Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Scalable Algorithms- Performance Parameters

Vassil Alexandrov, *ICREA* - **Barcelona Supercomputing Center, Spain**

Overview

- Sources of Overhead in Parallel Programs
- Performance Metrics for Parallel Systems
- Scalability of Parallel Systems and Algorithms
- Analysis of Parallel Programs

Some material from A.Grama, A. Gupta, G. Karypis, V.Kumar ``Introduction to Parallel Computing", Addison Wesley, 2003, http://www-users.cs.umn.edu/~karypis/parbook/

Parameters

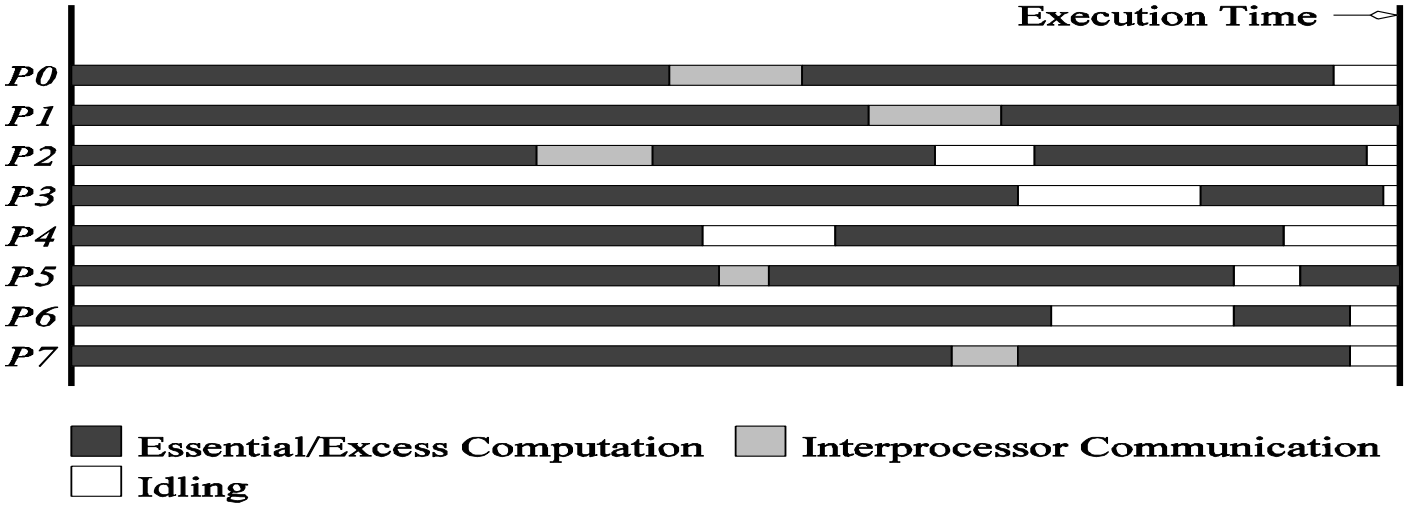
- A sequential algorithm is evaluated by its runtime (in general, asymptotic runtime as a function of input size).
- The asymptotic runtime of a sequential program is identical on any serial platform.
- The parallel runtime of a program depends on the input size, the number of processors, and the communication 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.

Parameters

- 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 follow up question – what is the baseline serial version with which we compare?

Sources of Overhead in Parallel Programs

- If I 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.



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 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

