

# Asynchronous Parallel Methods for Optimization and Linear Algebra

Stephen Wright

University of Wisconsin-Madison

Workshop on Optimization for Modern Computation,  
Beijing, September 2014

**Ji Liu** (UW-Madison → U. Rochester)

Victor Bittorf (UW-Madison → Cloudera)

Chris Ré (UW-Madison → Stanford)

Krishna Sridhar (UW-Madison → GraphLab)

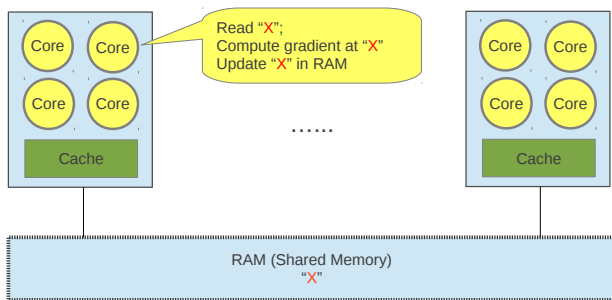
- 1 Asynchronous Random Kaczmarz
- 2 Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm with Inconsistent Read (ASYSPCD)

## Why study old, slow, simple algorithms?

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

# Asynchronous Parallel Optimization

Figure: *Asynchronous* parallel setup used in HOGWILD! [Niu, Recht, Ré, and Wright, 2011]



- All cores share the same memory, containing the variable  $x$ ;
- All cores run the same optimization algorithm independently;
- All cores update the coordinates of  $x$  concurrently *without* any software locking.

We use the same model of computation in this talk.

1 Asynchronous Random Kaczmarz

2 Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm with Inconsistent Read (ASYSPCD)

# 1. 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

$$A = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad \text{where } \|a_i\|_2 = 1, \quad \forall_i \quad (\text{normalized rows}).$$

Iteration  $j$  of Randomized Kaczmarz:

- Select row index  $i(j) \in \{1, 2, \dots, m\}$  randomly with equal probability.
- Set

$$x_{j+1} \leftarrow x_j - (a_{i(j)}^T x_j - b_{i(j)}) a_{i(j)}.$$

**Project  $x$  onto the plane of equation  $i(j)$ .**

# Relationship to Stochastic Gradient

Randomized Kaczmarz  $\equiv$  Stochastic Gradient applied 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  $x \rightarrow x^*$ . (The “variance” in the gradient estimate shrinks to zero.)



# Randomized Kaczmarz Convergence: Linear Rate

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

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

Taking expectations:

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

Strohmer and Vershynin (2009)

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

Each core runs a simple process, repeating indefinitely:

- Choose index  $i \in \{1, 2, \dots, m\}$  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  $\gamma$  (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 Randomized Kaczmarz in that each update is using **outdated information** and we **update just a single component of  $x$**  (in theory).

# 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  $t(j)$ :

$$\mathbf{x}_{j+1} = \mathbf{x}_j - \gamma \|a_{i(j)}\|_0 (a_{i(j)}^T \mathbf{x}_{k(j)} - b_{i(j)}) E_{t(j)} a_{i(j)},$$

where

- $k(j)$  is some iterate prior to  $j$  but **no more than  $\tau$  cycles old**:

$$j - k(j) \leq \tau;$$

- $E_t$  is the  $n \times n$  matrix of all zeros, except for 1 in the  $(t, t)$  location.

If all computational cores are roughly the same speed, we can think of the delay  $\tau$  as being **similar to the number of cores**.

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 two or more updates happen to the  $\text{supp}(a_{i(j)})$ -components of  $x$  while they are being read.)

When the vectors  $a_i$  are **sparse**, inconsistency is not too frequent.

**More on this later!**

# ASYRK Analysis: A Key Element

Key parameters:

- $\mu := \max_{i=1,2,\dots,m} \|a_i\|_0$  (maximum nonzero row count);
- $\alpha := \max_{i,t} \|a_i\|_0 \|AE_t a_i\| \leq \mu \|A\|$ ;
- $\lambda_{\max} = \max$  eigenvalue of  $A^T A$ .

