Xi'an Jiaotong-Liverpool University 西交利物浦大學

DTS205TC High Performance Computing

Lecture 8 Methodology 3

Di Zhang, Spring 2024

Analytical Modeling - Basics

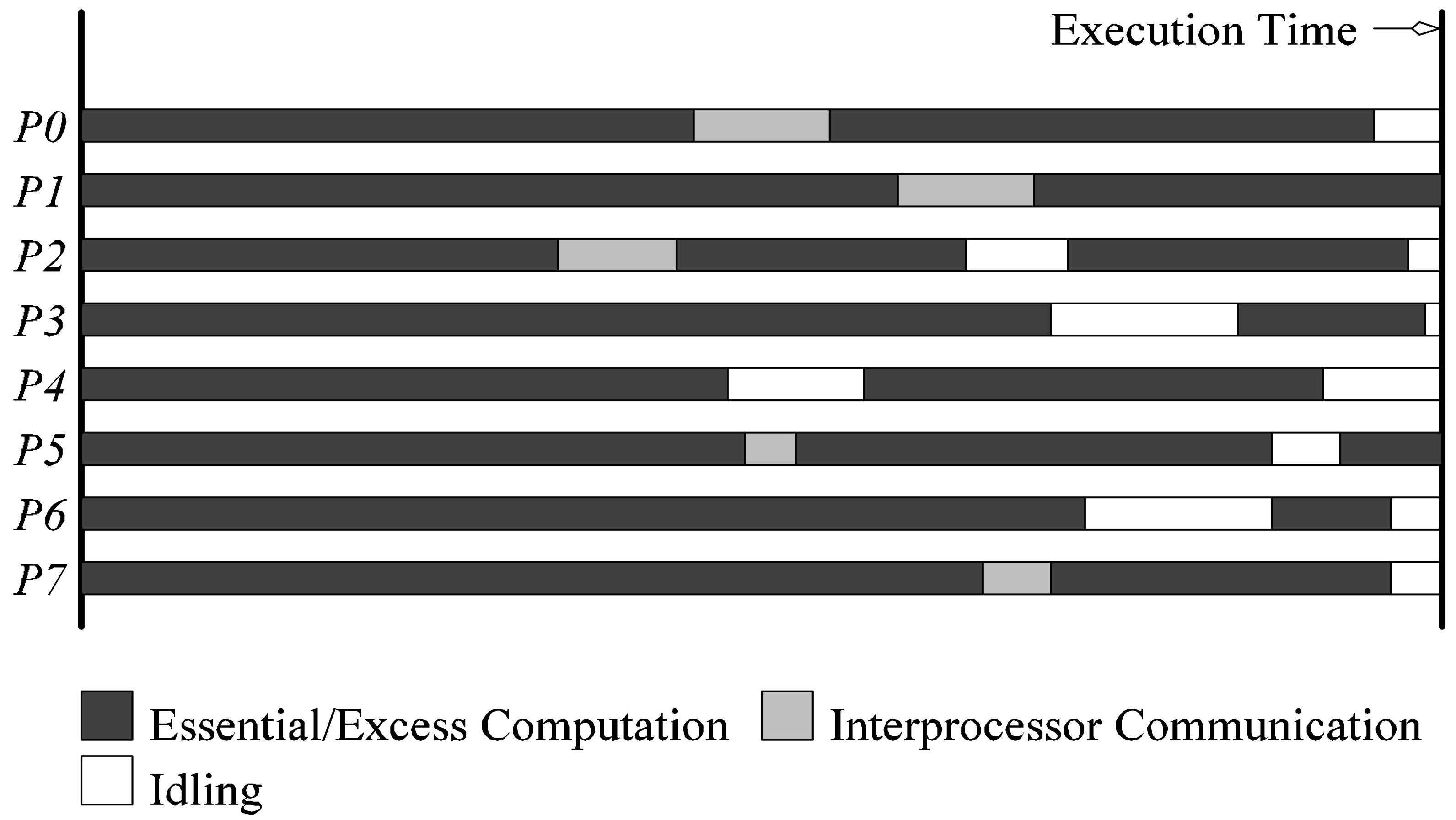
- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).
- The asymptotic runtime of a sequential program is identical on any serial platform.
- The parallel runtime of a program depends on the input size, the number of processors, and the communication 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 of the program is ported to another machine altogether?
- How much faster is the parallel version? This begs the obvious followup question - whats the baseline serial version with which we compare? Can we use a suboptimal serial program to make our parallel program look
- Raw FLOP count - What good are FLOP counts when they dont solve a problem?

