Parallel Computing

Ernest K. Ryu
Seoul National University

Mathematical and Numerical Optimization
Fall 2020

Last edited: 12/02/2020

Computational complexity and parallel computing

Briefly discuss computational complexity and parallel computing.

Useful for approximately analyzing run time of algorithms.

Outline

Floating-point operations

A floating-point operation (flop) is a single arithmetic operation on a computer such as addition, subtraction, multiplication, and division.

For simplicity, we also count a non-elementary function such as $\exp(x)$, $\log(x)$, or $\sqrt{x}$ as a single flop.

For example,

$$\|x\| = \sqrt{x_1^2 + \cdots + x_n^2}$$

for $x \in \mathbb{R}^n$ costs $2n = \mathcal{O}(n)$ flops to compute.
(n multiplications, $n-1$ additions, and 1 square root.)

Floating-point operations

- Ax costs $\mathcal{O}(mn)$ flops, where $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$.
- AB costs $\mathcal{O}(mnp)$ flops, where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$.
- For ABx, use $A(Bx)$, costing $\mathcal{O}(mn + np)$, instead of $(AB)x$, costing $\mathcal{O}(mnp)$, where $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, and $x \in \mathbb{R}^p$.
- A^{-1} costs $\mathcal{O}\left(n^3\right)$, where $A \in \mathbb{R}^{n \times n}$.

CPUs compute roughly 10^9 flops per second. Useful estimate in predicting run time and bottleneck of algorithms. But this is a very rough estimate; expect a 10-fold or even a 100-fold inaccuracy.

Algorithm vs. method

Algorithm and method both refer to a specification of how to compute a quantity of interest.

Difference:

- method is a higher-level description expressed in mathematical equations
- algorithm is a more literal step-by-step procedure unambiguously describing the steps the computer takes

Although this distinction is not precise, it is useful.

If an algorithm carries out the idea described by a method, we say the algorithm implements the method.

Algorithm vs. method

In a rigorous discussion, flop count ascribed to algorithm, not method.

For example, consider the method

$$x^{k+1} = x^k - \alpha A^\intercal (Ax^k - b)$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Algorithm corresponding to

$$A^\intercal (Ax^k - b)$$

costs $\mathcal{O}(mn)$ flops per iteration, but algorithm corresponding to

$$(A^\intercal A)x^k - A^\intercal b$$

costs $\mathcal{O}\left(n^2\right)$ flops per iteration, provided that $A^\intercal A \in \mathbb{R}^{n \times n}$ and $A^\intercal b \in \mathbb{R}^m$ have been precomputed and stored.

Often more than one way to implement a method. When implementation clear from the context, informally ascribe the flop count to the method.

Flop-count operator

Flop-count operator:

$$\mathcal{F}\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\} \,|\, \mathcal{A}\right]$$

number of flops $\mathcal{A}$ to compute $\{y_1, \ldots, y_m\}$ given $\{x_1, \ldots, x_n\}$.
(Algorithm $\mathcal{A}$, not a method, that determines the flop count.)

When algorithm is clear from context, omit $\mathcal{A}$ and write

$$\mathcal{F}\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\}\right].$$

For example, when $A \in \mathbb{R}^{m \times n}$

$$\begin{aligned}\mathcal{F}[A \mapsto (I + \alpha A^\mathsf{T} A)^{-1}] &= \mathcal{F}[A \mapsto I + \alpha A^\mathsf{T} A] + \mathcal{F}[I + \alpha A^\mathsf{T} A \mapsto (I + \alpha A^\mathsf{T} A)^{-1}] \\ &= \mathcal{O}\left(mn^2\right) + \mathcal{O}\left(n^3\right) = \mathcal{O}\left((m+n)n^2\right).\end{aligned}$$

