Measuring Speedup

Chapter 8

Performance Measurement Metrics

Problem size

A program’s problem size, N, is the number of computations the program performs to solve that problem.

For an image-processing problem, the problem size might be the number of pixels in the image. For an n × n-pixel image, the problem size would be $N = n^2$.

For the AES key search problem, the problem size is the number of keys tested: $N = 2^n$, where n is the number of missing key bits.

In general, the problem size is defined so that the amount of computation is proportional to N.

Running time

A program’s running time, T, is the amount of time the program takes to compute the answer to a problem.

Many factors influence T:

- Hardware characteristics—CPU speed, memory speed, caches, etc.
- the algorithm used to solve the problem.
- how the program is implemented; the same algorithm can sometimes run faster when coded differently.

When Comparing the program performances

- always run the programs on the same hardware
- the only factors that influence T must be N and K. K is the number of processors.

Running Time Notation: T(N,K)

the notation T(N,K) is used to emphasize that the running time is a function of the problem size and the number of processors.

Parallel and Sequential versions of Running Time

- $T_{seq}(N,K)$: Sequential Running Time
- $T_{\mathbf{par}}(N,K)$: Parallel Running Time

Speedup

A program’s speedup is the rate at which the parallel version of a program runs faster than the sequential version.

$$\mathrm{Speedup}(\mathrm{N},\mathrm{K}) = \frac{Tseq(N,K)}{Tpar(N,K)} \qquad \text{(eq 1)}$$

Ideal Speedup (Linear Speedup)

Ideally, a parallel program should run twice as fast on two processors as on one processor, three times as fast on three processors, four times as fast on four processors, and so on. Thus, ideally, Speedup should equal K.

Efficiency

Real parallel programs usually fall short of ideal speedup. Efficiency is a metric that captures how close to ideal a program's speedup is:

$$Eff(N, K) = \frac{Speedup(N,K)}{K} \qquad \text{(eq 2)}$$

An ideal parallel program has an efficiency of 1 for all problem sizes N and all numbers of processors K.

A real parallel program typically has an efficiency less than 1, so that the speedup is less than K—a sublinear speedup.

Amdahl's Law

Key insight: A certain portion of any program must be executed sequentially.

This portion consists of initialization, cleanup, thread synchronization, maybe some I/O, etc.

Single processor execution: The sequential portion can use only one processor, no matter how many processors are available in the parallel computer.

Upper bound on speedup: Consequently, there is an upper bound on the speedup a parallel program can achieve.