Sources of Overhead in Parallel Programs

- If I use two processors, shouldnt my program run twice as fast?
- No - a number of overheads, including wasted computation, communication, idling, and contention cause degradation in performance.



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

Xi'an Jiaotong-Liverpool University 西交利物浦大學

- Interprocess 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 and the parallel runtime by T_P.

Performance Metrics for Parallel Systems: Total Parallel Overhead

- Let T_{all} be the total time collectively spent by all the processing elements.
- T_S is the serial time.
- Observe that $T_{all} - T_S$ is then the total time spend by all processors combined in non-useful work. This is called the *total overhead*.
- The total time collectively spent by all the processing elements

  $T_{all} = p\, T_P$ (*p* is the number of processors).
- The overhead function (T_o) is therefore given by

Performance Metrics for Parallel Systems: Speedup

- What is the benefit from parallelism?

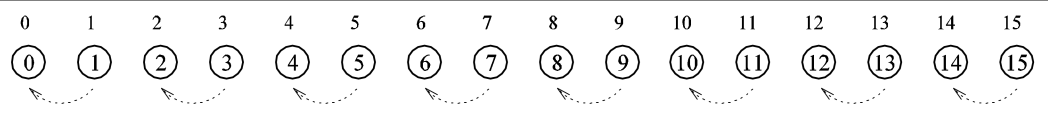
- Speedup (***S***) is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with ***p*** identical processing elements.

Performance Metrics: Example

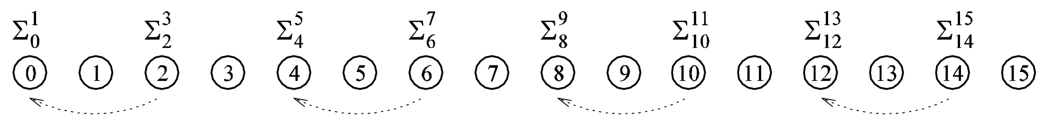
- Consider the problem of adding ***n*** numbers by using ***n*** processing elements.

- If ***n*** is a power of two, we can perform this operation in **log *n*** steps by propagating partial sums up a logical binary tree of processors.

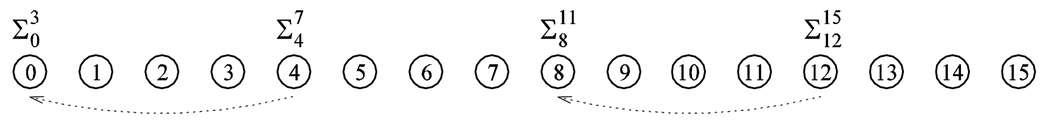
Performance Metrics: Example



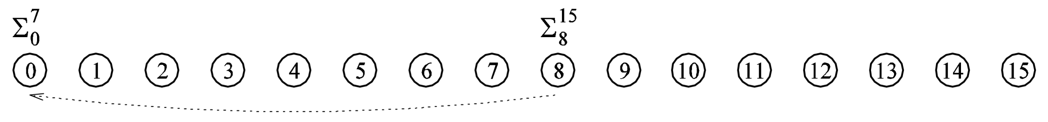
(a) Initial data distribution and the first communication step



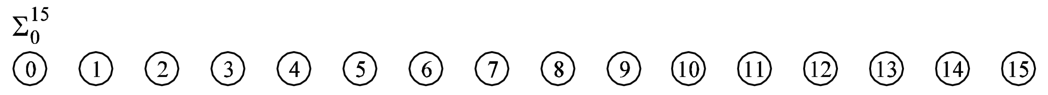
(b) Second communication step



(c) Third communication step



(d) Fourth communication step



(e) Accumulation of the sum at processing element 0 after the final communicatior

Computing the globalsum of 16 partial sums using 16 processing elements . Σ^j_i denotes the sum of numbers with consecutive labels from ***i*** to ***j***.

Performance Metrics: Example (continued)

- If an addition takes constant time, say, t_c and communication of a single word takes time $t_s + t_w$, we have the parallel time $T_P = \Theta(\log n)$
- We know that $T_S = \Theta(n)$
- 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 asymptotic runtimes and may be parallelizable to different degrees.

