High Performance Computing

Time Complexity and Parallisation

Martin Raum

Time Complexity

Given a program or algorithm (assuming a suitable machine model)
that depends on a parameter n, or input of size n,
we say that it has time complexity (or runtime) $\mathcal{O}(f(n))$
for a function $f : \mathbb{Z}_{\geq n_0} \to \mathbb{R}$,
defined on a $\mathbb{Z}_{\geq n_0} := \{n \in \mathbb{Z} : n \geq n_0\}$ for some integer n_0,
if there is a constant time span c and an integer $n_1 \geq n_0$ such that

$$\text{runtime for parameter/input size } n \leq c \cdot f(n) \quad \text{for all } n \geq n_1.$$

Alternative, one writes $\mathcal{O}(f(n))$, but this does not coincide with
the definition of $\mathcal{O}(\cdot)$ in mathematics.

Time Complexity

For example, summing the first n integers naively has time complexity $\mathcal{O}(n)$.

Summing the first n integers via the formula $n(n+1)/2$ time complexity $\mathcal{O}(\log(n)^*)$.

There are algorithms to solve the travelling salesperson problem in time complexity $\mathcal{O}(n^2 2^n)$. It is not known whether there is an algorithm that solves in $\mathcal{O}(p(n))$ for some polynomial function p.

Time complexity of parallelized programs

Parallelization gives rise to a new parameter r, the number of compute instances (compute nodes/processes/threads).

Contributions to the runtime $\mathrm{TIME}(n; r)$ split up into three fundamentally different terms:

- sequential $\mathrm{TIME}_{\mathrm{seq}}(n)$
- parallelizable $\mathrm{TIME}_{\mathrm{par}}(n)$, and
- communication/synchronization $\mathrm{TIME}_{\mathrm{sync}}(n)$.

A naive approach gives

$$\mathrm{TIME}(n; r) = \mathrm{TIME}_{\mathrm{seq}}(n) + \frac{1}{r}\mathrm{TIME}_{\mathrm{par}}(n) + r\mathrm{TIME}_{\mathrm{sync}}(n).$$

Optimally parallelized programs

$$\mathrm{TIME}(n;r) = \mathrm{TIME}_{\mathrm{seq}}(n) + \frac{1}{r}\mathrm{TIME}_{\mathrm{par}}(n) + r\,\mathrm{TIME}_{\mathrm{sync}}(n)$$

attains its minimum at

$$r = \sqrt{\frac{\mathrm{TIME}_{\mathrm{par}}(n)}{\mathrm{TIME}_{\mathrm{sync}}(n)}}$$

and the estimated optimal runtime is

$$\mathrm{TIME}_{\mathrm{seq}}(n) + 2\sqrt{\mathrm{TIME}_{\mathrm{par}}(n)\mathrm{TIME}_{\mathrm{sync}}(n)}.$$

Optimally parallelized programs

Due to the effort of synchronization the best possible number of compute instance is limited. In many problems, $\mathrm{TIME}_{\mathrm{sync}}(n)$ is so marginal, that effectively this bound is not reached:

$$\mathrm{TIME}_{\mathrm{seq}}(n) + 2\sqrt{\mathrm{TIME}_{\mathrm{par}}(n)\mathrm{TIME}_{\mathrm{sync}}(n)} \longrightarrow \mathrm{TIME}_{\mathrm{seq}}(n)$$

$$\text{as } \mathrm{TIME}_{\mathrm{sync}}(n) \to 0.$$

Sub-optimally parallelized programs

In many real-world attempts to "scalar up" a system, however, the impact of $\mathrm{TIME}_{\mathrm{sync}}(n)$ deteriorates efforts.

Assuming naively that an implementation that is not parallelized has runtime

$$\mathrm{TIME}_{\mathrm{seq}}(n) + \mathrm{TIME}_{\mathrm{par}}(n),$$

we have to require that

$$\mathrm{TIME}_{\mathrm{seq}}(n) \leq \frac{\mathrm{TIME}_{\mathrm{par}}(n)}{\sqrt{2}}$$

in order to beat it with a parallelized one.

But close to this cut-off the optimal number of computing instances is $\approx 1/\sqrt{2} < 1$, while in reality a systems "in the cloud" run on hundreds of nodes.

Sub-optimally parallelized programs

For given r, to achieve

$$\begin{aligned}\mathrm{TIME}(n;r) &= \mathrm{TIME}_{\mathrm{seq}}(n) + \frac{1}{r}\mathrm{TIME}_{\mathrm{par}}(n) + r\,\mathrm{TIME}_{\mathrm{sync}}(n) \\ &\leq \mathrm{TIME}_{\mathrm{seq}}(n) + \mathrm{TIME}_{\mathrm{par}}(n)\end{aligned}$$

we have to have

$$\frac{\mathrm{TIME}_{\mathrm{sync}}(n)}{\mathrm{TIME}_{\mathrm{par}}(n)} \leq \frac{1}{r}\left(1 - \frac{1}{r}\right) \to \frac{1}{r} \quad \text{as } r \to \infty.$$

If your big data computation on 1000 nodes in the cloud consists of matching ten words in a text, you might violate this bound.

Asymptotic time complexity

It is also legitimate to analyze $\mathrm{TIME}(n; r)$ for $n \to \infty$, but this requires estimates or expressions for the contributions $\mathrm{TIME}_{\mathrm{seq}}(n)$, $\mathrm{TIME}_{\mathrm{par}}(n)$, and $\mathrm{TIME}_{\mathrm{sync}}(n)$.

Asymptotic time complexity

The runtime of a matrix-vector multiplication of size n on a single node is approximately $2n^2t$, where t denotes the time consumed for one floating point operation.

A parallelized variant in which the matrix gets sliced could, for example, achieve

$$\frac{2n^2t}{r} + (r-1)\left(l + \frac{nb}{r}\right),$$

where l and b denote the latency and bandwidth of an underlying communication system.

As $n \to \infty$ is is asymptotically $2n^2t/r$.

Speedups

Instead of time complexity one can consider speedups. The following two approaches to speedups put emphasis on different aspects.

Focusing on the problem:

$$\text{SPEEDUP}(r) = \frac{\text{best possible runtime achieved on 1 node}}{\text{runtime achieved with } r \text{ nodes}}.$$

Focusing on algorithm or implementation:

$$\text{SPEEDUP}(r) = \frac{\text{time achieved on 1 node}}{\text{time achieved on } r \text{ nodes}}.$$

Superlinear speedup

Effects like memory locality can yield

$$\text{SPEEDUP}(r) > 1.$$

For instance, L1 cache is usually attached to each core, therefore there is more L1 cache available in total when parallelizing a program.