COMS4040A: High Performance Computing & Scientific Data Management
Analytical Modelling of Parallel Systems

Hairong Wang

School of Computer Science & Applied Mathematics
University of the Witwatersrand, Johannesburg

2019-4-11

Contents

Outline

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 runtime of a parallel program depends on the input size, the number of processing elements, and the communication parameters of the machine.
- A parallel 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 finishing time of the last processor in a parallel system.
  - How does this scale when the number of processors is changed or the program is ported to another machine?
- How much faster is the parallel version?
  - This leads to the obvious question – what's the baseline serial version with which we compare?
  - Can we use a suboptimal serial program to make our parallel program look better?

Sources of Overhead in Parallel Programs

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

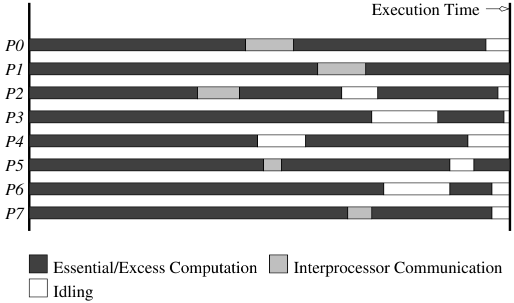


Figure: The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.

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 the 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.

Outline

Outline

Limits of Parallel Computing: Amdahl's Law

- Theoretical upper limits

  Amdahl's Law: Effect of multiple processors on speedup

$$S = \frac{T_s}{T_p} \leq \frac{1}{f_s + \frac{f_p}{P}} \tag{1}$$

- f_s: serial fraction of code
- f_p: parallel fraction of code
- P: number of processors
- S: speedup

Note: Amdahl's Law assumes that the problem size remains the same when parallelized.

Amdahl's Law Cont.

Amdahl's Law says, roughly, that unless virtually all of a serial program is parallelized, the possible speedup is going to be very limited – regardless of the number of processing elements available.

Example

Suppose, 90% of a serial program is perfectly parallelizable, and $T_{serial} = 20s$, what is the speedup achievable under Amdahl's Law?

Outline

Limits of Parallel Computing: Gustafson's Law

Gustafson's Law: Effect of multiple processors on run time of a problem with a fixed amount of parallel work per processor.

$$S \leq P - \alpha(P-1) \tag{2}$$

- α is the fraction of non-parallelized code where the parallel work per processor is fixed (not the same as f_p from Amdahl's) and can vary with P and with problem size.
- P is the number of processors
- S – scaled speedup.

Comparison of Amdahl's and Gustafson's Law

Amdahl: fixed work, $f_p = 0.5$

$$S \leq \frac{1}{f_s + f_p/N}$$

$$S_2 \leq \frac{1}{0.5 + 0.5/2} = 1.33$$

$$S_4 \leq \frac{1}{0.5 + 0.5/4} = 1.6$$

Gustafson: fixed work per processor, $\alpha = 0.5$

$$S_p \leq P - \alpha(P - 1)$$

$$S_2 \leq 2 - 0.5(2 - 1) = 1.5$$

$$S_4 \leq 4 - 0.5(4 - 1) = 2.5$$

Outline

Performance Metrics for Parallel Systems

- Execution Time:
  - Serial runtime T_s: The time that elapsed between the beginning and the end of its execution on a sequential computer.
  - Parallel runtime T_p: The time that elapses from the moment the first processor starts to the moment the last processor finishes execution.
- Total parallel overhead: The total time collectively spent by all the processing elements over and above that required by the fastest known sequential algorithm for solving the same problem on a single processing element. $T_o = pT_p - T_s$

Outline

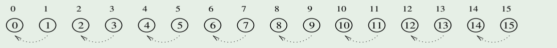
Performance Metrics for Parallel Systems: Speedup

- 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 processing elements.

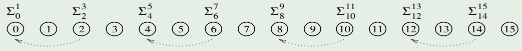
Example (1)

- 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.

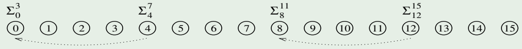
Example (1) Cont.



