Sparse Optimization
Lecture: Parallel and Distributed Sparse Optimization

Instructor: Wotao Yin

July 2013

online discussions on piazza.com

Those who complete this lecture will know

- basics of parallel computing
- how to parallel a bottleneck of existing sparse optimization method
- primal and dual decomposition
- GRock: greedy coordinate-block descent, and its parallel implementation

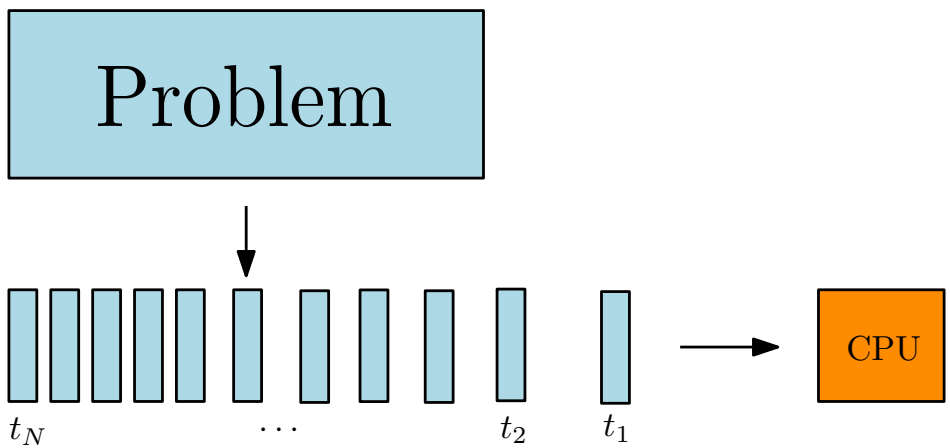
Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. Primal decomposition / dual decomposition
   - Parallel dual gradient ascent (linearized Bregman)
   - Parallel ADMM
4. Parallel greedy coordinate descent
5. Numerical results with big data

Serial computing

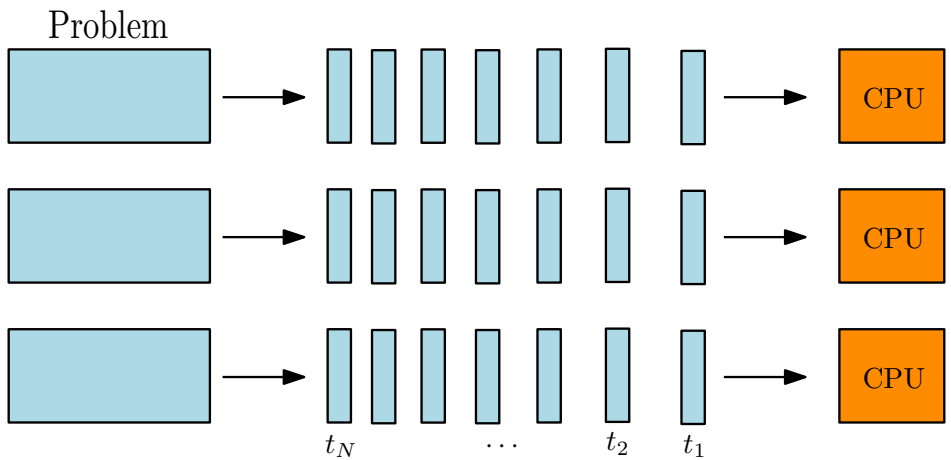
Traditional computing is **serial**

- a single CPU, one instruction is executed at a time
- a problem is broken into a series of instructions, executed one after another



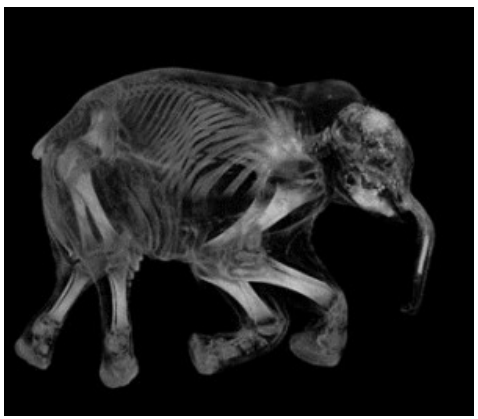
Parallel computing

- a problem is broken into *concurrent* parts, executed *simultaneously*, coordinated by a controller
- uses multiple cores/CPUs/networked computers, or their *combinations*
- expected to solve a problem in *less time*, or a *larger problem*



Commercial applications

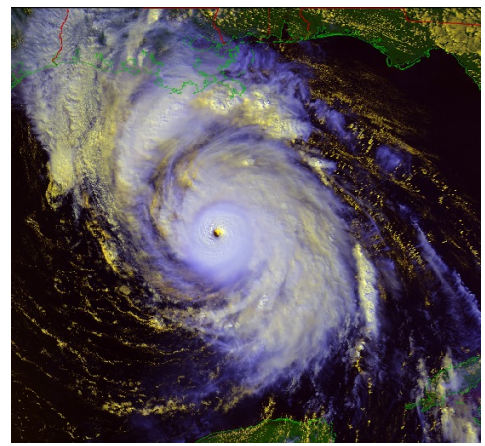
Examples: internet data mining, logistics/supply chain, finance, online ad, recommendation, sales data analysis, virtual reality, computer graphics (cartoon movies), medicine design, medical imaging, network video, ...



Applications in science and engineering

- **simulation:** many real-world events happen concurrently, interrelated
  examples: galaxy, ocean, weather, traffic, assembly line, queues
- **use in science/engineering:** environment, physics, bioscience, chemistry, geoscience, mechanical engineering, mathematics (computerized proofs), defense, intelligence





Benefits of being parallel

- **break single-CPU limits**
- **save time**
- **process big data**
- **access non-local resource**
- **handle concurrent data streams / enable collaboration**

Parallel overhead

- **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

Different parallel computers

- basic element is like a single computer, a 70-year old architecture
  components: CPU (control/arithmetic units), memory, input/output
- **SIMD**: single instruction multiple data, GPU-like
  applications: image filtering, PDE on grid, ...
- **MISD**: multiple instruction single data, *rarely seen*
  conceivable applications: cryptography attack, global optimization
- **MIMD**: multiple instruction multiple data
  most of today's supercomputer, clusters, clouds, multi-core PCs

Tech jargons

- HPC: high-performance computing
- node/CPU/processor/core
- **shared memory:** either all processors access a common *physical memory*, or parallel tasks have direct address of *logical memory*
- SMP: multiple processors share a single memory, *shared-memory computing*
- **distributed memory:** parallel tasks see *local* memory and use communication to access *remote* memory
- communication: exchange of data or synchronization controls; either through shared memory or network
- **synchronization:** coordination, one processor awaits others

Tech jargons

- **granularity:** *coarse* – more work between communication events; *fine* – less work between communication events
- Embarrassingly parallel: main tasks are independent, little need of coordination
- **speed scalability:** how speedup can *increases* with additional resources
  data scalability: how problem size can *increase* with additional resources
  scalability affected by: problem model, algorithm, memory bandwidth, network speed, parallel overhead
  sometimes, adding more processors/memory may not save time!
- **memory architectures:** uniform or non-uniform shared, distributed, hybrid
- **parallel models:** shared memory, message passing, data parallel, SPMD, MPMD, etc.

