DWQP: A large scale box-quadratic programming solver.

Srikrishna Sridhar

Computer Sciences
University of Wisconsin-Madison
http://www.cs.wisc.edu/~srikris/

Joint work with
Victor Bittorf, Christopher Ré and Stephen J. Wright
One slide summary

- **Objective:** Building a large-scale solver for convex optimization problems.
  - How large is large scale?
  - Current state of commercial solvers.

- Starting point: Box constrained quadratic programming problems (BQP).
- Feasible region: Asynchronous optimization algorithms.
- Stochastic coordinate descent (SCD) is an ideal candidate for large scale BQPs.
- Review convergence rates (serial and parallel) versions of SCD.
- Constraints: Implementation on multi-core processors.
  - Non-uniform memory access (NUMA).
  - Processor affinity.
- Optimal solution: Our BQP solver can be 100x faster than commercial solvers. (Optimistic preliminary results).
One slide summary

- **Objective**: Building a large-scale solver for convex optimization problems.
  - How large is large scale?
  - Current state of commercial solvers.

- **Starting point**: Box constrained quadratic programming problems (BQP).
One slide summary

- **Objective**: Building a large-scale solver for convex optimization problems.
  - How large is large scale?
  - Current state of commercial solvers.

- **Starting point**: Box constrained quadratic programming problems (BQP).

- **Feasible region**: Asynchronous optimization algorithms.
  - Stochastic co-ordinate descent (SCD) is an ideal candidate for large scale BQPs.
  - Review convergence rates (serial and parallel) versions of SCD.
One slide summary

- **Objective**: Building a large-scale solver for convex optimization problems.
  - How large is large scale?
  - Current state of commercial solvers.

- **Starting point**: Box constrained quadratic programming problems (BQP).

- **Feasible region**: Asynchronous optimization algorithms.
  - Stochastic co-ordinate descent (SCD) is an ideal candidate for large scale BQPs.
  - Review convergence rates (serial and parallel) versions of SCD.

- **Constraints**: Implementation on multi-core processors.
  - Non uniform memory access (NUMA).
  - Processor affinity.

Optimal solution: Our BQP solver can be 100x faster than commercial solvers. (Optimistic preliminary results).
One slide summary

▶ **Objective**: Building a large-scale solver for convex optimization problems.
  
  ▶ How large is large scale?
  
  ▶ Current state of commercial solvers.

▶ **Starting point**: Box constrained quadratic programming problems (BQP).

▶ **Feasible region**: Asynchronous optimization algorithms.
  
  ▶ Stochastic co-ordinate descent (SCD) is an ideal candidate for large scale BQPs.
  
  ▶ Review convergence rates (serial and parallel) versions of SCD.

▶ **Constraints**: Implementation on multi-core processors.
  
  ▶ Non uniform memory access (NUMA).
  
  ▶ Processor affinity.

▶ **Optimal solution**: Our BQP solver can be 100x faster than commercial solvers. (Optimistic preliminary results).
Motivation

Solving *large* optimization problems:

- many variables
- defined by a large data set.
Motivation

Solving *large* optimization problems:
  - many variables
  - defined by a large data set.

Stochastic iterative methods that take steps by looking just a *small piece of the data* because full scans of data are too expensive.
Motivation

Solving *large* optimization problems:

- many variables
- defined by a large data set.

Stochastic iterative methods that take steps by looking just a small piece of the data because full scans of data are too expensive.

Asynchronous, parallel variants are appealing for modern multicore architectures and clusters.

- **You can do a lot on one box!**
Motivation

Solving large optimization problems:

▶ many variables
▶ defined by a large data set.

Stochastic iterative methods that take steps by looking just a small piece of the data because full scans of data are too expensive.

Asynchronous, parallel variants are appealing for modern multicore architectures and clusters.

▶ You can do a lot on one box!

Compelling applications: learning from data, low-accuracy LP.
What is large scale?

Table: Solve times for commercial solvers using 32 cores on a dense box-constrained quadratic programs.

<table>
<thead>
<tr>
<th>Sl.</th>
<th>Vars</th>
<th>Size</th>
<th>Solve time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cplex(B)</td>
</tr>
<tr>
<td>1</td>
<td>123</td>
<td>0.2MB</td>
<td>0.031</td>
</tr>
<tr>
<td>2</td>
<td>12596</td>
<td>1.2GB</td>
<td>5882.5s</td>
</tr>
<tr>
<td>3</td>
<td>129136</td>
<td>125 GB</td>
<td>-</td>
</tr>
</tbody>
</table>

▶ S : Simplex  B : Barrier
▶ Time limit: 7200 secs (2 hours)
What is large scale?

Table: Solve times for commercial solvers using 32 cores on a dense box-constrained quadratic programs.