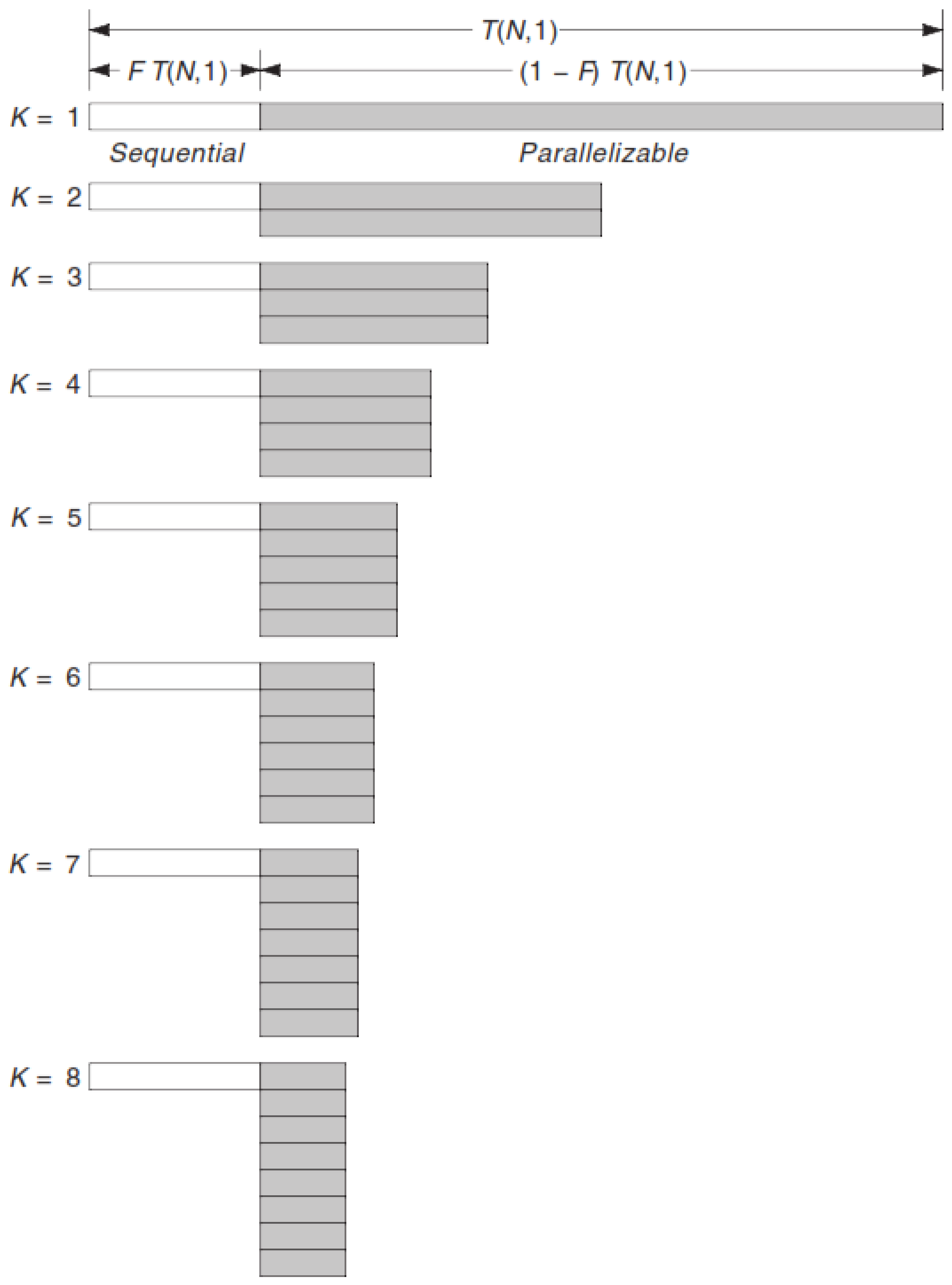


Figure 8.1 A parallel program with running time $T(N,1)$ and sequential fraction F executing with different numbers of processors K

Sequential fraction: F ($0 \le F \le 1$)

Let the sequential fraction F, where $0 \le F \le 1$, be the fraction of a program that must be executed sequentially.

If a program's running time on a single processor is T(N,1):

- F·T(N,1): the sequential portion of the running time
- (1–F)·T(N,1): parallel portion of the running time

If parallel portion is split equally among K processors (Figure 8.1), then the parallel program's running time is:

$$T(N,K) = F\cdot T(N,1) + \frac{1}{K}(1-F)\cdot T(N,1) \qquad \text{(eq 3)}$$

Speedup and efficiency with sequential portion

we can derive equations for speedup and efficiency as a function of the sequential fraction:

$$\text{Speedup}(N,K) = \frac{T(N,1)}{T(N,K)} = \frac{T(N,1)}{F\cdot T(N,1) + \frac{1}{K}(1-F)\cdot T(N,1)} = \frac{1}{F + \frac{1-F}{K}} \qquad \text{(eq 4)}$$

$$\text{Eff}(N,K) = \frac{Speedup(N,K)}{K} = \frac{1}{KF + 1-F} \qquad \text{(eq 5)}$$

Speedup On the Limit

If the number of processors (K) goes to infinity, speedup goes to 1/F.

At most we can get 1/F speedup.

Efficiency On the Limit

If the number of processors (K) goes to infinity, efficiency goes to 0.

As we add more processors, efficiency just gets worse.

Speedup and Efficieny Graphs

Speedup and Efficieny curves for various sequential fraction values:

