Asynchronous parallel stochastic Kaczmarz and stochastic coordinate descent algorithms

Ji Liu and Stephen Wright

University of Wisconsin-Madison

February 2014

Collaborators

Victor Bittorf (UW-Madison)
Yijun Huang
Chris Ré (UW-Madison $\rightarrow$ Stanford)
Krishna Sridhar (UW-Madison $\rightarrow$ GraphLab)

Rant

Why do we call it "SGD"?

Hypothesis: "Gradient Descent" became "*Stochastic* Gradient Descent"

Yann? Leon?

But it's not a "Descent" method!
The search directions are no longer necessarily "Descent" directions for F.

A Modest Suggestion: Stochastic Gradient Methods.

Motivation

Why study old, slow, simple algorithms?

- Often suitable for machine learning and big-data applications.
- Often a good fit for modern computers (multicore, NUMA, clusters): Parallel, asynchronous versions are possible.
- (Fairly) easy to implement.
- Interesting new analysis, tied to plausible models of parallel computation and data access.

"Asynchronicity is the key to speedup on modern architectures," says Bill Gropp (SIAM CS&E Plenary, Feb 2013).

Kaczmarz for $Ax = b$.

Consider linear equations $Ax = b$, where the equations are consistent and matrix A is $m \times n$, not necessarily square or full rank. Write

$$A = \begin{bmatrix} a_i^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad \text{where } \|a_i\|_2 = 1,\ i = 1, 2, \ldots, m.$$

Iteration j of Randomized Kaczmarz:

- Select row index $i_j \in \{1, 2, \ldots, m\}$ randomly with equal probability.
- Set
$$x_{j+1} \leftarrow x_j - (a_{i_j}^T x_j - b_{i_j}) a_{i_j}.$$

Relationship to SG

Randomized Kaczmarz is equivalent to applying SG to

$$f(x) := \frac{1}{2m} \sum_{i=1}^{m} (a_i^T x - b_i)^2 = \frac{1}{2m} \|Ax - b\|_2^2 = \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

with steplength $\alpha_k \equiv 1$.

However, it is a special case of SG, since the individual gradient estimates

$$\nabla f_i(x) = a_i(a_i^T x - b_i)$$

approach zero as $x \to x^*$. (The "variance" in the gradient estimate shrinks to zero.)

Randomized Kaczmarz Convergence: Linear Rate

Assume A scaled so that $\|a_i\| = 1$ for all i. $\lambda_{\min,\mathrm{nz}}$ denotes minimum nonzero eigenvalue of $A^T A$. $P(\cdot)$ is projection onto solution set.

$$\begin{aligned}\frac{1}{2}\|x_{j+1} - P(x_{j+1})\|^2 &\le \frac{1}{2}\|x_j - a_{i_j}(a_{i_j}^T x_j - b_{i_j}) - P(x_j)\|^2 \\ &= \frac{1}{2}\|x_j - P(x_j)\|^2 - \frac{1}{2}(a_{i_j}^T x_j - b_{i_j})^2.\end{aligned}$$

Taking expectations:

$$\begin{aligned}E\left[\frac{1}{2}\|x_{j+1} - P(x_{j+1})\|^2 \,|\, x_j\right] &\le \frac{1}{2}\|x_j - P(x_j)\|^2 - \frac{1}{2}E\left[(a_{i_j}^T x_j - b_{i_j})^2\right] \\ &= \frac{1}{2}\|x_j - P(x_j)\|^2 - \frac{1}{2m}\|Ax_j - b\|^2 \\ &\le \left(1 - \frac{\lambda_{\min,\mathrm{nz}}}{m}\right)\frac{1}{2}\|x_j - P(x_j)\|^2.\end{aligned}$$

Strohmer and Vershynin (2009)

Asynchronous Random Kaczmarz (Liu, Wright, 2014)

Assumes that x is stored in shared memory, accessible to all cores.

Each core runs a simple process, repeating indefinitely:

- Choose index $i \in \{1, 2, \ldots, m\}$ uniformly at random;
- Choose component $t \in \text{supp}(a_i)$ uniformly at random;
- Read the $\text{supp}(a_i)$-components of x (from shared memory), needed to evaluate $a_i^T x$;
- Update the t component of x:

$$(x)_t \leftarrow (x)_t - \gamma \|a_i\|_0 (a_i)_t (a_i^T x - b_i)$$

for some step size γ (a unitary operation);

Note that x can be updated by other cores between the time it is read and the time that the update is performed.

Differs from basic Random Kaczmarz in that each update is using slightly outdated information.

AsyRK: Global View

From a "central" viewpoint, aggregating the actions of the individual cores, we have the following: At each iteration j:

- Select i_j from $\{1, 2, \dots, m\}$ with equal probability;
- Select t_j from the support of a_{i_j} with equal probability;
- Update component t_j:

$$x_{j+1} = x_j - \gamma \|a_{i_j}\|_0 (a_{i_j}^T x_{k(j)} - b_{i_j}) E_{t_j} a_{i_j},$$

where

- $k(j)$ is some iterate prior to j but no more than τ cycles old: $j - k(j) \le \tau$;
- E_t is the $n \times n$ matrix of all zeros, except for 1 in the (t, t) location.

Assumes consistent reading, that is, the $x_{k(j)}$ used to evaluate the residual is an x that actually existed at some point in the shared memory.