$$T_o = p\ T_P - T_S \qquad (1)$$

Performance Metrics for Parallel Systems: Speedup

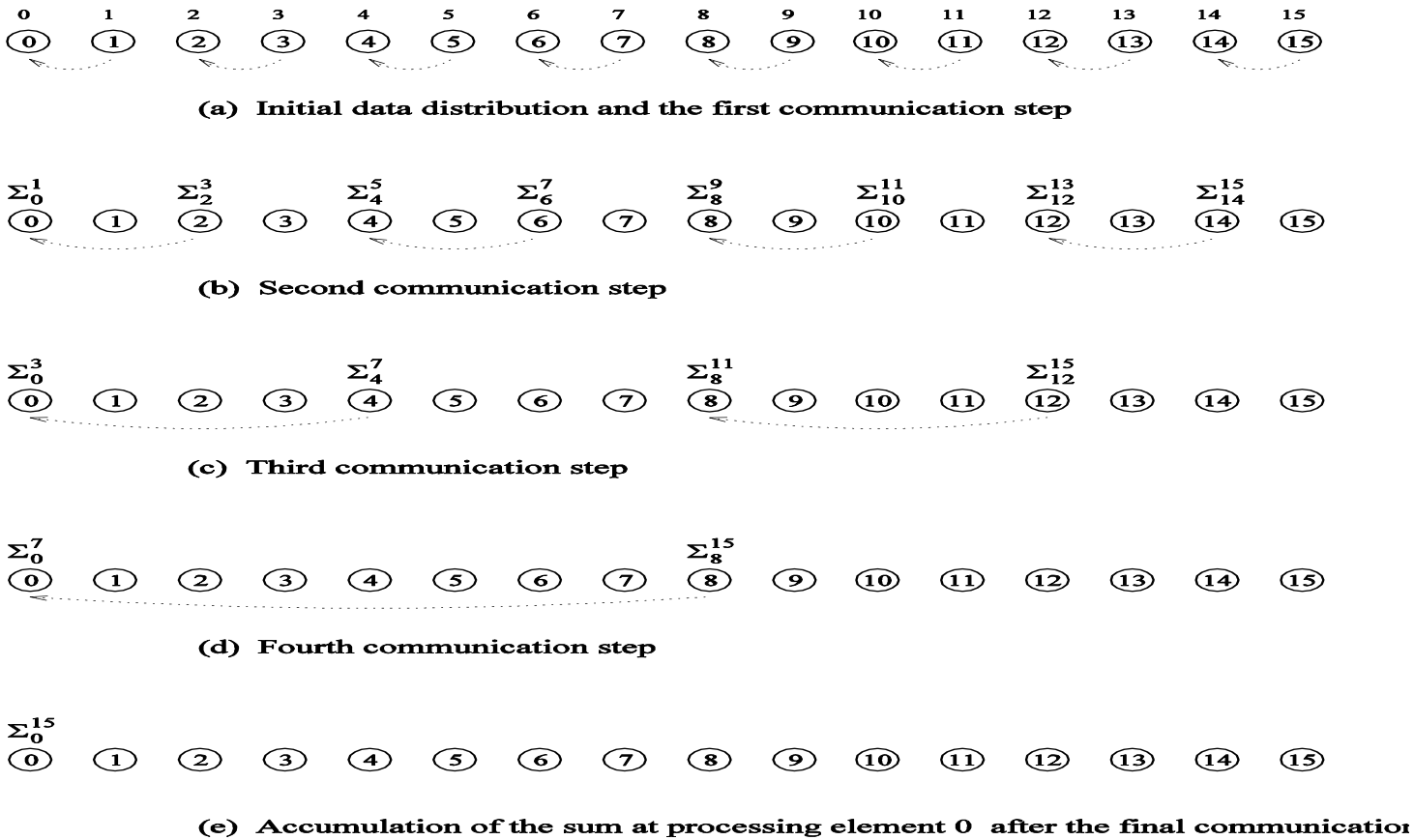
- What is the benefit from parallelism?
- Speedup (***S***) is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with ***p*** identical processing elements.
- Speedup $S_p = T_s / T_p$

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



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: 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: 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 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 times as many resources.
- Usually $0 < Sp < p$
- 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

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

- $0 < E < 1$
- 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

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

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

Effect of Granularity on Performance

- Often, using fewer processors improves performance 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 n / p, the computation at each processing element increases by a factor of 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 building granularity.

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, except, 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.

Granularity: Example (continued)

