Tutorial: Parallel, Distributed, and Decentralized Optimization Methods

Wotao Yin
(UCLA Math & Alibaba US Damo Academy)

East Coast Optimization Meeting 2021

Objective

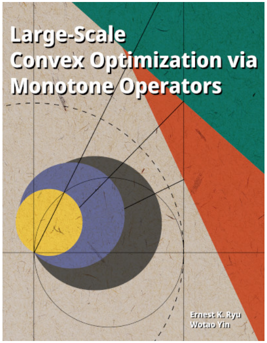
Large arbitrary-scale optimization methods

Explore structures, e.g., finite sums and block diagonals, and create parallel subproblems

Make communication efficient by going decentralized

Live demonstration of parallel, distributed, and decentralized optimization

Book: Large-Scale Convex Optimization via Monotone Operators



`large-scale-book.mathopt.com`

Ernest Ryu (Seoul Nat'l U) and **Wotao Yin**

Taught twice at SNU and UCLA

Audience: mathematicians and engineers

Goals:

- present convex optimization through the abstraction of monotone operators
- cover parallel, distributed, decentralized, stochastic, and block-coordinate methods
- include just the theories to develop and use methods correctly, not analysis focused

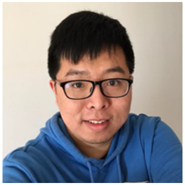
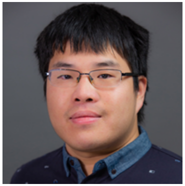
Open-source software: BlueFog

`github.com/Bluefog-Lib/bluefog`

Bicheng Ying (Google), **Kun Yuan** (Alibaba), **Hanbin Hu** (UCSB), **Ji Liu** (Baidu), and Wotao Yin

A software framework for decentralized optimization

- implements various decentralized topologies
- is capable of non-blocking asynchronous steps
- also supports parallel and (centralized) distribute computing
- can be used to run gradient, stochastic gradient, proximal, ... methods

Parallel computing

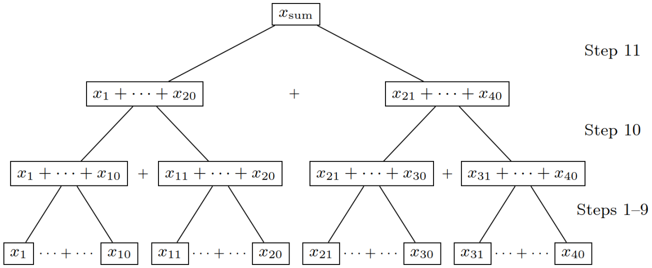
- **definition:** Calculations are carried out simultaneously by multiple computing nodes (CPU cores in one or multiple computers).
- Not all computational tasks can benefit from parallel computing. Loosely speaking, a method is *parallelizable* if it has a parallel implementation that provides a significant speedup using many agents.
- **Embarrassingly parallel** is great.
  Example: matrix sum $C = A + B$

```
parallel for i=1,...,m, j=1,...,n {
  C[i,j] = A[i,j]+B[i,j]
}
```

Parallel reduction

- Reduction combines a set of numbers into one number with an associative binary operator.
  Example: $x_{\text{sum}} = \sum_{i=1}^{n} x_i$
- With $p \geq \lfloor n/2 \rfloor$ agents, reduction take $\mathcal{O}(\log n)$ steps.
- With $p < \lfloor n/2 \rfloor$ agents, reduction take $\mathcal{O}(n/p + \log p)$ steps.

  Example: sum of 40 numbers on 4 agents


x_{sum}
Step 11
$x_1 + \cdots + x_{20}$
+
$x_{21} + \cdots + x_{40}$
Step 10
$x_1 + \cdots + x_{10}$
+
$x_{11} + \cdots + x_{20}$
$x_{21} + \cdots + x_{30}$
+
$x_{31} + \cdots + x_{40}$
Steps 1–9
x_1 $\cdots + \cdots$ x_{10}
x_{11} $\cdots + \cdots$ x_{20}
x_{21} $\cdots + \cdots$ x_{30}
x_{31} $\cdots + \cdots$ x_{40}

Discussion: cost of parallel computing

Adding parallel costs of single steps may be *inadequate* because:

- good performance requires proper data organization;
- different steps of a method may prefer different data organizations;
- also:

Other costs:

- Latency
- Data transmission time
- Coordination time (barrier, memory lock) — sometimes hard to quantify

Amdahl's law

Consider

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) + \frac{1}{m}\sum_{i=1}^{m} h_i(x),$$

and its *proximal gradient* method

$$v^k = -\frac{\alpha}{m}\sum_{i=1}^{m} \nabla h_i(x^k)$$
$$x^{k+1} = \mathbf{prox}_{\alpha f}\left(x^k + v^k\right).$$

Suppose it takes 6ms to evaluate v^k on one agent and 3ms to evaluate x^{k+1}.

