Analytical Modeling of Parallel Systems

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University

Thank Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar for providing slides.

Topic Overview

- Sources of Overhead in Parallel Programs
- Performance Metrics for Parallel Systems
- Effect of Granularity on Performance
- Scalability of Parallel Systems
- Minimum Execution Time and Minimum Cost-Optimal Execution Time
- Asymptotic Analysis of Parallel Programs
- Other Scalability Metrics

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

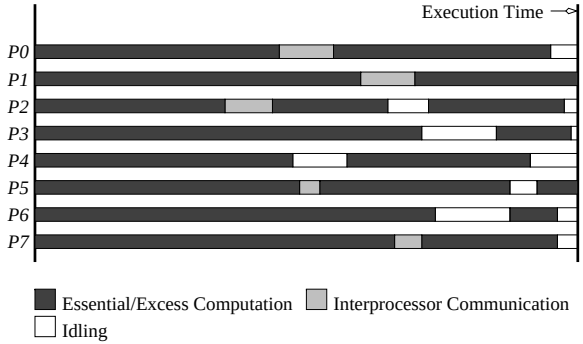
- Wall clock time – the time from the start of the first processor to the stopping time of the last processor in a parallel ensemble. But how does this scale when the number of processors is changed ?

- 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 good ?

- We need to understand if the parallel runtime scales with the number of processors being used.

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

- 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 T_S 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 $T_{all} = pT_P$ (p is the number of processors).
- The overhead function (T_o) is therefore given by

$$T_o = pT_P - T_S. \tag{1}$$

Performance Metrics for Parallel Systems: Speedup

- What is the benefit from parallelism?
- Speedup

  $$S = \frac{T_S}{T_P}$$

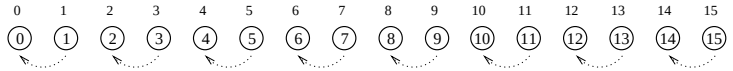
  is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processing elements.
- We expect to get big S.
- Can $S > P$? In general, No !! However, it could happen if T_S is suboptimal.
- Resource-based superlinearity: The higher aggregate cache/memory bandwidth can result in better cache-hit ratios, and therefore superlinearity.

Performance Metrics: Example

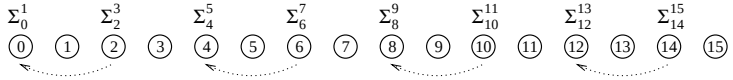
- Consider the problem of adding n numbers by using n processing elements.

- If n is a power of two, we can perform this operation in $\log_2 n$ steps by propagating partial sums up a logical binary tree of processors.

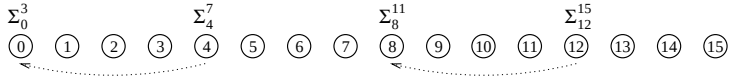
Performance Metrics: Example



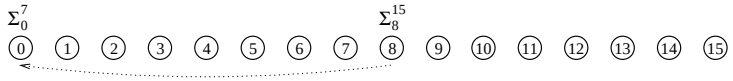
(a) Initial data distribution and the first communication step



(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 global sum of 16 partial sums using 16 processing elements. Σ_i^j 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 $T_P = (\log_2 n)(t_c + t_w) = \mathcal{O}(\log_2 n)$.
- We know that $T_S = (n-1)t_c = \mathcal{O}(n)$
- Speedup S is given by $S = \mathcal{O}\left(\frac{n}{\log_2 n}\right)$

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 system.

Performance Metrics: Speedup Bounds

- Speedup can be as low as 0 (the parallel program never terminates).
- 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 timeslided to achieve a faster serial program, which contradicts our assumption of fastest serial program as basis for speedup.

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

$$E = \frac{T_S}{pT_P}.$$

- Following the bounds on speedup, efficiency can be as low as 0 and as high as 1.