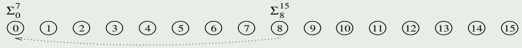
(a) Initial data distribution and the first communication step


Σ_0^1 Σ_2^3 Σ_4^5 Σ_6^7 Σ_8^9 Σ_{10}^{11} Σ_{12}^{13} Σ_{14}^{15}
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

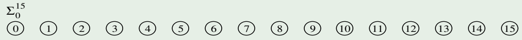
(b) Second communication step


Σ_0^3 Σ_4^7 Σ_8^{11} Σ_{12}^{15}
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

(c) Third communication step



(d) Fourth communication step



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

Figure: 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.

Example (1) Cont.

- If an addition takes constant time, say, t_c and communication of a single word takes time $t_s + t_w$ (t_s for startup time and t_w for per word transfer time) — these are constant time.
- Thus, we have the parallel time $T_p = \Theta(\log n)$
- We know that $T_s = \Theta(n)$
- Speedup S is given by $S = \Theta\left(\frac{n}{\log n}\right)$

WITS UNIVERSITY

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 should always consider the best sequential program as the baseline.

Speedup Cont.

- 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.
- Linear speedup – If $T_p = T_s/p$, we say the parallel program has linear speedup.
- 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: Superlinear Speedup

- A speedup of greater than p is known as superlinear speedup
- Resource-based superlinearity: The higher aggregate cache/memory bandwidth can result in better cache-hit ratios, and therefore superlinearity.

Example (2)

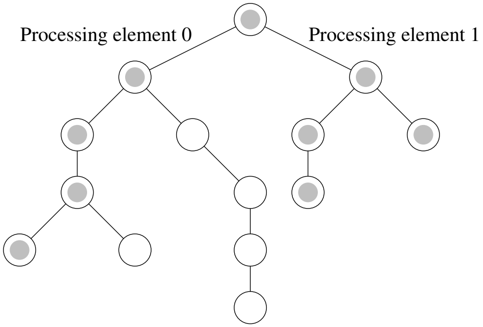
Consider the execution of a parallel program on a two processor parallel system. One processor with $64KB$ of cache yields an 80% hit ratio. If two such processors are used, since the problem size/processor is smaller, the hit ratio goes up to 90%. Of the remaining 10% access, 8% come from local memory and 2% from remote memory. If DRAM access time is $100ns$, cache access time is $2ns$, and remote memory access time is $400ns$, give an estimate of the speedup.

Superlinear Speedup Cont.

- If the parallel version does less work than the corresponding serial one, superlinearity is also possible.

Example (3)

Searching an unstructured tree using DFS in the following tree.



Outline

Performance Metrics: Efficiency

- Efficiency is a measure of processor utilization.
- Mathematically, it is given by

$$E = \frac{S}{p}. \tag{3}$$

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

Performance Metrics: Efficiency Cont.

Example (4)

- The speedup S of adding n numbers on n processors is given by $S = \frac{n}{\log n}$.
- Efficiency E is given by

$$\begin{aligned} E &= \frac{\Theta\left(\frac{n}{\log n}\right)}{n} \\ &= \Theta\left(\frac{1}{\log n}\right) \end{aligned}$$

Example (5)

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

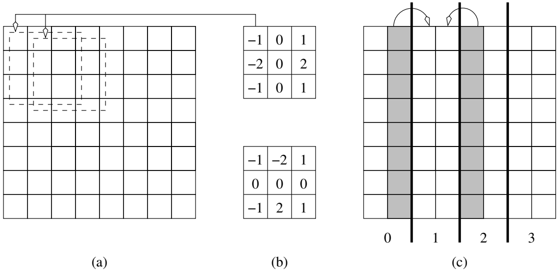


Figure: Example of edge detection: (a) an 8×8 image; (b) typical masks for detecting edges; and (c) partitioning of the image across four processors with shaded regions indicating image data that must be communicated from neighbouring processors to processor 1.

Example (5) Cont.

- 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 $2(t_s + t_w n)$.
- masks may now be applied to all n^2/p pixels in time $T_s = 9t_c n^2/p$.

Example (5) Cont.