Flop-count operator

As another example, consider

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|Ax-b\|^2+\lambda\|x\|_1,$$

where $A\in\mathbb{R}^{m\times n}$, $b\in\mathbb{R}^m$, and $\lambda>0$. DRS is

$$\begin{aligned} x^{k+1/2} &= (I+\alpha A^\mathsf{T}A)^{-1}(z^k+\alpha A^\mathsf{T}b) \\ x^{k+1} &= S(2x^{k+1/2}-z^k;\alpha\lambda) \\ z^{k+1} &= z^k+x^{k+1}-x^{k+1/2}, \end{aligned}$$

where S is soft-thresholding. A naive implementation costs

$$\begin{aligned} \mathcal{F}\left[z^k\mapsto z^{k+1}\right] &= \mathcal{F}\left[A\mapsto(I+\alpha A^\mathsf{T}A)^{-1}\right]+\mathcal{F}\left[\{z^k,(I+\alpha A^\mathsf{T}A)^{-1}\}\mapsto x^{k+1/2}\right] \\ &\quad +\mathcal{F}\left[\{x^{k+1/2},z^k\}\mapsto x^{k+1}\right]+\mathcal{F}\left[\{z^k,x^{k+1/2},x^{k+1}\}\mapsto z^{k+1}\right] \\ &= \mathcal{O}\left((m+n)n^2\right)+\mathcal{O}\left((m+n)n\right)+\mathcal{O}\left(n\right)+\mathcal{O}\left(n\right) \\ &= \mathcal{O}\left((m+n)n^2\right). \end{aligned}$$

Flop-count operator

Reduce this cost. When $m \geq n$, precompute $(I + \alpha A^\intercal A)^{-1}$ with cost

$$\mathcal{F}\left[A \mapsto (I + \alpha A^\intercal A)^{-1}\right] = \mathcal{O}\left(mn^2\right)$$

and $\alpha A^\intercal b$ with cost

$$\mathcal{F}\left[\{\alpha, A, b\} \mapsto \alpha A^\intercal b\right] = \mathcal{O}\left(mn\right).$$

In subsequent iterations,

$$\begin{aligned}
&\mathcal{F}\left[\{z^k, (I + \alpha A^\intercal A)^{-1}, \alpha A^\intercal b\} \mapsto z^{k+1}\right] \\
&\quad = \mathcal{F}\left[\{z^k, (I + \alpha A^\intercal A)^{-1}, \alpha A^\intercal b\} \mapsto x^{k+1/2}\right] + \mathcal{F}\left[\{x^{k+1/2}, z^k\} \mapsto x^{k+1}\right] \\
&\qquad + \mathcal{F}\left[\{z^k, x^{k+1/2}, x^{k+1}\} \mapsto z^{k+1}\right] \\
&\quad = \mathcal{O}\left(n^2\right) + \mathcal{O}\left(n\right) + \mathcal{O}\left(n\right) \\
&\quad = \mathcal{O}\left(n^2\right).
\end{aligned}$$

Outline

Simplified view of parallel computing

(Over)simplified view of parallel computing: group of computational agents working on a single task. Example: CPU cores, GPU cores, or computers connected over the internet.

If p processors, $A, B \in \mathbb{R}^{m \times n}$, and $p \leq mn$, then $C = A + B$ requires $\mathcal{O}(mn/p)$ flops for each processor:

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

Embarrassingly parallel

Task is embarrassingly parallel if trivial to parallelize.
(Embarrassingly parallel is good.)

For example, $v = Ax$ is embarrassingly parallel:

```
parallel for i=1,...,m {
  v[i] = 0;
  for j=1,...,n
    v[i] += A[i,j]*x[j]
}
```

Not everything is parallelizable

Some tasks can't be parallelized. Consider DRS:

$$\begin{aligned} x^{k+1/2} &= \mathrm{Prox}_{\alpha f}(z^k) \\ x^{k+1} &= \mathrm{Prox}_{\alpha g}(2x^{k+1/2} - z^k) \\ z^{k+1} &= z^k + x^{k+1} - x^{k+1/2}. \end{aligned}$$

$\mathrm{Prox}_{\alpha g}$ and $\mathrm{Prox}_{\alpha f}$ cannot be computed simultaneously, in parallel.

