Lecture 13: Parallel Optimization Algorithms

Prof. Krishna R. Pattipati
Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu (860) 486-2890

ECE 6437
Computational Methods for Optimization
Fall 2009
December 1, 2009
Outline of Lecture 13

- Key Ideas of Parallel Algorithms
- Jacobi Algorithm
- Parallel implementation of Jacobi method and its variants
- Convergence Analysis
- Non-linear Jacobi and Gauss-Seidel Algorithms
- Constrained Optimization
- Decomposition Methods
Almost all of the parallel optimization algorithms are based on one of the two ideas:

1. Jacobi and Gauss-Seidel type relaxation schemes
2. Dual or decomposition methods…. mainly for convex constrained optimization methods.

Let us illustrate these ideas by means of a series of examples:

Consider \( \min_{x \in \mathbb{R}^n} x^T Q x - b^T x \)

\[ \Rightarrow \text{Solve } Q x = b \]

\( i^{th} \) equation is: \( \sum_{j=1}^{n} q_{ij} x_j = b_i \); \( i = 1, 2, \ldots, n \)

If \( q_{ii} \neq 0 \) (which it will be if \( Q > 0 \))

\[ q_{ii} x_i = b_i - \sum_{j \neq i} q_{ij} x_j \]

\[ x_i = \frac{-1}{q_{ii}} \left[ \sum_{j \neq i} q_{ij} x_j - b_i \right] \]
Jacobi Algorithm

Starting with an initial vector \( \underline{x}(0) \in R^n \), evaluate \( \underline{x}(k), k = 0, 1, \ldots \)
using the iteration:

\[
x_i(k + 1) = \frac{-1}{q_{ii}} \left[ \sum_{j \neq i} q_{ij} x_j(k) - b_i \right]
\]

It is well known that convergence is enhanced if \( i^{th} \) component is solved by an equation

\[
| q_{ii} | > | q_{ij} | \quad \forall j \neq i
\]
Gauss-Seidel Algorithm: On a serial computer, we can enhance convergence if we use new estimates of $x_j, j<i$ when updating $x_i$

$$x_i(k+1) = \frac{-1}{q_{ii}} \left[ \sum_{j<i} q_{ij} x_j(k+1) + \sum_{j>i} q_{ij} x_j(k) - b_i \right]$$
Richardson-Gauss-Seidel Algorithm:

Solution of $Qx = b \Rightarrow$ solve $x \leftarrow x - \alpha [Ax - b]$

$$x_i(k + 1) = x_i(k) - \alpha \left[ \sum_{j<i} q_{ij} x_j(k + 1) + \sum_{j \geq i} q_{ij} x_j(k) - b_i \right]$$

Note the similarity to steepest descent with a constant step-size $\alpha$

Parallel implementation of methods:

- Jacobi and Jacobi version of Richardson's algorithms are straightforward to implement in parallel. 3 two ways:
  - If we have $n$ processors, processor $i$ knows row $i$ of matrix $Q$. Each processor broadcasts $x_i(k)$ to all other $(n-1)$ processors. In a hypercube, this can be done in $O(n/\log_2 n)$ time, since $\exists \log_2 n$ distinct paths between any two processors.
  - If we have $n$ processors, processor $i$ knows column $i$ of $Q$. Each processor computes $q_{ji} x_i \forall j = 1, 2, ..., n$. Then for each $j$, the quantities $q_{ji} x_i$ are sent to processor $j$ with partial sums accumulated along the way.
Parallel Implementation Methods

If number of available processors $p < n$, assign $\left\lfloor \frac{n}{p} \right\rfloor$ components to each processor.

- Gauss-Siedel is, unfortunately, not well-suited for parallel implementation. When matrix $A$ is sparse, it is sometimes possible to exploit the non-dependencies.

