Communication Avoiding Optimization Algorithms

A. DEVARAKONDA, K. FOUNTOULAKIS, J. DEMMEL, M. MAHONEY
Motivation

Need for faster optimization/ML algorithms with less communication

Processor speed $\ll$ Communication speed
Gap is growing
Trade-offs and existing approaches

Current approach: choose an algorithm based on computation and communication trade-off
Trade-offs and existing approaches

What happens if there is no algorithm with the required trade-off?

We need to wait until a mathematician comes up with a solution.
Our approach

Take existing algorithms and make them communication avoiding

- Newton
- Some algorithm
- Coordinate Descent
Outline of the approach and results

Choose your favorite algorithm

Re-organize it to make it communication avoiding

Load balanced processors

Scalability to 1000+ of processors or more
minimize $\lambda g(x) + \frac{1}{2} \|Ax - b\|^2_2$

**Optimization/ML**

- Sparse regression $g(x) = \|x\|_1$
- Elastic net $g(x) = \frac{\eta}{2} \|x\|^2_2 + (1 - \eta) \|x\|_1$
- Group lasso $g(x) = \sum_{j=1}^{K_j} \|x_j\|_{K_j}$
- Sparse group lasso

For what problems?

**Linear Regression**

minimize $\|Ax - b\|^2_2$
An example: coordinate descent

Pseudo-code

- Sample a column of data
- Compute partial derivative
- Update solution
- Repeat

1 communication per iteration
An example: communication avoiding coordinate descent

- 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

Pseudo-code

1 communication round per s iterations
More details about the results

Decrease communication by a factor of \(s\)

No free lunch: increase message size and flops by a factor of \(s\)

Flops are distributed across processors

Logarithmic dependence of communication cost on number of processors
Scalable results for all data layouts

* Best performance depends on dataset and algorithm
Other examples

- Block coordinate descent
- Accelerated block coordinate descent
- Gradient descent
- Any proximal method
## Datasets

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

![Graph showing convergence of re-organized algorithms](image)
Scalability performance

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

Performance scaling: url

Performance scaling: epsilon
Scaleability performance

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

Performance scaling: news20

Performance scaling: covtype
Speed up breakdown

**Large communication speedup** until bandwidth takes a hit

**Computation is maintained** due to local cache-efficient (BLAS-3) computations

![Graph showing speedup with recurrence unrolling parameter (s)](image)

- **Speedup: url**
  - Total speedup
  - Communication speedup
  - Computation speedup
  - Speedup = 1
  - Best s = 64

- **Speedup: epsilon**
  - Total speedup
  - Communication speedup
  - Computation speedup
  - Speedup = 1
  - Best s = 64
Speed up breakdown

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

**Speedup: news20**

- Total speedup
- Communication speedup
- Computation speedup
- Best s = 16

**Speedup: covtype**

- Total speedup
- Communication speedup
- Computation speedup
- Best s = 32
Summary

- Generalize from linear algebra to optimization/ML
- Provably avoid communication
- Scalability to 10,000+ processors
- Applies to many algorithms
Thank You!

Questions?