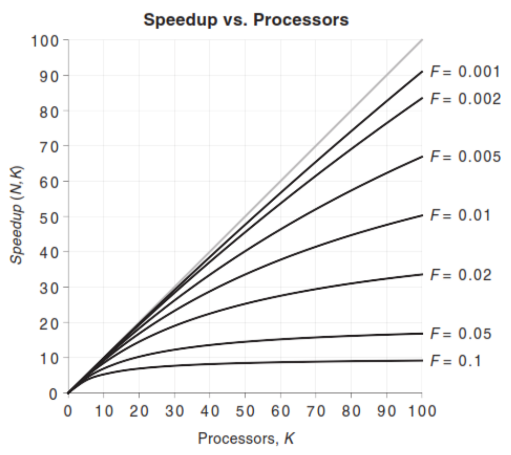
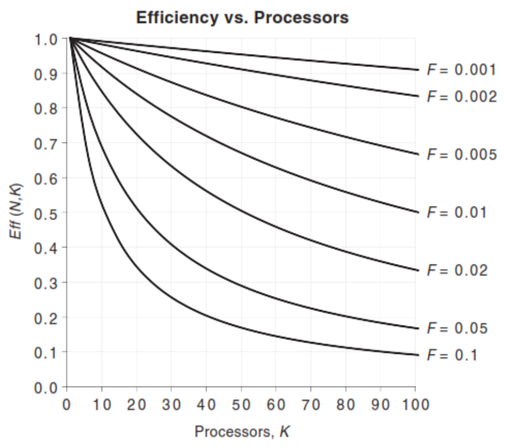


Figure 8.2 Speedup predicted by Amdahl's Law



8.3 Efficiency predicted by Amdahl's Law

Experimentally determined sequential fraction (EDSF)

We can estimate the sequential fraction of a program by rearranging Eq 3:

$$F = \frac{K \cdot \mathrm{T}(\mathrm{N},\mathrm{K}) - \mathrm{T}(\mathrm{N},1)}{K \cdot \mathrm{T}(\mathrm{N},1) - \mathrm{T}(\mathrm{N},1)} \qquad \text{(Eq 6)}$$

When F is determined from the data in this way, it is called the experimentally determined sequential fraction, EDSF.

Expected EDFS Behaviour: If the program is behaving as shown in Figure 8.1. A sequential portion with a fixed running time plus a parallel portion with a running time inversely proportional to K—then F should be a constant, and a plot of EDSF versus K should be a horizontal line.

If the measured EDSF plot does not show up as roughly a horizontal line, it's another indication that something is going on that might require changing the program's design.

Measuring Running Time

Different Running Times: Multiple runs of a program produces different running times. Which one should we use?

How about using average running time: First thought is to take several readings and use their average. However, this is not the proper procedure for measuring a program's running time.

Not a zero-mean Gaussian Distribution for errors: when measuring a program's wall-clock running time, the measurement error distribution is not a zero-mean Gaussian.

Averaging only makes sense when: Taking the average of a series of measurements gives a close approximation to the true value only when the errors obey a Gaussian probability distribution with a mean of zero as in Fig 8.5.

Symmetric errors: A Gaussian distribution is symmetric; positive and negative errors both are equally possible.

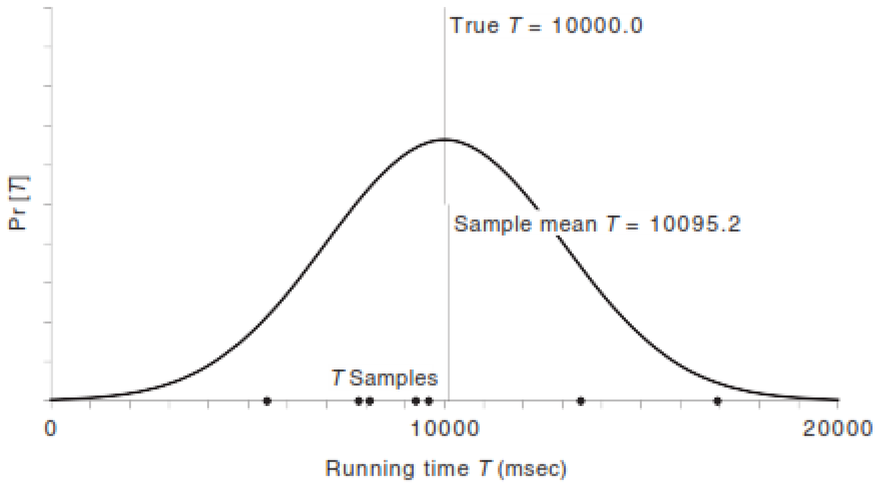


Figure 8.5 Running-time measurement with a Gaussian error distribution

Running time errors are not symmetric: the chief source of errors in the program's wall-clock running time does not have a symmetric distribution. When the operating system takes the CPU away from the parallel program to respond to an interrupt or to let another process run, the parallel program's running-time measurement can only increase, never decrease.