 Updating Order

- $x_1(k+1) = f_1(x_1(k), x_3(k))$
- $x_3(k+1) = f_3(x_2(k), x_3(k), x_4(k))$
- $x_4(k+1) = f_4(x_2(k), x_4(k))$
- $x_2(k+1) = f_2(x_1(k+1), x_2(k))$

If we have two processors, then it can be done in two steps.
Parallel Algorithms for unconstrained minimization

Jacobi:

\[ x_i(k+1) = x_i(k) - \alpha h_i(x_k) \frac{\partial f(x_k)}{\partial x_i} \]

\[ h_i(x(k)) = \left[ \frac{\partial^2 f(x_k)}{\partial x_i^2} \right]^{-1} \]

\[ h_i = 1 \Rightarrow \text{Gradient or Richardson's Algorithm} \]

Gauss-Seidel:

\[ x_i(k+1) = x_i(k) - \frac{\alpha}{\partial^2 f[z_i(k)]} \frac{\partial f}{\partial x_i} [z_i(k)] \frac{\partial^2 f}{\partial x_i^2} \]

where \[ z_i(k) = [x_1(k+1), \ldots, x_{i-1}(k+1), x_i(k), \ldots, x_n(k)] \]

These are Coordinate descent Algorithms.
Convergence Analysis:

Assumptions: $f(x)$ is bounded below $\Rightarrow$ existence of a minimum
$\nabla f(x)$ is Lipschitz continuous $\Rightarrow f(x)$ is continously differentiable with bounded derivatives

$$\left\| \nabla f(x) - \nabla f(y) \right\| \leq K \left\| x - y \right\|_2, \ \forall x, y \in \mathbb{R}^n$$

Key to proving convergence is the following DESCENT LEMMA

Under the above assumption

$$f(x + y) \leq f(x) + y^T \nabla f(x) + \frac{K}{2} y^T y \ \forall x, y \in \mathbb{R}^n$$

Proof: Let $g(\alpha) = f(x + \alpha y)$, $\frac{dg(\alpha)}{d\alpha} = \nabla f^T (x + \alpha y) y$

$$f(x + y) - f(x) = g(1) - g(0) = \int_0^1 \frac{dg}{d\alpha} d\alpha$$

$$= \int_0^1 y^T \nabla f(x + \alpha y) d\alpha \leq \int_0^1 y^T \nabla f(x) d\alpha + \left| \int_0^1 y^T [\nabla f(x + \alpha y) - \nabla f(x)] d\alpha \right|$$

$$\leq \int_0^1 y^T \nabla f(x) d\alpha + \int_0^1 \|y\|_2 K \alpha \|y\|_2 d\alpha = y^T \nabla f(x) + \frac{K}{2} \|y\|_2^2$$
Convergence Theorem:

Suppose \( x(k + 1) = x(k) + \alpha d(k) \)

where \( \|d(k)\|_2 \geq K_1 \left\| \nabla f(x(k)) \right\|_2 \quad \forall k \)

\( d^T(k) \nabla f(x_k) \leq -K_2 \|d(k)\|^2 \)

Then, gradient and Jacobi algorithms converge if \( 0 < \alpha < \frac{2K_2}{K} \)

Gradient: \( K_1 = 1 \) and \( K_2 = 1 \)

Jacobi: \( K_2 \leq \min_i \frac{\partial^2 f}{\partial x_i^2}, \quad \frac{1}{K_1} \geq \max_i \frac{\partial^2 f}{\partial x_i^2} \)

From Descent Lemma

\[
\begin{align*}
 f(x(k + 1)) & \leq f(x(k)) + \alpha d^T(k) \nabla f(x(k)) + \frac{K}{2} \alpha^2 d^T(k) d(k) \\
& \leq f(x(k)) - \alpha \left( K_2 - \frac{K \alpha}{2} \right) \|d(k)\|^2 \\
& \Rightarrow f(x(k + 1)) \leq f(x(k))
\end{align*}
\]
Convergence Theorem - 2

Since \( f(\bar{x}) \) is bounded from below, we obtain optimum (local) if \( 0 < \alpha < \frac{2K_2}{K} \)

Gradient: \( 0 < \alpha < \frac{2}{K} \)

\[ \frac{2 \min \partial^2 f}{\partial x_i^2} \]