(This condition may be violated if more than one update happens to the supp(a_{i_j})-components of x while they are being read.)

AsyRK Analysis: A Key Element

Key parameters:

- $\mu := \max_{i=1,2,\dots,m} \|a_i\|_0$ (maximum nonzero row count);
- $\alpha := \max_{i,t} \|a_i\|_0 \|AE_t a_i\| \le \mu\|A\|$.

Idea of analysis: Choose some $\rho > 1$ and choose steplength γ small enough that

$$\rho^{-1}\mathbb{E}(\|Ax_j - b\|^2) \le \mathbb{E}(\|Ax_{j+1} - b\|^2) \le \rho\mathbb{E}(\|Ax_j - b\|^2).$$

Not too much change to the residual at each iteration. Hence, don't pay too much of a price for using outdated information in the asynchronous algorithm.

But don't want γ to be too tiny, otherwise overall progress is too slow. Strike a balance.

Main Theorem

Theorem

Choose any $\rho > 1$ and define γ via the following:

$$\psi = \mu + \frac{2\lambda_{\max}\tau\rho^{\tau}}{m}$$

$$\gamma \le \min\left\{\frac{1}{\psi},\ \frac{m(\rho-1)}{2\lambda_{\max}\rho^{\tau+1}},\ m\sqrt{\frac{\rho-1}{\rho^{\tau}(m\alpha^2+\lambda_{\max}^2\tau\rho^{\tau})}}\right\}$$

Then have

$$\rho^{-1}\mathbb{E}(\|Ax_j-b\|^2) \le \mathbb{E}(\|Ax_{j+1}-b\|^2) \le \rho\mathbb{E}(\|Ax_j-b\|^2)$$

$$\mathbb{E}(\|x_{j+1}-P(x_{j+1})\|^2) \le \left(1-\frac{\lambda_{\mathrm{min,nz}}\gamma}{m\mu}(2-\gamma\psi)\right)\mathbb{E}(\|x_j-P(x_j)\|^2),$$

A particular choice of ρ leads to simplified results, in a reasonable regime.

A Particular Choice

Corollary

Assume

$$2e\lambda_{\max}(\tau+1) \le m$$

and set $\rho = 1 + 2e\lambda_{\max}/m$. *Can show that* $\gamma = 1/\psi$ *for this case, so expected convergence is*

$$\mathbb{E}(\|x_{j+1} - P(x_{j+1})\|^2) \le \left(1 - \frac{\lambda_{\mathrm{min,nz}}}{m(\mu+1)}\right) \mathbb{E}(\|x_j - P(x_j)\|^2).$$

In the regime $2e\lambda_{\max}(\tau+1) \le m$ considered here the delay τ doesn't really interfere with convergence rate.

High-Probability Result

Obtained with Markov inequality.

$\textsc{AsyRK}$ with the choices of ρ and γ above converges to precision ϵ with probability at least $1 - \eta$ in

$$K \geq \frac{m(\mu + 1)}{\lambda_{\min,\text{nz}}} \left| \log \frac{\|x_0 - P(x_0)\|^2}{\eta\epsilon} \right| \quad \text{iterations.}$$

Discussion

For random matrices A with unit rows, we have $\lambda_{\max} \approx (1 + O(m/n))$, with high probability, so that $\tau = O(m) = O(n)$.

Conditions on τ are less strict than for asynchronous random algorithms for optimization problems. (Typically $\tau = O(n^{1/4})$ or $\tau = O(n^{1/2})$ for coordinate descent methods.)

Complexity Comparisons

- δ is fractional sparsity of the matrix A;
- RK is regular (serial) randomized Kaczmarz;
- $\textsc{AsySCD}$ is an asynchronous stochastic coordinate descent method applied to $(1/2)\|Ax-b\|^2$;
- $L_{\max}$ is max diagonal of A^TA; L_{res} is max row norm of A^TA;

algorithms	RK	AsySCD	AsyRK
# ops/iter.	$O(\delta n)$	$\min\{O(\delta^2 mn),\ O(n)\}$	$O(\delta n)$
rate (iter)	$1-\lambda_{\min}/m$	$1-\lambda_{\min}/(2nL_{\max})$	$1-\lambda_{\min}/(m(\mu+1))$
# procs	1	$O\left(\sqrt{n}L_{\max}/L_{\text{res}}\right)$	$O\left(m/\lambda_{\max}\right)$
rate (runtime)	$1-O\left(\frac{\lambda_{\min}}{\delta mn}\right)$	$1-O\left(\frac{\lambda_{\min}}{n^{1.5}L_{\text{res}}\min\{\delta^2 m,\ 1\}}\right)$	$1-O\left(\frac{\lambda_{\min}}{\delta^2 n^2\lambda_{\max}}\right)$

Comparing runtimes, we see that for normalized "random" matrices ($\lambda_{\max}=O(1)$, $L_{\text{res}}=O(1)$):

- $\textsc{AsyRK}$ faster than RK unless $m<\delta n$.
- $\textsc{AsyRK}$ faster than $\textsc{AsySCD}$ unless $\delta > O(n^{-1/4})$.

Sparsity is better for $\textsc{AsyRK}$.

