ECE5610/CSC6220
Introduction to Parallel and Distribution Computing

Lecture 7: Analytical Modeling of Parallel Systems (Part1)
Outline

• Sources of Overhead in Parallel Programs
• Performance Metrics for Parallel Systems
• Effect of Granularity on Performance
• Scalability of Parallel Systems
• Asymptotic Analysis of Parallel Programs
Analytical Modeling - Basics

- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).

- The asymptotic runtime of a sequential program is identical on any serial platform.

- The parallel runtime of a program depends on the input size, the number of processors, and the communication parameters of the machine.

- An algorithm must therefore be analyzed in the context of the underlying platform.

- A parallel system is a combination of a parallel algorithm and an underlying platform.
Analytical Modeling - Basics

- A number of performance measures are intuitive.

- Wall clock time - the time from the start of the first processor to the stopping time of the last processor in a parallel ensemble. But how does this scale when the number of processors is changed when the program is ported to another machine?

- How much faster is the parallel version?
  - what’s the baseline serial version with which we compare? Can we use a suboptimal serial program that is more amenable to parallel processing?

- Raw FLOP count - What good are FLOP counts when they are used to solve a problem?
Sources of Overhead in Parallel Programs

If we use two processors, shouldn’t my program run twice as fast?

No - a number of overheads, including wasted computation, communication, idling, and contention, cause degradation in performance.

![Execution Time Graph](image)

**Figure 5.1** The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.
Sources of Overheads in Parallel Programs

• Inter-process interactions: Processors working on any non-trivial parallel problem will need to talk to each other.

• Idling: Processes may idle because of load imbalance, synchronization, or serial components.

• Excess Computation: This is computation not performed by the serial version. This might be because the serial algorithm is difficult to parallelize, or that some computations are repeated across processors to minimize communication.
Performance Metrics for Parallel Systems: Execution Time

• Serial runtime of a program is the time elapsed between the beginning and the end of its execution on a sequential computer.

• The parallel runtime is the time that elapses from the moment the first processor starts to the moment the last processor finishes execution.

• We denote the serial runtime by $T_s$ and the parallel runtime by $T_p$. 
Performance Metrics for Parallel Systems: Total Parallel Overhead

- \( T_{all} = p \ T_P \) \hspace{1em} (\( p \) is the number of processors): the total time collectively spent by all the processing elements.

- Total overhead \((T_{all} - T_S)\): the total time spend by all processors combined for non-useful work.

- The overhead function: \( T_o = pT_P - T_S \)

- Speedup \((S(p) = T_s/T_p)\) is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with \( p \) identical processing elements.
Complexity of Functions and Order Analysis

- Complexity of Functions
  - **Exponential function**: Let $a$ be a real number ($a>1$), the exponential function with base $a$ is $f(x) = a^x$.
  - **Polynomial function**: let $b$ be a positive real number, the polynomial function of degree $b$ is $f(x) = x^b$.
  - **Logarithmic functions**: Let $b$ be a real number ($b>1$), the logarithmic function with base $b$ is $f(x) = \log_b x$.
  - Function $f$ is said to dominate function $g$ is $f(x)$ grows at a faster rate than $g(x)$.

- Order Analysis
  - $f(x) = \Theta(g(x))$: $f(x)$ is equivalent to $g(x)$ in the order analysis of performance
  - $f(x) = O(g(x))$: the complexity of $f(x)$ is upper-bounded by $g(x)$
  - $f(x) = \Omega(g(x))$: the complexity of $f(x)$ is lower-bounded by $g(x)$
Performance Metrics: Example

Consider the problem of adding \( n \) numbers by using \( n \) processing elements.

If \( n \) is a power of two, perform this operation in \( \log n \) steps by propagating partial sums up a logical binary tree of processors.

Figure 5.2  Computing the globalsum of 16 partial sums using 16 processing elements. \( \Sigma_i^j \) denotes the sum of numbers with consecutive labels from \( i \) to \( j \).
Performance Metrics: Example (continued)

• If an addition takes constant time $t_c$ and communication of a single word takes time $t_s$ + $t_w$, then the addition and communication operations take a constant amount of time.

  So the parallel time $T_P = \Theta (\log n)$, and sequential time $T_S = \Theta (n)$

• Speedup $S$ is given by $S = \Theta (n / \log n)$
**Performance Metrics: Speedup**

- For a given problem, there might be many serial algorithms available. These algorithms may have different runtimes and may be parallelizable to different degrees.

- For the purpose of computing speedup, we always consider the best sequential program as the baseline.

Example: parallel bubble sort.

The serial time for bubblesort is 150 seconds.

The parallel time for odd-even sort (efficient parallelization of bubble sort) is 40 seconds.

The speedup would appear to be \( \frac{150}{40} = 3.75 \).

But is this really a fair assessment of the system?

What if serial quicksort only took 30 seconds? In this case, the speedup is \( \frac{30}{40} = 0.75 \). This is a more realistic assessment of the system.
Performance Metrics: Speedup Bounds

• Speedup, in theory, should be upper bounded by $p$ - after all, we can only expect a $p$-fold speedup if we use $p$ times as many resources.

  ➢ A speedup greater than $p$ is possible only if each processing element spends less than time $T_s/p$ solving the problem.

  ➢ In this case, a single processor could be time-sliced to achieve a faster serial program, which contradicts our assumption of fastest serial program as basis for speedup.
Performance Metrics: Superlinear Speedups

• In practice, superlinear speedup exists.
• Example: The higher aggregate cache/memory bandwidth can result in better cache-hit ratios, and therefore superlinearity.