Jacobi: \( 0 < \alpha < \frac{\partial^2 f}{\partial x_i^2} \)

For Gauss–Seidel, one obtains a stronger result. If \( 0 < \gamma_i \leq \frac{\partial^2 f}{\partial x_i^2} \leq \Gamma_i \) for all \( i \)

\( \bar{x} \in \mathbb{R}^n \), then convergence occurs if \( 0 < \alpha < \frac{2\gamma_i}{\Gamma_i} \) for all \( i \)

In the quadratic case, \( \frac{1}{2} x^T Q x - b^T x \)

\[ \frac{\partial^2 f}{\partial x_i^2} = q_{ii} = \gamma_i = \Gamma_i \Rightarrow \text{Convergence occurs if } 0 < \alpha < 2. \]

Jacobi requires \( \alpha \) small enough for convergence.
Nonlinear Jacobi and Gauss-Seidel

Non-linear Jacobi and Gauss-Seidel Algorithm

**Jacobi**: \( x_i(k+1) = \arg \min_{x_i} f(x_1(k), \ldots, x_{i-1}(k), x_i, x_{i+1}(k), \ldots, x_n(k)) \)

(or) solve \( \frac{\partial f(x_1(k), \ldots, x_{i-1}(k), x_i, x_{i+1}(k), \ldots, x_n(k))}{\partial x_i} = 0 \)

JACOBI

Gauss-Seidel
Gauss-Seidel:

\[ x_i(k + 1) = \arg \min_{x_i} f(x_1(k + 1), \ldots, x_{i-1}(k + 1), x_i, x_{i+1}(k), \ldots, x_n(k)) \]

Gauss–Seidel algorithms are guaranteed to converge for **convex** functions. Note that the function value is strictly decreasing at each iteration. The Jacobi algorithm requires more stringent restrictions (e.g., \( x_k - \gamma \nabla f(x_k) \) must be a contraction map \( \Rightarrow \| x_k - \alpha \nabla f(x_k) \| < 1 \))

for \( \alpha > 0 \) \( \Rightarrow \nabla^2 f(\bar{x}) \) must be **diagonally dominant**.

- **Constrained Optimization:**
  Recall the gradient projection algorithm

\[ \bar{x}(k + 1) = \left[ \bar{x}(k) - \alpha H_k \nabla f(x_k) \right]^+ \]

Projection on to the feasible set \( \Omega \)
Constrained Optimization (continued):

Unfortunately the gradient projection algorithm is not amenable to parallel implementation unless the constraints are of the “box type”, i.e., upper and lower bound constraints on each variable.
Recall that projection of \([x - \alpha \nabla f(x)]\) into \(\Omega\) is equivalent to
\[
\min_{y} \| y - x + \alpha \nabla f(x) \|^2_2 \quad \text{s.t.} \quad y \in \Omega
\]

(or) \[
\min_{y} (y - x)^T \nabla f(x) + \frac{1}{2\alpha} (y - x)^T (y - x) \quad \text{s.t.} \quad y \in \Omega
\]

If \(\Omega = \prod_{i=1}^{n} [x_{i_{\min}}, x_{i_{\max}}]\), then the problem is equivalent to
\[
\sum_{i=1}^{n} \min_{x_{i_{\min}} \leq x \leq x_{i_{\max}}} \frac{1}{2\alpha} (y_i - x_i)^2 + (y_i - x_i) \frac{\partial f}{\partial x_i} \quad \Rightarrow \text{Minimizations can be carried out independently.}
\]

- Convergence of Gauss-Seidel and Jacobi follow for small enough \(\alpha\)
- Their non-linear versions converge under conditions similar to unconstrained case \(x_i(k+1) = \arg \min_{x_i \in \Omega_i} f(x_1(k), x_2(k), ..., x_i, ..., x_n(k))\), etc.
Decomposition Methods: By far the best for constrained minimization. We will present ideas via a series of examples.