Parallel speedup

- **definition:**

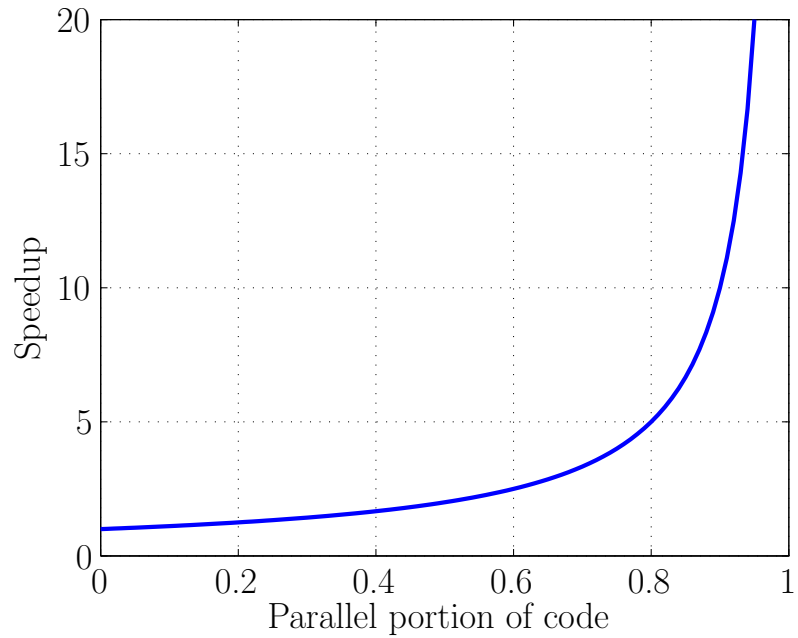
$$\text{speedup} = \frac{\text{serial time}}{\text{parallel time}}$$