Computational bottleneck usually in $\mathrm{Prox}_{\alpha g}$ or $\mathrm{Prox}_{\alpha f}$. If $\mathrm{Prox}_{\alpha f}$ and $\mathrm{Prox}_{\alpha g}$ are not parallelizable, DRS is not parallelizable.

Parallel flop count operator

Let $\mathcal{A}$ be an algorithm that utilizes p parallel computing units.
($\mathcal{A}$ can process up to p flops in parallel each step.)

Parallel flop-count operator:

$$\mathcal{F}_p \left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\} \,|\, \mathcal{A} \right]$$

number of steps $\mathcal{A}$ takes to compute $\{y_1, \ldots, y_m\}$ given $\{x_1, \ldots, x_n\}$.

Parallelizable methods and operators

Method is parallelizable if many processors (large p) provide a significant speedup and is serial otherwise. ("Significant" depends on context.)

Parallelizability of $\{y_1, \ldots, y_m\}$ given $\{x_1, \ldots, x_n\}$ defined with $\mathcal{F}_p$:

$$\mathcal{F}_p\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\}\right] \ll \mathcal{F}\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\}\right]$$

for large enough p.

Meaning of $\ll$ depends on context, but if

$$\mathcal{F}_p\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\}\right] \sim \frac{C}{p}\mathcal{F}\left[\{x_1, \ldots, x_n\} \mapsto \{y_1, \ldots, y_m\}\right]$$

for some $C > 0$ not too large, then parallelizable.

Operator $\mathbb{T}$ is parallelizable if

$$\mathcal{F}_p\left[x \mapsto \mathbb{T}x\right] \ll \mathcal{F}\left[x \mapsto \mathbb{T}x\right].$$

Reduction

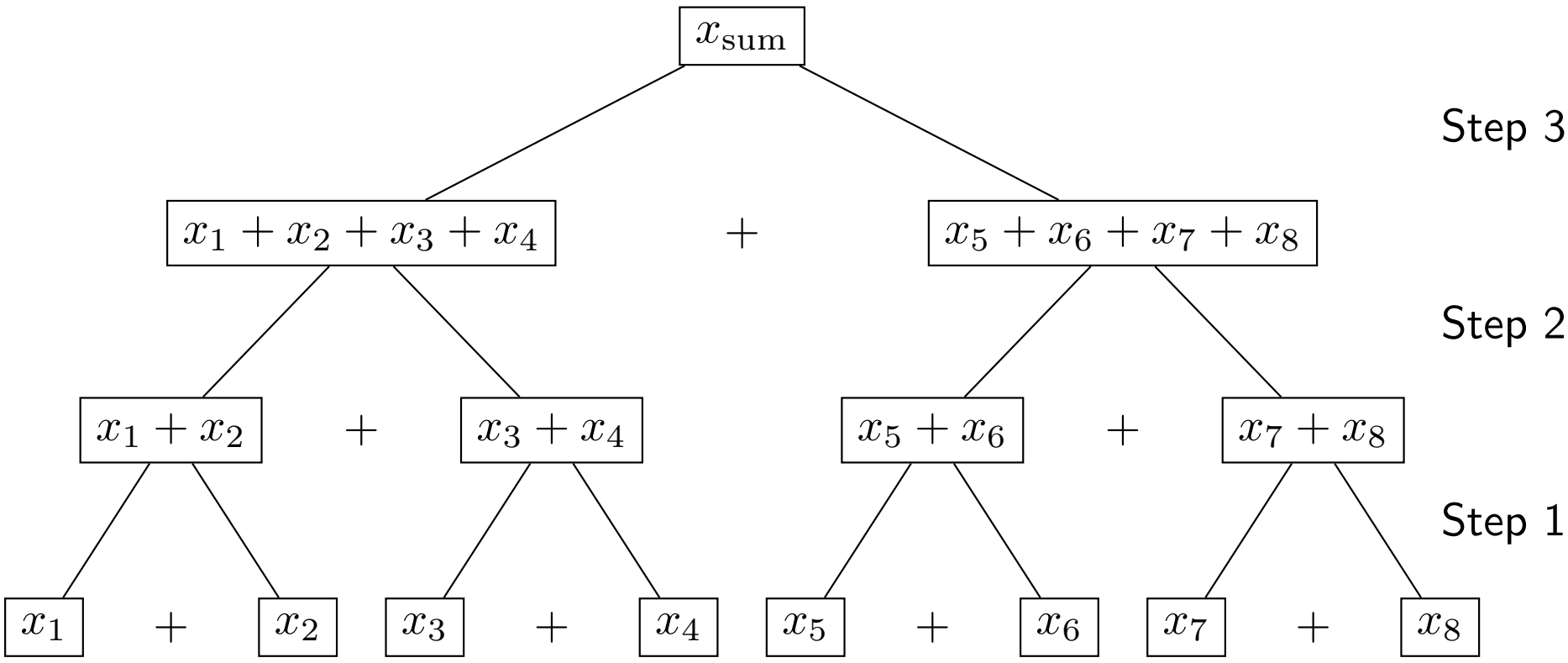
Reduction combines a set of numbers into one with an associative binary operator. A common example is the sum