Performance Metrics: Efficiency Example

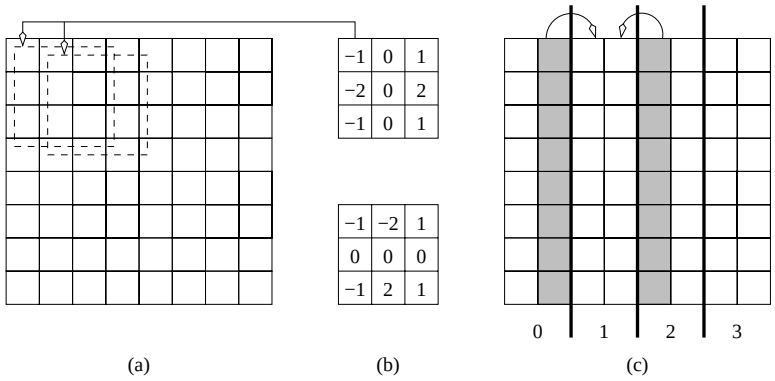
- The speedup S of adding n numbers on n processors is given by $S = \mathcal{O}\left(\frac{n}{\log_2 n}\right)$.

- Efficiency E is given by

$$\begin{aligned} E &= \mathcal{O}\left(\frac{n}{n \log_2 n}\right) \\ &= \mathcal{O}\left(\frac{1}{\log_2 n}\right). \end{aligned}$$

Parallel Time, Speedup, and Efficiency Example

Consider the problem of edge-detection in images. The problem requires us to apply a 3×3 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 $T_S = 9\,t_c n^2$.



Example of edge detection: (a) an 8×8 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 (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 $2(t_s + t_w n)$.

- Templates may now be applied to all n^2/p pixels in time $9\,t_c n^2/p$.

Parallel Time, Speedup, and Efficiency Example (continued)

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

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

- The corresponding values of speedup and efficiency are given by:

$$S = \frac{9\,t_c n^2}{9\,t_c\frac{n^2}{p} + 2(t_s + t_w n)}, \quad E = \frac{1}{1 + \frac{2\,p(t_s + t_w n)}{9\,t_c n^2}}.$$

- Keep n fixed, then
  - if $t_c \uparrow$, then E $\rightarrow$1. If $p \uparrow$, then E $\downarrow$.
- Keep p fixed, then
  - if $n \uparrow$, then E $\uparrow$.

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.
- Since $E = T_S/pT_P$, for cost optimal systems, $E = O(1)$.
- Cost is sometimes referred to as *work* or *processor-time product*.

Cost of a Parallel System: Example

Consider the problem of adding n numbers on n processors.

- We have, $T_P = \mathcal{O}(\log_2 n)$ (for $p = n$).
- The cost of this system is given by $pT_P = \mathcal{O}(n \log_2 n)$.
- Since the serial runtime of this operation is $\mathcal{O}(n)$, the algorithm is not cost optimal.
- Speedup could go down as the problem size n is increased for a given p!

Scaling Characteristics of Parallel Programs

- The efficiency of a parallel program can be written as:

$$E = \frac{S}{p} = \frac{T_S}{pT_P}$$

or

$$E = \frac{1}{1 + \frac{T_o}{T_S}}.$$

- The total overhead function T_o is an increasing function of p.

- For a given problem size (i.e., the value of T_S remains constant), as we increase the number of processing elements, T_o increases.

- The overall efficiency of the parallel program goes down. This is the case for all parallel programs.

Scaling Characteristics of Parallel Programs: Example

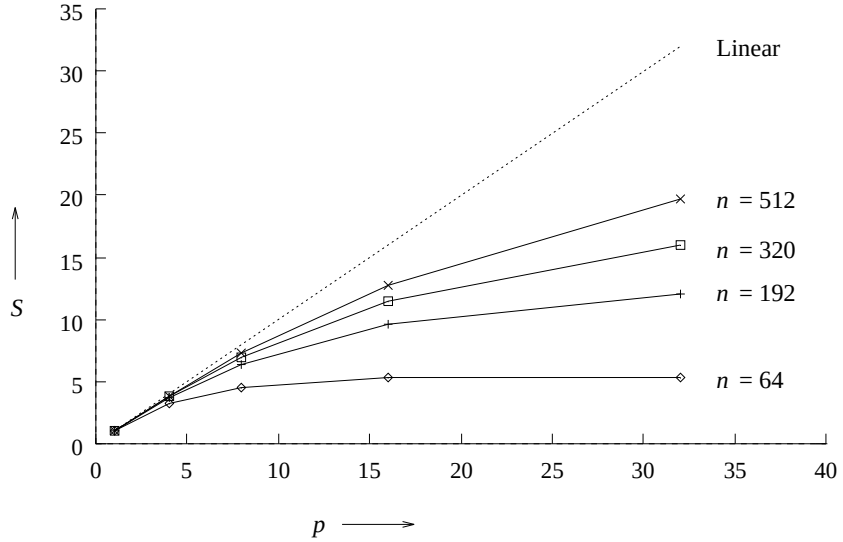
Consider the problem of adding n numbers on p processing elements.

