Parallel Methods for Convex Optimization

A. Devarakonda, J. Demmel, K. Fountoulakis, M. Mahoney
Problems

minimize $\lambda g(x) + f(x; A, b)$

- Sparse regression
  
  $g(x) = \|x\|_1$  
  $f(x) = \|Ax - b\|^2_2$

- Sparse SVM
  
  $g(x) = \|x\|_1$  
  $f(x) = \sum_{i=1}^{m} \max(1 - b_i A_i^T x, 0)^2$

- Elastic net
  
  $g(x) = \frac{\eta}{2} \|x\|^2_2 + (1 - \eta) \|x\|_1$

- Group lasso
  
  $g(x) = \sum_{j=1}^{J} \|x_j\| K_j$
Numerous applications

- Medical sciences
  - Cancer diagnosis
  - Prediction of common diseases: diabetes, pre-diabetes

- Biological sciences
  - Protein remote homology detection
  - Protein function prediction
  - Gene expression data classification

- Finance sciences
  - Financial time series forecasting
  - Stock market prediction

- Web related applications
  - Web searching
  - Email filtering
  - Relevance feedback
  - Classification of articles, emails, web-pages

- Geo and spatiotemporal environmental analysis.
Existing directions and our goals

- Parallelizing linear algebra is monopolized by HPC community
- Parallelizing optimization methods is monopolized by ML community

<table>
<thead>
<tr>
<th>ML community</th>
<th>HPC community</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐ Data parallel approach</td>
<td>☐ Parallelizes LA</td>
</tr>
<tr>
<td>☐ Uses spark</td>
<td>☐ Uses MPI (lingua franca of HPC)</td>
</tr>
</tbody>
</table>

**Our goal:** generalize HPC/LA techniques to convex optimization
First attempt to parallelism

In ML first-order methods is often the king
  - Optimal worst-case FLOPS
  - Excellent for low accuracy solutions

Let’s parallelize computation of the gradient of first-order methods. What could go wrong?

Communication becomes the bottleneck and the methods are slow.
Partition the data across nodes. Each node works independently on its data. The outputs are averaged.

Control FLOPS/communication trade-off by number and quality of partitions.

<table>
<thead>
<tr>
<th>Correlated</th>
<th>Uncorrelated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Large #partitions</strong></td>
<td>Many FLOPS</td>
</tr>
<tr>
<td></td>
<td>Much communication</td>
</tr>
<tr>
<td><strong>Small #partitions</strong></td>
<td>Excessive FLOPS</td>
</tr>
<tr>
<td></td>
<td>Low communication</td>
</tr>
</tbody>
</table>
Parallelism, the ML way for l1-regularization

\[
\text{minimize } \lambda \| x \|_1 + f(x; A, b)
\]

Partition the data and coordinates across nodes. Each node works independently on its data and coordinates. The gradients are averaged.

The optimal solution is often very sparse...

The algorithm turns from parallel to serial as the algorithm converges since many nodes become idle.
Parallelism, the HPC way

**Our objective:** solve the communication bottleneck of first-order methods using HPC techniques.

- Exchange FLOPS with communication, directly.
- We get optimal worst-case FLOPS + communication efficiency
- We get good scalability for all sparse data
- This talk + preliminary experiments
An example: coordinate descent

Pseudo-code

1 communication per iteration

- Sample a column of data
- Compute partial derivative
- Update solution
- Repeat
An example: communication avoiding coordinate descent

Pseudo-code

- Compute in parallel anticipated computations for the next “s” iterations
- Redundantly store the result in all processors
- Each processor independently computes the next “s” iterations
- Repeat

1 communication round per s iterations
Other examples

- Block coordinate descent
- Accelerated block coordinate descent
- Newton-type (first-order++) block coordinate descent
- Proximal versions of the above
Running time in Big-O

\[ \gamma \times \left( s \frac{H \mu^2 f m}{P} + H \mu^3 \right) + \alpha \times \frac{H \log P}{s} + \beta \times s H \mu^2 \log P \]

**No free lunch:** Exchange FLOPS and bandwidth for latency

\[ \alpha \quad \text{Time per message} \quad P \quad \text{Number of cores} \]

\[ \beta \quad \text{Time per word} \quad \mu \quad \text{Number of coordinates at each iteration} \]

\[ \gamma \quad \text{Time per FLOP} \quad H \quad \text{Number of iterations} \]

\[ f = \text{nnz}(A)/mn \]
Optimal parameter “s”

\[ s = \sqrt{\frac{\alpha P \log P}{\gamma \mu^2 fm + \beta \mu^2 P \log P}} \]

- \( \alpha \): Time per message
- \( P \): Number of cores
- \( \beta \): Time per word
- \( \mu \): Number of coordinates at each iteration
- \( \gamma \): Time per FLOP
- \( f = \text{nnz}(A)/mn \)

Let’s assume \( A \) is very sparse and \( \mu = 1 \) (single coordinate descent)

\[ s \approx \sqrt{\frac{\alpha}{\beta}} = o\left(\sqrt{\frac{10^{-6}}{10^{-10}}}\right) = 100 \]
Scalable results for all data layouts

* Best performance depends on dataset and algorithm
### Summary of (LIBSVM) datasets

<table>
<thead>
<tr>
<th>Name</th>
<th>#Features</th>
<th>#Data points</th>
<th>Density of non-zeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>3,231,961</td>
<td>2,396,130</td>
<td>0.0036%</td>
</tr>
<tr>
<td>epsilon</td>
<td>2,000</td>
<td>400,000</td>
<td>100%</td>
</tr>
<tr>
<td>news20</td>
<td>62,021</td>
<td>15,935</td>
<td>0.13%</td>
</tr>
<tr>
<td>covtype</td>
<td>54</td>
<td>581,012</td>
<td>22%</td>
</tr>
</tbody>
</table>

C++ using the Message Passing Interface (MPI). Intel MKL library for sparse and dense BLAS routines. All methods were tested on a Cray XC30.
Convergence of re-organized algorithms

**Convergence rate remains the same** in exact arithmetic
Empirically stable convergence: no divergence between methods
Scalability performance

The more processors the better
The gap between CA and non-CA increases w.r.t. #processors
Scalability performance

The more processors the better
The gap between CA and non-CA increases w.r.t. #processors
Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations
Speed up breakdown

Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations
Future directions

- Comparisons among CA and ML approaches
- Hybrid CA and ML,
  - Partition based on ML, use CA for local problems
  - Solve the local subproblems using GPUs + CA first order++
- Currently the ML worst-case theory is worse than sequential
  - Possible solution, find a way to quantify and incorporate correlation among partitions into theory.
Thank You!

Questions?