$$x_{\text{sum}} = \sum_{i=1}^{n} x_i,$$

where $x_1, \ldots, x_n \in \mathbb{R}$. With $p = 1$ processor, reduction costs $\mathcal{O}(n)$.

Parallel reduction

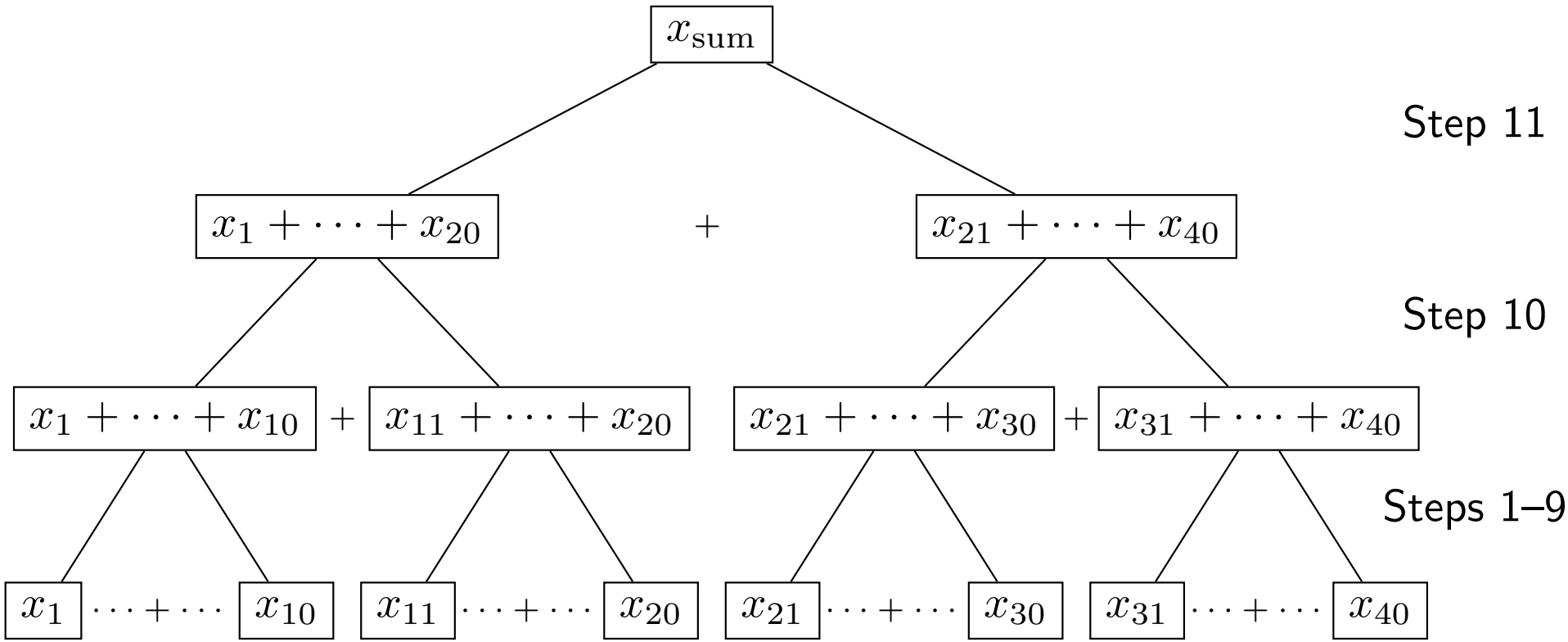
With $p \geq \lfloor n/2 \rfloor$ processors, reduction takes $\mathcal{O}(\log n)$ steps. In the following example with $n = 8$ and $p = 4$, $\mathcal{F}_p[\{x_1, \ldots, x_8\} \mapsto x_{\text{sum}}] = 3$.