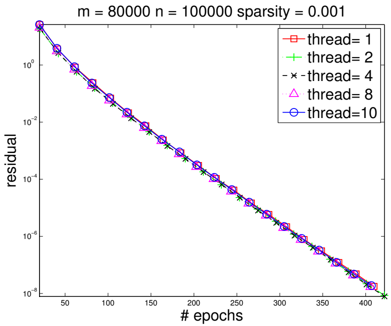
AsyRK: Near-Linear Speedup

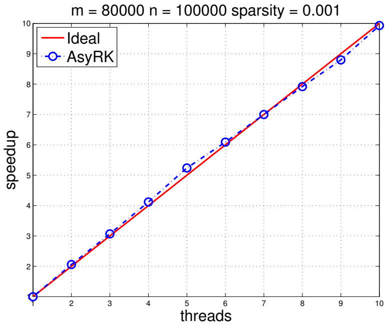
Run on an Intel Xeon 40-core machine. Used one socket — 10 cores).[1]

Say more about the implementation - epochs with shuffling in between.

Sparse Gaussian random matrix $A \in \mathbb{R}^{m \times n}$ with $m = 100000$ and $n = 80000$, sparsity $\delta = .001$.

See linear speedup in the number of cores.



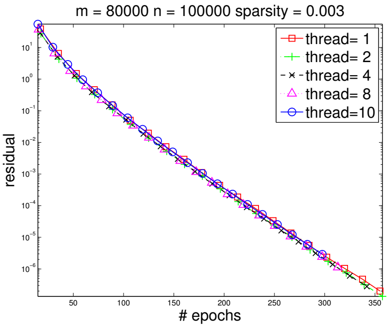


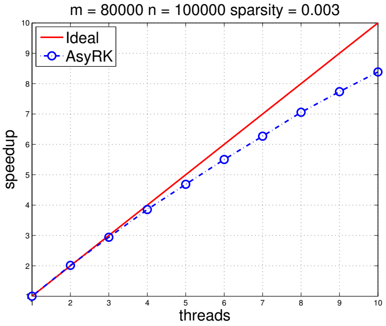
[1]Programming becomes much trickier to exploit nonuniform memory access (NUMA) across multiple sockets; see later!

AsyRK: Near-Linear Speedup

Sparse Gaussian random matrix $A \in \mathbb{R}^{m \times n}$ with $m = 100000$ and $n = 80000$, sparsity $\delta = .003$.

See slight dropoff from linear speedup for this slightly less-sparse problem.





Experiment: Comparison with AsySCD

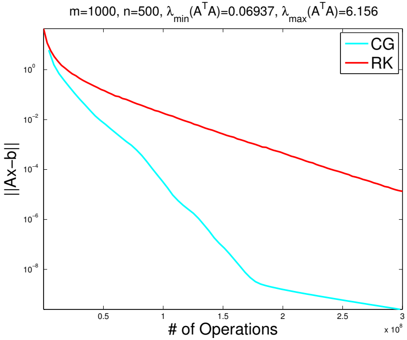
Compare of running time and number of epochs (matrix-vector multiplications) on 10 cores, to attain a residual of 10^{-5}.

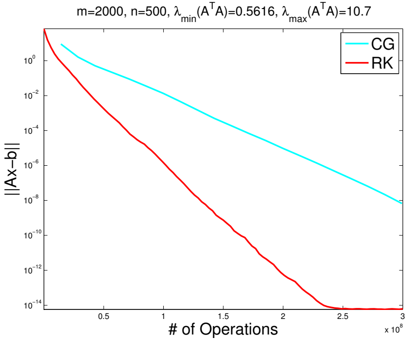
synthetic data			size (MB)	running time (sec)		epochs	
m	n	δ		AsySCD	AsyRK	AsySCD	AsyRK
80k	100k	0.0005	43	39.	3.6	199	195
80k	100k	0.001	84	170.	7.6	267	284
80k	100k	0.003	244	1279.	18.4	275	232
500k	1000k	0.00005	282	54.	5.8	19	19
500k	1000k	0.0001	550	198.	10.4	24	30
500k	1000k	0.0002	1086	734.	15.0	29	31

Per-iteration complexity of a coordunate-descent method is much higher, due to lower sparsity of $A^T A$.

RK vs Conjugate Gradient

We compare serial implementations of RK and CG. (The benefits of multicore implementation are similar for both.) Random A, $\delta = .1$.





CG does better in the more ill-conditioned case, probably due to nice distribution of dominant eigenvalues of A^TA. (Note slower convergence in later stages.) RK is competitive in the well-conditioned case.

Overview

Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm (AsySPCD)

$$\min_x : \ F(x) := f(x) + g(x) \tag{1}$$

- $f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$ is convex and differentiable;
- $g(\cdot) : \mathbb{R}^n \mapsto \mathbb{R} \cup \{+\infty\}$ is a proper closed convex real value extended function;
- $g(x)$ is separable: $g(x) = \sum_{i=1}^n g_i((x)_i)$, $g_i(\cdot) : \mathbb{R} \mapsto \mathbb{R} \cup \{+\infty\}$.

Instances of $g(x)$:

- Unconstrained: $g(x) = \text{constant}$.
- Box constrained: $g(x) = \sum_{i=1}^n \mathbf{1}_{[a_i,b_i]}((x)_i)$ where $\mathbf{1}_{[a_i,b_i]}$ is an indicator function for $[a_i, b_i]$;
- ℓ_p norm regularization: $g(x) = \|x\|_p^p$ where $p \geq 1$.

