1 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.

1.1 Computational Models

1.1.1 Equal Duration Model

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.

- Speed Up;

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

- 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

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

- This equation indicates that the **relative values of t_s and t_c affect the achieved speed-up factor**.
- A number of cases can then be studied:

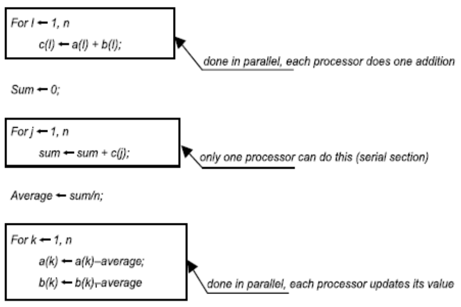
  1. if $t_c \ll t_s$ then the potential speed-up factor is approximately n
  2. if $t_c \gg t_s$ then the potential speed-up factor is $t_s/t_c \ll 1$
  3. if $t_c = t_s$ then the potential speed-up factor is $n/n + 1 \cong 1$, for $n \gg 1$.

- 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

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

- 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.



```
For l ← 1, n
      c(l) ← a(l) + b(l);

Sum ← 0;

For j ← 1, n
      sum ← sum + c(j);

Average ← sum/n;

For k ← 1, n
      a(k) ← a(k)–average;
      b(k) ← b(k)-average
```

done in parallel, each processor does one addition
only one processor can do this (serial section)
done in parallel, each processor updates its value

Figure 1: Example program segments.

- In Fig. 1 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.

1.1.2 Parallel Computation with Serial Sections Model

- 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.
- 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

- The maximum speed-up factor under such conditions is given by

$$S(n) = \frac{t_s}{t_s * f + (1-f) * (t_s/n) + t_c} = \frac{n}{(n-1) * f + 1 + n * (t_c/t_s)} \quad (5)$$

$$lim_{n\rightarrow\infty} S(n) = lim_{n\rightarrow\infty} \frac{n}{(n-1) * f + 1 + n * (t_c/t_s)} = \frac{1}{f + (t_c/t_s)}$$

- The above formula indicates that the maximum speed-up factor is determined not by the number of parallel processors employed but *by the fraction of the computation that is not parallelized and the communication* overhead.

- Recall that the efficiency is defined as the ratio between the speed-up factor and the number of processors, n.

- The efficiency can be computed as:

$$\begin{array}{r} E(no\ communication\ overhead) = \frac{1}{1+(n-1)*f} \\ E(with\ communication\ overhead) = \frac{1}{(n-1)*f+1+n*(t_c/t_s)} \end{array} \quad (6)$$

- As the number of processors increases, it may become difficult to use those processors efficiently.

1.2 Skeptic Postulates For Parallel Architectures

1.2.1 Grosch's Law

A number of postulates were introduced by some well-known computer architects expressing about the usefulness of parallel architectures.

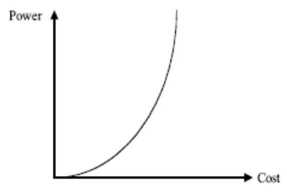


Figure 2: Power-cost relationship according to Grosch's law.

- It was as early as the late 1940s that H. Grosch studied the relationship between the power of a computer system, P, and its cost, C.

- He postulated that $P = K * C^s$, where s and K are positive constants. Grosch postulated further that the value of s would be close to 2.
- Simply stated, Grosch's law implies that the power of a computer system increases in proportion to the square of its cost (see Fig. 6).
- Alternatively, one can express the cost of a system as $C = sqrt(P/K)$ assuming that $s = 2$.
- According to Grosch's law, in order to sell a computer for twice as much, it must be four times as fast.
- Alternatively, to do a computation twice as cheaply, one has to do it four times slower.
- With the advances in computing, it is easy to see that Grosch's law is overturned, and it is possible to build faster and less expensive computers over time.

1.2.2 Amdahl's Law

- Similar to Grosch's law, 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 $lim_{n\to\infty} f(n) = 0$

$$lim_{n\to\infty} S(n) = lim_{n\to\infty} \frac{n}{1 + (n-1) * f(n)} = n \tag{7}$$

- 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.

- For example, researchers at the Sandia National Laboratories have shown that using a 1024-processor hypercube multiprocessor system for a number of engineering problems, a linear speed-up factor can be achieved.
- Consider, for example, the well-known engineering problem of multiplying a large square matrix $A(m \times m)$ by a vector $X(m)$ to obtain a vector, that is, $C(m)$.
- Assume further that the solution of such a problem is performed on a binary tree architecture consisting of n nodes (processors).
- Initially, the root node stores the vector $X(m)$ and the matrix $A(m \times m)$ is distributed row-wise among the n processors such that the maximum number of rows in any processor is $m/n + 1$.

A simple algorithm to perform such computation consists of the following three steps:

1. The root node sends the vector $X(m)$ <u>to all processors</u> in the order of $O(m * logn)$
2. All processors perform the product $C_i = \sum_{j=1}^{m} a_{ij} * x_j$ in
$$O(m * (m/n + 1)) = O(m) + O(\frac{m^2}{n})$$
3. All processors send their C_i values to the root node in $O(m * logn)$.

- According to the above algorithm, the amount of computation needed is
$$O(m * logn) + O(m) + O(\frac{m^2}{n}) + O(m * logn) = O(m^2)$$
- The indivisible part of the computation is equal to
$$O(m) + O(m * logn)$$
- Therefore, the fraction of computation that is indivisible
$$f(m) = \frac{(O(m) + O(m * logn))}{O(m^2)} = O(\frac{(1 + logn)}{m})$$
- Notice that $lim_{m \to \infty} f(m) = 0$.

- Hence, contrary to Amdahl's law, a linear speed-up can be achieved for such a large-sized problem.
- It should be noted that in presenting the above scenario for solving the matrix vector multiplication problem, we have assumed that the memory size of each processor is large enough to store the maximum number of rows expected.
- This assumption amounts to us saying that with n processors, the memory is n times larger.
- Naturally, this argument is more applicable to message passing parallel architectures than it is to shared memory ones.
- The Gustafson-Barsis law makes use of this argument.

1.2.3 Gustafson-Barsis's Law

- In 1988, Gustafson and Barsis at Sandia Laboratories studied the paradox created by Amdahl's law.
- Then It is the fact that parallel architectures comprised of hundreds of processors were built with **substantial improvement in performance**.
- In introducing their law, Gustafson recognised that the fraction of indivisible tasks in a given algorithm might not be known a priori.
- They argued that in practice, the problem size scales with the number of processors, n.
- Recall that Amdahl's law assumes that the amount of time spent on the parts of the program that can be done in parallel, $(1-f)$, is independent of the number of processors, n.
- Gustafson and Brasis postulated that when using a more powerful processor, the problem tends to make use of the **increased resources**.
- They found that to a first approximation the parallel part of the program, not the serial part, scales up with the problem size.
- They postulated that if s and p represent respectively the serial and the parallel time spent on a parallel system,

- Then $s + p * n$ represents the time needed by a serial processor to perform the computation.

- They therefore, introduced a new factor, called the scaled speed-up factor, $SS(n)$, which can be computed as:

$$SS(n) = \frac{s + p * n}{s + p} = s + p * n = s + (1 - s) * n = n + (1 - n) * s \quad (8)$$

- This equation shows that the resulting function is a straight line with a $slope = (1 - n)$.

- This shows clearly that it is possible, even easier, to achieve efficient parallel performance than is implied by Amdahl's speed-up formula.

- Speed-up should be measured by scaling the problem to the number of processors, not by fixing the problem size.