

Parallel Computing

Ernest K. Ryu and Wotao Yin

Large-Scale Convex Optimization via Monotone Operators

Computational complexity and parallel computing

Briefly discuss computational complexity.

Use examples to introduce parallel algorithms.

Outline

Computational complexity via flop count

Parallel computing

Floating-point operations

A floating-point operation (flop) is a single arithmetic operation on a floating-point number or a pair of those numbers such as addition, subtraction, multiplication, and division.

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

Floating-point operations

A floating-point operation (flop) is a single arithmetic operation on a floating-point number or a pair of those numbers such as addition, subtraction, multiplication, and division.

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

For example, given $x \in \mathbb{R}^n$,

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

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 $\mathcal{O}(n^3)$, where $A \in \mathbb{R}^{n \times n}$.

Processing power in flops

Flop per second is an indicator of the processing power of (a core of) of a CPU/GPU.

Each CPU core can process roughly 10^9 flops per second.

Processing power in flops

Flop per second is an indicator of the processing power of (a core of) of a CPU/GPU.

Each CPU core can process roughly 10^9 flops per second.

(flops needed)/(flops offered) roughly predicts the run time of an algorithm on a CPU. But this is a very rough estimate; expect a 10-fold or even a 100-fold inaccuracy.

Algorithm vs. method

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

But, they sit at different levels of specifications.

Algorithm vs. method

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

But, they sit at different levels of specifications.

Difference:

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

Algorithm vs. method

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

But, they sit at different levels of specifications.

Difference:

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

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 is ascribed to algorithm, not method.

Algorithm vs. method

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

Example: consider $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and the method

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

Algorithm corresponding to

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

costs $\mathcal{O}(mn)$ flops per iteration. But, by precomputing and storing $A^\top A \in \mathbb{R}^{n \times n}$ and $A^\top b \in \mathbb{R}^n$, algorithm corresponding to

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

costs $\mathcal{O}(n^2)$ flops per iteration.

Flop-count operator

Define: flop-count operator

$$\mathcal{F} [\{x_1, \dots, x_n\} \mapsto \{y_1, \dots, y_m\} \mid \mathcal{A}] :$$

number of flops \mathcal{A} to compute $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$.
(Algorithm \mathcal{A} , not a method, that determines the flop count.) When \mathcal{A} is clear from context, write $\mathcal{F} [\{x_1, \dots, x_n\} \mapsto \{y_1, \dots, y_m\}]$.

Flop-count operator

Define: flop-count operator

$$\mathcal{F} [\{x_1, \dots, x_n\} \mapsto \{y_1, \dots, y_m\} \mid \mathcal{A}] :$$

number of flops \mathcal{A} to compute $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$.
(Algorithm \mathcal{A} , not a method, that determines the flop count.) When \mathcal{A} is clear from context, write $\mathcal{F} [\{x_1, \dots, x_n\} \mapsto \{y_1, \dots, y_m\}]$. For example, when $A \in \mathbb{R}^{m \times n}$

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

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

$$x^{k+1/2} = (I + \alpha A^T A)^{-1} (z^k + \alpha A^T b)$$

$$x^{k+1} = S(2x^{k+1/2} - z^k; \alpha\lambda)$$

$$z^{k+1} = z^k + x^{k+1} - x^{k+1/2},$$

where S is soft-thresholding.

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

$$x^{k+1/2} = (I + \alpha A^T A)^{-1} (z^k + \alpha A^T b)$$

$$x^{k+1} = S(2x^{k+1/2} - z^k; \alpha \lambda)$$

$$z^{k+1} = z^k + x^{k+1} - x^{k+1/2},$$

where S is soft-thresholding. A naive implementation costs

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

Flop-count operator

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

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

and $\alpha A^T b$ with cost

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

Flop-count operator

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

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

and $\alpha A^T b$ with cost

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

In subsequent iterations,

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

Outline

Computational complexity via flop count

Parallel computing

Simplified view of parallel computing

(Over)simplified view of parallel computing: a group of computational agents working *simultaneously* on the same task.

Example of agents: CPU cores, GPU cores, or computers connected via LAN or over the internet.

Simplified view of parallel computing

(Over)simplified view of parallel computing: a group of computational agents working *simultaneously* on the same task.

Example of agents: CPU cores, GPU cores, or computers connected via LAN or 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

A task is embarrassingly parallel if trivial to parallelize.
(The fact that little effort is needed is “embarrassing.”)

Embarrassingly parallel

A task is embarrassingly parallel if trivial to parallelize.
(The fact that little effort is needed is “embarrassing.”)

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 are difficult to parallelize. Consider DRS:

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

The three steps must be computed serially.