General strategy: follow a binary tree with depth $\log_2 n$.

Parallel reduction

With $p < \lfloor n/2 \rfloor$ processors, reduction takes $\mathcal{O}(n/p + \log p)$ steps. In the following example with $n = 40$ and $p = 4$,
$\mathcal{F}_p[\{x_1, \ldots, x_{40}\} \mapsto x_{\text{sum}}] = 40/4 - 1 + \log_2 4 = 11$.



General strategy: (i) partition n numbers into p groups of sizes roughly n/p, (ii) reduce within p groups with $\mathcal{O}(n/p)$ steps, (iii) reduce p numbers with $\mathcal{O}(\log p)$ steps.

Parallel reduction

To summarize,

$$\mathcal{F}_p\left[\{x_1,\ldots,x_n\}\mapsto x_{\text{sum}}\right]=\begin{cases}\mathcal{O}\left(n\right) & \text{if } p=1\\ \mathcal{O}\left(n/p+\log p\right) & \text{if } 1<p<\lfloor n/2\rfloor\\ \mathcal{O}\left(\log n\right) & \text{if } p\geq\lfloor n/2\rfloor.\end{cases}$$

Likewise, we can compute

- minimum and maximum of $x_1,\ldots,x_n\in\mathbb{R}$,
- arithmetic mean, geometric mean, and product of $x_1,\ldots,x_n\in\mathbb{R}$,
- $\langle x,y\rangle$ for $x,y\in\mathbb{R}^n$, and
- $\|x\|_1$ and $\|x\|_\infty$ for $x\in\mathbb{R}^n$.

Parallel matrix-vector multiplication

Let $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$ and consider $\{A, x\} \mapsto b$.

$$\mathcal{F}_p\left[\{A, x\} \mapsto b\right] = \begin{cases} \mathcal{O}\left(mn\right) & \text{if } p = 1 \\ \mathcal{O}\left(mn/p\right) & \text{if } p \leq m \\ \mathcal{O}\left(mn/p + \log(p/m)\right) & m < p < mn/2 \\ \mathcal{O}\left(\log n\right) & \text{if } mn/2 \leq p. \end{cases}$$

For $m < p$, assign $\frac{p}{m}$ processors to compute $b_i = \sum_{j=1}^{n} A_{i,j} x_j$ with parallel reduction.

Other costs: coordination and communication

On a multi-core CPU, counting flops only is a useful approximation.