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

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

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

But **don't want  $\gamma$  to be too tiny**, otherwise overall progress is too slow.

**Strike a balance!**

## Theorem

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

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

Then have

$$\rho^{-1}\mathbb{E}(\|Ax_j - b\|^2) \leq \mathbb{E}(\|Ax_{j+1} - b\|^2) \leq \rho\mathbb{E}(\|Ax_j - b\|^2)$$
$$\mathbb{E}(\|x_{j+1} - P(x_{j+1})\|^2) \leq \left(1 - \frac{\lambda_{\min, \text{nz}}\gamma}{m\mu}(2 - \gamma\psi)\right) \mathbb{E}(\|x_j - P(x_j)\|^2),$$

A particular choice of  $\rho$  leads to simplified results, in a reasonable regime.

## Corollary

Assume

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

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

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

In the regime  $2e\lambda_{\max}(\tau + 1) \leq m$  considered here the delay  $\tau$  doesn't really interfere with convergence rate. In this regime, **speedup is linear in the number of cores!**

- Rate is consistent with serial randomized Kaczmarz: extra factor of  $1/(\mu + 1)$  arises because we update just one component in  $x$ , not all the components in  $a_{i(j)}$ .
- For random matrices  $A$  with unit rows, we have  $\lambda_{\max} \approx (1 + O(m/n))$ , with high probability, so that  $\tau$  can be  $O(m)$  without compromising linear speedup.
- Conditions on  $\tau$  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.) See below....



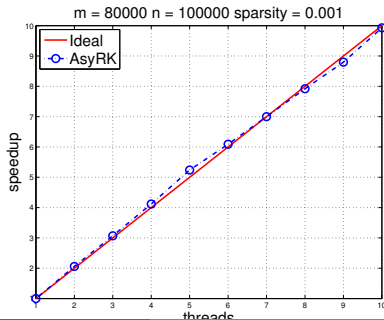
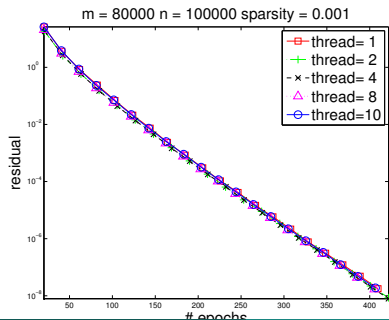
# ASYRK: Near-Linear Speedup

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

Diverges a bit from the analysis:

- We update *all* components of  $x$  for  $a_{i(j)}$  (not just one);
- We use sampling without replacement to work through the rows of  $A$ , reordering after each “epoch”

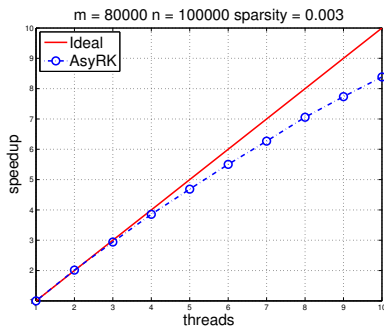
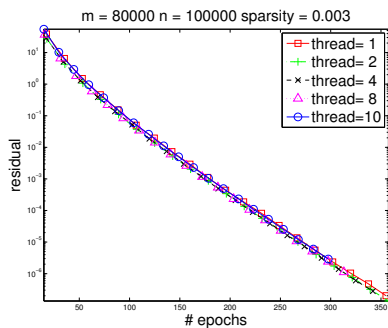
Sparse Gaussian random matrix  $A \in \mathbb{R}^{m \times n}$  with  $m = 100000$  and  $n = 80000$ , sparsity  $\delta = .001$ . See linear speedup.