A processor with 64KB of cache yields an 80% hit ratio. If two processors are used, since the problem size/processor is smaller, the hit ratio goes up to 90%. Of the remaining 10% access, 8% come from local memory and 2% from remote memory.

If DRAM access time is 100 ns, cache access time is 2 ns, and remote memory access time is 400ns, then each processor may spend less than half of serial execution time (average access times: 2*0.8 + 100 * 0.2 = 21.6ns vs 2*0.9 + 100*0.08 + 400*0.02 = 17.8ns)!
Performance Metrics: Efficiency

- Efficiency is a measure of the fraction of time for which a processing element is usefully employed.

- Mathematically, it is given by \( E = \frac{S}{p} \).

- Following the bounds on speedup, efficiency can be as low as 0 and as high as 1.
Performance Metrics: Efficiency Example

The speedup of adding numbers on processors is given by

\[ S = \Theta \left( \frac{n}{\log n} \right) \]

Efficiency is given by

\[ E = \Theta \left( \frac{n}{\log n} \right) \]

\[ E = \Theta \left( \frac{1}{\log n} \right) \]
Parallel Time, Speedup, and Efficiency Example

Consider the problem of edge-detection in images. The problem requires to apply a $3 \times 3$ template to each pixel. If each multiply-add operation takes time $t_c$, the serial time for an $n \times n$ image is given by $T_S = 9t_c n^2$.

\[
\text{Comp: } 9t_c n^2 / p \quad \text{Comm: } 2(t_s + t_w n).
\]

**Figure 5.4** Example of edge detection: (a) an $8 \times 8$ image; (b) typical templates for detecting edges; and (c) partitioning of the image across four processors with shaded regions indicating image data that must be communicated from neighboring processors to processor 1.
Parallel Time, Speedup, and Efficiency Example (cont’d)

The total time for the algorithm is therefore given by:

$$TP = 9t_c \frac{n^2}{p} + 2(t_s + t_w n)$$

The corresponding values of speedup and efficiency are given by:

$$S = \frac{9t_c n^2}{9t_c \frac{n^2}{p} + 2(t_s + t_w n)}$$

and

$$E = \frac{1}{1 + \frac{2p(t_s + t_w n)}{9t_c n^2}}.$$
Cost of a Parallel System

- Cost is the product of parallel runtime and the number of processing elements used ($p \times T_P$).

- Cost reflects the sum of the time that each processing element spends solving the problem.

- A parallel system is said to be *cost-optimal* if the cost of solving a problem on a parallel computer is asymptotically identical to serial cost (in $\Theta$ terms).

- Since $E = T_S / (p \times T_P)$, for cost optimal systems, $E = \Theta(1)$.

- Cost is sometimes referred to as *work* or *processor-time product*. 
Cost of a Parallel System: Example

Consider the problem of adding numbers on processors.

We have, $T_p = \Theta (\log n)$ (for $p = n$).

The cost of this system is given by $p T_p = \Theta (n \log n)$.

Since the serial runtime of this operation is $\Theta(n)$, the algorithm is not cost optimal.
Effect of Granularity on Performance

• Often, using fewer processors improves efficiency of parallel systems.

• Using fewer than the maximum possible number of processing elements to execute a parallel algorithm is called *scaling down* a parallel system.

• A naive way of scaling down is to think of each processor in the original case as a virtual processor and to assign virtual processors equally to scaled down processors.

• Since the number of processing elements decreases by a factor of \( \frac{n}{p} \), the computation at each processing element increases by a factor of \( \frac{n}{p} \).

• The communication cost should not increase by this factor since some of the virtual processors assigned to a physical processors might talk to each other. This is the basic reason for the improvement from increasing granularity.
Building Granularity: Example

- Consider the problem of adding \( n \) numbers on \( p \) processing elements such that \( p < n \) and both \( n \) and \( p \) are powers of 2.
- Use the parallel algorithm for \( n \) processors. However, in this case we think of them as virtual processors.
- Each of the \( p \) processors is now assigned \( n / p \) virtual processors.
- The first \( \log p \) of the \( \log n \) steps of the original algorithm are simulated in \( (n / p) \log p \) steps on \( p \) processing elements.
- Subsequent \( \log n - \log p \) steps do not require any communication.
- The overall parallel execution time of this parallel system is

\[
\Theta ( (n / p) \log p + n/p) = \Theta ( (n / p) \log p).
\]

- The cost is \( \Theta (n \log p) \), which is asymptotically higher than the \( \Theta (n) \) cost of adding \( n \) numbers sequentially. Therefore, the parallel system is not cost-optimal.
23

(a) Four processors simulating the first communication step of 16 processors

(b) Four processors simulating the second communication step of 16 processors

Figure 5.5  Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (first two steps). \( \Sigma^j_i \) denotes the sum of numbers with consecutive labels from \( i \) to \( j \).
Figure 5.5 (continued) Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (last three steps).
Increasing Granularity: Example (continued)

Can we have a smarter assignment of data to achieve cost-optimality?

- Each processing element locally adds its $n/p$ numbers in time $\Theta(n/p)$.
- The $p$ partial sums on $p$ processing elements can be added in time $\Theta(\log p)$.

![Diagram showing the process of adding partial sums.](image)

**Figure 5.6** A cost-optimal way of computing the sum of 16 numbers using four processing elements.
Increasing Granularity: Example (continued)

The parallel runtime of this algorithm is

\[ T_P = \Theta(n/p + \log p), \]

As long as

\[ n = \Omega(p \log p) \]

The cost is

\[ \Theta(n + p \log p) \]

This is cost-optimal!