Parallel computing on a graphics processing unit (GPU) relies on thousands of slower processors. Cost of coordination may be significant.

In distributed and decentralized computing, many computers operate in parallel and communicate. Cost of communication may be significant.

Parallelizing linear algebra vs. high-level parallelism

When a method relies on linear algebraic operations (like $\{A, x\} \mapsto Ax$), it is possible to parallelize the linear algebra.

In some cases, a method itself is parallelizable at a higher level.

Example: Sum of smooth functions

Consider

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

where $h_1, \ldots, h_m$ are differentiable. FBS is

$$\begin{aligned} v^k &= -\frac{\alpha}{m}\sum_{i=1}^{m} \nabla h_i(x^k) \\ x^{k+1} &= \mathrm{Prox}_{\alpha f}\left(x^k + v^k\right) \end{aligned}$$

Assume $\mathrm{Prox}_{\alpha f}$ costs C_f flops and ∇h_i costs C_h flops (or fewer). Then

$$\begin{aligned} \mathcal{F}_p\left[x^k \mapsto x^{k+1}\right] &= \mathcal{F}_p\left[x^k \mapsto \{\nabla h_i(x^k)\}_{i=1}^m\right] + \mathcal{F}_p\left[\{\nabla h_i(x^k)\}_{i=1}^m \mapsto v^k\right] \\ &\qquad + \mathcal{F}_p\left[\{x^k, v^k\} \mapsto x^{k+1}\right] \\ &= \mathcal{O}\left(mC_h/p\right) + \mathcal{O}\left(mn/p\right) + \mathcal{O}\left(n/p + C_f\right) \\ &= \mathcal{O}\left((C_h + n)m/p + C_f\right). \end{aligned}$$

for $p \leq \min\{m, n\}$. Method parallelizable if $C_f = \mathcal{O}\left((C_h + n)m/p\right)$.

Example: Sum of proximable functions

Consider

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

Using the consensus technique, reformulate into

$$\underset{x_1,\ldots,x_m\in\mathbb{R}^n}{\text{minimize}} \quad f(x_1) + \delta_C(x_1,\ldots,x_m) + \frac{1}{m}\sum_{i=1}^{m} g_i(x_i),$$

where $C = \{(x_1,\ldots,x_m) \,|\, x_1 = \cdots = x_m\}$. DRS is

$$\begin{aligned} x^{k+1/2} &= \mathrm{Prox}_{\alpha f}\left(\frac{1}{m}\sum_{i=1}^{m} z_i^k\right), \\ x_i^{k+1} &= \mathrm{Prox}_{\alpha g_i}(2x^{k+1/2} - z_i^k) \\ z_i^{k+1} &= z_i^k + x_i^{k+1} - x^{k+1/2} \qquad \text{for } i = 1,\ldots,m. \end{aligned}$$

Assume $\mathrm{Prox}_{\alpha f}$ costs C_f and $\mathrm{Prox}_{\alpha g_i}$ costs C_g (or fewer). For $p \le m$,

$$\begin{aligned} \mathcal{F}_p\left[\mathbf{z}^k \mapsto \mathbf{z}^{k+1}\right] &= \mathcal{F}_p\left[\mathbf{z}^k \mapsto x^{k+1/2}\right] + \mathcal{F}_p\left[\{\mathbf{z}^k, x^{k+1/2}\} \mapsto \mathbf{z}^{k+1}\right] \\ &= \mathcal{O}\left(mn/p + C_f + C_g m/p\right). \end{aligned}$$