Imagine we reduce 6ms to 0ms by parallel computing but cannot improve that 3ms. Then, the speedup is $\frac{6+3}{0+3} = 3$.

If the parallelizable part takes time $\eta \in [0,1]$ in proportion and we speed it up by s times, then the total speedup is

$$\frac{1}{1-\eta+\eta/s}.$$

SPMD (single program, multiple data) and Allreduce

SPMD (single program, multiple data)

One code for all agents; but different agents have different data and their unique ranks.

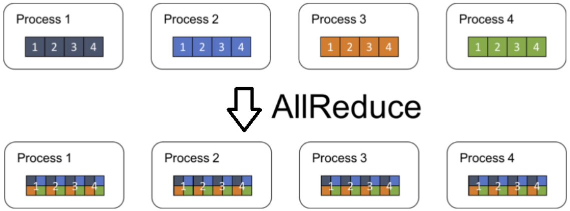
```
# hello_world.py
import bluefog.torch as bf

bf.init()
print("I'm rank {} of {}".format(bf.rank(), bf.size()))
```

```
> bfrun -np 4 python hello_world.py
I'm rank 1 of 4
I'm rank 0 of 4
I'm rank 3 of 4
I'm rank 2 of 4
```

You can use “`if bf.rank()==0:[tasks]`” to execute something only on one agent

Allreduce

- AllReduce is a reduce operation that *returns the result to all agents*



- Often used in SPMD implementations of iterative methods
- Different implementations:
  - dedicate a master agent (gather, local reduce, distribute), not scalable
  - butterfly allreduce: theoretically both latency and bandwidth optimal, but causing contentions
  - ring allreduce: bandwidth optimal, less contention, but not latency optimal

Allreduce demo

- BlueFog's `allreduce` (by default) is a ring-allreduce of averaging
- Demo: compute the average of the ranks of all agents

$$\frac{1}{n}\left(0+1+\cdots+(n-1)\right)=\frac{n-1}{2}$$

- Demo: solve

$$\underset{x\in\mathbb{R}^d}{\text{minimize}}\sum_{i=1}^{n}h_i(x)$$

by distributed gradient descent

$$x^{k+1}=x^k-\frac{\alpha}{n}\sum_{i=1}^{n}\nabla h_i(x^k).$$

Primal decomposition and dual decomposition

Primal decomposition

Consider

$$\underset{x_1,\dots,x_n,y}{\text{minimize}}\ r(y)+\sum_{i=1}^{n} h_i(x_i,y)$$

When y is fixed, we can deal with $x_1,\dots,x_n$ separately.

Define $f_i(y)=\min_{x_i} h_i(x_i,y)$ and obtain master problem:

$$\underset{y}{\text{minimize}}\ r(y)+\sum_{i=1}^{n} f_i(y).$$

Various methods update $x_1,\dots,x_n$ and y alternatively.

When $f_1,\dots,f_n$ are smooth, applying the proximal-gradient method on y:

$$\begin{aligned} x_i^\star(y^k) &\in \underset{x_i}{\arg\min}\, f_i(x_i,y^k) \\ (0,g_i^k) &\in \partial f_i(x_i^\star(y^k),y^k) \\ y^{k+1} &= \mathbf{prox}_{\alpha r}\left(y^k-\alpha\sum_{i=1}^{n} g_i^k\right). \end{aligned}$$

Example: resource sharing problem with total resource penalty

- Consider minimizing (total cost) + (sum of local costs)

$$\underset{x_1,\ldots,x_n}{\text{minimize}}\, r\left(\sum_{i=1}^n B_i x_i\right) + \sum_{i=1}^n h_i(x_i).$$