Instances

Problems that fit this framework include the following:

- least squares: $\min_x \quad \frac{1}{2}\|Ax-b\|^2$;
- LASSO: $\min_x \quad \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1$;
- support vector machine (SVM) with squared hinge loss:

$$\min_w \quad C\sum_i \max\{y_i(x_i^T w - b), 0\}^2 + \frac{1}{2}\|w\|^2$$

- support vector machine: dual form with bias term:

$$\min_{0\le\alpha\le C\mathbf{1}} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i.$$

Instances (continued)

- logistic regression with ℓ_p norm regularization ($p = 1, 2$):

$$\min_x \quad \frac{1}{n} \sum_i \log(1 + \exp(-y_i {x_i}^T w)) + \lambda \|w\|_p^p$$

- semi-supervised learning (Tikhonov Regularization)

$$\min_f \quad \sum_{i \in \{\text{labeled data}\}} (f_i - y_i)^2 + \lambda f^T L f$$

  where L is the Laplacian matrix.

- relaxed linear program:

$$\min_{x \geq 0} \quad c^T x \quad s.t. \quad Ax = b \quad \Rightarrow \quad \min_{x \geq 0} \quad c^T x + \lambda \|Ax - b\|^2$$

Stochastic Proximal Coordinate Descent SPCD

Define prox-operator $\mathcal{P}_h$ for a convex function h:

$$\mathcal{P}_h(y) = \arg\min_x \frac{1}{2}\|x - y\|^2 + h(x).$$

(It's nonexpansive: $\|\mathcal{P}_h(y) - \mathcal{P}_h(z)\| \le \|y - z\|$.)

Repeat: Select a coordinate i and compute the coordinate gradient;

$$(x)_i \leftarrow \mathcal{P}_{\gamma g_i}(\underbrace{(x)_i - \alpha \nabla_i f(x)}_{\text{coordinate gradient}}),$$

for some step length α.

Prox-Operator Examples

Prox-operators can be executed efficiently in many cases.

Examples:

- $g(x) = |x|$: soft thresholding operation

$$\mathcal{P}_{\lambda g}(z) = \operatorname{sgn}(z) \max\{|z| - \lambda, 0\}.$$

- $g(x) = \mathbf{1}_{[a,b]}$: projection operation

$$\mathcal{P}_{\lambda g}(z) = \arg\min_{x \in [a,b]} \frac{1}{2}\|x - z\|^2 = \operatorname{mid}(a, b, z).$$

Local View of AsySPCD

Steplength depends on $L_{\max}$: component Lipschitz constant ("max diagonal of Hessian")

$$\|\nabla f(x + te_j) - \nabla f(x)\|_\infty \le L_{\max}|t| \quad \forall x \in \mathbb{R}^n, t \in \mathbb{R}, j = 1, 2, \dots, n.$$

All processors run a stochastic coordinate descent process concurrently and without synchronization:

- Select a coordinate $i \in \{1, 2, \dots, n\}$ uniformly at random;
- Read "x" from the shared memory and compute the i gradient component using "x":
$$d_i \leftarrow \nabla_i f(x);$$
- Update "x" in the shared memory by the proximal operation, performed atomically:
$$(x)_i \leftarrow \mathcal{P}_{(\gamma/L_{\max})g_i}\left((x)_i - \frac{\gamma}{L_{\max}} d_i\right),$$
for some steplength $\gamma > 0$.

Global View of AsySPCD

Aggregate the actions of the individual cores against a central clock.
Iteration j:

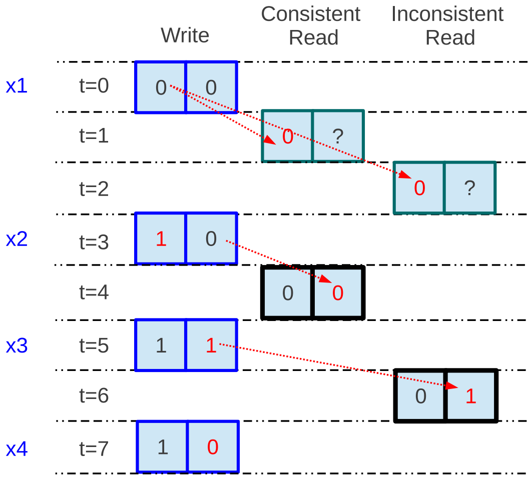
- Choose $i(j) \in \{1, 2, \cdots, n\}$ uniformly at random;
- Read components of x from shared memory needed to compute $\nabla_{i(j)} f$, denoting the local version of x by $\hat{x}_j$;
- Update compoment $i(j)$ of x (atomically):

$$(x_{j+1})_{i(j)} \leftarrow \mathcal{P}_{(\gamma/L_{\max})g_{i(j)}}\left((x_j)_{i(j)} - \frac{\gamma}{L_{\max}} \nabla_{i(j)} f(\hat{x}_j)\right).$$

Note that $\hat{x}_j$ may not never appear in shared memory at any point in time. The elements of x may have been updated repeatedly during reading of $\hat{x}_j$, which means that the components of $\hat{x}_j$ may have different "ages."

We call this phenomenon **inconsistent read**.