# ASYRK: Near-Linear Speedup

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

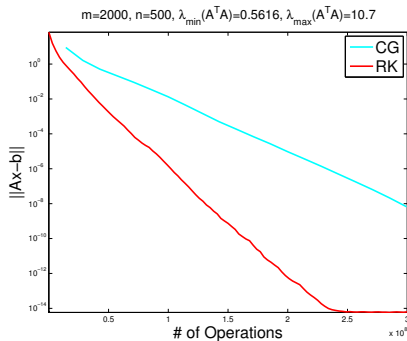
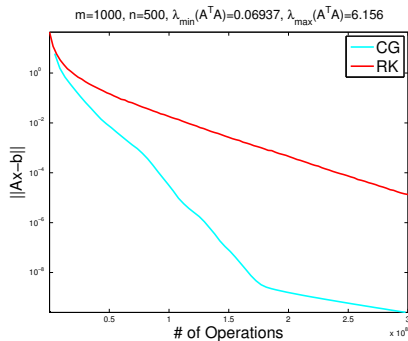
See slight dropoff from linear speedup for this slightly less-sparse problem.



(Runtime: 18.4 seconds on 10 cores.)

# 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  $A^T A$ . (Note slower convergence in later stages.) RK is competitive in the well-conditioned case.

1 Asynchronous Random Kaczmarz

2 Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm with Inconsistent Read (ASYSPCD)

## 2. Asynchronous Parallel Stochastic Proximal Coordinate Descent Algorithm (ASYSPCD)

$$\min_x : F(x) := f(x) + g(x) \quad (1)$$

- $f(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}$  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  $[a_i, b_i]$ ;
- $\ell_p$  norm regularization:  $g(x) = \|x\|_p^p$  where  $p \geq 1$ .

Problems that fit this framework include the following:

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

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

- support vector machine: dual form with bias term:

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

# Instances (continued)

- logistic regression with  $\ell_p$  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.

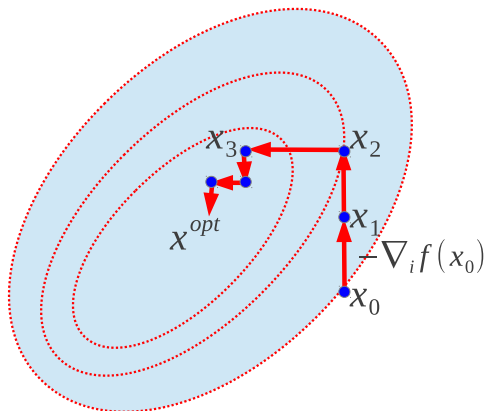
- relaxed linear program:

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





# Stochastic Coordinate Descent



- Take a step of fixed length along partial derivative (not exact)
- Choose components randomly (don't have control over the sequence).

# 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:  $\|\mathcal{P}_h(y) - \mathcal{P}_h(z)\| \leq \|y - z\|$ .)

**Basic Step:** Select a coordinate  $i$  and compute the coordinate gradient  $\nabla_i f(x)$ ; take a step along this direction and “shrink” to account for  $g_i$ .

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

for some step length  $\alpha$ .

This is equivalent to solving an approximate version of the coordinate- $i$  problem in which  $f$  is replaced by a simple quadratic:

$$\min_{(z)_i} \nabla_i f(x)^T [(z)_i - (x)_i] + \frac{1}{2\alpha} [(z)_i - (x)_i]^2 + g_i((z)_i).$$

# Prox-Operator Examples

Prox-operators can be executed efficiently in many cases.

- $g_i(t) = |t|$ : soft thresholding operation

$$\mathcal{P}_{\lambda g_i}(t) = \text{sgn}(t) \max\{|t| - \lambda, 0\}.$$

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

$$\mathcal{P}_{\lambda g_i}(t) = \arg \min_{s \in [a,b]} \frac{1}{2} \|s - t\|^2 = \text{mid}(a, b, t).$$

# Local View of ASySPCD

Step length depends on  $L_{\max}$ : component Lipschitz constant (“max diagonal of Hessian”)

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

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 step length  $\gamma > 0$ .