  time is in the *wall-clock* sense

- **Amdahl's Law:** assume infinite processors and no overhead

$$\text{ideal max speedup} = \frac{1}{1-\rho}$$

  where $\rho =$ percentage of parallelized code, can depend on input size

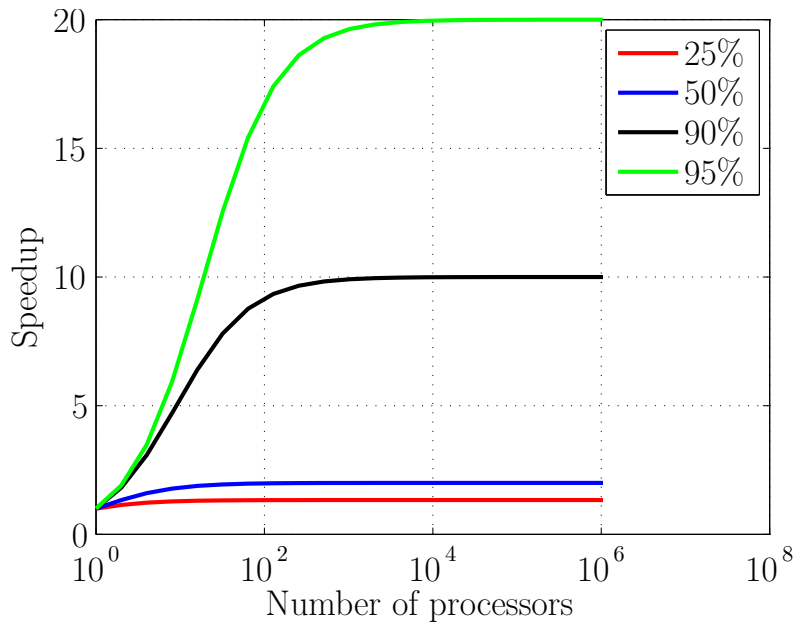


Parallel speedup

- assume N processors and no overhead

$$\text{ideal speedup} = \frac{1}{(\rho/N) + (1-\rho)}$$

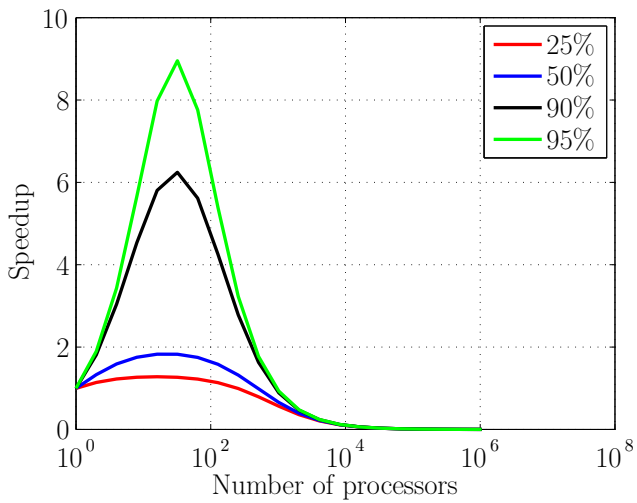
- ρ may also depend on N



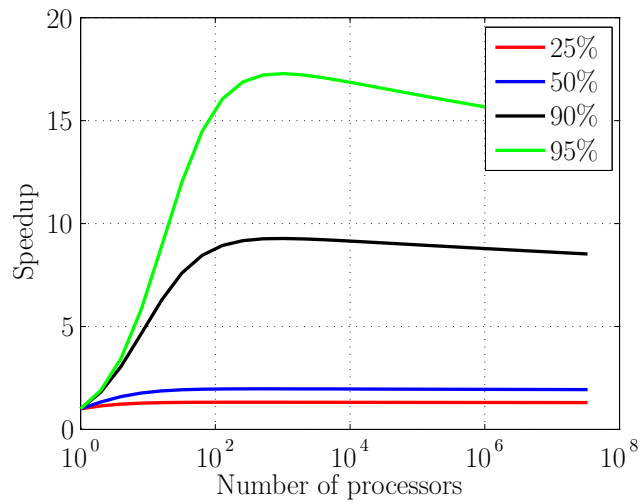
Parallel speedup

- $\varepsilon :=$ parallel overhead
- **in the real world**

$$\text{actual speedup} = \frac{1}{(\rho/N) + (1-\rho) + \varepsilon}$$

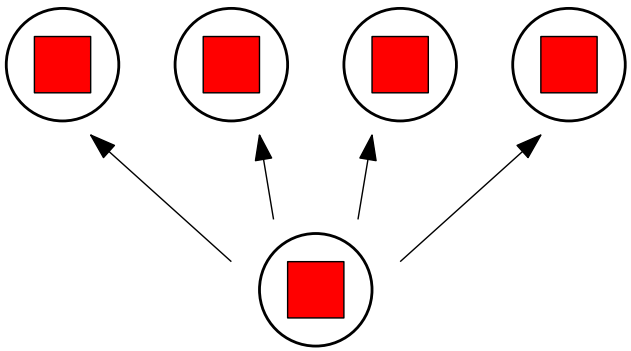


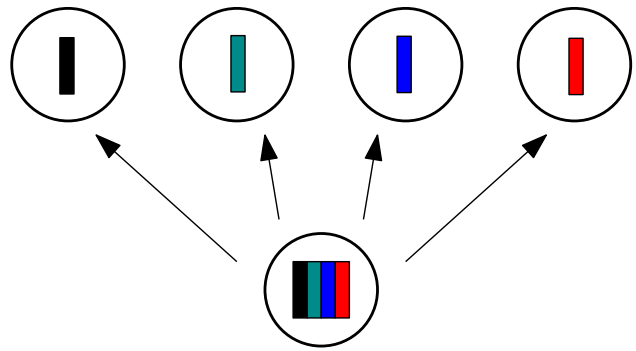
when $\varepsilon = N$

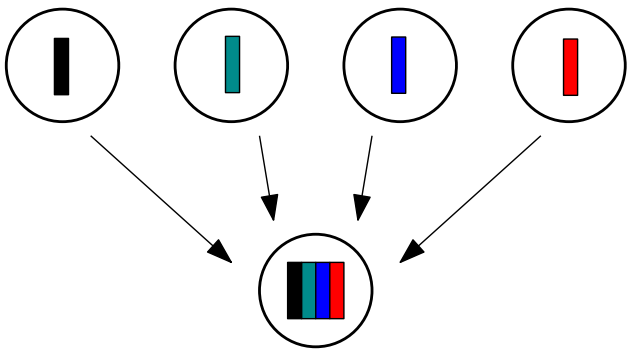


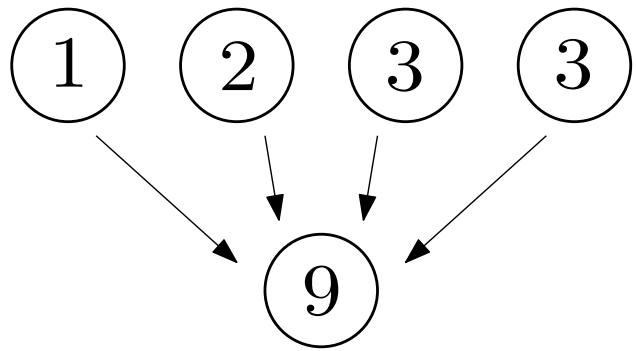
when $\varepsilon = \log(N)$

Basic types of communication


Broadcast


Scatter

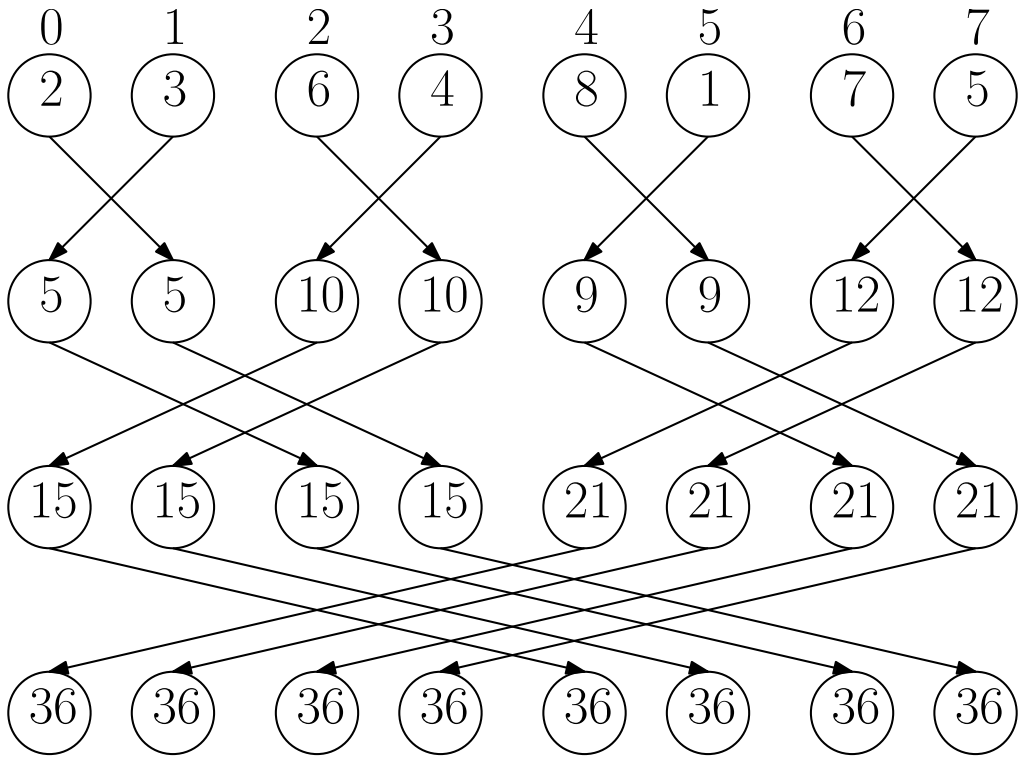

Gather



Reduce

Allreduce

Allreduce *sum* via butterfly communication

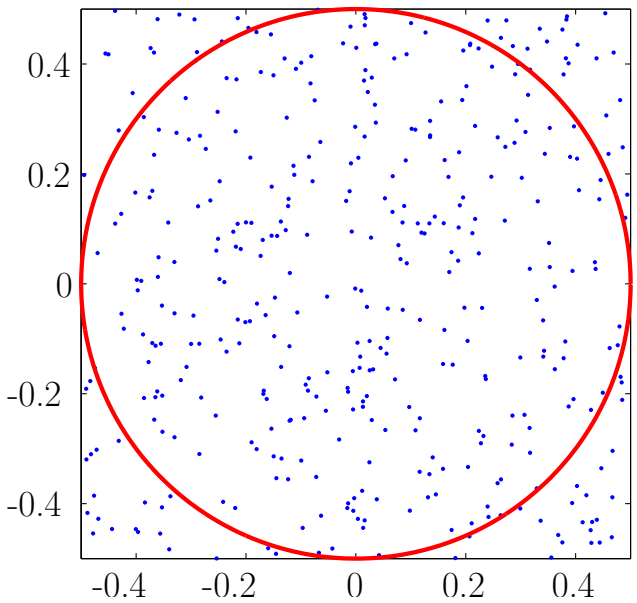


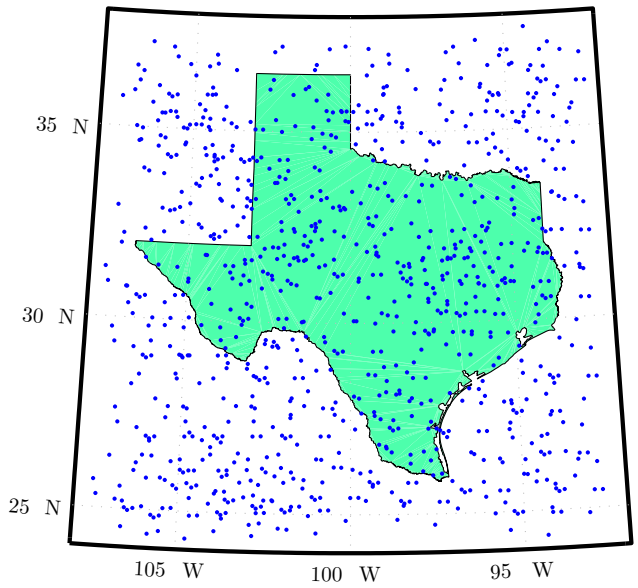
$\log(N)$ layers, N parallel communications per layer

Additional important topics

- **synchronization types:** barrier, lock, synchronous communication
- **load balancing:** static or dynamic distribution of tasks/data so that all processors are busy all time
- **granularity:** fine grain *vs* coarse grain
- **I/O:** a traditional *parallelism killer*, but modern database/parallel file system alleviates the problem (e.g., Hadoop Distributed File System (HDFS))

Example: π, area computation



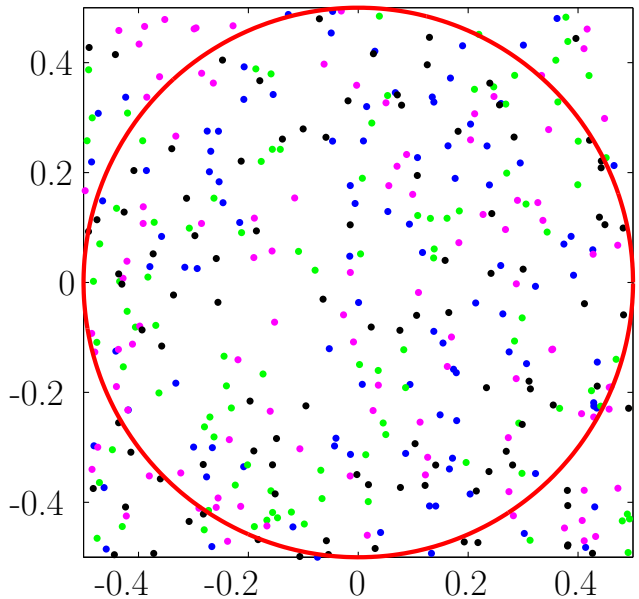


- box area is 1×1
- disc area is $\pi/4$
- ratio of area $\approx$ % inside points
- $\hat{\pi} = 4 \, \frac{\#\text{ points in the disc}}{\#\text{ all the points}}$