Consistent Read vs. Inconsistent Read



Consistent / Inconsistent Read

Consistent vs Inconsistent

Most recent analyses of multicore coordinate descent algorithms have used some sort of consistent-read assumption or a synchronization step, though computational results usually are obtained without enforcing these conditions.

A notable exception is Bertsekas and Tsitsiklis (1989), who describe analyze an asynchronous coordinate method for fixed-point problem $x = q(x)$. over a separable closed convex feasible region. Linear convergence is established if the component ages are bounded and q is a maximum-norm contraction.

To apply this approach to our problem can define (for some $\alpha > 0$)

$$q(x) := \mathcal{P}_{\alpha g}(x - \alpha \nabla f(x)).$$

When applied to minimization of a convex quadratic, the contraction condition amounts to a diagonal dominance condition on $\nabla^2 f$ — a stronger condition than strong convexity, and stronger than what we assume here.

Expressing Read-Inconsistency

The difference between $\hat{x}_j$ and x_j is expressed in terms of the "missed updates:"

$$x_j = \hat{x}_j + \sum_{t \in K(j)} (x_{t+1} - x_t)$$

where $K(j)$ defines the iterate set of updates missed in reading $\hat{x}_j$.

Here we assume τ to be the upper bound of ages of all elements in $K(j)$ for all j:

$$\tau \geq j - \min\{t \mid t \in K(j)\}.$$

Example: our assumptions would be satisfied with $\tau = 10$ when

$$x_{100} = \hat{x}_{100} + \sum_{t \in \{91,95,98,99\}} (x_{t+1} - x_t)$$

τ is related strongly to the number of cores / processors that can be used in the computation. The number of updates we would expect to miss between reading and updating x is about equal to the number of cores.

Notation

- $L_{\max}$: component Lipschitz constant ("max diagonal of Hessian")

$$\|\nabla f(x + te_j) - \nabla f(x)\|_\infty \leq L_{\max}|t| \quad \forall x \in \mathbb{R}^n, t \in \mathbb{R}, i;$$

- L_{res}: restricted Lipschitz constant ("max row-norm of Hessian")

$$\|\nabla f(x + te_i) - \nabla f(x)\| \leq L_{\text{res}}|t| \quad \forall x \in \mathbb{R}^n, t \in \mathbb{R}, i;$$

- $\Lambda := L_{\text{res}}/L_{\max}$ measures the degree of diagonal dominance. It's 1 for separable f, 2 for convex quadratic f with diagonally dominant Hessian, $\sqrt{n}$ for general quadratic.
- S: the solution set of (1);

Key to Analysis

Recall iteration:

$$(x_{j+1})_{i(j)} = \mathcal{P}_{(\gamma/L_{\max})g_{i(j)}}\left((x_j)_{i(j)} - \frac{\gamma}{L_{\max}}\nabla_{i(j)} f(\hat{x}_j)\right).$$

Choose some $\rho > 1$ and choose γ so that

$$\mathbb{E}(\|x_j - x_{j-1}\|^2) \le \rho\mathbb{E}(\|x_{j+1} - x_j\|^2) \quad \text{“}\rho\text{-condition”}.$$

Not too much change in the gradient over each iteration, so not too much price to pay for using inexact information, in the asynchronous setting.

Want to choose γ small enough to satisfy this property but large enough to get a better convergence rate.

Main Assumption: Optimal Strong Convexity (OSC)

Optimal strong convexity parameter $l > 0$

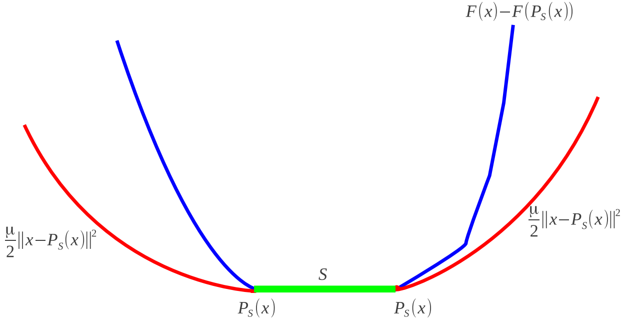
$$F(x) - F(\mathcal{P}_S(x)) \geq \frac{l}{2}\|x - \mathcal{P}_S(x)\|^2$$

for all $x \in \text{dom}F$.

Weaker than usual strong convexity — allows nonunique solutions, for a start. Examples:

- $F(x) = f(Ax)$ with strongly convex f.
- (Can add an indicator function for a closed convex set, or a convex regularizer like $\|x\|_p^p$ to this.)
- Squared hinge loss: $F(x) = \sum_k \max(a_k^T x - b_k,\ 0)^2$;

An OSC (but not strongly convex) function:


$F(x) - F(P_S(x))$
$\frac{\mu}{2}\|x - P_S(x)\|^2$
$\frac{\mu}{2}\|x - P_S(x)\|^2$
S
$P_S(x)$
$P_S(x)$

Main Theorem: OSC yields a Linear Rate

Theorem

For any $\rho > 1 + 4/\sqrt{n}$, define

$$\theta := \frac{\rho^{(\tau+1)/2} - \rho^{1/2}}{\rho^{1/2} - 1} \quad \theta' := \frac{\rho^{(\tau+1)} - \rho}{\rho - 1} \quad \psi := 1 + \frac{\tau\theta'}{n} + \frac{\Lambda\theta}{\sqrt{n}}.$$

and choose

$$\gamma \le \min\left\{\frac{1}{\psi}, \frac{\sqrt{n}(1-\rho^{-1}) - 4}{4(1+\theta)\Lambda}\right\}.$$