  Introduce $y_i = B_i x_i$ and write

$$\underset{y_1,\ldots,y_n}{\text{minimize}}\, f_0\left(\sum_{i=1}^n y_i\right) + \frac{1}{2}\sum_{i=1}^n \min_{x_i}\{h_i(x_i) : B_i x_i = y_i\}.$$

- Demo: use $r = \|\cdot\|_1$, $h_i(x_i) = \frac{1}{2}\|Ax_i - b_i\|^2$, and invertible B_i

Dual decomposition

Consider:

$$\begin{aligned} \underset{x_i \in \mathbb{R}^{d_i}, y \in \mathbb{R}^q}{\text{minimize}} \quad & r(y) + \frac{1}{n}\sum_{i=1}^{n} f_i(x_i) \\ \text{subject to} \quad & \sum_{i=1}^{n} B_i x_i \le y, \end{aligned}$$

This primal problem is generated by the Lagrangian

$$L(x_1, \ldots, x_n, y, u) = r(y) - \langle u, y \rangle + \frac{1}{n}\sum_{i=1}^{n} \left(f_i(x_i) + \langle u, B_i x_i \rangle \right) - \delta_{\mathbb{R}^n_+}(u).$$

With

$$\psi_i(u) = \sup_{x_i \in \mathbb{R}^{p_i}} \left(\langle -u, B_i x_i \rangle - f_i(x_i) \right)$$

we obtain the master dual problem

$$\underset{u \in \mathbb{R}^q}{\text{maximize}} \quad -r^*(u) - \delta_{\mathbb{R}^q_+}(u) - \frac{1}{n}\sum_{i=1}^{n} \psi_i(u).$$

Under certain strict convexity assumptions, primal solutions can be recovered from the dual problem.

Alternating Direction of Multipliers (ADM or ADMM)

Consider

$$\begin{aligned} \underset{x,y}{\text{minimize}} \quad & f(x) + g(y) \\ \text{subject to} \quad & \underbrace{Ax}_{=z} + By = b \end{aligned}$$

With *infimal postcomposition*

$$(A \triangleright f)(z) := \inf\{f(x) : Ax = z\}$$

we obtain the master problem

$$\underset{z}{\text{minimize}} \;\; (A \triangleright f)(z) + (B \triangleright g)(b - z).$$

Applying Douglas-Rachford Splitting, we get ADMM:

$$\begin{aligned} x^{k+1} &\in \arg\min_x L_\alpha(x, y^k, u^k) \\ y^{k+1} &\in \arg\min_y L_\alpha(x^{k+1}, y, u^k) \\ u^{k+1} &= u^k + \alpha(Ax^{k+1} + By^{k+1} - c). \end{aligned}$$

where $L_\alpha(x, y, u) = f(x) + g(y) + \langle u, Ax + By - c\rangle + \frac{\alpha}{2}\|Ax + By - c\|^2$.

Distributed ADMM

When $x = [x_1; \ldots; x_n]$, $f(x) = \sum_{i=1}^{n} f_i(x_i)$, and block diagonal A such that

$$Ax = \begin{bmatrix} A_1 x_1 \\ \vdots \\ A_n x_n \end{bmatrix},$$

ADMM generates separable subproblems subject to parallel computing

$$\begin{aligned}
x_i^{k+1} &\in \arg\min_{x_i} f(x_i) + \langle u^k, A_i x_i \rangle + \frac{\alpha}{2} \| A_i x_i + (By^k)_i - c_i \|^2, \quad i = 1, \ldots, n \\
y^{k+1} &\in \arg\min_{y} L_\alpha(x^{k+1}, y, u^k) \\
u_i^{k+1} &= u_i^k + \alpha(A_i x_i^{k+1} + (By^{k+1})_i - c_i), \quad i = 1, \ldots, n.
\end{aligned}$$

Steps 1 and 3 are embarrassingly parallel.

Summary of part 1

Finite-sum and block diagonal structures lend themselves to parallel computing

Infimal postcomposition can break coupling

Dual decomposition (Lagrange multipliers) can decouple linear constraints

Various software packages (e.g. PyTorch, BlueFog) have made distributed optimization easy, *not just for machine learning*.

Part 2: reduce communication costs by avoiding long-distance messages (i.e., going decentralized)