# Global View of ASYSPCD

Global counter  $j$  incremented when one of the cores makes an update:

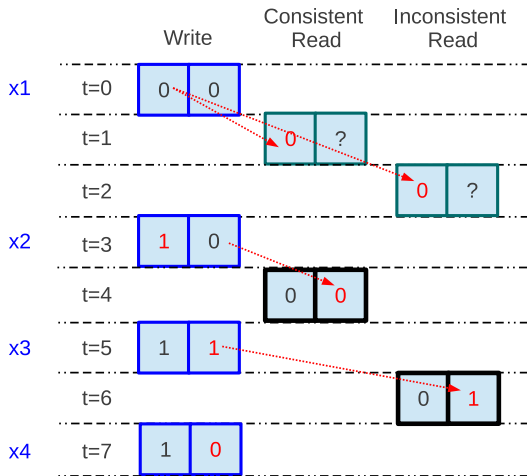
- Choose  $i(j) \in \{1, 2, \dots, n\}$  uniformly at random;
- Read components of  $x$  from shared memory needed to compute  $\nabla_{i(j)} f$ , denoting the local version of  $x$  by  $\hat{x}_j$ ;
- Update component  $i(j)$  of  $x$  (atomically):

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

Note that  $\hat{x}_j$  may not ever 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

# Expressing Read-Inconsistency

Difference between  $\hat{x}_j$  and  $x_j$  is expressed in terms of “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  $\hat{x}_j$ .

We assume  $\tau$  to be the upper bound of ages of all elements in  $K(j)$ :

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

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

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

$\tau$  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.

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

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

- $L_{\text{res}}$ : restricted Lipschitz constant (“max row-norm of Hessian”)

$$\|\nabla f(x + te_i) - \nabla f(x)\|_2 \leq L_{\text{res}}|t| \quad \forall x \in \mathbb{R}^n, t \in \mathbb{R}, i;$$

- $\Lambda := L_{\text{res}}/L_{\max}$  measures the degree of diagonal dominance.
  - 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  $\gamma$  so that

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

**Not too much change** in the step at each iteration

⇒ not too much change in the gradient

⇒ not too much price to pay for using outdated information.

Want to choose  $\gamma$  **small enough** to satisfy this property but **large enough** to get a better convergence rate.

**Strike a balance**, as in asynchronous randomized Kaczmarz.

# Main Assumption: Optimal Strong Convexity (OSC)

Optimal strong convexity parameter  $\mu > 0$

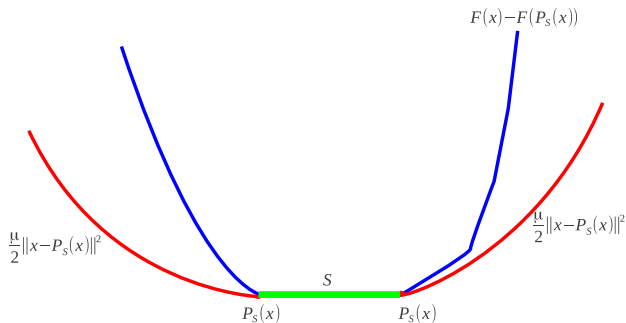
$$F(x) - F(\mathcal{P}_S(x)) \geq \frac{\mu}{2} \|x - \mathcal{P}_S(x)\|^2$$

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  $f$ .
- Squared hinge loss:  $F(x) = \sum_k \max(a_k^T x - b_k, 0)^2$ ;

An OSC (but not strongly convex) function:



# 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

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

Then the “ $\rho$ -condition” is satisfied at all  $j$ , and we have

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

Rate depends intuitively on the various quantities involved:

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

Dependence on  $\rho$  is a bit more complicated, but best to choose  $\rho$  near its lower bound.