Then the "ρ-condition" is satisfied at all j, and we have

$$\begin{aligned}&\mathbb{E}\|x_j - \mathcal{P}_S(x_j)\|^2 + 2\gamma(\mathbb{E}F(x_j) - F^*)\\ &\le \left(1 - \frac{l}{n(l+\gamma^{-1})}\right)^j \left(\|x_0 - \mathcal{P}_S(x_0)\|^2 + 2\gamma(F(x_0) - F^*)\right).\end{aligned}$$

Notes on the Result

Rate depends intuitively on the various quantities involved:

- Smaller $\gamma \Rightarrow$ slower rate;
- Smaller $I \Rightarrow$ slower rate;
- Larger $\Lambda = L_{\text{res}}/L_{\text{max}}$ implies smaller γ and thus slower rate.
- Larger delay $\tau \Rightarrow$ slower rate.

Dependence on ρ is a but more complicated, but best to choose ρ near its lower bound.

Special Case

Corollary

Consider the regime in which τ *satisfies*

$$4e\Lambda(\tau+1)^2 \le \sqrt{n},$$

and define

$$\rho = \left(1 + \frac{4e\Lambda(\tau+1)}{\sqrt{n}}\right)^2.$$

Thus we can choose $\gamma = \frac{1}{2}$, *and the rate simplifies to:*

$$\mathbb{E}(F(x_j) - F^*) \le \left(1 - \frac{l}{n(l + 2L_{\max})}\right)^j (L_{\max}\|x_0 - \mathcal{P}_S(x_0)\|^2 + F(x_0) - F^*).$$

If the diagonal dominance properties are good ($\Lambda \sim 1$) we have $\tau \sim n^{1/4}$.

In earlier work, with consistent read and no regularization, get $\tau \sim n^{1/2}$.

General Convex (without OSC): Sublinear Rate

Theorem

Define ψ and γ as in the main theorem, have

$$\mathbb{E}(F(x_j) - F^*) \leq \frac{n(L_{\max}\gamma^{-1}\|x_0 - \mathcal{P}_S(x_0)\|^2 + 2(F(x_0) - F^*))}{2(j+n)}.$$

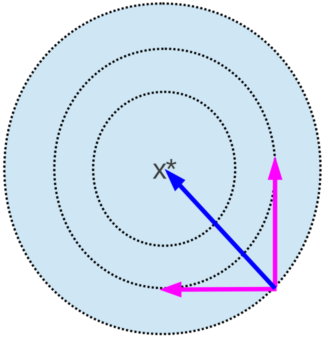
Roughly "$1/j$" behavior (sublinear rate)

Corollary

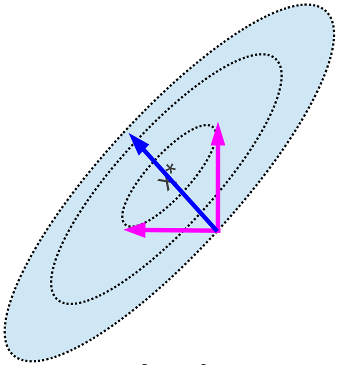
Assuming $4e\Lambda(\tau+1)^2 \leq \sqrt{n}$ and setting ρ and $\gamma = 1/2$ as above, we have

$$\mathbb{E}(F(x_j) - F^*) \leq \frac{n(L_{\max}\|x_0 - \mathcal{P}_S(x_0)\|^2 + F(x_0) - F^*)}{j+n}.$$

Diagonalicity



$\Lambda \approx 1$



$\Lambda \gg 1$

Computational Experiments

Implemented on a 40-core Intel Xeon, containing 4 sockets $\times$ 10 cores.

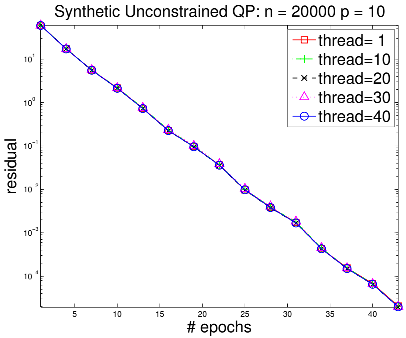
We don't do "sampling with replacement" as in the algorithm described above. Rather, each thread/core is assign a subset of gradient components, and sweeps through these in order: "sampling without replacement."

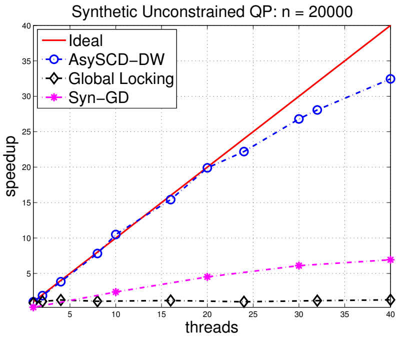
The order of indices is shuffled periodically - either between every pass, or less frequently.