- 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

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

Outline

Performance Metrics: The Karp-Flatt Metric

- Karp-Flatt metric — the experimentally determined serial fraction
- The time of a parallel program executing on p processes:

$$T(n,p) = T_s + T_p + T_o.$$

- The serial time for the computation:

$$T(n,1) = T_s + T_1,$$

  assuming T_1 is the portion of the computation can be parallelized; T_s is the inherently serial component of the computation, and T_o is the overhead. Note that $T_p = T_1/p$.
- Define the *experimentally determined serial fraction e*

$$e = (T_s + T_o)/T(n,1)$$

- Hence $T_s + T_o = T(n,1)e$.

The Karp-Flatt Metric Cont.

- The parallel time can be written as

$$T(n,p) = T(n,1)e + T(n,1)(1-e)/p$$

- The speedup $S = T(n,1)/T(n,p)$, and hence, $T(n,1) = T(n,p)S$,

$$\begin{aligned} T(n,p) &= T(n,p)Se + T(n,p)S(1-e)/p \\ 1 &= Se + S(1-e)/p \\ 1/S &= e + (1-e)/p \\ 1/S &= e(1-1/p) + 1/p \\ e &= \frac{1/S - 1/p}{1 - 1/p} \end{aligned}$$

The Karp-Flatt Metric Cont.

The Karp-Flatt metric — Given a parallel computation with speedup S on p processors, where $p > 1$, the experimentally determined serial fraction e is defined to be

$$e = \frac{1/S - 1/p}{1 - 1/p}$$

Example (6)

Benchmarking a parallel program on $1, 2, \ldots, 8$ processors produces the following speedup results:

p	2	3	4	5	6	7	8
S	1.82	2.50	3.08	3.57	4.00	4.38	4.71
e	0.10	0.10	0.10	0.10	0.10	0.10	0.10

What could be the primary reason for the parallel program achieving only 4.71 on 8 processors? — Limited opportunities for parallelism

The Karp-Flatt Metric Cont.

Example (7)

Benchmarking a parallel program on $1, 2, \ldots, 8$ processors produces the following speedup results:

p	2	3	4	5	6	7	8
S	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	0.070	0.075	0.080	0.085	0.090	0.095	0.10

What could be the primary reason for the parallel program achieving only 4.71 on 8 processors? — Parallel overhead.

Outline

WITS UNIVERSITY

Scalability of Parallel Systems

- How do we extrapolate performance from small problems and small systems to larger problems on larger configurations?
- Consider three parallel algorithms for computing an *n*-point Fast Fourier Transform (FFT) on 64 processing elements.
- Clearly, it is difficult to infer scaling characteristics from observations on small datasets on small machines.

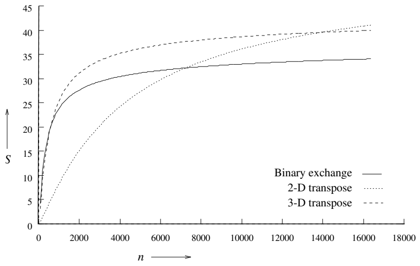


Figure: A comparison of the speedups obtained by the binary-exchange, 2-D transpose and 3-D transpose algorithms on 64 processing elements.

Outline

Scaling Characteristics of Parallel Systems

- Scalability of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processing elements. It reflects a parallel system's ability to utilize increasing resources effectively.
- The efficiency of a parallel program can be written as:

$$E = \frac{S}{p} = \frac{T_s}{pT_p}$$

or

$$E = \frac{1}{1 + \frac{T_o}{T_s}}. \tag{4}$$

- The total overhead function $T_o = pT_p - T_s$ is an increasing function of p.

Scaling Characteristics of Parallel Systems

- 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 Systems Cont.

- Plotting the speedup for various input sizes gives us:
- Speedup tends to saturate and efficiency drops.

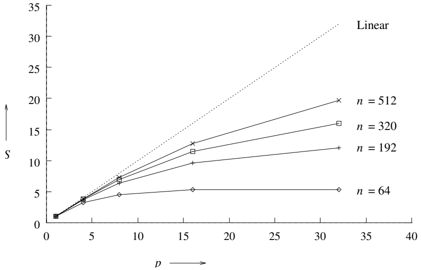


Figure: Speedup versus the number of processing elements for adding a list of numbers.

