PERFORMANCE

Speedup and efficiency

Usually the best we can hope to do is to equally divide the work among the cores,while at the same time introducing no additional work for the cores. If we succeedin doing this, and we run our program with p cores, one thread or process on eachcore, then our parallel program will run p times faster than the serial program. If wecall the serial run-time Tserial and our parallel run-time Tparallel, then the best we canhope for is Tparallel=Tserial/p. When this happens, we say that our parallel programhas linear speedup.

In practice, we're unlikely to get linear speedup because the use of multiple processes/threads almost invariably introduces some overhead. For example, sharedmemoryprograms will almost always have critical sections, which will requirethat we use some mutual exclusion mechanism such as a mutex. The calls to themutex functions are overhead that's not present in the serial program, and the use ofthe mutex forces the parallel program to serialize execution of the critical section.

Distributed-memory programs will almost always need to transmit data across thenetwork, which is usually much slower than local memory access. Serial programs,on the other hand, won't have these overheads. Thus, it will be very unusual for usto find that our parallel programs get linear speedup. Furthermore, it's likely that theoverheads will increase as we increase the number of processes or threads, that is,more threads will probably mean more threads need to access a critical section. Moreprocesses will probably mean more data needs to be transmitted across the network.So if we define the speedup of a parallel program to be

$$S = T_{serial}/T_{parallel}$$

- Then linear speedup has S = p, which is unusual. Furthermore, as p increases, we expect S to become a smaller and smaller fraction of the ideal, linear speedup p.Another way of saying this is that S/p will probably get smaller and smaller as pincreases.

- This value, S/p, is sometimes called the efficiency of the parallel program. If we substitute the formula for S, we see that the efficiency is

$$E = \frac{S}{p} = \frac{\left(\frac{T_{serial}}{T_{parallel}}\right)}{p} = \frac{T_{serial}}{p \cdot T_{parallel}}.$$

Many parallel programs are developed by dividingthe work of the serial program among the processes/threads and adding in the necessary"parallel overhead" such as mutual exclusion or communication. Therefore, ifToverhead denotes this parallel overhead, it's often the case that

Tparallel=Tserial/pCToverhead

Amdahl's Law

The performance gain that can be obtained by improving some portion of a computercan be calculated using Amdahl's Law. Amdahl's Law states that the performanceimprovement to be gained from using some faster mode of execution islimited by the fraction of the time the faster mode can be used.

Amdahl's Law defines the speedup that can be gained by using a particularfeature. What is speedup? Suppose that we can make an enhancement to a machinethat will improve performance when it is used. Speedup is the ratio

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

it can also be expressed as

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Speedup tells us how much faster a task will run using the machine with the enhancementas opposed to the original machine.Amdahl's Law gives us a quick way to find the speedup from some enhancement,which depends on two factors:

1. The fraction of the computation time in the original machine that can be

converted to take advantage of the enhancement—For example, if 20seconds of the execution time of a program that takes 60 seconds in total canuse an enhancement, the fraction is 20/60. This value, which we will callfractionenhanced, is always less than or equal to 1.

2. The improvement gained by the enhanced execution mode; that is, how much

faster the task would run if the enhanced mode were used for the entire program—

This value is the time of the original mode over the time of the enhancedmode: If the enhanced mode takes 2 seconds for some portion of theprogram that can completely use the mode, while the original mode took 5 secondsfor the same portion, the improvement is 5/2. We will call this value,which is always greater than 1, Speedupenhanced.

The execution time using the original machine with the enhanced mode will be

the time spent using the unenhanced portion of the machine plus the time spent

using the enhancement:

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Scalability

- The word “scalable” has a wide variety of informal uses. Roughly speaking, a technology is scalable if it can handleever-increasing problem sizes.
- Suppose we run a parallelprogram with a fixed number of processes/threads and a fixed input size, and weobtain an efficiency E.
- Suppose we now increase the number of processes/threadsthat are used by the program. If we can find a corresponding rate of increase inthe problem size so that the program always has efficiency E, then the program isscalable.
- As an example, suppose that Tserial D n, where the units of Tserial are in microseconds,and n is also the problem size. Also suppose that Tparallel D n=pC1. Then

$$E = \frac{n}{p(n/p+1)} = \frac{n}{n+p}.$$

- If the program is scalable, we increase the number of processes/threads by a factor of k, and we want to find the factor x that we need to increase the problemsize by so that E is unchanged. The number of processes/threads will be kp and theproblem size will be xn, and we want to solve the following equation for x:

$$E = \frac{n}{n+p} = \frac{xn}{xn+kp}.$$

- Well, if x D k, there will be a common factor of k in the denominatorXn+kp=kn+kp= k.(n+p), and we can reduce the fraction to get

$$\frac{xn}{xn+kp} = \frac{kn}{k(n+p)} = \frac{n}{n+p}. \frac{xn}{xn+kp} = \frac{kn}{k(n+p)} = \frac{n}{n+p}.$$

- In other words, if we increase the problem size at the same rate that we increase thenumber of processes/threads, then the efficiency will be unchanged, and our programis scalable.

PARALLEL PROGRAM DESIGN

- If we want to parallelize a serial program., we need to divide the work among the processes/threads so that each process getsroughly the same amount of work and communication is minimized. In most cases,we also need to arrange for the processes/threads to synchronize and communicate.

- Unfortunately, there isn't some mechanical process we can follow; if there were, wecould write a program that would convert any serial program into a parallel program,but, as we noted in Chapter 1, in spite of a tremendous amount of work and someprogress, this seems to be a problem that has no universal solution.

- However, Ian Foster provides an outline of steps:

1. Partitioning. Divide the computation to be performed and the data operated on bythe computation into small tasks. The focus here should be on identifying tasksthat can be executed in parallel.
2. Communication. Determine what communication needs to be carried out amongthe tasks identified in the previous step.
3. Agglomeration or aggregation. Combine tasks and communications identified inthe first step into larger tasks. For example, if task A must be

executed before taskB can be executed, it may make sense to aggregate them into a single compositetask.

4. Mapping. Assign the composite tasks identified in the previous step to processes/threads. This should be done so that communication is minimized, and eachprocess/thread gets roughly the same amount of work.

This is sometimes called Foster’s methodology.