Unconstrained: 4-socket, 40-core Intel Xeon

$$\min_x \quad \|Ax - b\|^2 + 0.5\|x\|^2$$

where $A \in \mathbb{R}^{m \times n}$ is a Gaussian random matrix ($m = 6000$, $n = 20000$, data size≈3GB, columns are normalized to 1). $\Lambda \approx 2.2$. Choose $\gamma = 1$. 3-4 seconds to achieve the accuracy 10^{-5} on 40 cores.

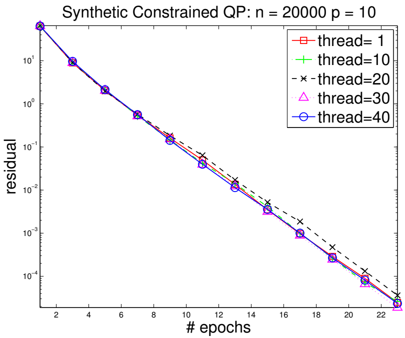


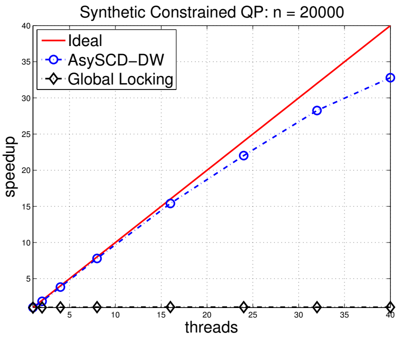


Constrained: 4-socket, 40-core Intel Xeon

$$\min_{x \geq 0} \quad (x - z)^T (A^T A + 0.5I)(x - z) \quad ,$$

where $A \in \mathbb{R}^{m \times n}$ is a Gaussian random matrix ($m = 6000$, $n = 20000$, columns are normalized to 1) and z is a Gaussian random vector. $L_{\text{res}}/L_{\max} \approx 2.2$. Choose $\gamma = 1$.

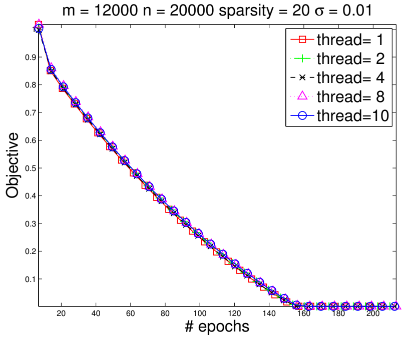


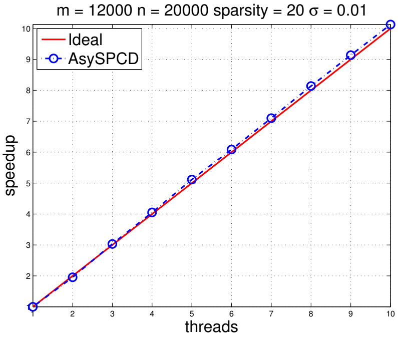


Experiments: 1-socket, 10-core Intel Xeon

$$\min_{x} \quad \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1,$$

where $A \in \mathbb{R}^{m\times n}$ is a Gaussian random matrix ($m = 12000$, $n = 20000$, data size≈3GB),$b = A * \text{sprandn}(n,1,20) + 0.01 * \text{randn}(n,1)$, and $\lambda = 0.2\sqrt{m\log(n)}$. $L_{\text{res}}/L_{\text{max}} \approx 2.2$. Choose $\gamma = 1$.





Extreme Linear Programming

State-of-the-art solvers for large linear programs (LPs) are based on simplex and interior-point methods. An alternative approach based on

- augmented Lagrangian / proximal-point
- iterative solvers for the bounded-QP subproblems (SOR, CG)

were studied in the late 1980s

- O. L. Mangasarian and R. DeLeone, “Serial and Parallel Solution of Large-Scale Linear Program by Augmented Lagrangian Successive Overrelaxations,” 1987.
- S. J. Wright, “Implementing Proximal-Point Methods for Linear Programming,” JOTA, 1990

These showed some promise on random, highly degenerate problems, but were terrible on the netlib test set and other problems arising in practice.

But this approach has potential appeal for:

- Cases in which only crude approximate LP solutions are needed.
- No matrix factorizations or multiplications are required. (Thus may be good for special problems, at extreme scale.)
- Multicore implementation is easy, when asynchronous solver is used on the QP subproblems.

Basics of the Approach

Primal-dual pair:

$$\min_x \; c^T x \;\text{ s.t. }\; Ax = b, \;\; x \ge 0$$

$$\max_u \; b^T u \;\text{ s.t. }\; A^T u \le c.$$

"Proximal method of multipliers" subproblem is a bound-constrained convex QP:

$$x(\beta) := \arg\min_{x \ge 0} \; c^T x - \bar{u}^T (Ax - b) + \frac{\beta}{2} \|Ax - b\|^2 + \frac{1}{2\beta} \|x - \bar{x}\|_2^2,$$

where $(\bar{x}, \bar{u})$ is an estimate of the primal-dual solution and β is a penalty parameter.

Can solve a sequence of these, with updates to $\bar{u}$ and $\bar{x}$, and possible increases in β, in the familiar style of augmented Lagrangian.

A Perturbation Result

Make use of Renegar's measures δ_P and δ_D of relative distance to primal and dual infeasibility.