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

Performance Metrics: Speedup Example

- Consider the problem of 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 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 30/40 = 0.75. This is a more realistic assessment of the

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

$$E = \frac{S}{p}. \quad (2)$$

- 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

$$S = \frac{n}{\log n}$$

- Efficiency is given by E

$$\frac{\Theta\left(\frac{n}{\log n}\right)}{n}$$

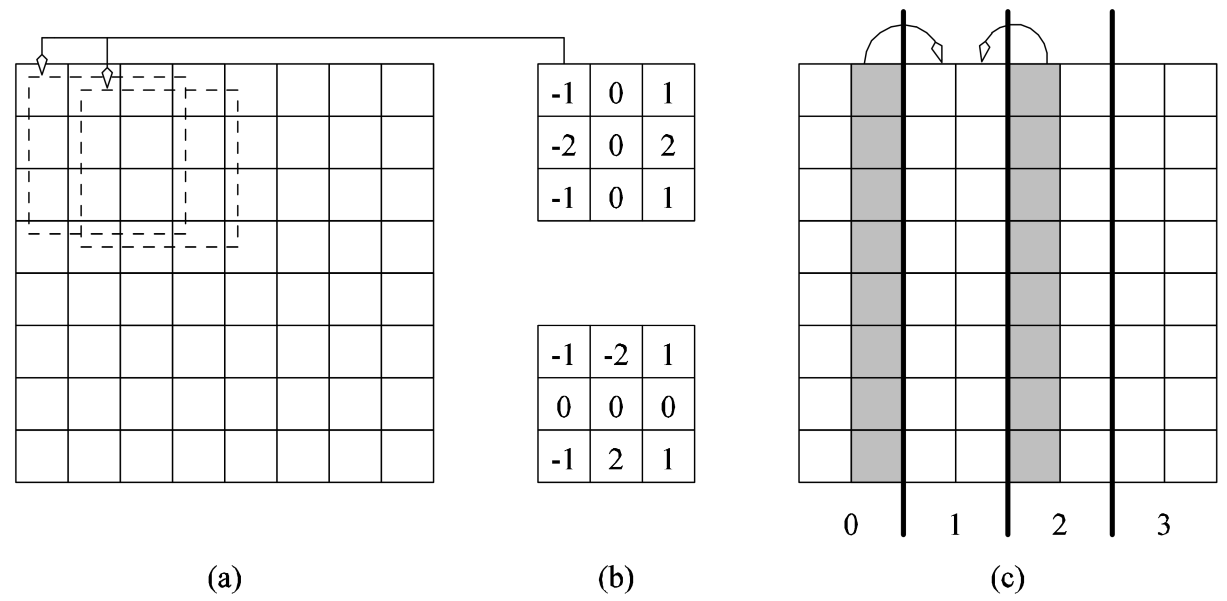
=

$$\Theta\left(\frac{1}{\log n}\right)$$

=

Parallel Time, Speedup, and Efficiency Example

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



Example of edge detection: (a) an ***8*** x ***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

Parallel Time, Speedup, and Efficiency Example (continued)

- One possible parallelization partitions the image equally into vertical segments, each with ***n*²** / ***p*** pixels.

- The boundary of each segment is ***2n*** pixels. This is also the number of pixel values that will have to be communicated. This takes time $\boldsymbol{2}(\boldsymbol{t_s} + \boldsymbol{t_w n})$.

- Templates may now be applied to all ***n*²** / ***p*** pixels in time

  $\boldsymbol{T_S = 9\, t_c n^2} / \boldsymbol{p}$.

Parallel Time, Speedup, and Efficiency Example (continued)

- The total time for the algorithm is therefore given by:

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

- 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 ($\boldsymbol{p}$ x $\boldsymbol{T_P}$).

- 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.

- Since $\boldsymbol{E} = \boldsymbol{T_S} / \boldsymbol{p}\ \boldsymbol{T_P}$, for cost optimal systems, $\boldsymbol{E} = O(1)$.

- 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, $\boldsymbol{T_P}$ = **log** $\boldsymbol{n}$ (for $\boldsymbol{p}$ = $\boldsymbol{n}$).
- The cost of this system is given by $\boldsymbol{p\ T_P}$ = $\boldsymbol{n}$ **log** $\boldsymbol{n}$.
- Since the serial runtime of this operation is $\Theta(\boldsymbol{n})$, the algorithm is not cost optimal.

Impact of Non-Cost Optimality

Consider a sorting algorithm that uses ***n*** processing elements to sort the list in time (**log** ***n***)^2.

- Since the serial runtime of a (comparison-based) sort is ***n*** **log** ***n***, the speedup and efficiency of this algorithm are given by ***n*** / **log** ***n*** and 1 / **log** ***n***, respectively.

- The ***p*** ***T_P*** product of this algorithm is ***n*** (**log** ***n***)^2.

- This algorithm is not cost optimal but only by a factor of ***log n***.

- If ***p*** < ***n***, assigning ***n*** tasks to ***p*** processors gives T_P = ***n*** (**log** ***n***)^2 / ***p*** .