- **Example 1:**

  \[
  \text{min } \frac{1}{2} x^T x \quad \text{“Project the Origin on the Constraint Set” }
  \]

  s.t. \( Ax \leq c \quad A \) is an \( m \) by \( n \) matrix

  \[
  q(\mu) = \min_x \frac{1}{2} x^T x + \mu^T (Ax - c)
  \]

  Optimal \( x^* \) for a given \( \mu \):

  \[
  x + A^T \mu = 0 \Rightarrow x = -A^T \mu
  \]

  \[
  \therefore q(\mu) = -\frac{1}{2} \mu^T AA^T \mu - \mu^T c
  \]

  Dual: \[
  \max_{\mu \geq 0} \quad q(\mu) \quad \text{or} \quad \min_{\mu \geq 0} \frac{1}{2} \mu^T P \mu + \mu^T c, \quad P = AA^T
  \]

  Unlike the primal problem, the dual is parallelizable

  \[
  \Rightarrow \text{Unconstrained min } \tilde{\mu}_j = -\frac{1}{p_{jj}} \left[ c_j + \sum_{k \neq j} p_{jk} \mu_k \right] = \mu_j - \frac{1}{p_{jj}} \left[ c_j + \sum_{k=1}^{m} p_{jk} \mu_k \right]
  \]
The iteration is: $\mu_j = \max (0, \tilde{\mu}_j) = \max \left(0, \mu_j - \frac{1}{p_{jj}} \left[ c_j + \sum_{k=1}^{m} p_{jk} \mu_k \right] \right)$ (*)

The matrix $A$ has a sparse structure in practice and we must exploit it (even if $A$ is sparse, $P = AA^T$ need not be!!!). Define $\underline{y} = -A^T \underline{\mu} \Rightarrow P \underline{\mu} = -A \underline{y}$

So, $\sum_{k=1}^{m} p_{jk} \mu_k = \underline{p}_j^T \underline{\mu} = -\underline{a}_j^T \underline{y}$, $\underline{a}_j^T = j^{th}$ row of $A$ or $j^{th}$ column of $A^T = \underline{a}_j$

Also, $p_{jj} = (AA^T)_{jj} = \underline{a}_j^T \underline{a}_j$; So, iteration (*) can be rewritten as two iterations:

$$\mu_j = \max(0, \mu_j - \frac{1}{\underline{a}_j^T \underline{a}_j} (c_j - \underline{a}_j^T \underline{y})) = \mu_j - \min \left( \mu_j, \frac{1}{\underline{a}_j^T \underline{a}_j} (c_j - \underline{a}_j^T \underline{y}) \right)$$

A Gauss–Seidel algorithm is of the form: $\underline{\mu} = \underline{\mu} - \min \left( \mu_j, \frac{1}{\underline{a}_j^T \underline{a}_j} (c_j - \underline{a}_j^T \underline{y}) \right) \underline{e}_j$

Corresponding iteration for $\underline{y}$ is $\underline{y} = -A^T \underline{\mu} \Rightarrow \underline{y} = \underline{y} + \min \left( \mu_j, \frac{1}{\underline{a}_j^T \underline{a}_j} (c_j - \underline{a}_j^T \underline{y}) \right) \underline{a}_j$
Geometric Interpretation:

Half Space

\[ H_j = \{a_j^T x \leq c_j\} \]

\[
\hat{y} = \begin{cases} 
  y + a_j \mu_j & \text{if } \mu_j < \frac{c_j - a_j^T y}{a_j^T a_j} \\
  y + \frac{1}{a_j^T a_j} [c_j - a_j^T y] a_j & \text{otherwise}
\end{cases}
\]

If \( y \notin H_j \), set it to

\[
\hat{y} = y + \frac{1}{a_j^T a_j} [c_j - a_j^T y] a_j
\]
Extension to General QPP

For general quadratic programming problems, the procedure extends easily.

\[
\begin{align*}
\min & \frac{1}{2} x^T Q x - b^T x \\
\text{s.t.} & \quad A x \leq c \\
\iff & \quad \min_{\mu \geq 0} \frac{1}{2} \mu^T P \mu + r^T \mu \\
\text{where} & \quad P = A Q^{-1} A^T ; \quad r = c - A Q^{-1} b
\end{align*}
\]