Scaling Characteristics of Parallel Systems Cont.

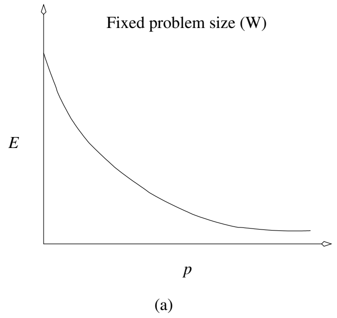
- Total overhead function T_o is a function of both problem size T_s and the number of processing elements p.
- In many cases, T_o grows sublinearly with respect to T_s.
- In such cases, the efficiency increases if the problem size is increased keeping the number of processing elements constant.
- For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

Outline

Isoefficiency Metric of 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.

Isoefficiency Metric of Scalability Cont.



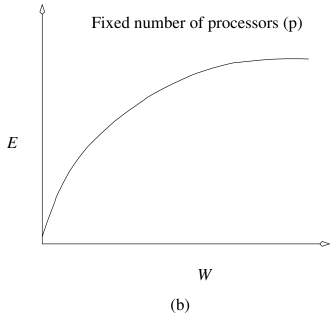


Figure: Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Isoefficiency Metric of Scalability Cont.

- 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; $T_o(W, p)$ – parallel overhead; p – number of processing elements.

Isoefficiency Metric of Scalability Cont.

- We can write parallel runtime as:

$$T_p = \frac{W + T_o(W,p)}{p} \tag{5}$$

- The resulting expression for speedup is

$$\begin{aligned} S &\leq \frac{W}{T_p} \\ &= \frac{Wp}{W + T_o(W,p)}. \end{aligned} \tag{6}$$

- Finally, we write the expression for efficiency as

$$\begin{aligned} E &\leq \frac{S}{p} \\ &= \frac{W}{W + T_o(W,p)} \\ &= \frac{1}{1 + T_o(W,p)/W}. \end{aligned} \tag{7}$$

Isoefficiency Metric of Scalability Cont.

$$\begin{aligned} E &\leq \frac{1}{1 + T_o(W,p)/W} \\ \frac{T_o(W,p)}{W} &\leq \frac{1-E}{E} \\ W &\geq \frac{E}{1-E} T_o(W,p) \end{aligned} \tag{8}$$

Let $K = E/(1-E)$ be a constant depending on the efficiency to be maintained.

$$W \geq K T_o(W,p). \tag{9}$$

- We call this function isoefficiency relation of the parallel system.
- In order to maintain the same level of efficiency as the number of processors increases, n must be increased so that the efficiency is maintained.

WITS UNIVERSITY

Example (8)

If the overhead function for the problem of adding n numbers on p processing elements is approximately $2p \log p$, what is the isoefficiency function for this parallel program?

$$W = K2p \log p.$$

That is, if p is increased from p to p', the problem size must be increased by a factor of $p' \log p' / p \log p$ to get the same efficiency as on p processing elements. That is,

$$\frac{W'}{W} = \frac{p' \log p'}{p \log p}$$

Isoefficiency Metric of Scalability Cont.

Example (9)

Consider the edge detection problem again. For the problem of $n \times n$ grid, we decompose the problem into subgrids of $n/\sqrt{p} \times n/\sqrt{p}$.
During each iteration of the algorithm every processor sends boundary values to its 4 neighbours; the time for these communications is $\Theta(n/\sqrt{p})$ each iteration.
The isoefficiency function for this parallel system is

$$n^2 \geq Kp(n/\sqrt{p}) \Rightarrow n \geq K\sqrt{p}.$$

Scalability: Strong vs Weak

- We want to know how quickly we can complete analysis on a particular data set by increasing the processing element count
- Or, equivalently, when we increase the number of processes/threads, we can keep the efficiency fixed without increasing the problem size
  - Amdahl's Law
  - Known as strongly scalable
- We want to know if we can analyze more data in approximately the same amount of time by increasing the processing element count
- Or, equivalently, we can keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads
  - Gustafson's Law
  - Known as weakly scalable

References

- A. Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing, 2nd Edition, Chapter 5.
- Quinn's book, Chapter 7.