A New Sequential Algorithm: Sparse Proximal SAGA

The algorithm relies on the following quantities:
- Extended support $T_i$: set of blocks that intersect with $\nabla f_i$.
- $D_{ij} = \{B : \supp(\nabla f_j) \cap B \neq \emptyset, B \in B\}$
- $D_i$: is a diagonal matrix defined block-wise as $[D_i]_{Bj} = \supp(\nabla f_B) I_j I_j$
- $\phi_i$: is a block-wise reweighting of $h$: $\phi_i = \sum_{Bj} d_i h_i(B(x))$

Justification:
- The following properties are verified:
  - $\phi_i(x)$ is zero outside $T_i$
  - $D_i x$ is zero outside $T_i$ (sparsity)
  - $E[\phi_i(h_i)] = h_i$ $E[D_i] = I$ (unbiasedness)

Algorithm:
- As SAGA, it maintains current iterate $x$ and table of historical gradients $\alpha_i \in \mathbb{R}^{np}$. At each iteration, it samples an index $i \in \{1, \ldots, n\}$ and computes next iterate ($x^\alpha$, $x^\gamma$) as:
  $$ x^\alpha = \nabla f_i(x) - \alpha_i + D_i \delta T $$
  $$ x^\gamma = \text{prox}_{\alpha_i} (x - \alpha_i)^\top : \alpha_i^\top = \nabla f_i(x) $$

Features:
- Per iteration cost in $\mathcal{O}(|T_i|)$.
- Easy to implement (compared to the averaged update approach [3]).
- Amenable to parallelization.

Convergence Analysis

For step size $\gamma = \frac{\tau}{\delta}$ and $f$ $\mu$-strongly convex ($\mu > 0$), Sparse Proximal SAGA converges geometrically in expectation. At iteration $t$ we have
  $$ E[\|x^\gamma - x^\alpha\|^2] \leq (1 - \frac{1}{2}\min(\frac{\tau}{\delta}, 1))^t C_0 $$
  $$ C_0 = \|x_0 - x^\gamma\|^2 + \frac{1}{2}\sum_{i=1}^n \|\alpha_i - \nabla f_i(x)^\gamma\|^2 + \kappa = \frac{1}{\delta} $$

(Condition number).

Implications
- Same convergence rate than SAGA with cheaper updates.
- In the “big data regime” ($n \gg \kappa$): rate in $O(1/n)$.
- In the “ill-conditioned regime” ($n \ll \kappa$): rate in $O(1/\kappa)$.

Adaptivity to strong convexity, i.e., no need to know strong convexity parameter to obtain linear convergence.

A New Parallel Algorithm: Proximal Asynchronous SAGA (ProxASAGA)

Proximal Asynchronous SAGA (ProxASAGA) runs Sparse Proximal SAGA asynchronously and without locks and updates $x$, $\alpha_i$ and $\tau_i$ in shared memory.

All read/write operations to shared memory are inconsistent, i.e., no vector-level locks while reading/writing.

Convergence guarantee of ProxASAGA

Suppose $\tau \leq \frac{\alpha_i}{\delta}$, then:
- If $\gamma > \frac{n}{\delta}$, then with step size $\gamma = \frac{1}{\delta/n}$, ProxASAGA converges geometrically with rate factor $\Omega(\frac{1}{\delta})$.
- If $\gamma < \frac{n}{\delta}$, then using the step size $\gamma = \frac{1}{\delta/n}$, ProxASAGA converges geometrically with rate factor $\Omega(\frac{1}{\delta})$.

In both cases, the convergence rate is the same as Sparse Proximal SAGA. ProxASAGA is linearly faster up to constant factor. In both cases, the step size does not depend on $T_i$.

If $\tau \geq 6\kappa$, a universal step size of $\theta(1/\alpha)$ achieves a similar rate than Sparse Proximal SAGA, making it adaptive to local strong convexity (knowledge of $\kappa$ not required).

Experimental Results

Comparison on 3 large-scale datasets on an elastic-net regularized logistic regression model:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>I</th>
<th>m</th>
<th>n</th>
<th>p</th>
<th>KDD 2010</th>
<th>KDD 2012</th>
<th>Criteo</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>10</td>
<td>8,494</td>
<td>1,000,000</td>
<td>10</td>
<td>28</td>
<td>12</td>
<td>0.15</td>
</tr>
<tr>
<td>m</td>
<td>149,639,105</td>
<td>54,686,452</td>
<td>4,000,000</td>
<td>4,000,000</td>
<td>4,000,000</td>
<td>4,000,000</td>
<td>4,000,000</td>
</tr>
<tr>
<td>n</td>
<td>19,264,097</td>
<td>1,163,024</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>p</td>
<td>28.12</td>
<td>0.15</td>
<td>0.15</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>k</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

Highlights: ProxASAGA significantly outperforms existing methods, significant speedup (6x to 12x) over the sequential version.

References