Example: Sum of proximable functions and a strongly convex function

Consider primal problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad f(x) + \sum_{i=1}^{m} g_i(a_i^\mathsf{T} x - b_i)$$

and dual problem

$$\underset{u_1,\ldots,u_m\in\mathbb{R}}{\text{maximize}} \quad -f^*\left(-\sum_{i=1}^{m} u_i a_i\right) - \sum_{i=1}^{m}(g_i^*(u_i) + b_i u_i)$$

generated by

$$\mathbf{L}(x, u_1, \ldots, u_m) = f(x) + \sum_{i=1}^{m}\langle u_i, a_i^\mathsf{T} x - b_i\rangle - \sum_{i=1}^{m} g_i^*(u_i),$$

where $a_1, \ldots, a_m \in \mathbb{R}^n$, $b_1, \ldots, b_m \in \mathbb{R}$, f is a strongly convex CCP function on $\mathbb{R}^n$, and $g_1, \ldots, g_m$ are proximable CCP functions on $\mathbb{R}$.

Example: Sum of proximable functions and a strongly convex function

FBS applied to the dual is

$$x^k = \nabla f^* \left(- \sum_{i=1}^m u_i^k a_i \right)$$
$$u_i^{k+1} = \mathrm{Prox}_{\alpha g_i^*} \left(u_i^k + \alpha (a_i^\intercal x^k - b_i) \right) \qquad \text{for } i = 1, \ldots, m.$$

(Since f is strongly convex, f^* is smooth.) Assume ∇f^* costs C_f flops and $\mathrm{Prox}_{\alpha g_i^*}$ costs C_g flops. Then for $p \leq m$ and $p \leq n$,

$$\mathcal{F}_p \left[\{u_1^k, \ldots, u_m^k\} \mapsto \{u_1^{k+1}, \ldots, u_m^{k+1}\} \right] = \mathcal{O}\left((C_g + n)m/p + C_f \right).$$

Example: Support-vector machine

In the support-vector machine (SVM) setup of machine learning, we solve

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{\lambda}{2}\|x\|^2 + \sum_{i=1}^{m} \max\{1 - y_i a_i^\intercal x, 0\},$$

where $a_1, \ldots, a_m \in \mathbb{R}^n$, $y_1, \ldots, y_n \in \{-1, 1\}$, and $\lambda > 0$. FBS applied to the dual is

$$x^k = \frac{1}{2\lambda}\left(-\sum_{i=1}^{m} u_i^k y_i a_i\right)$$
$$u_i^{k+1} = \Pi_{[-1,0]}\left(u_i^k - \alpha(1 - y_i a_i^\intercal x^k)\right) \qquad \text{for } i = 1, \ldots, m.$$

Parallelizable since

$$\mathcal{F}_p\left[\{u_1^k, \ldots, u_m^k\} \mapsto \{u_1^{k+1}, \ldots, u_m^{k+1}\}\right] = \mathcal{O}\left(nm/p\right)$$

for $p \leq \min\{m, n\}$.

Amdahl's law

Imagine the algorithm

$$x^{k+1/2} = x^k - \alpha \nabla f(x^k)$$
$$x^{k+1} = \mathrm{Prox}_{\alpha g}(x^{k+1/2})$$

takes $6\mathrm{ms}$ to evaluate $x^{k+1/2}$ and $3\mathrm{ms}$ to evaluate x^{k+1}.

If we reduce the computation of $x^{k+1/2}$ from $6\mathrm{ms}$ to $0\mathrm{ms}$, speedup is

$$\frac{9}{3+0} = 3.$$

Upper bounds the maximum speedup achievable by reducing the computation time of $x^{k+1/2}$.

Amdahl's law

If a part of a task takes time $\eta \in [0, 1]$, in proportion, and we speedup the part by s, then the total speedup is

$$S(s) = \frac{1}{1 - \eta + \eta/s}.$$

This formula is Amdahl's law.

The $s = \infty$ case $1/(1 - \eta)$ upper bounds the speedup.

A part of an algorithm is only worth accelerating if it occupies a significant portion of the runtime (if η is large).

Conclusion

The notion of computational cost we briefly considered is incomplete as it only accounts for flops while ignoring coordination and communication.

Nevertheless, this framework is a useful approximation for analyzing the running time of algorithms.