Define \( y = -A^T \mu \) and \( P \mu = -A Q^{-1} y \Rightarrow p^T_j \mu = -w^T_j y ; \quad w^T_j = j^{th} \text{ row of } A Q^{-1} \)

The iterations are:

\[
\mu = \mu + \min \left( \mu_j, \frac{1}{w^T_j a_j} \left[ r_j - w^T_j y \right] \right) a_j
\]

\[
y = y + \min \left( \mu_j, \frac{1}{w^T_j a_j} \left[ r_j - w^T_j y \right] \right) a_j
\]

• **Example 2:** Finding a point in a set intersection by parallel projections

\[
\min \frac{1}{2} \sum_{i=1}^m \| x_i - x \|^2
\]

\[
\text{s.t. } x \in R^n \]

and \( x_i \in \Omega_i , \quad i = 1, 2, \ldots, m \)

This is a simple separable problem. Start with any \( x(k) \) and solve for
Set Intersection Problem

\[ x_i(k) = \arg \min_{x_i \in \Omega_i} \| x_i - x(k) \|^2 \]

Then, \( x(k + 1) = \frac{1}{m} \sum_{i=1}^{m} x_i(k) \)

Convergence can be extremely poor in some cases, especially when \( \Omega_1 \) and \( \Omega_2 \) are "nearly colinear"

- **Example 3:** Linear programming problems
  \[
  \text{min } c^T x \\
  \text{s.t. } Ax = b \\
  0 \leq x \leq d
  \]

  Using the method of multipliers: at step \( k \)

  \[
  \text{Primal: } x_{k+1} = \arg \min_{0 \leq x \leq d} \left[ c^T x + \frac{h_k}{2} (Ax - b)^T (Ax - b) + \lambda_k^T (Ax - b) \right] \\
  \text{Dual: } \lambda_{k+1} = \lambda_k + h_k (A\lambda_{k+1} - b)
  \]
− **Approach 1**: The primal is equivalent to:

\[
\min_{x} \frac{1}{2} x^T A^T A x + \frac{1}{h_k} \left[ c + A^T \left( \lambda_k - h_k b \right) \right]^T x; \quad P = A^T A
\]

This is a quadratic programming problem, similar to the one we studied earlier. All we need to do is to project the unconstrained minimum along each direction \( i \) onto \([0, d_i]\). The unconstrained minimum \( \tilde{x}_i \) is

\[
\tilde{x}_i = \frac{-1}{(a_i^T a_i) h_k} \left[ c_i + a_i^T \left( \lambda_k - h_k b + h_k \sum_{j \neq i} a_j x_j \right) \right]
\]

or

\[
x_i = \left\{ x_i - \frac{1}{(a_i^T a_i) h_i} \left[ c_i + a_i^T \lambda_k - h_k \left( b + y \right) \right] \right\} #
\]

where \( y = -Ax \)  #Projection onto \([0, d_i]\)
The update for $y$ is:

$$y : y + \left[ x_i - \left\{ x_i - \frac{1}{(a_i^T a_i) h_i} \left[ c_i + a_i^T \lambda_k - h_k (b + y) \right] \right\}^+ \right] a_i$$

**Approach 2:** Let $I(i) = \{ j | a_{ij} \neq 0 \}; i = 1, 2, \ldots, m$

$$z_{ij} = a_{ij} x_j \quad i = 1, 2, \ldots, m; \quad j = 1, 2, \ldots, n$$

(1)

$$\Rightarrow \sum_{j \in I(i)} z_{ij} = b_i \quad i = 1, 2, \ldots, m$$

Let $\lambda_{ij}$ be the set of Lagrange multipliers associated with (1).