Serial pseudo-code

```
1: total = 100000
2: in_disc = 0
3: for k = 1 to total do
4:     x = a uniformly random point in [−1/2, 1/2]^2
5:     if x is in the disc then
6:         in_disc++
7:     end if
8: end for
9: return  4*in_disc/total
```

Parallel π computation



- each sample is independent of others
- use multiple processors to run the for-loop
- use SPMD, process #1 collects results and return $\hat{\pi}$

SPMD pseudo-code

```
1: total = 100000
2: in_disc = 0
3: P = find_total_parallel_processors
4: i = find_my_processor_id
5: total = total/P
6: for k = 1 to total do
7:     x ← a uniformly random point in [0, 1]^2
8:     if x is in the disc then
9:         in_disc++
10:    end if
11: end for
12: if i == 1 then
13:    total_in_disc = reduce(in_disc, SUM)
14:    return 4*total_in_disc/(total* P)
15: end if
```

- requires only one collective communication;
- speedup $= \frac{1}{1/N+O(\log(N))}$ $= \frac{N}{1+cN\log(N)}$, where c is a very small number;
- main loop doesn't require any synchronization.

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. **Parallelize existing algorithms**
3. **Primal decomposition / dual decomposition**
   - **Parallel dual gradient ascent (linearized Bregman)**
   - **Parallel ADMM**
4. Parallel greedy coordinate descent
5. Numerical results with big data

Decomposable objective and constraints enables parallelism

- variables $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$
- embarrassingly parallel (not an interesting case): $\min \sum_i f_i(\mathbf{x}_i)$
- consensus minimization, no constraints

$$\min f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}) + r(\mathbf{x})$$

  LASSO example: $f(\mathbf{x}) = \frac{\lambda}{2} \sum_i \|\mathbf{A}_{(i)}\mathbf{x} - \mathbf{b}_i\|_2^2 + \|\mathbf{x}\|_1$

- coupling variable $\mathbf{z}$

$$\min_{\mathbf{x},\mathbf{z}} f(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{N} [f_i(\mathbf{x}_i, \mathbf{z}) + r_i(\mathbf{x}_i)] + r(\mathbf{z})$$

- coupling constraints

$$\min f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i) + r_i(\mathbf{x}_i)$$

  subject to

  - equality constraints: $\sum_{i=1}^{N} \mathbf{A}_i \mathbf{x}_i = b$
  - inequality constraints: $\sum_{i=1}^{N} \mathbf{A}_i \mathbf{x}_i \leq b$
  - graph-coupling constraints $\mathbf{A}\mathbf{x} = b$, where $\mathbf{A}$ is sparse

- combinations of independent variables, coupling variables and constraints

  example:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{w},\mathbf{z}} \quad & \textstyle\sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z}) \\ \text{s.t.} \quad & \textstyle\sum_i \mathbf{A}_i \mathbf{x}_i \leq \mathbf{b} \\ & \mathbf{B}_i \mathbf{x}_i \leq \mathbf{w}, \ \forall i \\ & \mathbf{w} \in \Omega \end{aligned}$$

- variable coupling $\longleftrightarrow$ constraint coupling:

$$\min \sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z}) \longrightarrow \min \sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z}_i) \quad \text{s.t. } \mathbf{z}_i = \mathbf{z} \ \ \forall i$$

Approaches toward parallelism

- Keep an existing serial algorithm, parallelize its expensive part(s)
- Introduce a new parallel algorithm by
  - formulating an equivalent model
  - replacing the model by an approximate model

where the new model lands itself for parallel computing

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. **Parallelize existing algorithms**
3. Primal decomposition / dual decomposition
   - Parallel dual gradient ascent (linearized Bregman)
   - Parallel ADMM
4. Parallel greedy coordinate descent
5. Numerical results with big data

Example: parallel ISTA

$$\underset{\mathbf{x}}{\text{minimize}}\, \lambda\|\mathbf{x}\|_1 + f(\mathbf{x})$$

- Prox-linear approximation:

$$\arg\min_{\mathbf{x}} \lambda\|\mathbf{x}\|_1 + \langle \nabla f(\mathbf{x}), \mathbf{x} - \mathbf{x}^k \rangle + \frac{1}{2\delta_k}\|\mathbf{x} - \mathbf{x}^k\|_2^2$$

- Iterative soft-threshold algorithm:

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

  - $\text{shrink}(\mathbf{y}, t)$ has a simple close form: `max(abs(y)-t,0).*sign(y)`
  - the dominating computation is $\nabla f(\mathbf{x})$

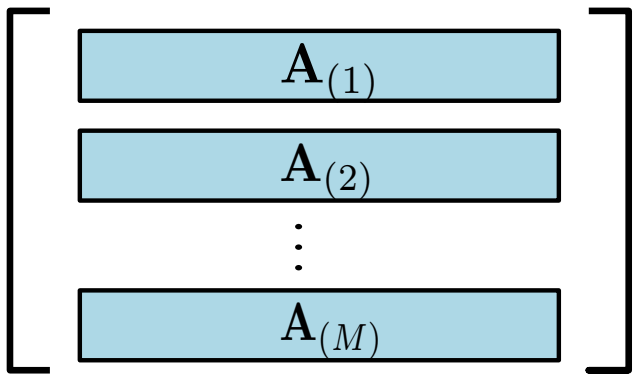
- examples: $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, logistic loss of $\mathbf{A}\mathbf{x} - \mathbf{b}$, hinge loss ... where $\mathbf{A}$ is often dense and can go very large-scale

Parallel gradient computing of $f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b})$: row partition

Assumption: $\mathbf{A}$ is big and dense; $g(\mathbf{y}) = \sum g_i(\mathbf{y}_i)$ decomposable

Then $f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b}) = \sum g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i)$

- let $\mathbf{A}_{(i)}$ and $\mathbf{b}_i$ be the ith row block of $\mathbf{A}$ and $\mathbf{b}$, respectively



- store $\mathbf{A}_{(i)}, \mathbf{b}_i$ on node i
- goal: to compute $\nabla f(\mathbf{x}) = \sum \mathbf{A}_{(i)}^T \nabla g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i)$, or similarly $\partial f(\mathbf{x})$

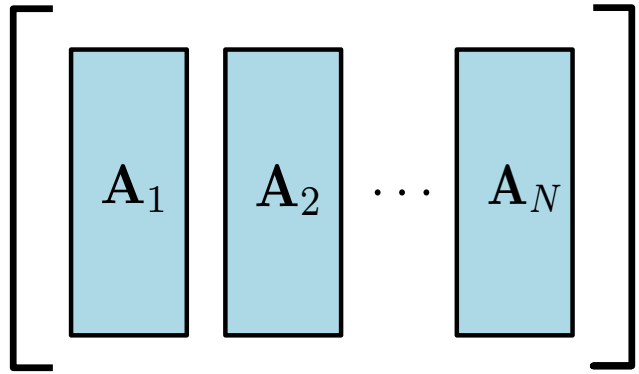
Parallel gradient computing of $f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b})$: row partition

Steps

1. compute $\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i \ \forall i$ in parallel;
2. compute $\mathbf{g}_i = \nabla g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i) \ \forall i$ in parallel;
3. compute $\mathbf{A}_{(i)}^T \mathbf{g}_i \ \forall i$ in parallel;
4. allreduce $\sum_{i=1}^M \mathbf{A}_{(i)}^T \mathbf{g}_i$.

- step 1, 2 and 3 are local computation and communication-free;
- step 4 requires an allreduce sum communication.