Running time errors are only positive: The measurement-error probability distribution ends up looking more like Figure 8.6, which shows that the sample mean always ends up being larger than the true value.

Min value is best estimator: When the measurement error is always positive, the best estimator of the true value is the minimum of the measurements, not the average.

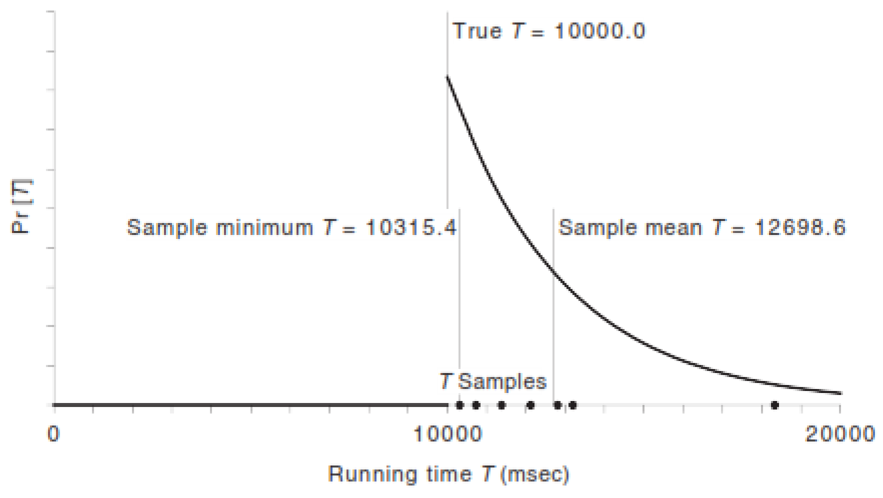


Figure 8.6 Running time measurement with a positive error distribution

The steps to measure Running Times

Here is the procedure to measure the running times:

1. Ensure that the parallel program is the only user process running. Don't let other users log in while timing measurements are being made. Don't run other programs such as editors, Web browsers, or e-mail clients.
2. To the extent possible, don't have any server or daemon processes running, such as Web servers, e-mail servers, file servers, network time daemons, and so on.
3. Prepare several input data sets, covering a range of problem sizes N. Choose the smallest problem size so that $T_{seq}(N,1)$ is at least 60 seconds.
4. For each N (each input data set), run the sequential version of the program seven times. (The number of runs, seven, is just an arbitrary choice.) Take the smallest of the running times as the measured $T_{seq}(N,1)$ value.
5. For each N, and for each K from 1 up to the number of available processors, run the parallel version of the program seven times. Take the smallest of the running times as the measured $T_{par}(N,K)$ value.

FindKeySmp Running Time Measurements

The six input data sets had from n = 24 to 29 missing key bits, yielding problem sizes of N = 16M, 32M, 64M, 128M, 256M, and 512M encryption keys tested. ("M" stands for 2^{20}).

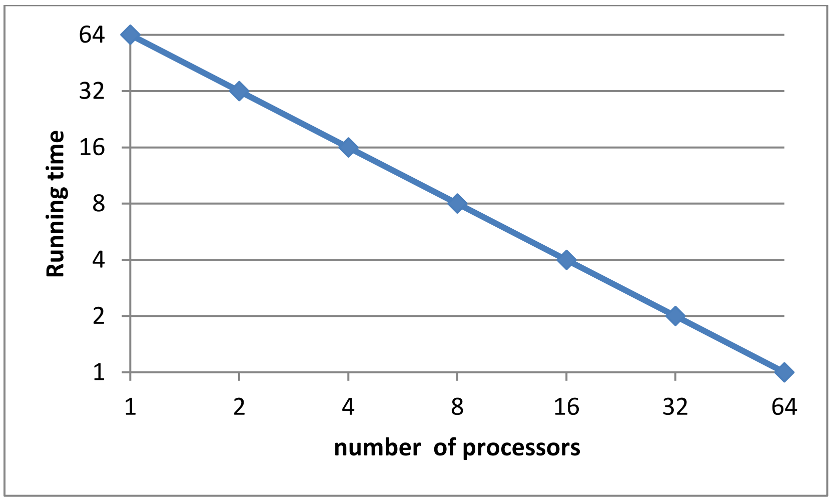
Running time vs the number of processor plots.