Theorem

Suppose that δ_P and δ_D are both positive, and let (x^*, u^*) be any primal-dual solution pair. If we define $C_* := \max(\|x^* - \bar{x}\|, \|u^* - \bar{u}\|)$, then the unique solution $x(\beta)$ of the QP subproblem satisfies

$$\|Ax(\beta) - b\| \le (1/\beta)(1 + \sqrt{2})C_*, \quad \|x(\beta) - x^*\| \le \sqrt{6}C_*.$$

If in addition we have

$$\beta \ge \frac{10C_*}{\|d\| \min(\delta_P, \delta_D)},$$

then

$$|c^Tx^* - c^Tx(\beta)| \le \frac{1}{\beta}\left[\frac{25C_*}{2\delta_P\delta_D} + 6C_*^2 + \sqrt{6}\|\bar{x}\|C_*\right].$$

LP Rounding Approximations

There are numerous NP-hard problems for which approximate solutions can be found using linear programming followed by rounding. Typical process:

- Construct a MIP formulation;
- Relax to an LP (replace binary variables by $[0, 1]$ intervals);
- Solve the LP approximately;
- Use LP solution to construct a feasible MIP solution ("rounding").

Examples: Vertex cover, set cover, set packing, multiway cut, maximal independent set.

Vertex Cover

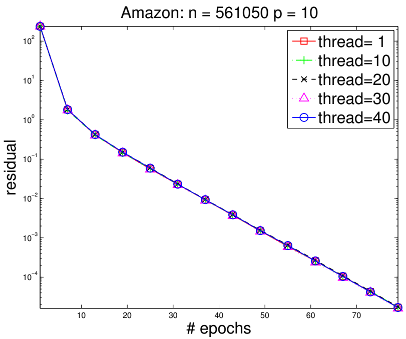
Given a graph with edge set E, vertex set V, seek a subset of vertices such that every edge touches the subset. Cost to select a vertex v is c_v.

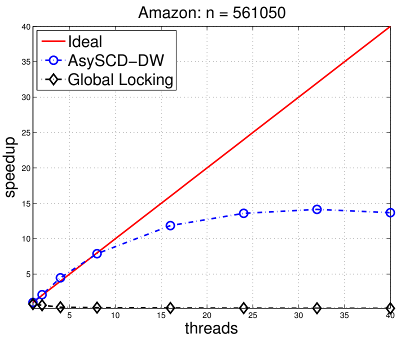
Binary programming formulation:

$$\min \sum_{v \in V} c_v x_v \;\text{ s.t. }\; x_u + x_v \geq 1 \;\text{ for } (u,v) \in E; \;\; x_v \in \{0,1\} \text{ for all } v \in V.$$

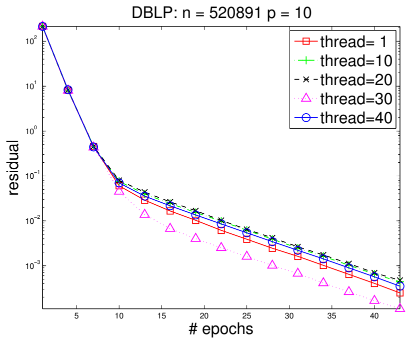
Relax the binary constraint to $x_v \in [0,1]$ to get an LP. Very large, but matrix A is highly sparse and structured.

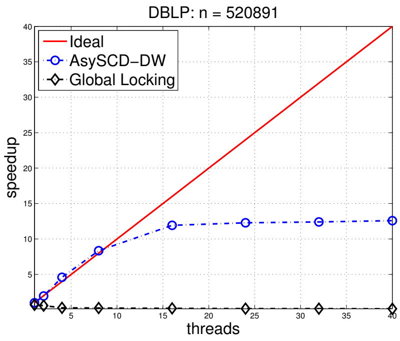
Vertex Cover (Amazon): 4-socket, 40-core Intel Xeon





Vertex Cover (DBLP): 4-socket, 40-core Intel Xeon





Running Time

Problem	1 core	40 cores
QP	98.4	3.03
QPc	59.7	1.82
Amazon	17.1	1.25
DBLP	11.5	.91

Table: Runtimes (s) for the four test problems on 1 and 40 cores.

Sample Results

instance	vertex cover		multiway cuts	
	n	size(A)	n	size(A)
frb59-26-1	126K	616K	1.3M	3.6M
Amazon	203K	956K	6.8M	21.3 M
DBLP	146K	770K	10.7M	33.7M
Google+	82K	1.5M	7.6M	24.1M

Table: Problem Sizes

Computation Times (Seconds)

Run on 32 cores Intel machine for max of one hour. Compared with Cplex IP and LP solvers. Times shown for reaching solutions of similar quality.

instance	Cplex IP	Cplex LP	Us
frb59-26-1 VC	-	5.1	0.65
Amazon VC	44	22	4.7
DBLP VC	23	21	3.2
Google+ VC	-	62	6.2
frb59-26-1 MC	54	360	29
Amazon MC	-	-	131
DBLP MC	-	-	158
Google+ MC	-	-	570

(Cplex IP sometimes faster than LP because the IP preprocessing can drastically simplify the problem, for some data sets.)

Conclusions

- Old methods are interesting again, because of modern computers and modern applications (particularly in machine learning).
- We can analyze asynchronous parallel algorithms, with a computing model that approximates reality pretty well. (There are other models.)

FIN