}^{k+1} - \mathbf{p}_i^k)$ for all nodes
4. *parallel*: $\mathbf{x}_{(i)}^{k+1} \leftarrow \mathbf{x}_{(i)}^k - \gamma(\mathbf{A}_i^T\mathbf{z}_{(i)}^{k+1} - \mathbf{y}_i^{k+1})$ for every i
5. *parallel*: $\mathbf{p}_i^{k+1} \leftarrow \mathbf{p}_i^k - \gamma(\mathbf{z}_{(i)}^{k+1} - \mathbf{z}^{k+1})$ for every i

Outline

1. sparse optimization background
2. model and data distribution
3. parallel and distributed algorithms
   - parallel/distributed prox-linear iteration
   - GRock: greedy coordinate-block descent
   - parallel/distributed ADMM
4. numerical experiments
   - Rice cluster STIC
   - Amazon EC2
5. conclusions

test on STIC, a cluster at Rice university

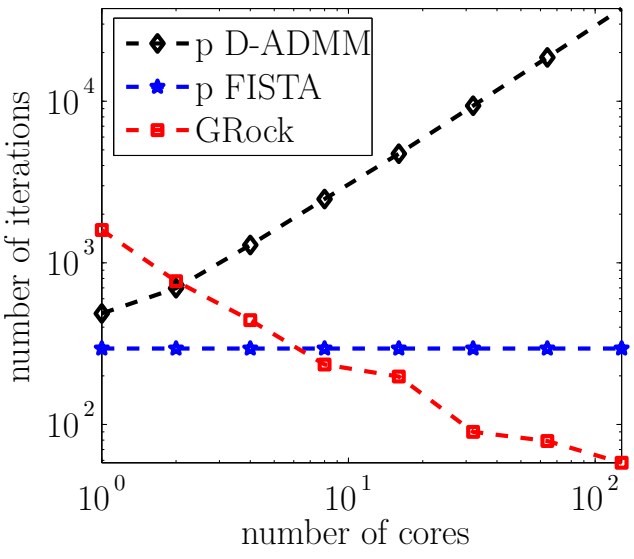
specs:

- 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

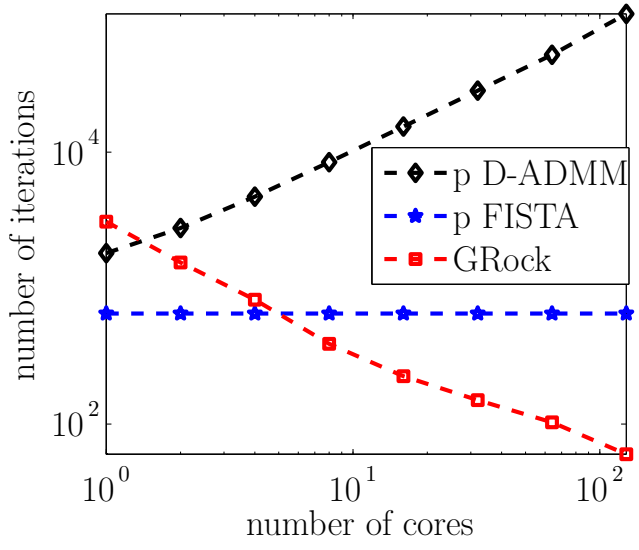
test dataset:

	$\mathbf{A}$ 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

cores vs iterations

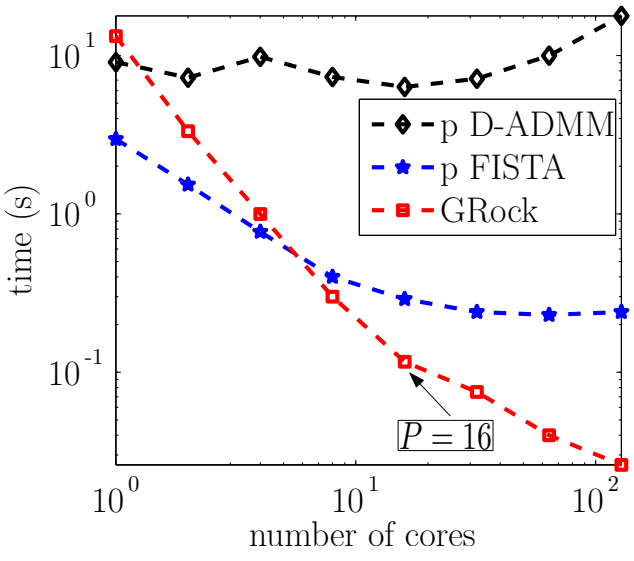


dataset I

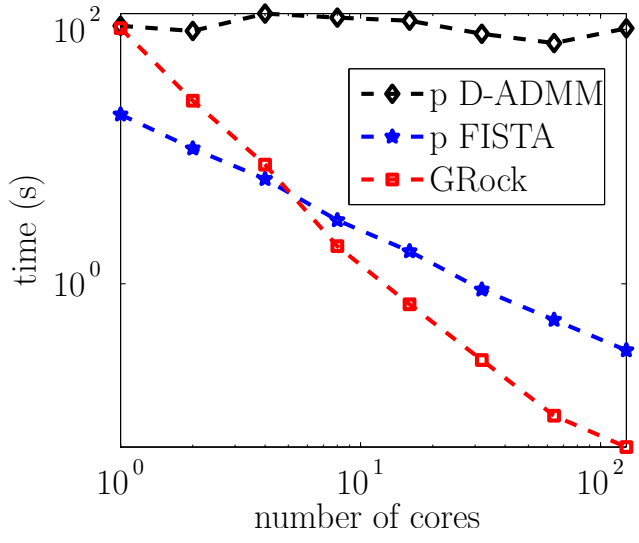


dataset II

cores vs time

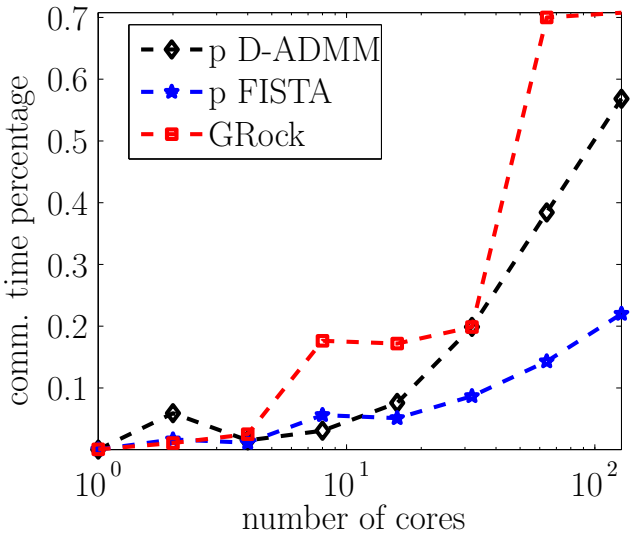


dataset I

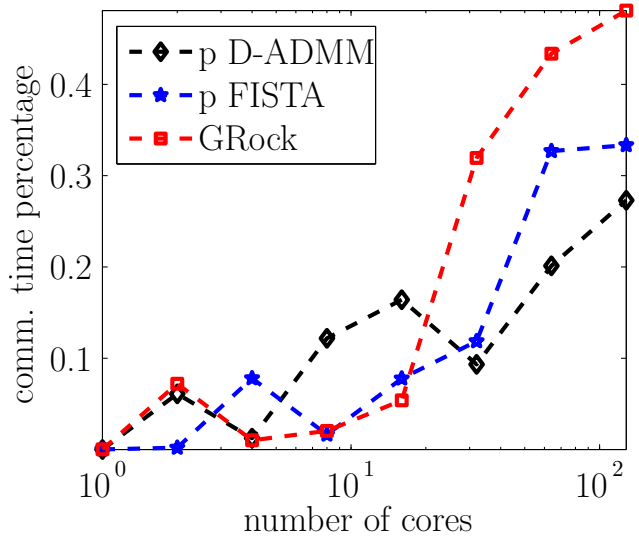


dataset II

cores vs (% communication/total time)



dataset I



dataset II

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

- 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 dual ADMM vs 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's performance depends on penalty parameter β
  we picked β as the best out of only a few trials (we cannot afford more)
- parallel dual ADMM stopped at 2500 iterations
- GRock stopped at relative error 1E-5

conclusions

summary of results

- ADMM is flexible but does not scale well; its iterations increase with distribution
- separable objectives enable parallel and distributed prox-linear iterations, which scale very well
- nearly orthogonal $\mathbf{A}$ further enables parallel GRock, which scales even better
- in general, model structures and data properties of sparse optimization enable very efficient parallel and distributed approaches

acknowledgements: NSF DMS and ECCS, ARO/ARL MURI

open resources: report, slides, and codes available on our websites

References:

David Leigh Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4): 1289–1306, 2006.

Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *Information Theory, IEEE Transactions on*, 52(2):489–509, 2006.

Elaine T Hale, Wotao Yin, and Yin Zhang. Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3):1107–1130, 2008.

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

E. van den Berg and M. P. Friedlander. Probing the pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008. doi: $10.1137/080714488$. URL `http://link.aip.org/link/?SCE/31/890`.

Wotao Yin. Analysis and generalizations of the linearized bregman method. *SIAM Journal on Imaging Sciences*, 3(4):856–877, 2010.

Junfeng Yang and Yin Zhang. Alternating direction algorithms for ℓ_1-problems in compressive sensing. *SIAM journal on scientific computing*, 33(1):250–278, 2011.

Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.

Deanna Needell and Joel A Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

Yingying Li and Stanley 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.

Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 28–36, 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.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.