PERFORMANCE ANALYSIS

- Introduction to performance analysis
- Amdahl’s law
- Speed-up, Efficiency
- Scalability

**Speed-up and efficiency**

- Parallel run time of a program = time elapsed between beginning of parallel program and completion of last (parallel) process.

  **Speed-up:**
  \[
  S(p) = \frac{\text{Run-time on one processor}}{\text{Run-time on } p \text{ processors}}
  \]

  - NOTE: In practice, speed-up refers to observed speed-up. It can be estimated theoretically.

**Example:** Sum of \(n\) numbers with cascade sum. Sequential time is \((n - 1) \times \tau\). If communication time is neglected (PRAM model), then time on \(n/2\) processors is \(\log_2(n)\). So speed-up using \(n/2\) processors is

  \[
  S\left(\frac{n}{2}\right) \approx \frac{n}{\log_2(n)}
  \]

  - Speed-up is normally \(\leq p\). However, “superlinear” speed-up can be observed in some cases.
  - Examples: (1) effect of cache; (2) Better vectorization of parallel algorithm, (3) Parallel algorithm performs different operations.

**Efficiency:**

\[
E(p) = \frac{S(p)}{p}
\]

- In general \(E(p) \leq 1\), but there are cases when \(E(p) > 1\) (see above). Efficiencies close to one are hard to achieve.

**Cost:** Sum of times spend by all processors in the execution of the algorithm.

- An algorithm is cost-optimal if cost is of the same order as cost on one processor.

**Example:** In the example of the cascade sum above is the algorithm cost-optimal?
Classical performance model: Amdahl's Law

- Main point: speedup of a parallel program is limited by the time needed for the serial fraction of the problem

- Let $T(p) =$ run-time for a parallel program on $p$ processors

- If a problem has $W$ operations of which a component of $W_s$ operations are serial, the best achievable time on $p$ Processors is:

$$T(p) = \frac{W-W_s}{p} + W_s$$

- So best achievable speed-up is

$$S(p) = \frac{W}{(W-W_s)/p+Ws}$$

- Let $f = \frac{W_s}{W}$. Then

$$S(p) = \frac{1}{(1-f)/p+f}$$ and $$E(p) = \frac{1}{1+f(p-1)}$$

- Known as Amdahl's law (1967)

- As $p$ goes to infinity we have $S(p) < \frac{W}{W_s}$ – which is $1/f$ in the previous notation.

- Find $S(p)$ when $W_s/W = 0.2$ and the limits of $S(p)$ and efficiency $E(p)$ in this case

- Determine the number of processors $p_{1/2}$ for which efficiency is 50 percent. Determine the number of processors $p_e$ for which efficiency is equal to $e$ (with $0 < e < 1$).

Example: Sum of $n$ numbers on $p$ processors. There is a sequential part (summation of subsums) and the parallel part (computing the subsums in parallel).

- Example of the sum of $n$ numbers see earlier where the summation of subsums is done with the cascade algorithm.

Consequences of Amdahl's law:

- A small part of sequential code can have a big effect on performance

- Effort in parallelizing a small fraction of sequential code may yield a big pay-off
**Scalability**

- In rough terms: An algorithm is scalable if increasing the number of processors does not degrade efficiency. An algorithm is not scalable if efficiency goes to zero as \( p \to \infty \).
- The total overhead of a parallel program is defined as
  \[
  T_o(p) = pT(p) - T(1)
  \]
  Here \( T(1) \) is the sequential time (sometimes denoted by \( T_S \)).
  Therefore:
  \[
  E(p) = \frac{T(1)}{pT(p)} = \frac{1}{1 + T_o(p)/T(1)}
  \]
- Typically, \( T_o(p) \) increases with \( p \). Note that \( T_o \) includes times for sequential parts. It grows at least linearly with \( p \).

**Gustafson’s law**

- In this model, the ratio \( T_o(p)/T(1) \) is reduced by increasing the problem size, i.e., \( T(1) \). Rationale: practically \( p \) often increased in order to solve a bigger problem.
- Within Amdahl’s model, this means we need to increase size to keep \( f \) constant.
- Equivalently: assume that the time on a \( p \)-processor system is fixed and let \( f \) be the fraction of sequential code on the \( p \)-processors,
  \[
  fT(p) = \text{run time for sequential part}
  
  (1 - f)T(p) = \text{run time for parallel part}
  \]

**Scaled Speed-up – Weak Scaling**

- Time it would take the same program on one PE:
  \[
  T(1) = fT(p) + p(1 - f)T(p) \quad \rightarrow \quad S(p) = f + (1 - f)p
  \]
  which is linear in \( p \). Note that \( E(p) = 1 - f + f/p \)
- Known as Gustafson’s law or Gustafson-Barsis law

**Problem size** \( W \) here is amount of sequential work (\# seq. operations).

- Note: problem size increased linearly.
- In practice, calculate scaled speed-ups by allowing the problem size to be as large as can be fit in memory.
Formula for scaled speed-up in practice:

\[ S_p^G = \omega_p \times \frac{\text{Time for solving } Q_1}{\text{Time for solving } Q_p} \]

where \( Q_i \) = maximum size problem that can be solved on an \( i \)-processor computer, and \( \omega_i \) is an adjustment factor:

\[ \omega_p = \frac{\# \text{ ope's for solving } Q_p}{\# \text{ ope's for solving } Q_1} \]

This sort of analysis is known as a \textbf{Weak Scaling analysis}.

\textit{Example:} Gaussian elimination (GE) If a problem of size \( n \times n \) can fit on one processor, then a problem of size \( (np^{1/2}) \times (np^{1/2}) \) can fit on a \( p \)-processor system (assuming memory size is proportional to the number of PEs).

Asymptotically in GE: \# ope's = \( O(n^3) \) - therefore \( \omega_p \approx p^{3/2} \).

Thus, when going from 1 to 16 processors (a \( 4 \times 4 \) grid) the matrix size increases by a factor of 4, so \( \omega_p = 4^{3/2} = 64 \).

If it takes 1s to solve the \( n \times n \) problem on one processor and 8s to solve the \( 4n \times 4n \) on the 16-node machine, then the scaled speed-up would be

\[ S_{16}^G = 64 \times \frac{1}{8} = 8. \]

A certain parallel algorithm dealing with matrices is determined to have parallel complexity \( T(p) = n^2/p + k \times p \times n \) where \( n \) is the matrix size and \( k \) is a certain constant. The sequential algorithm runs in time \( T(1) = n^2 \). Determine the overhead function.

Note: In the end, you need to express everything in terms of \( W \) not \( n \).

Following up on Lab 1, plot the scaled speed-up you obtain for the matrix-matrix product – You can increase the size \( n \) linearly with the number of processors (up to a maximum of 32 processes).