Log-log plot is better suited for plotting running times vs the number of processors.

If it is an ideal speedup, then the running time vs processors plot will be a linear with a slope of -1.

Ex:

running time	processors	Log_2(running time)	Log_2(processors)
64	1	6	0
32	2	5	1
16	4	4	2
8	8	3	3
4	16	2	4
2	32	1	5
1	64	0	6



Running Time plot for key search from the book:

Both axises are in log scale. Some of them are not straight lines. Particularly when the number of processors approach 8.

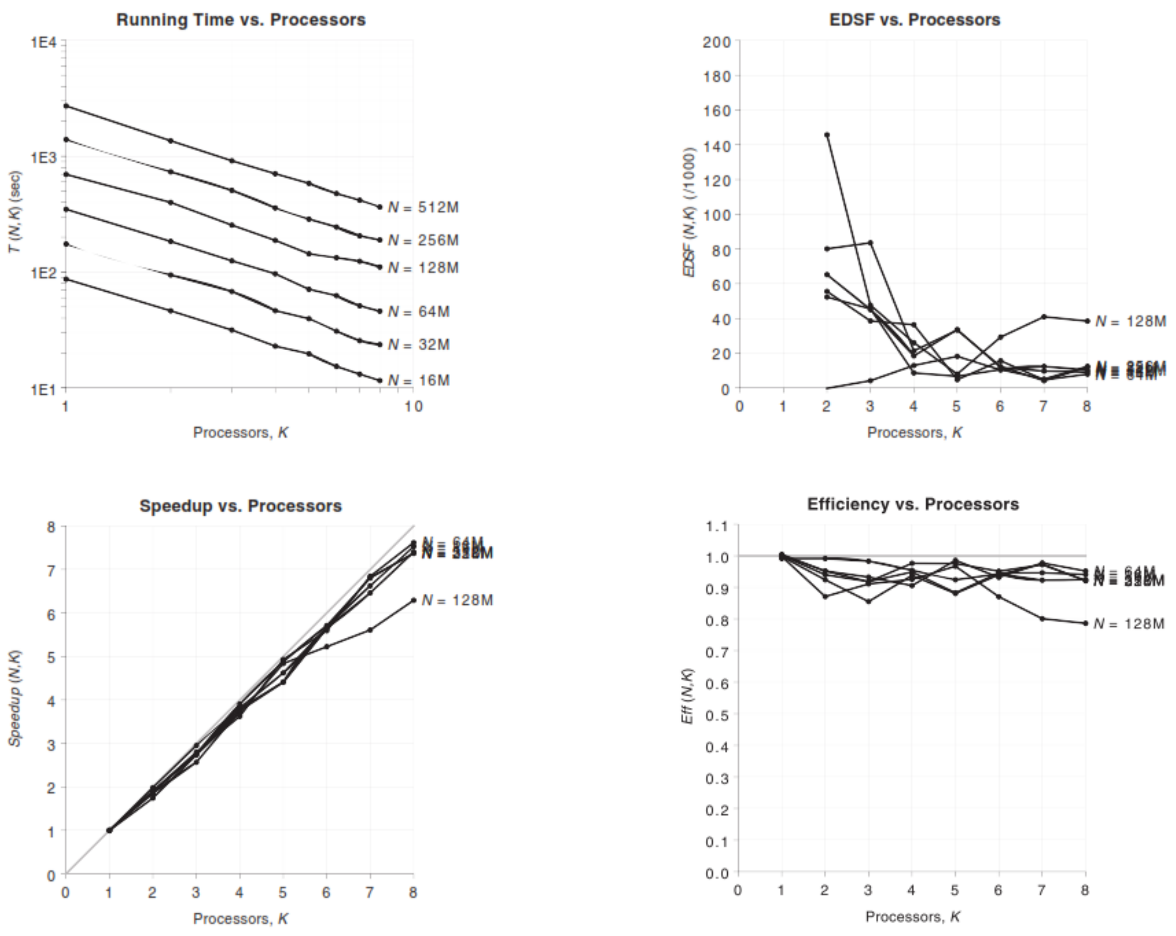


Figure 8.7 FindKeySeq/FindKeySmp running-time metrics

Speedup, Efficiency and EDSF plots for key search from the book:

All speedup, efficiency and EDSF plots fluctuate when the number of processors increase.