Predicting and Measuring Parallel Performance

<keywords: threading, performance, Amdahl, Gustafson, efficiency, speedup>

Abstract

Building parallel versions of software can enable applications to run a given data set in less time, run multiple data sets in a fixed amount of time, or run large-scale data sets that are prohibitive with unthreaded software. The success of parallelization is typically quantified by measuring the speedup of the parallel version relative to the serial version. In addition to that comparison, however, it is also useful to compare that speedup relative to the upper limit of the potential speedup. That issue can be addressed using Amdahl's Law and Gustafson's Law.

This article is part of the larger series, "The Intel Guide for Developing Multithreaded Applications which provides guidelines for developing efficient multithreaded applications for Intel® platforms.

Background

The faster an application runs, the less time a user will need to wait for results. Shorter execution time also enables users to run larger data sets (e.g., a larger number of data records, more pixels, or a bigger physical model) in an acceptable amount of time. One computed number that offers a tangible comparison of serial and parallel execution time is *speedup*.

Simply stated, speedup is the ratio of serial execution time to parallel execution time. For example, if the serial application executes in 6720 seconds and a corresponding parallel application runs in 126.7 seconds (using 64 threads and cores), the speedup of the parallel application is 53X (6720/126.7 = 53.038).

For an application that scales well, the speedup should increase at or close to the same rate as the increase in the number of cores (threads). When increasing the number of threads used, if measured speedups fail to keep up, level out, or begin to go down, the application doesn't scale well on the data sets being measured. If the data sets are typical of actual data sets on which the application will be executed, the application won't scale well.

Related to speedup is the metric of *efficiency*. While speedup is a metric to determine how much faster parallel execution is versus serial execution, efficiency indicates how well software utilizes the computational resources of the system. To calculate the efficiency of parallel execution, take the observed speedup and divide by the number of cores used. This number is then expressed as a percentage. For example, a 53X speedup on 64 cores equates to an efficiency of 82.8% (53/64 = 0.828). This means that, on average, over the course of the execution, each of the cores is idle about 17% of the time.

Amdahl's Law

Before starting a parallelization project, developers may wish to estimate the amount of performance increase (speedup) that they can realize. If the percentage of serial code execution that could be executed in parallel is known (or estimated), one can use Amdahl's Law to compute an upper bound on the speedup of an application without actually writing any concurrent code. Several variations of the Amdahl's Law formula have been put forth in the literature. Each uses the percentage of (proposed) parallel execution time (*pctPar*), serial execution time ($1 - pctPar$), and the number of threads/cores (p). A simple formulation of Amdahl's Law to estimate speedup of a parallel application on p cores is given here:

$$Speedup \leq \frac{1}{(1 - pctPar) + \frac{pctPar}{p}}.$$

The formula is simply the serial time, normalized to 1, divided by the estimated parallel execution time, using percentages of the normalized serial time. The parallel execution time is estimated to be the percentage of serial execution (*1 – pctPar*) and the percentage of execution that can be run in parallel divided by the number of cores to be used (*pctPar/p*). For example, if 95% of a serial application's run time could be executed in parallel on eight cores, the estimated speedup, according to Amdahl's Law, could as much 6X (1 / (0.05 + 0.95/8) = 5.925).

In addition to the less than or equal relation (≤) in the formula, the formulations of Amdahl's Law assume that those computations that can be executed in parallel will be divisible by an infinite number of cores. Such an assumption effectively removes the second term in the denominator, which means that the most speedup possible is simply the inverse of the percentage of remaining serial execution.

Amdahl's Law has been criticized for ignoring real-world overheads such as communication, synchronization, and other thread management, as well as the assumption of infinite-core processors. In addition to not taking into account the overheads inherent in concurrent algorithms, one of the strongest criticisms of Amdahl's Law is that as the number of cores increases, the amount of data handled is likely to increase as well. Amdahl's Law assumes a fixed data set size for whatever number of cores is used and that the percentage of overall serial execution time will remain the same.

Gustafson's Law

If a parallel application using eight cores were able to compute a data set that was eight times the size of the original, does the execution time of the serial portion increase? Even if it does, it does not grow in the same proportion as the data set. Real-world data suggests that the serial execution time will remain almost constant.