<table>
<thead>
<tr>
<th>Sl.</th>
<th>Vars</th>
<th>Size</th>
<th>Solve time (secs)</th>
<th>Cplex(B)</th>
<th>Cplex(S)</th>
<th>Gurobi(B)</th>
<th>Gurobi(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>0.2MB</td>
<td></td>
<td>0.031</td>
<td>0.065</td>
<td>0.081</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>12596</td>
<td>1.2GB</td>
<td></td>
<td>5882.5s</td>
<td>1690.79s</td>
<td>3002.1</td>
<td>2707.8</td>
</tr>
<tr>
<td>3</td>
<td>129136</td>
<td>125 GB</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

▶ S : Simplex  B : Barrier
▶ Time limit: 7200 secs (2 hours)
Box constrained quadratic programs

Applications and algorithms
Problem description

Box constrained quadratic program

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x \\
\text{s.t.} \quad l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}
\]

Some applications...

- Support vector machines.
Problem description

Box constrained quadratic program

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x \\
\text{s.t. } l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}
\]

Some applications...

- Support vector machines.
- Least squares regression.
Problem description

Box constrained quadratic program

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x \\
\text{s.t. } l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}
\]

Some applications...

- Support vector machines.
- Least squares regression.
- Subproblems in constrained optimization.
Problem description

Box constrained quadratic program

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x$$

s.t. \quad l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}

Some applications...

- Support vector machines.
- Least squares regression.
- Subproblems in constrained optimization.
- and many more...
Problem description

Box constrained quadratic program

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x$$

s.t. \( l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\} \)

Optimize for certain structures

- Q matrix is dense/sparse. (e.g. support vector machines)
Problem description

Box constrained quadratic program

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x$$

s.t. \quad l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}

Optimize for certain structures

- Q matrix is dense/sparse. (e.g support vector machines)
- Q matrix is of the form $A^T A$. (e.g least squares, linear SVMs)
Full gradient based methods

Gradient descent

while not converged do
    for $i \in \{1, 2, \ldots, n\}$ do
        Compute $\nabla f = Qx + p$;
        $x \leftarrow \max(x - \alpha \nabla f, l)$;
        $x \leftarrow \min(x, u)$;
    end for
end while

Full gradients are expensive!
Stochastic coordinate descent (SCD)

SCD is ideal for large scale!

- Computing partial gradients are cheap.

Serial SCD

- Step 1: Compute the gradient $\nabla f_i$ along a single coordinate $i$.
- Step 2: Take a step along a single coordinate.
- Step 3: Projection to the feasible set of that coordinate $[l_i, u_i]$. 
Stochastic coordinate descent (SCD)

**SCD algorithm**

```plaintext
while not converged do
  for $i \in \{1, 2, \ldots, n\}$ do
    Compute $\nabla f_i = Q_i x + p_i$;
    $x_i \leftarrow \max(x_i - \nabla f_i / Q_{ii}, l_i)$;
    $x_i \leftarrow \min(x_i, u_i)$;
  end for
end while
```

- Nesterov (2012) showed that with high probability convergence of $f(\cdot)$ to within a specified threshold $\epsilon$ of $f(x^*)$ in about $O(1/k)$ iterations.
- **Linear** convergence, in expectation, when $f(\cdot)$ is strongly convex.
Parallel Stochastic co-ordinate descent (PSCD)

Serial SCD

while not converged do
    for $i \in \{1, 2, \ldots, n\}$ do
        Compute $\nabla f_i = Q_i \cdot x + p_i$;
        $x_i \leftarrow \max(x_i - \nabla f_i/Q_{ii}, l_i)$;
        $x_i \leftarrow \min(x_i, u_i)$;
    end for
end while

Parallel SCD

while not converged do
    for $i \in \{1, 2, \ldots, n\}$ in parallel do
        Read the current state of $x$;
        Compute $\nabla f_i = Q_i \cdot x + p_i$;
        $x_i \leftarrow \max(x_i - \nabla f_i/Q_{ii}, l_i)$;
        $x_i \leftarrow \min(x_i, u_i)$;
    end for
end while
Parallel Stochastic co-ordinate descent (PSCD)

**Asynchronous: **Hogwild! style

- Each core grabs the centrally-stored $x$ and evaluates $\nabla f_i$ and then writes the updates back into $x$. (Niu, Ré, Recht, Wright, NIPS, 2011).
Parallel Stochastic co-ordinate descent (PSCD)

Asynchronous: **Hogwild! style**

- Each core grabs the centrally-stored $x$ and evaluates $\nabla f_i$ and then writes the updates back into $x$. (Niu, Ré, Recht, Wright, NIPS, 2011).
- Updates can be old by the time they are applied.
Parallel Stochastic co-ordinate descent (PSCD)

Asynchronous: **Hogwild! style**

- Each core grabs the centrally-stored $\mathbf{x}$ and evaluates $\nabla f_i$ and then writes the updates back into $\mathbf{x}$. (Niu, Ré, Recht, Wright, NIPS, 2011).
- Updates can be old by the time they are applied.
- Processors don’t overwrite each other’s work!
Parallel Stochastic co-ordinate descent (PSCD)

Asynchronous: *Hogwild!* style

- Each core grabs the centrally-stored $x$ and evaluates $\nabla f_i$ and then writes the updates back into $x$. (Niu, Ré, Recht, Wright, NIPS, 2011).
- Updates can be old by the time they are applied.
- Processors don’t overwrite each other’s work!
- Asynchronous SCD analyzed by Richtarik and Takac (2012)
Implementation issues
Problem description