Computational bottleneck usually in $\text{PROX}_{\alpha f}$ or $\text{PROX}_{\alpha g}$. If the bottleneck step (or steps) is not parallelizable by itself, DRS is not parallelizable.

Parallel flop count operator

Assume \mathcal{A} has access to p processors.
(But, \mathcal{A} may or may not process p flops in each parallel step.)

Notation: parallel flop-count operator:

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

the *number of parallel steps* that algorithm \mathcal{A} takes to compute $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$ and p processors.

Parallelizable methods and operators

An algorithm is parallel if it utilizes multiple computing units and is serial otherwise.

Parallelizable methods and operators

An algorithm is parallel if it utilizes multiple computing units and is serial otherwise.

But, A method is parallelizable if it has an parallel algorithm to provide a significant speedup.

Parallelizable methods and operators

An algorithm is parallel if it utilizes multiple computing units and is serial otherwise.

But, A method is parallelizable if it has an parallel algorithm to provide a significant speedup.

Computing $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$ is parallelizable if

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

for large enough p . Meaning of \ll depends on context.

Parallelizable methods and operators

An algorithm is parallel if it utilizes multiple computing units and is serial otherwise.

But, A method is parallelizable if it has an parallel algorithm to provide a significant speedup.

Computing $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$ is parallelizable if

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

for large enough p . Meaning of \ll depends on context. If

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

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

Parallelizable methods and operators

An algorithm is parallel if it utilizes multiple computing units and is serial otherwise.

But, A method is parallelizable if it has an parallel algorithm to provide a significant speedup.

Computing $\{y_1, \dots, y_m\}$ given $\{x_1, \dots, x_n\}$ is parallelizable if

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

for large enough p . Meaning of \ll depends on context. If

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

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

Operator \mathbb{T} is parallelizable if $\mathcal{F}_p [x \mapsto \mathbb{T}x] \ll \mathcal{F} [x \mapsto \mathbb{T}x]$.

Reduction

Reduction combines a set of numbers into one with an associative binary operator.

Reduction

Reduction combines a set of numbers into one with an associative binary operator.

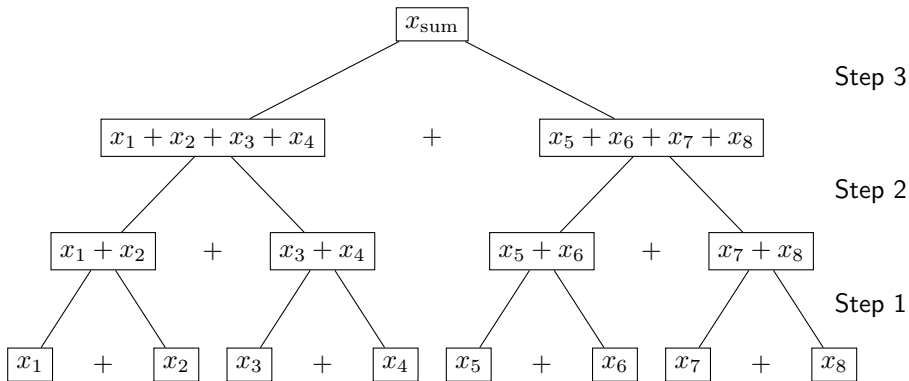
A common example is the sum

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

where $x_1, \dots, x_n \in \mathbb{R}$. 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 $p = 4$, $\mathcal{F}_p[\{x_1, \dots, x_8\} \mapsto x_{\text{sum}}] = 3$.

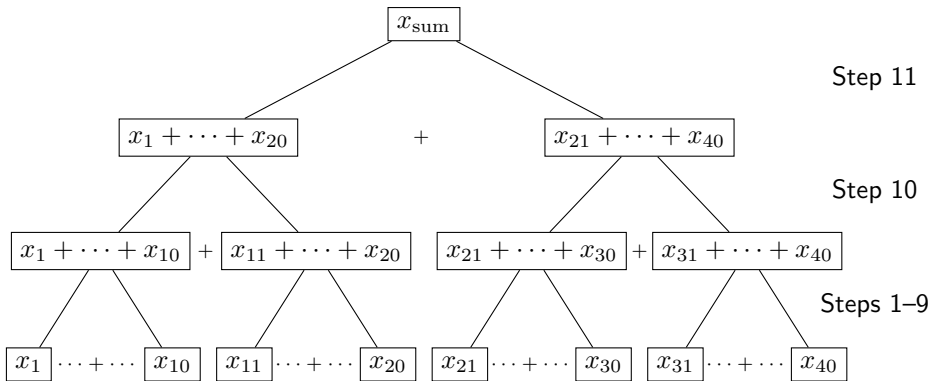


General strategy: follow a binary tree with depth $\log_2 n$.