Parallel gradient computing of $f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b})$: column partition



column block partition

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{N} \mathbf{A}_i \mathbf{x}_i \implies f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b}) = g(\sum \mathbf{A}_i \mathbf{x}_i + \mathbf{b})$$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \mathbf{A}_1^T \nabla g(\mathbf{A}\mathbf{x} + \mathbf{b}) \\ \mathbf{A}_2^T \nabla g(\mathbf{A}\mathbf{x} + \mathbf{b}) \\ \vdots \\ \mathbf{A}_N^T \nabla g(\mathbf{A}\mathbf{x} + \mathbf{b}) \end{pmatrix}, \quad \text{similar for } \partial f(\mathbf{x})$$

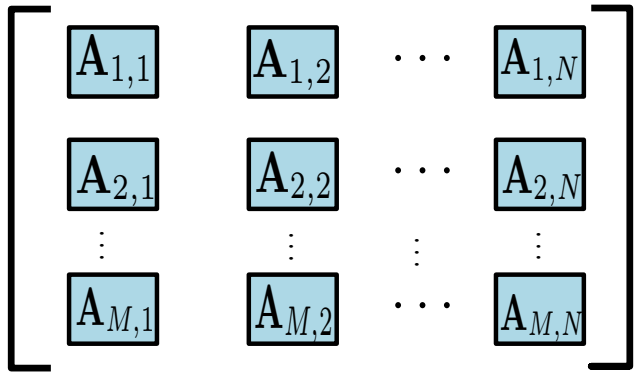
Parallel computing of $\nabla f(\mathbf{x}) = g(\mathbf{A}\mathbf{x} + \mathbf{b})$: column partition

Steps

1. compute $\mathbf{A}_i\mathbf{x}_i + \frac{1}{N}\mathbf{b}$ $\forall i$ in parallel
2. allreduce $\mathbf{A}\mathbf{x} + \mathbf{b} = \sum_{i=1}^{N} \mathbf{A}_i\mathbf{x}_i + \frac{1}{N}\mathbf{b}$;
3. evaluate $\nabla g(\mathbf{A}\mathbf{x} + \mathbf{b})$ $\forall i$ in each process;
4. compute $\mathbf{A}_i^T \nabla g(\mathbf{A}\mathbf{x} + \mathbf{b})$ $\forall i$ in parallel.

- each processor keeps $\mathbf{A}_i$ and $\mathbf{x}_i$, as well as a copy of $\mathbf{b}$;
- step 2 requires an allreduce sum communication;
- step 3 involves duplicated computation in each process (hopefully, g is simple).

Two-way partition



$$f(\mathbf{x}) = \sum g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i) = \sum_i g_i(\sum_j \mathbf{A}_{i,j}\mathbf{x}_j + \mathbf{b}_i)$$

$$\nabla f(\mathbf{x}) = \sum_i \mathbf{A}_{(i)}^T \nabla g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i) = \sum_i \mathbf{A}_{(i)}^T \nabla g_i(\sum_j \mathbf{A}_{i,j}\mathbf{x}_j + \mathbf{b}_i)$$

Two-way partition

Step

1. compute $\mathbf{A}_{i,j}\mathbf{x}_j + \frac{1}{N}\mathbf{b}_i \ \forall i$ in parallel;
2. allreduce $\sum_{j=1}^{N} \mathbf{A}_{i,j}\mathbf{x}_j + \mathbf{b}_i$ for every i;
3. compute $\mathbf{g}_i = \nabla g_i(\mathbf{A}_{(i)}\mathbf{x} + \mathbf{b}_i) \ \forall i$ in parallel;
4. compute $\mathbf{A}_{j,i}^T \mathbf{g}_i \ \forall i$ in parallel;
5. allreduce $\sum_{i=1}^{N} \mathbf{A}_{j,i}^T \mathbf{g}_i$ for every j.

- processor (i, j) keeps $\mathbf{A}_{i,j}$, $\mathbf{x}_j$ and $\mathbf{b}_i$;
- step 2 and 5 require parallel allreduce sum communication;

Example: parallel ISTA for LASSO

LASSO

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

algorithm

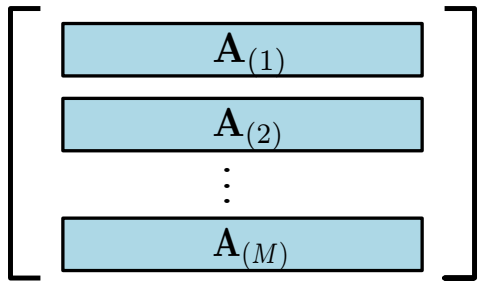
$$\mathbf{x}^{k+1} = \text{shrink}\left(\mathbf{x}^k - \delta_k \mathbf{A}^T\mathbf{A}\mathbf{x}^k + \delta_k \mathbf{A}^T\mathbf{b},\ \lambda\delta_k\right)$$

Serial pseudo-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 ISTA for LASSO

distribute blocks of rows to M nodes

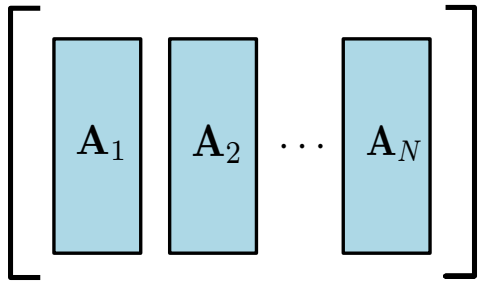


1: initialize $\mathbf{x} = \mathbf{0}$ and δ;
2: $i =$ find_my_processor_id
3: processor i loads $\mathbf{A}_{(i)}$
4: pre-compute $\delta \mathbf{A}^T \mathbf{b}$
5: **while** not converged **do**
6: processor i computes $\mathbf{c}_i = \mathbf{A}_{(i)}^T \mathbf{A}_{(i)} \mathbf{x}$
7: $\mathbf{y} =$ 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
  $\approx \frac{1}{\rho/M + (1-\rho) + O(\log(M))}$
- P 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 $\approx \frac{1}{\rho/N + (1-\rho) + O(\frac{m}{n}\log(N))}$
- P is close to 1
- requires synchronization
- if $m \ll n$, this approach is faster

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. **Primal decomposition / dual decomposition**[3]
   - Parallel dual gradient ascent (linearized Bregman)
   - Parallel ADMM
4. Parallel greedy coordinate descent
5. Numerical results with big data

[3] Dantzig and Wolfe [1960], Spingarn [1985], Bertsekas and Tsitsiklis [1997]

Unconstrained minimization with coupling variables

$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$.

$\mathbf{z}$ is the **coupling/bridging/complicating** variable.

$$\min_{\mathbf{x},\mathbf{z}} f(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z})$$

equivalently to separable objective with **coupling constraints**

$$\min_{\mathbf{x},\mathbf{z}} \sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z}_i) \quad \text{s.t. } \mathbf{z}_i = \mathbf{z} \;\; \forall i$$

Examples

- network utility maximization (NUM), extend to multiple layers[4]
- domain decomposition ($\mathbf{z}$ are overlapping boundary variables)
- system of equations: $\mathbf{A}$ is block-diagonal but with a few dense columns

[4] see tutorial: Palomar and Chiang [2006]

Primal decomposition (bi-level optimization)

Introduce **easier subproblems**

$$g_i(\mathbf{z}) = \min_{\mathbf{x}_i} f_i(\mathbf{x}_i, \mathbf{z}), \quad i = 1, \ldots, N$$

then

$$\min_{\mathbf{x},\mathbf{z}} \sum_{i=1}^{N} f_i(\mathbf{x}_i, \mathbf{z}) \iff \min_{\mathbf{z}} \sum g_i(\mathbf{z})$$

the RHS is called the **master problem**

parallel computing: iterate

1. fix $\mathbf{z}$, update $\mathbf{x}_i$ and compute $g_i(\mathbf{z})$ *in parallel*
2. update $\mathbf{z}$ using a standard method (typically, subgradient descent)

- good for small-dimensional $\mathbf{z}$
- fast if the master problem converges fast