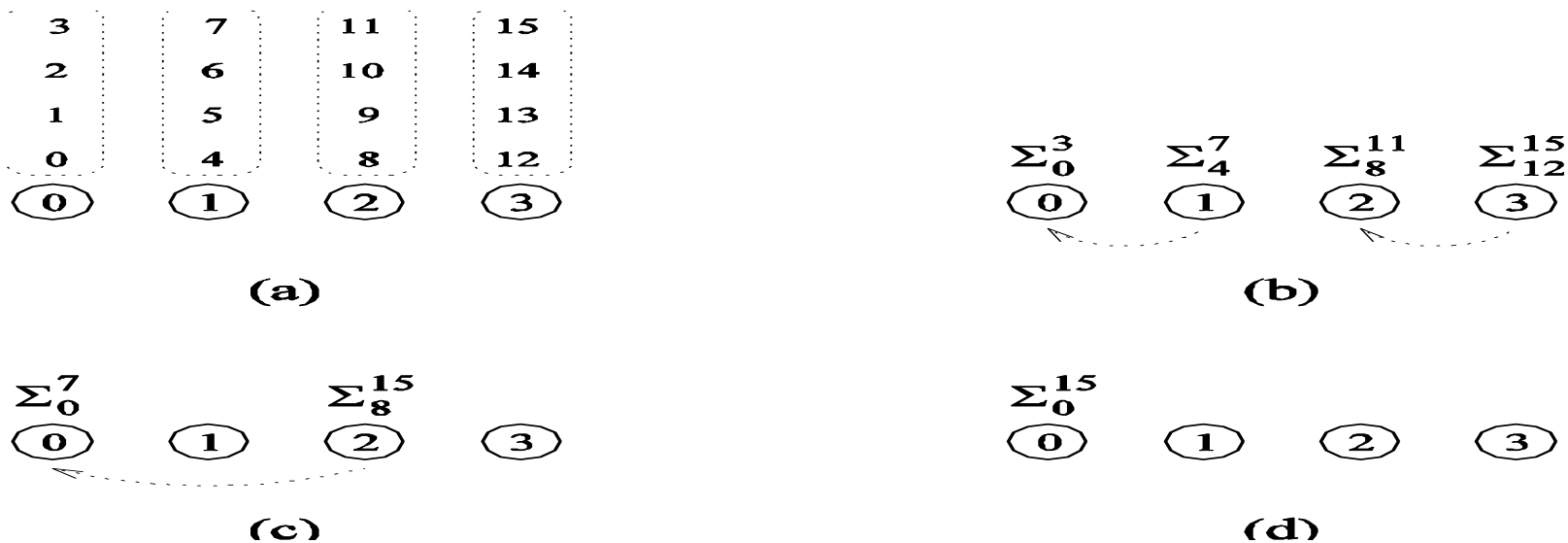
- The overall parallel execution time now is $\Theta\left((\boldsymbol{n} / \boldsymbol{p}) \log \boldsymbol{p}\right)$.

- The cost is $\Theta\left(\boldsymbol{n} \log \boldsymbol{p}\right)$, which is asymptotically higher than the $\Theta\left(\boldsymbol{n}\right)$ cost of adding $\boldsymbol{n}$ numbers sequentially. Therefore, the parallel (algorithm) system is not cost-optimal.

Granularity: Example (continued)

Can we build granularity in the example in a cost-optimal fashion?

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



A cost-optimal way of computing the sum of 16 numbers using four processing elements.

Granularity: Example (continued)

- The parallel runtime of this algorithm is

$$T_P = \Theta(n/p + \log p), \tag{3}$$

$$n = \Omega(p \log p)$$

- The cost is $\Theta(n + p \log p)$

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

$$E = \frac{1}{1 + \frac{T_o}{T_S}}. \qquad (4)$$

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

Scaling Characteristics of Parallel Programs

- 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 numbers on processing elements.
- We have seen that:

