Lecture 4

Performance Analysis

Performance Metrics, Postulates

IKC-MH.57 *Introduction to High Performance and Parallel Computing* at November 03, 2023

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University

Contents

Performance Analysis

- Analysis of the performance measures of parallel programs.
- Two computational models;
  1. *the equal duration* processes
  2. *parallel computation with serial sections*.
- Two measures;
  1. speed-up factor
  2. efficiency.
- The impact of the communication overhead on the overall speed performance of multiprocessors.
- The scalability of parallel systems.

Computational Models - Equal Duration Model I

Assume that a given computation can be divided into concurrent tasks for execution on the multiprocessor.

- In this model (t_s: execution time of the whole task using a single processor),
  - a given task can be divided into n equal subtasks,
  - each of which can be executed by one processor,
  - the time taken by each processor to execute its subtask is

$$t_p = \frac{t_s}{n}$$

  - since all processors are executing their subtasks simultaneously, then the time taken to execute the whole task is

$$t_p = \frac{t_s}{n}$$

- The speed-up factor of a parallel system can be defined as
  - the ratio between the time taken by a single processor to solve a given problem
  - to the time taken by a parallel system consisting of n processors to solve the same problem.

Computational Models - Equal Duration Model II

- Speed Up;

$$S(n) = \frac{t_s}{t_p} = \frac{t_s}{t_s/n} = n \quad (1)$$

- This equation indicates that, according to the equal duration model, the speed-up factor resulting from using n processors is equal to the number of processors used (n).
- One important factor has been ignored in the above derivation.
- This factor is the communication overhead, t_c, which results from the time needed for processors to communicate and possibly exchange data while executing their subtasks.
- Then the actual time taken by each processor to execute its subtask is given by

$$S(n) = \frac{t_s}{t_p} = \frac{t_s}{t_s/n + t_c} = \frac{n}{1 + n * t_c/t_s} \quad (2)$$

- This equation indicates that the **relative values of t_s and t_c affect the achieved speed-up factor**.

Computational Models - Equal Duration Model III

- A number of cases can then be studied:
  1. if $t_c \ll t_s$ then the potential speed-up factor is approximately n
  2. if $t_c \gg t_s$ then the potential speed-up factor is $t_s/t_c \ll 1$
  3. if $t_c = t_s$ then the potential speed-up factor is $n/n + 1 \cong 1$, for $n \gg 1$.
- In order to scale the speed-up factor to a value between 0 and 1, we divide it by the number of processors, n.
- The resulting measure is called the efficiency, E.
- The efficiency is a measure of the **speed-up achieved per processor**.
- According to the simple equal duration model, the efficiency E is equal to 1, if the communication overhead is ignored.
- However if the communication overhead is taken into consideration, the efficiency can be expressed as

$$E = \frac{1}{1 + n * t_c/t_s} \tag{3}$$

Computational Models - Equal Duration Model IV

- Although simple, the equal duration model is however unrealistic.
- This is because it is based on the assumption that a given task can be divided into a number of equal subtasks.
- However, real algorithms contain some (serial) parts that cannot be divided among processors.
- These (serial) parts must be executed on a single processor.

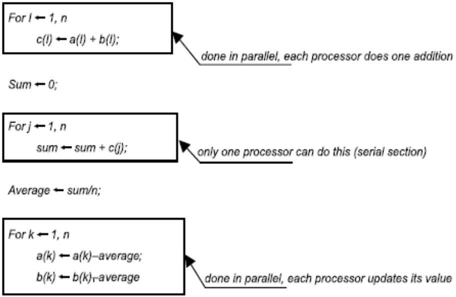

Figure: Example program segments.

Computational Models - Equal Duration Model V

- In Figure program segments, we assume that we start with a value from each of the two arrays (vectors) *a* and *b* stored in a processor of the available *n* processors.
  - The first program block can be done in parallel; that is, each processor can compute an element from the array (vector) *c*. The elements of array *c* are now distributed among processors, and each processor has an element.
  - The next program segment cannot be executed in parallel. This block will require that the elements of array *c* be communicated to one processor and are added up there.
  - The last program segment can be done in parallel. Each processor can update its elements of *a* and *b*.

Computational Models - Parallel Computation I

- It is assumed (or known) that **a fraction** f of the given task (computation) is not dividable into concurrent subtasks.
- The remaining part $(1 - f)$ is assumed to be dividable into concurrent subtasks.
- The time required to execute the task on n processors is

$$t_p = t_s * f + (1 - f) * (t_s/n)$$

- The speed-up factor is therefore given by

$$S(n) = \frac{t_s}{t_s * f + (1 - f) * (t_s/n)} = \frac{n}{1 + (n - 1) * f} \qquad (4)$$

- According to this equation, the potential speed-up due to the use of n processors is determined primarily by the fraction of code that cannot be divided.
- If the task (program) is completely serial, that is, $f = 1$, then no speed-up can be achieved regardless of the number of processors used.

Computational Models - Parallel Computation II

- This principle is known as Amdahl's law.
- It is interesting to note that according to this law, the maximum speed-up factor is given by

$$lim_{n \to \infty} S(n) = \frac{1}{f}$$

- Therefore, the improvement in performance (speed) of a parallel algorithm over a sequential one is
  - limited not by the number of processors employed
  - but rather
    by the fraction of the algorithm that cannot be parallelized.
- According to Amdahl's law, researchers were led to believe that a substantial increase in speed-up factor would **not be possible** by using parallel architectures.
- NOT parallelizable;
  - communication overhead,
  - a sequential fraction, f

Postulates - Amdahl's Law I

- Amdahl's law made it so pessimistic to build parallel computer systems.
- Due to the intrinsic limit set on the performance improvement (speed) regardless of the number of processors used.
- An interesting observation to make here is that according to Amdahl's law, f is fixed and does not scale with the problem size, n.
- However, it has been practically observed that some **real parallel algorithms** have a fraction that is a function of n.
- Let us assume that f is a function of n such that $lim_{n\to\infty} f(n) = 0$

$$lim_{n\to\infty} S(n) = lim_{n\to\infty} \frac{n}{1 + (n-1) * f(n)} = n \qquad (5)$$

- This is clearly in contradiction to Amdahl's law.
- It is therefore **possible to achieve a linear speed-up factor** for large-sized problems, given that

$$lim_{n\to\infty} f(n) = 0$$

a condition that has been practically observed.