## Corollary

Consider the regime in which  $\tau$  satisfies

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

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:

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

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  $\psi$  and  $\gamma$  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  $\rho$  and  $\gamma = 1/2$  as above, we have

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

# 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.”

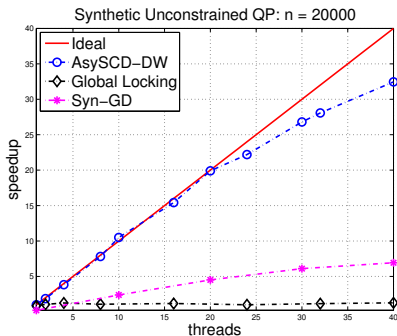
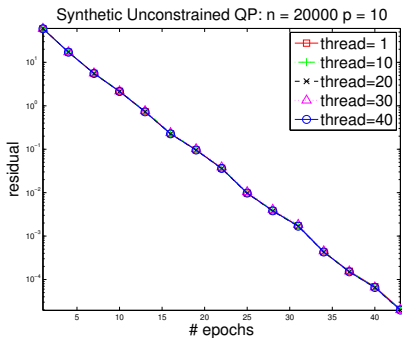
The order of indices is shuffled periodically - either between every pass, or less frequently.



# Unconstrained: 4-socket, 40-core Intel Xeon

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

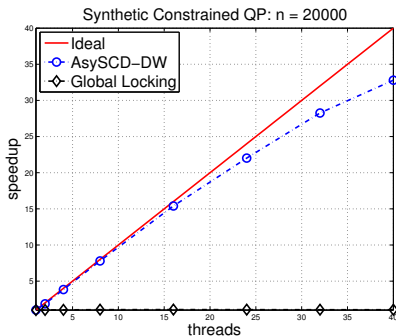
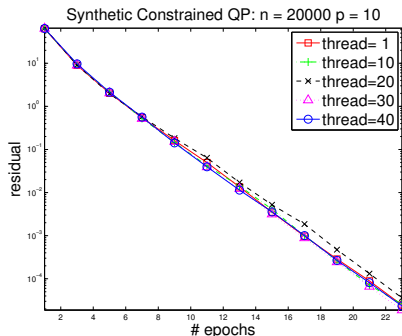
where  $A \in \mathbb{R}^{m \times n}$  is a Gaussian random matrix ( $m = 6000$ ,  $n = 20000$ , data size  $\approx 3$ GB, columns are normalized to 1).  $\Lambda \approx 2.2$ . Choose  $\gamma = 1$ . 3-4 seconds to achieve the accuracy  $10^{-5}$  on 40 cores.



# Constrained: 4-socket, 40-core Intel Xeon

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

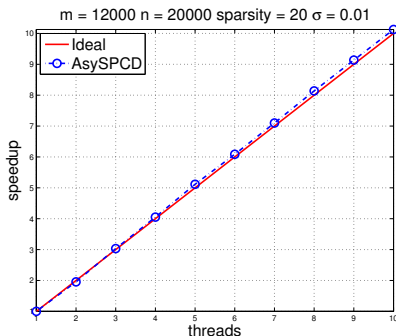
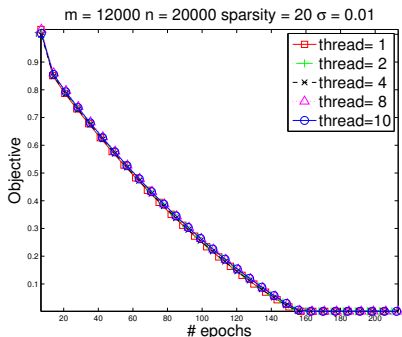
where  $A \in \mathbb{R}^{m \times n}$  is a Gaussian random matrix ( $m = 6000$ ,  $n = 20000$ , columns are normalized to 1) and  $z$  is a Gaussian random vector.  
 $L_{\text{res}}/L_{\text{max}} \approx 2.2$ . Choose  $\gamma = 1$ .



# Experiments: 1-socket, 10-core Intel Xeon

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

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



- 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.