Dual decomposition

Consider

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & \sum_i f_i(\mathbf{x}_i, \mathbf{z}_i) \\ \text{s.t.} \quad & \sum_i \mathbf{A}_i \mathbf{z}_i = \mathbf{b} \end{aligned}$$

Introduce

$$g_i(\mathbf{z}) = \min_{\mathbf{x}_i} f(\mathbf{x}_i, \mathbf{z})$$

$$g_i^*(\mathbf{y}) = \max_{\mathbf{z}} \mathbf{y}^T \mathbf{z} - g_i(\mathbf{z}) \qquad \text{//convex conjugate}$$

Dualization:

$$\Leftrightarrow \quad \min_{\mathbf{x},\mathbf{z}} \max_{\mathbf{y}} \sum_i f_i(\mathbf{x}_i, \mathbf{z}_i) + \mathbf{y}^T (\mathbf{b} - \sum_i \mathbf{A}_i \mathbf{z}_i)$$

(generalized minimax thm.) $$\Leftrightarrow \quad \max_{\mathbf{y}_i} \min_{\mathbf{x}_i,\mathbf{z}_i} \left\{ \sum_i \left(f_i(\mathbf{x}_i, \mathbf{z}_i) - \mathbf{y}^T \mathbf{A}_i \mathbf{z}_i \right) + \mathbf{y}^T \mathbf{b} \right\}$$

$$\Leftrightarrow \quad \max_{\mathbf{y}_i} \left\{ \min_{\mathbf{z}_i} \sum_i \left(g_i(\mathbf{z}_i) - \mathbf{y}^T \mathbf{A}_i \mathbf{z}_i \right) + \mathbf{y}^T \mathbf{b} \right\}$$

$$\Leftrightarrow \quad \min_{\mathbf{y}} \sum_i g_i^*(\mathbf{A}_i^T \mathbf{y}) - \mathbf{y}^T \mathbf{b}$$

Example: consensus and exchange problems

Consensus problem

$$\min_{\mathbf{z}} \sum_i g_i(\mathbf{z})$$

Reformulate as

$$\min_{\mathbf{z}} \sum_i g_i(\mathbf{z}_i) \quad \text{s.t. } \mathbf{z}_i = \mathbf{z} \ \ \forall i$$

Its dual problem is the *exchange problem*

$$\min_{\mathbf{y}} \sum_i g_i^*(\mathbf{y}_i) \quad \text{s.t. } \sum_i \mathbf{y}_i = 0$$

Primal-dual objective properties

Constrained separable problem

$$\min_{\mathbf{z}} \sum_i g_i(\mathbf{z}) \quad \text{s.t. } \sum_i \mathbf{A}_i \mathbf{z}_i = \mathbf{b}$$

Dual problem

$$\min_{\mathbf{y}} \sum_i g_i^*(\mathbf{A}_i^T \mathbf{y}) - \mathbf{b}^T \mathbf{y}$$

- If g_i is proper, closed, convex, $g_i^*(\mathbf{a}_i^T \mathbf{y})$ is sub-differentiable
- If g_i is strictly convex, $g_i^*(\mathbf{a}_i^T \mathbf{y})$ is differentiable
- If g_i is strongly convex, $g_i^*(\mathbf{a}_i^T \mathbf{y})$ is differentiable and has Lipschitz gradient
- Obtaining $\mathbf{z}^*$ from $\mathbf{y}^*$ generally requires strict convexity of g_i or strict complementary slackness

Example:

- $g_i(\mathbf{z}) = |z_i|$ and $g_i^*(\mathbf{a}_i^T \mathbf{y}) = \iota\{-1 \leq \mathbf{a}_i^T \mathbf{y} \leq 1\}$
- $g_i(\mathbf{z}) = \frac{1}{2}|z_i|^2$ and $g_i^*(\mathbf{a}_i^T \mathbf{y}) = \frac{1}{2}|\mathbf{a}_i^T \mathbf{y}|^2$
- $g_i(\mathbf{z}) = \exp(z_i)$ and $g_i^*(\mathbf{a}_i^T \mathbf{y}) = (\mathbf{a}_i^T \mathbf{y}) \ln(\mathbf{a}_i^T \mathbf{y}) - \mathbf{a}_i^T \mathbf{y} + \iota\{\mathbf{a}_i^T \mathbf{y} \geq 0\}$

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. **Primal decomposition / dual decomposition**
   - **Parallel dual gradient ascent (linearized Bregman[5])**
   - Parallel ADMM
4. Parallel greedy coordinate descent
5. Numerical results with big data

[5] Yin, Osher, Goldfarb, and Darbon [2008]

Example: augmented ℓ_1

Change $|z_i|$ into strongly convex $g_i(z_i) = |z_i| + \frac{1}{2\alpha}|z_i|^2$

Dual problem

$$\min_{\mathbf{y}} \sum_i g_i^*(\mathbf{A}_i^T\mathbf{y}) - \mathbf{b}^T\mathbf{y} = \sum_i \underbrace{\frac{\alpha}{2}|\operatorname{shrink}(\mathbf{A}_i^T\mathbf{y})|^2}_{h_i(\mathbf{y})} - \mathbf{b}^T\mathbf{y}$$

Dual gradient iteration (equivalent to linearized Bregman):

$$\mathbf{y}^{k+1} = \mathbf{y}^k - t^k\left(\sum_i \nabla h_i(\mathbf{y}^k) - \mathbf{b}\right), \quad \text{where } \nabla h_i(\mathbf{y}) = \alpha\mathbf{A}_i \operatorname{shrink}(\mathbf{A}_i^T\mathbf{y}).$$

- **Dual is C^1 and unconstrained.** One can apply (accelerated) gradient descent, line search, quasi-Newton method, etc.
- Recover $\mathbf{x}^* = \alpha \operatorname{shrink}(\mathbf{A}_i^T\mathbf{y}^*)$.
- **Easy to parallelize**. Node i computes $\nabla h_i(\mathbf{y})$. One collective communication per iteration.

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. **Primal decomposition / dual decomposition**
   - Parallel dual gradient ascent (linearized Bregman)
   - **Parallel ADMM**[8]
4. Parallel greedy coordinate descent
5. Numerical results with big data

[8] Bertsekas and Tsitsiklis [1997]

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 80s, revived very recently, new convergence results recently

ADMM and parallel/distributed computing

Two approaches:

- parallelize the serial algorithm(s) for ADMM subproblem(s),
- turn problem into a parallel-ready form[9]

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

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

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

Issue: to have *block diagonal* $\mathbf{A}$ and *simple* $\mathbf{B}$, additional variables are often needed, thus increasing the problem size and parallel overhead.

[9] a recent survey Boyd, Parikh, Chu, Peleato, and Eckstein [2011]

Parallel Linearized Bregman vs ADMM on basis pursuit

Basis pursuit:

