Parallel Program Performance Analysis

CSCI 4850/5850 High-Performance Computing

Spring 2018

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
Suppose we want to compute in parallel

\[
\text{for } (i = 0; i < N; i++) \\
\quad z[i] = x[i] + y[i];
\]

Then the obvious choice is to split the iteration space in \( P \) equal-sized \( N/P \) chunks and let each processor share the work (worksharing) of the loop:

\[
\text{for each processor } p \text{ from 0 to } P-1 \text{ do:} \\
\quad \text{for} \\
\quad \quad z[i] = x[i] + y[i];
\]

We would assume that this parallel version runs \( P \) times faster, that is, we hope for \textit{linear speedup}

Unfortunately, in practice this is not the case because of overhead in communication and synchronization
Speedup

- Suppose we want to compute in parallel
  \[
  \text{for } (i = 0; i < N; i++) \\
  \quad z[i] = x[i] + y[i];
  \]

- Then the obvious choice is to split the iteration space in \( P \) equal-sized \( N/P \) chunks and let each processor share the work (worksharing) of the loop:
  \[
  \text{for each processor } p \text{ from 0 to } P-1 \text{ do: } \\
  \quad \text{for } (i = p*N/P; i < (p+1)*(N/P); i++) \\
  \quad \quad z[i] = x[i] + y[i];
  \]

- We would assume that this parallel version runs \( P \) times faster, that is, we hope for **linear speedup**

- Unfortunately, in practice this is not the case because of overhead in communication and synchronization
**Speedup**

- **Definition:** the *speedup* of an algorithm using $P$ processors is defined as

  $$S_P = \frac{t_s}{t_P}$$

  where $t_s$ is the execution time of the *best available sequential algorithm* and $t_P$ is the execution time of the parallel algorithm.

- The speedup is *perfect* or *ideal* if $S_P = P$

- The speedup is *linear* if $S_P \approx P$

- The speedup is *superlinear* when, for some $P$, $S_P > P$
Speedup

- super-linear speedup (wonderful)
- linear speedup
- sub-linear speedup (common)

\[ Sp \]

\[ \# \text{ of processors} \]
Relative Speedup

- **Definition:** The *relative speedup* is defined as
  \[ S^1_P = \frac{t_1}{t_P} \]
  where \( t_1 \) is the execution time of the parallel algorithm on one processor.

- Similarly, \( S^k_P = \frac{t_k}{t_P} \) is the relative speedup with respect to \( k \) processors, where \( k < P \).

- The relative speedup \( S^k_P \) is used when \( k \) is the smallest number of processors on which the problem will run.

- \textit{Relative Speedup}(N,P) = \text{Time to solve a problem with 1 processor} / \text{Time to solve a problem with } P \text{ processors}
**Efficiency**

- **Definition:** the *efficiency* of an algorithm using $P$ processors is

  \[ E_P = \frac{S_P}{P} = \frac{T_S}{P \times T_P} \]

- Efficiency estimates how well-utilized the processors are in solving the problem, compared to how much effort is lost in idling and communication/synchronization.

- Ideal (or perfect) speedup means 100% efficiency $E_P = 1$.

- Many difficult-to-parallelize algorithms have efficiency that approaches zero as $P$ increases.
Scalability

- Speedup describes how the parallel algorithm’s performance changes with increasing $P$

- **Scalability** concerns the efficiency of the algorithm with changing problem size $N$ by choosing $P$ dependent on $N$ so that the efficiency of the algorithm is bounded below

- Simply, **Scalability** is a measure of a parallel system’s capacity to increase speedup in proportion to the number of processors.
Scalability Analysis

- Predicts performance of the parallel program for large \( W, p \)
- The larger the overhead the lower the efficiency

\[
E = \frac{S}{P} = \frac{T_S}{pT_P} = \frac{T_S}{T_O + T_S} = \frac{1}{1 + \frac{T_O(W,p)}{T_S}}
\]

where \( T_O = pT_P - T_S \)
Amdahl’s Law

- Strong scaling: same problem solved on more processors.
- \( W = T_S = \text{constant}, \ p = \text{increases} + \)
- \( t_{\text{SER}} = \text{inherently serial part of the code} \ T_S \)

\[
T_P = t_{\text{ser}} + \frac{T_S - t_{\text{ser}}}{p}
\]

\[
S = \frac{T_S}{T_P} = \frac{T_S}{t_{\text{ser}} + \frac{T_S - t_{\text{ser}}}{p}} \xrightarrow{p \uparrow \infty} \frac{T_S}{t_{\text{ser}}}
\]

- E.g., serial part is 5%. Even with \( p = 1024 \) processors the speedup is only close to 20.
Suppose the run-time of a serial program is given by $T_{\text{serial}} = n^2$, where the units of the run-time are in microseconds. Suppose that a parallelization of this program has run-time $T_{\text{parallel}} = n^2/p + \log_2(p)$. Write a program that finds the speedups and efficiencies of this program for various values of $n$ and $p$. Run your program with $n = 10, 20, 40, \ldots, 320$, and $p = 1, 2, 4, \ldots, 128$. What happens to the speedups and efficiencies as $p$ is increased and $n$ is held fixed? What happens when $p$ is fixed and $n$ is increased?