We have seen that:

$$\begin{aligned} T_P &= \frac{n}{p} + 2\log_2 p \\ S &= \frac{n}{\frac{n}{p} + 2\log_2 p} \\ E &= \frac{1}{1 + \frac{2p\log_2 p}{n}} \end{aligned}$$

Scaling Characteristics of Parallel Programs: Example (continued)

Plotting the speedup for various input sizes gives us:



Speedup versus the number of processing elements for adding a list of n umbers.

Speedup tends to saturate and efficiency drops as a consequence of Amdahl's law.

Scaling Characteristics of Parallel Programs

- Total overhead function T_o is a function of both problem size T_S and the number of processing elements p.
- In many cases, T_o grows sublinearly with respect to T_S.
- In such cases, the efficiency increases if the problem size is increased keeping the number of processing elements constant.
- For such systems, we can simultaneously increase the problem size and number of processors to keep efficiency constant.
- We call such systems ***scalable parallel systems***.

Other Scalability Metrics

- A number of other metrics have been proposed, dictated by specific needs of applications.
- For real-time applications, the objective is to scale up a system to accomplish a task in a specified time bound.
- In memory constrained environments, metrics operate at the limit of memory and estimate performance under this problem growth rate.

Other Scalability Metrics: Scaled Speedup

- Speedup obtained when the problem size is increased linearly with the number of processing elements.

- If scaled speedup is close to linear, the system is considered scalable.

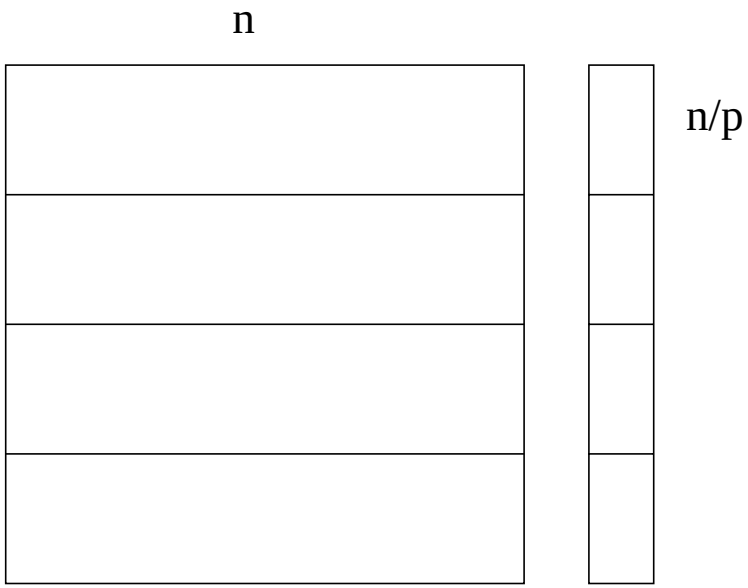
Scaled Speedup: Example

- The serial runtime of multiplying a matrix of dimension $n \times n$ with a vector is $t_c n^2$.

- For a given parallel algorithm,

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

- Total memory requirement of this algorithm is $\mathcal{O}(n^2)$.



Scaled Speedup: Example (continued)

Consider the case of memory-constrained scaling.

- We have $n^2 = p \times c$.
- Memory constrained scaled speedup is given by

$$S' = \frac{t_c\, c \times p}{t_c \frac{c \times p}{p} + t_s\, p + t_w \sqrt{c \times p}}$$

  or $S' = O(1)$.

- This is not a particularly scalable system.
- Homework: Scaled analysis.