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 $p = 4$,

$$\mathcal{F}_p[\{x_1, \dots, x_{40}\} \mapsto x_{\text{sum}}] = 40/4 - 1 + \log_2 4 = 11.$$



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,

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

Parallel reduction

To summarize,

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

Likewise, we can compute

- ▶ minimum and maximum of $x_1, \dots, x_n \in \mathbb{R}$,
- ▶ arithmetic mean, geometric mean, and product of $x_1, \dots, x_n \in \mathbb{R}$,
- ▶ $\langle x, y \rangle$ 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 $\{A, x\} \mapsto b$.

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

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 only flops is a useful approximation. Flops maybe *inadequate* since

- ▶ data organization is important
- ▶ latency, data transmission, and coordination may take significant time.

Other costs: coordination and communication

On a multi-core CPU, counting only flops is a useful approximation. Flops maybe *inadequate* since

- ▶ data organization is important
- ▶ latency, data transmission, and coordination may take significant time.

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

Other costs: coordination and communication

On a multi-core CPU, counting only flops is a useful approximation. Flops may be *inadequate* since

- ▶ data organization is important
- ▶ latency, data transmission, and coordination may take significant time.

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

When going distributed and decentralized, many computers (as opposed to CPU cores on a chip) operate in parallel and communicate. Cost of communication (latency, transmission, and coordination) becomes more 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.

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.

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.

There are also cases to run a method with different data; this task is parallelizable at an even 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, \dots, h_m are differentiable.

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, \dots, h_m are differentiable. FBS is

$$v^k = -\frac{\alpha}{m} \sum_{i=1}^m \nabla h_i(x^k)$$
$$x^{k+1} = \text{Prox}_{\alpha f}(x^k + v^k)$$

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, \dots, h_m are differentiable. FBS is

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

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

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

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

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

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

where $C = \{(x_1, \dots, x_m) \mid x_1 = \dots = x_m\}$.

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

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

where $C = \{(x_1, \dots, x_m) \mid x_1 = \dots = x_m\}$. DRS is

$$x^{k+1/2} = \text{Prox}_{\alpha f} \left(\frac{1}{m} \sum_{i=1}^m z_i^k \right),$$

$$x_i^{k+1} = \text{Prox}_{\alpha g_i} (2x^{k+1/2} - z_i^k)$$

$$z_i^{k+1} = z_i^k + x_i^{k+1} - x^{k+1/2} \quad \text{for } i = 1, \dots, m.$$

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

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

where $C = \{(x_1, \dots, x_m) \mid x_1 = \dots = x_m\}$. DRS is

$$x^{k+1/2} = \text{Prox}_{\alpha f} \left(\frac{1}{m} \sum_{i=1}^m z_i^k \right),$$

$$x_i^{k+1} = \text{Prox}_{\alpha g_i} (2x^{k+1/2} - z_i^k)$$

$$z_i^{k+1} = z_i^k + x_i^{k+1} - x^{k+1/2} \quad \text{for } i = 1, \dots, m.$$

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

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

Example: Sum of proximable functions and a strongly convex function

Consider primal problem

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

and dual problem

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

generated by

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

where $a_1, \dots, a_m \in \mathbb{R}^n$, $b_1, \dots, b_m \in \mathbb{R}$, f is a strongly convex CCP function on \mathbb{R}^n , and g_1, \dots, 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

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

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

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

Example: Support-vector machine

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

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

where $a_1, \dots, a_m \in \mathbb{R}^n$, $y_1, \dots, y_m \in \{-1, 1\}$, and $\lambda > 0$.

Example: Support-vector machine

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

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

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

$$\begin{aligned} x^k &= \frac{1}{2\lambda} \left(- \sum_{i=1}^m (u_i^k y_i) a_i \right) \\ u_i^{k+1} &= \Pi_{[-1,0]} (u_i^k - \alpha(1 - y_i a_i^\top x^k)) \quad \text{for } i = 1, \dots, m. \end{aligned}$$

Example: Support-vector machine

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

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

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

$$\begin{aligned} x^k &= \frac{1}{2\lambda} \left(- \sum_{i=1}^m (u_i^k y_i) a_i \right) \\ u_i^{k+1} &= \Pi_{[-1,0]} \left(u_i^k - \alpha(1 - y_i a_i^\top x^k) \right) \quad \text{for } i = 1, \dots, m. \end{aligned}$$

Parallelizable since

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

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

Amdahl's law

Imagine the algorithm

$$x^{k+1/2} = x^k - \alpha \nabla f(x^k) \quad \text{takes 6ms}$$

$$x^{k+1} = \text{Prox}_{\alpha g}(x^{k+1/2}) \quad \text{takes 3ms.}$$

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

$$\frac{6 + 3}{0 + 3} = 3.$$

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 data organization and communication.

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