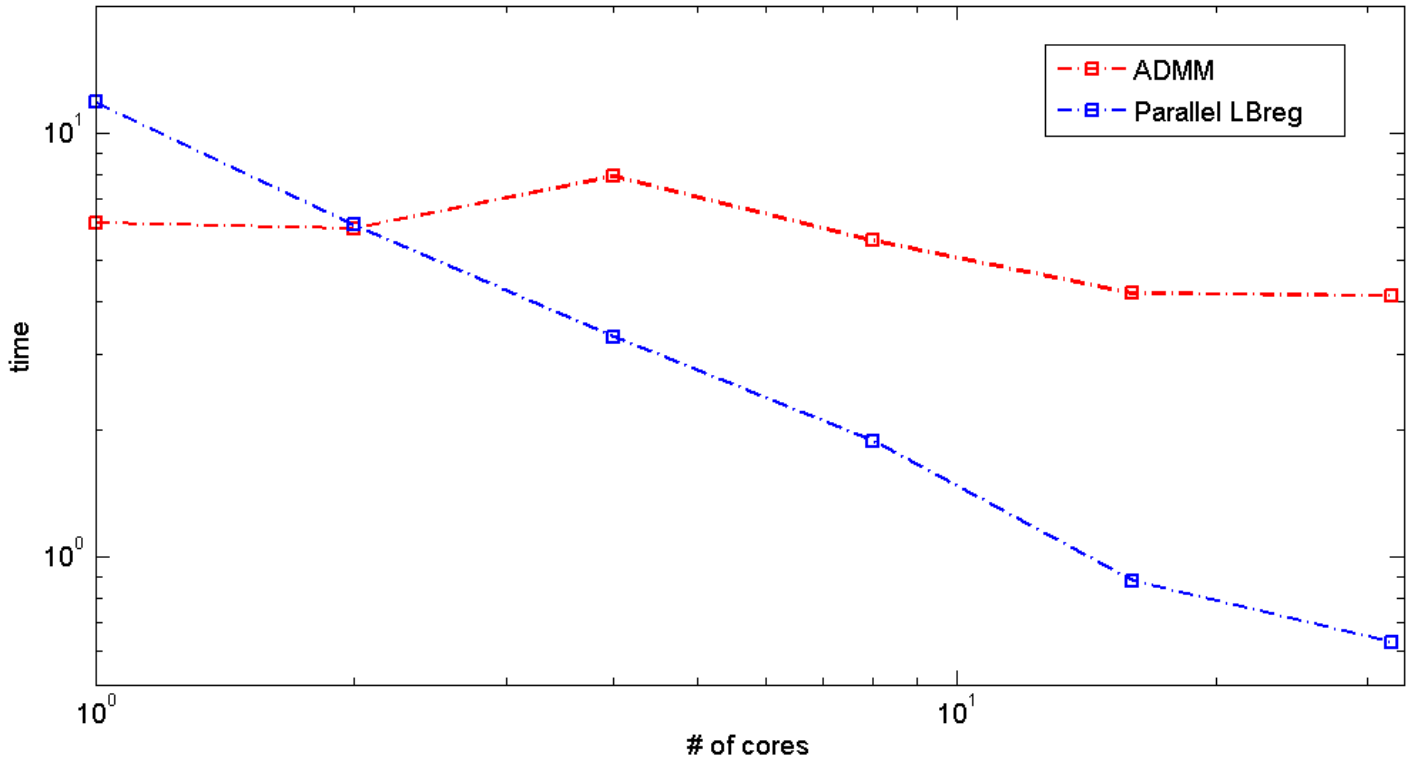
$$\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}.$$

- example with **dense** $\mathbf{A} \in \mathbb{R}^{1024 \times 2048}$;
- distribute $\mathbf{A}$ to N computing nodes

$$\mathbf{A} = [\mathbf{A}_1 \ \mathbf{A}_2 \ \cdots \ \mathbf{A}_N];$$

- compare
  1. parallel ADMM in the survey paper Boyd, Parikh, Chu, Peleato, and Eckstein [2011];
  2. parallel linearized Bregman (dual gradient descent), un-accelerated.
- tested $N = 1, 2, 4, \ldots, 32$ computing nodes;

parallel ADMM v.s. parallel linearized Bregman



parallel linearized Bregman scales much better

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. Primal decomposition / dual decomposition
   - Parallel dual gradient ascent (linearized Bregman)
   - Parallel ADMM
4. **Parallel greedy coordinate descent**
5. Numerical results with big data

(Block)-coordinate descent/update

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

Advantage: update is simple

Disadvantages:

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

Block selection: cycle (Gauss-Seidel), parallel (Jacobi), random, greedy

Specific for sparse optimization: greedy approach[10] can be exceptionally effective (because most time correct variables are selected to update; most zero variables are never touched; "reducing" the problem to a much smaller one.)

[10] Li and Osher [2009]

GRock: greedy coordinate-block descent

Example:

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

- 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 a merit value for each coordinate i

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

2. (*parallel*) compute a merit value for each block j

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

   let s_j be the index of the maximal coordinate within 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 $\mathbf{A}\mathbf{x}$

How large P can be?

P depends on the **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

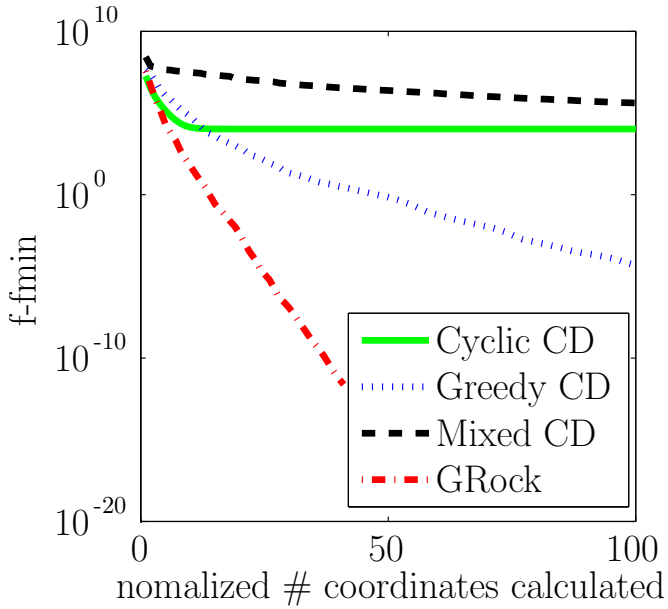
Assume each column of $\mathbf{A}$ has unit 2-norm. If $\rho_P < 2$, GRock with P parallel updates per iteration gives

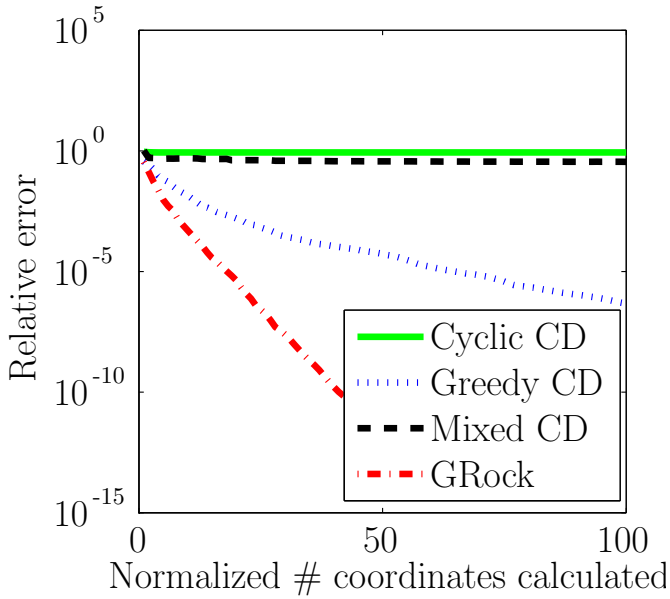
$$\mathcal{F}(\mathbf{x} + \mathbf{d}) - \mathcal{F}(\mathbf{x}) \leq \frac{\rho_P - 2}{2}\beta\|\mathbf{d}\|_2^2$$

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[11] uses $P = 1$
- mixed CD[12] selects $P = 8$ random blocks and best coordinate in each iteration

[11] Li and Osher [2009]

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

Outline

1. Background of parallel computing: benefits, speedup, and overhead
2. Parallelize existing algorithms
3. Primal decomposition / dual decomposition
   - Parallel dual gradient ascent (linearized Bregman)
   - Parallel ADMM
4. Parallel greedy coordinate descent
5. **Numerical results with big data**