Box constrained quadratic program

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T Q x + p^T x
\]

s.t. \( l_i \leq x_i \leq u_i \) \( \forall i \in \{1, 2 \ldots n\} \)

Optimize for certain structures

- Q matrix is dense/sparse. (e.g support vector machines)
Problem description

Box constrained quadratic program

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^\top Q x + p^\top x \\
\text{s.t. } l_i \leq x_i \leq u_i \quad \forall i \in \{1, 2 \ldots n\}
\]

Optimize for certain structures

- Q matrix is dense/sparse. (e.g support vector machines)
- Q matrix is of the form \(A^T A\). (e.g least squares, linear SVMs)
Dual SVM with linear kernel is a bound-constrained QP, with $Q = A^T A$. Each row of $A$ is the feature vector for a single item of data.

- **Eager**: $Q$ is precomputed.
- **Lazy**: Use $A$; don’t compute $Q$ explicitly.

In Lazy, the key operation at each SCD iteration is

$$Q_i \cdot x = A_i^T A x.$$ 

With sparse $A$, implement this by

- compute $A_j \cdot x$ for those $j$ for which $A_{ij} \neq 0$;

$$\sum_{j : A_{ij} \neq 0} A_{ij} (A_j \cdot x).$$
NUMA aware SCD: 4 socket, 40 cores
Each SCD step requires access to only a single column of $Q$.

Distribute columns of the $Q$ matrix to separate cores.

Each core accesses a pre-determined set of columns of $Q$. 
Full-blown Eager

- Cores (40) update components of $x$ asynchronously, in parallel.
Full-blown Eager

- Cores (40) update components of $x$ asynchronously, in parallel.

- Reshuffling between epochs: The slice allocated to each of the 4 sockets is not changed, but the ordering and assignment to core within each slice is shuffled.
Full-blown Eager

- Cores (40) update components of $x$ asynchronously, in parallel.

- Reshuffling between epochs: The slice allocated to each of the 4 sockets is not changed, but the ordering and assignment to core within each slice is shuffled.

- To do a coordinate descent step, a core must read the latest $x$. Most components are already in its cache — it needs to fetch only those components recently changed.
Full-blown Eager

- Cores (40) update components of $x$ asynchronously, in parallel.

- Reshuffling between epochs: The slice allocated to each of the 4 sockets is not changed, but the ordering and assignment to core within each slice is shuffled.

- To do a coordinate descent step, a core must read the latest $x$. Most components are already in its cache — it needs to fetch only those components recently changed.

- When a core writes to $x_i$, the hardware ensures that this $x_i$ is simultaneously removed from the cache of other cores.
Computational Experiments

Do 20 epochs of SCD on the problem with 1.2 GB of data ($n = 12596$).

- **Lazy**: Store $A$ (on every socket), not $Q$.
- **Eager**: Algorithm described above, with $Q$ precomputed (which takes approximately 6 seconds)
- **Eager**: Spin-Naive: Lock $x$ while reading and writing.
- **Eager**: NUMA-naive: Cores select index $i$ to update without regard to where $Q_i$ is stored — possibly need to fetch it from another socket.
- **Parallel Sum**: Speed limit: simply sum the elements of $Q$, 20 times. SCD “Eager” cannot be faster than this.
Runtimes vs Threads for 20 Epochs of SCD

- lazy
- eager
- eager parallel sum
- eager numa-naïve
- eager spin-naïve

Runtime (s) vs # threads
Preliminary results: Comparison with commercial solvers

Table: Solve time on 32 cores for dense box-constrained quadratic programs.

<table>
<thead>
<tr>
<th>Sl.</th>
<th>Vars</th>
<th>Non-zeros</th>
<th>Size</th>
<th>Solve time (secs)</th>
<th>Cplex(B)</th>
<th>Cplex(S)</th>
<th>Gurobi(B)</th>
<th>Gurobi(S)</th>
<th>PSCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>15.12K</td>
<td>0.2MB</td>
<td></td>
<td>0.031</td>
<td>0.065</td>
<td>0.081</td>
<td>0.03</td>
<td>0.05 (10e-5)</td>
</tr>
<tr>
<td>2</td>
<td>12596</td>
<td>158.6M</td>
<td>1.18GB</td>
<td>5882.5s</td>
<td>1690.79s</td>
<td>3002.1</td>
<td>2707.8</td>
<td>6.9 (10e-5)</td>
<td>686.934 (10e-2)</td>
</tr>
<tr>
<td>3</td>
<td>129136</td>
<td>16.6B</td>
<td>124.8GB</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- S : Simplex  B : Barrier
- Time limit: 7200 secs (2 hours)
Snow storm arriving. Time to go home.

```
<table>
<thead>
<tr>
<th># of threads</th>
<th>Speedups vs Threads for 20 Epochs of SCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>lazy</td>
</tr>
<tr>
<td>5</td>
<td>eager</td>
</tr>
<tr>
<td>10</td>
<td>eager numa-naïve</td>
</tr>
<tr>
<td>15</td>
<td>eager spin-naïve</td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>
```

Srikrishna Sridhar (UW-Madison)