The problem is:

$$\min c^T x = \sum_{j=1}^{n} c_j x_j$$

s.t. $a_{ij} x_j = z_{ij}$

and $\sum_{j \in I(i)} z_{ij} = b_i$
Form augmented Lagrangian
\[
\sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \sum_{j \in I(i)} \lambda_{ijk} \left( a_{ij} x_j - z_{ij} \right) + \frac{h_k}{2} \sum_{i=1}^{m} \sum_{j \in I(i)} \left( a_{ij} x_j - z_{ij} \right)^2
\]
subject to \[
\sum_{j \in I(i)} z_{ij} = b_i; \ i = 1, 2, \ldots, m; \quad 0 \leq x_i \leq d_i
\]
so,
\[
x_j = \arg \min_{0 \leq x_j \leq d_j} \left\{ c_j x_j + \sum_{i \in I(i)} \lambda_{ijk} a_{ij} x_j + \frac{h_k}{2} \left( a_{ij} x_j - z_{ij} \right)^2 \right\}
\]
and
\[
\left\{ z_{ij} \mid j \in I(i) \right\} = \arg \min_{z_{ij} : \sum_{j \in I(i)} z_{ij} = b_i} \left\{ - \sum_{j \in I(i)} \lambda_{ijk} z_{ij} + \frac{h_k}{2} \sum_{j \in I(i)} \left( a_{ij} x_j - z_{ij} \right)^2 \right\}
\]
The optimization w.r.t. \( z_{ij} \) is a quadratic programming problem with a single equality constraint. We can write optimal solution directly.
If $p_i$ is the Lagrange multiplier, it is given by:

$$p_{i,k+1} = \frac{1}{m_i} \sum_{j \in I(i)} \lambda_{ijk} + \frac{h_k}{m_i} \sum_{j \in I(i)} \left(a_{ij}x_j - b_i\right) ; i = 1, 2, \ldots, m$$

where $m_i = |I(i)|$ the number of non zero elements in row $i$ of $A$

$$z_{ijk+1} = a_{ij}x_{jk+1} + \frac{\lambda_{ijk} - p_{i,k+1}}{h_k}$$

Note that

$$\lambda_{ijk+1} = \lambda_{ijk} + h_k \left[a_{ij}x_{jk+1} - z_{ijk+1}\right]$$

$$= p_{i,k+1} \Rightarrow \text{Don't actually need } \lambda_{ij}, \text{ need only } p_i$$

so, $p_{i,k+1} = p_{i,k} + \frac{h_k}{m_i} \sum_{j \in I(i)} \left[a_{ij}x_{jk+1} - b_i\right]$ ;

so, $z_{ij,k+1} = a_{ij}x_{jk+1} + \frac{p_{i,k} - p_{i,k+1}}{h_k} = a_{ij}x_{jk+1} - \frac{1}{m_i} \sum_{j \in I(i)} \left[a_{ij}x_{jk+1} - b_i\right]$ 

$$= a_{ij}x_{jk+1} - \frac{1}{m_i} \left[a_i^T x_{k+1} - b_i\right]$$

Parallel LP - 4
\[ x_j(k+1) = \arg\min_{0 \leq x_j \leq d_j} \left\{ c_j + \sum_{i \in I(i)} p_{i,k} a_{ij} x_j + \frac{h_k}{2} \sum_{i \in I(i)} \left[ a_{ij} (x_j - x_{j,k}) + w_i \right]^2 \right\} \]

where \( w_i = \frac{1}{m_i} (a_i^T x_k - b_i) \) \( \Rightarrow \) one dimensional minimization

Since quadratic, can find unconstrained minimum and project onto \((0, d_j)\)

The solution is:

\[ x_j(k+1) = \left\{ x_{j,k} - \frac{1}{h_k \left( a_j^T a_j \right)} \left[ c_j + a_j^T b + h_k a_j^T w \right] \right\}^\# \]

The methods can be extremely slow. May need to use Diagonal scaling for faster convergence. The method extends easily to separable nonlinear functions

\[ f(x) = \sum_{i=1}^{n} f_i(x_i) \]
Summary

- Key Ideas of Parallel Algorithms
- Jacobi Algorithm
- Parallel implementation of methods
- Convergence Analysis
- Non-linear Jacobi and Gauss-Seidel Algorithms
- Constrained Optimization
- Decomposition Methods