Test on Rice cluster STIC

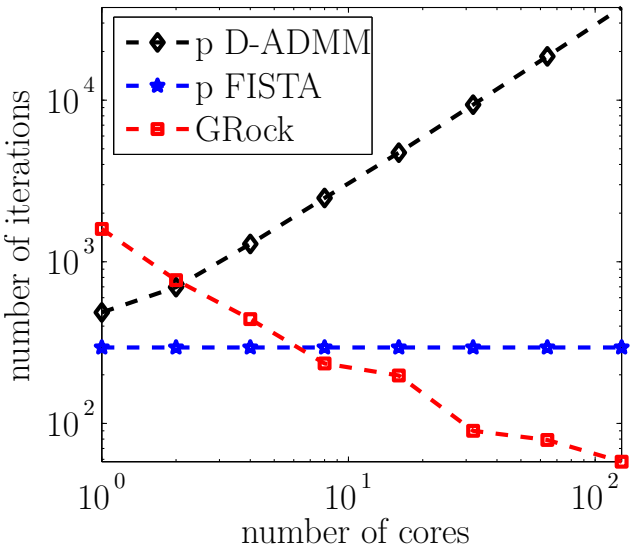
specs:

- 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

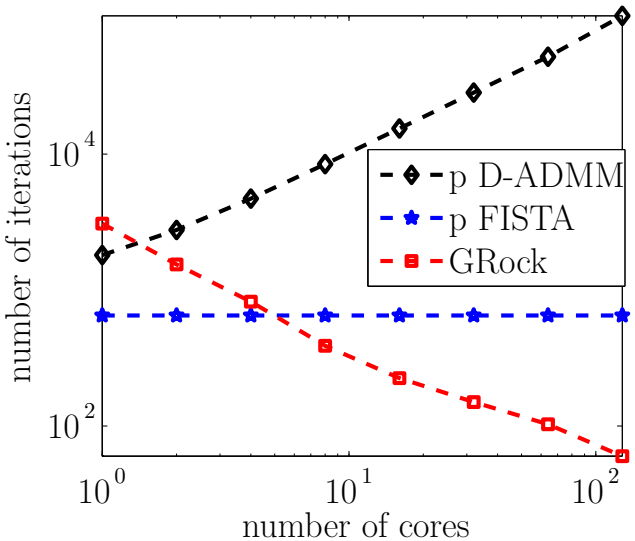
test dataset:

	$\mathbf{A}$ 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

iterations vs cores



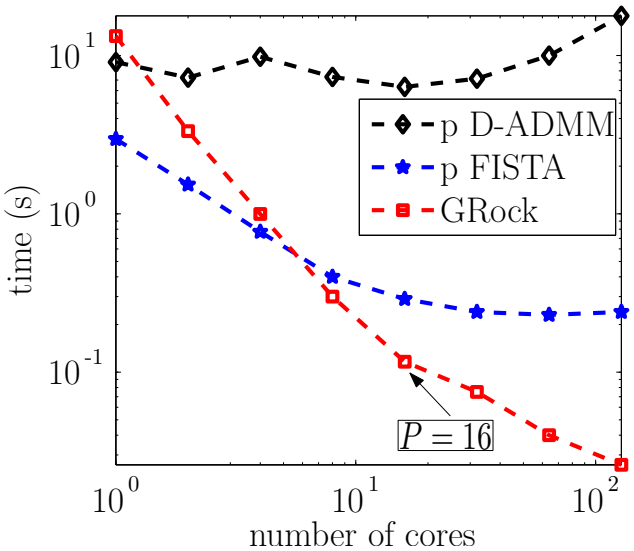
dataset I



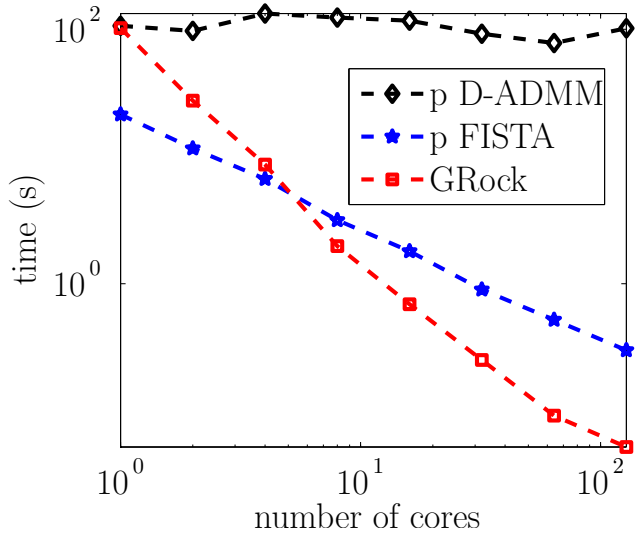
dataset II

Note: we set $P =$ number of cores

time vs cores



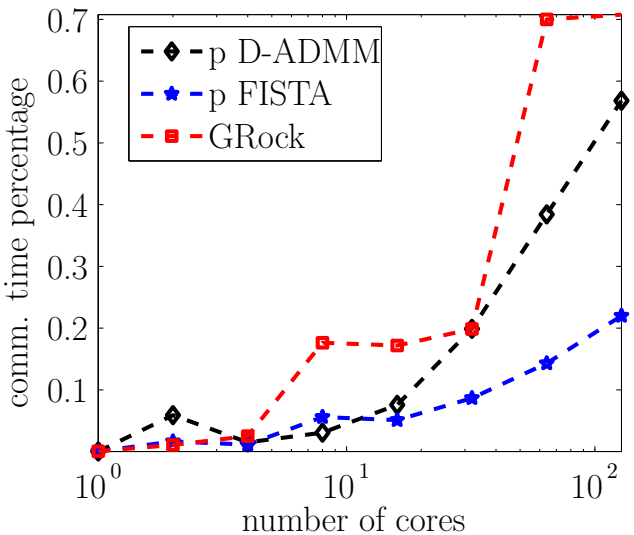
dataset I



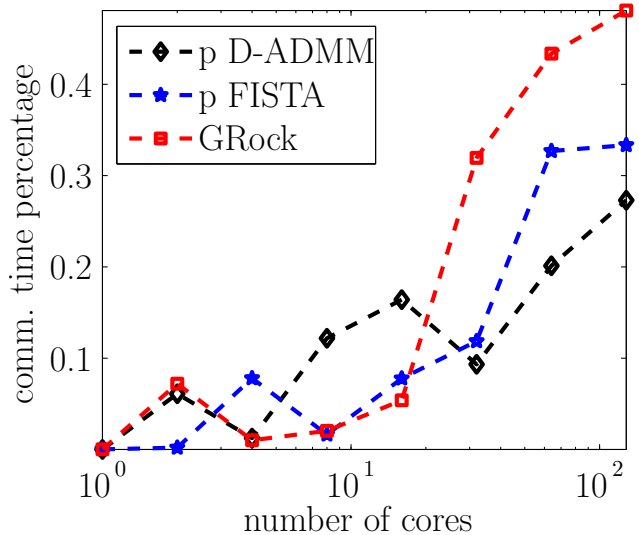
dataset II

Note: we set $P =$ number of cores

(% communication 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

- 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 FISTA[13] 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 and FISTA were capped at 2500 iterations
- GRock used adaptive P and stopped at relative error 1E-5

[13] Beck and Teboulle [2009]

Software codes can be found in instructor's website.

Acknowledgements: Zhimin Peng (Rice), Ming Yan (UCLA). Also, Qing Ling (USTC) and Zaiwen Wen (SJTU)

References:

B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24:227–234, 1995.

D. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries vis ℓ_1 minimization. *Proceedings of the National Academy of Sciences*, 100:2197–2202, 2003.

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

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

G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

J.E. Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32(2):199–223, 1985.

D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods, Second Edition*. Athena Scientific, 1997.

D.P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8): 1439–1451, 2006.

W. Yin, S. Osher, D. Goldfarb, and J. Darbon. Bregman iterative algorithms for l1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168, 2008.

M.P. Friedlander and P. Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18(4):1326–1350, 2007.

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

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.

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.

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

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.