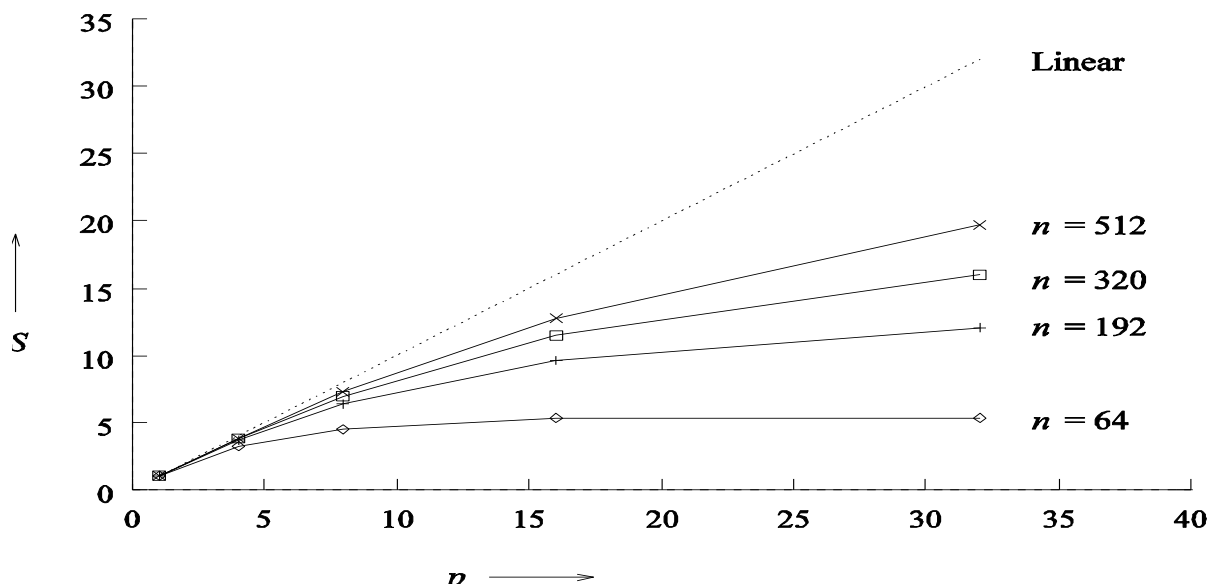
$$T_P = \frac{n}{p} + 2\log p \tag{5}$$

$$S = \frac{n}{\frac{n}{p} + 2\log p} \tag{6}$$

$$E = \frac{1}{1 + \frac{2p\log p}{n}} \tag{7}$$

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 numbers.

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.

Scalability

- For a given problem size, as we increase the number of processing elements, the overall efficiency of the parallel system goes down for all systems.
- For some systems, the efficiency of a parallel system increases if the problem size is increased while keeping the number of processing elements constant.

Scalability

- What is the rate at which the problem size must increase with respect to the number of processing elements to keep the efficiency fixed?
- This rate determines the scalability of the system. The slower this rate, the better.
- Before we formalize this rate, we define the problem size ***W*** as the asymptotic number of operations associated with the best serial algorithm to solve the problem.

Isoefficiency Metric of Scalability

- We can write parallel runtime as:

$$T_P = \frac{W + T_o(W,p)}{p} \tag{8}$$

- The resulting expression for speedup is

$$\begin{aligned} S &= \frac{W}{T_P} \\ &= \frac{Wp}{W + T_o(W,p)}. \end{aligned} \tag{9}$$

- Finally, we write the expression for efficiency as

$$\begin{aligned} E &= \frac{S}{p} \\ &= \frac{W}{W + T_o(W,p)} \\ &= \frac{1}{1 + T_o(W,p)/W}. \end{aligned}$$

Isoefficiency Metric of Scalability

- For scalable parallel systems, efficiency can be maintained at a fixed value (between 0 and 1) if the ratio T_o / W is maintained at a constant value.
- For a desired value E of efficiency,

$$\begin{aligned} E &= \frac{1}{1 + T_o(W,p)/W}, \\ \frac{T_o(W,p)}{W} &= \frac{1-E}{E}, \\ W &= \frac{E}{1-E} T_o(W,p). \end{aligned} \tag{11}$$

- If $K = E / (1 - E)$ is a constant depending on the efficiency to be maintained, since T_o is a function of W and p, we have

$$W = K T_o(W,p). \tag{12}$$

Isoefficiency Metric of Scalability

- The problem size ***W*** can usually be obtained as a function of ***p*** by algebraic manipulations to keep efficiency constant.
- This function is called the *isoefficiency function*.
- This function determines the ease with which a parallel system can maintain a constant efficiency and hence achieve speedups increasing in proportion to the number of processing elements

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.

Amdahl's Law
Fixed problem size

In many practical applications the computational workload is fixed:
Two parts for the problem with size W :
sequential and parallel part

$W = \alpha W + (1- \alpha)W$

$Sp = W / (\alpha W + (1- \alpha)(W/p))$

$= p / (1 + (p - 1) \alpha) \rightarrow 1 / \alpha$ as $p \rightarrow \infty$

Speedup is limited by $1 / \alpha$