Gustafson's Law, also known as *scaled speedup,* takes into account an increase in the data size in proportion to the increase in the number of cores and computes the (upper bound) speedup of the application, as if the larger data set could be executed in serial. The formula for scaled speedup is as follows:

$$Speedup \leq p + (1 - p)s.$$

As with the formula for Amdahl's Law, *p* is the number of cores. To simplify the notation, *s* is the percentage of serial execution time in the parallel application for a given data set size. For example, if 1% of execution time on 32 cores will be spent in serial execution, the speedup of this application over the same data set being run on a single core with a single thread (assuming that to be possible) is:

$$Speedup \leq 32 + (1 - 32)(0.01) = 32 - 0.31 = 31.69\text{X}.$$

Consider what Amdahl's Law would estimate for the speedup with these assumptions. Assuming the serial execution percentage to be 1%, the equation for Amdahl's Law yields 1/(0.01 + (0.99/32)) = 24.43X. This is a false computation, however, since the given percentage of serial time is relative to the 32-core execution. The details of this example do not indicate what the corresponding serial execution percentage would be for more cores or fewer cores (or even one core). If the code is perfectly scalable and the data size is scaled with the number of cores, then

this percentage could remain constant, and the Amdahl's Law computation would be the predicted speedup of the (fixed-size) single-core problem on 32 cores.

On the other hand, if the total parallel application execution time is known in the 32-core case, the fully serial execution time can be calculated and the speed up for that fix-sized problem (further assuming that it could be computed with a single core) could be predicted with Amdahl's Law on 32 cores. Assuming the total execution time for a parallel application is 1040 seconds on 32 cores, then 1% of that time would be serial only, or 10.4 seconds. By multiplying the number of seconds (1029.6) for parallel execution on 32 cores, the total amount of work done by the application takes 1029.6*32+10.4 = 32957.6 seconds. The nonparallel time (10.4 seconds) is 0.032% of that total work time. Using that figure, Amdahl's Law calculates a speedup of 1/(0.00032 + (0.99968/32)) = 31.686X.

Since the percentage of serial time within the parallel execution must be known to use Gustafson's Law, a typical usage for this formula is to compute the speedup of the scaled parallel execution (larger data sets as the number of cores increases) to the serial execution of the same sized problem. From the above examples, a strict use of the data about the application executions within the formula for Amdahl's Law gives a much more pessimistic estimate than the scaled speedup formula.

Advice

When computing speedup, the best serial algorithm and fastest serial code must be compared. Frequently, a less than optimal serial algorithm will be easier to parallelize. Even in such a case, it is unlikely that anyone would use serial code when a faster serial version exists. Thus, even though the underlying algorithms are different, the best serial run time from the fastest serial code must be used to compute the speedup for a comparable parallel application.

When stating a speedup value, a multiplier value should be used. In the past, the speedup ratio has been expressed as a percentage. In this context, using percentages can lead to confusion. For example, if it were stated that a parallel code is 200% faster than the serial code, does it run in half the time of the serial version or one-third of the time? Is 105% speedup almost the same time as the serial execution or more than twice as fast? Is the baseline serial time 0% speedup or 100% speedup? On the other hand, if the parallel application were reported to have a speedup of 2X, it is clear that it took half the time (i.e., the parallel version could have executed twice in the same time it took the serial code to execute once).

In very rare circumstances, the speedup of an application exceeds the number of cores. This phenomenon is known as super-linear speedup. The typical cause for super-linear speedup is that decomposition of the fixed-size data set has become small enough per core to fit into local cache. When running in serial, the data had to stream through cache, and the processor was made to wait while cache lines were fetched. If the data was large enough to evict some previously used cache lines, any subsequent reuse of those early cache lines would cause the processor to wait once more for cache lines. When the data is divided into chunks that all fit into the cache on a core, there is no waiting for reused cache lines once they have all been placed in the cache. Thus, the use of multiple cores can eliminate some of the overhead associated with the serial code executing on a single core. Data sets that are too small—smaller than a typical data set size—can give a false sense of performance improvement.

Usage Guidelines

Other parallel execution models have been proposed that attempt to make reasonable assumptions for the discrepancies in the simple model of Amdahl's Law.

Still, for its simplicity and the understanding by the user that this is a theoretical upper bound, which is very unlikely to be achieved or surpassed, Amdahl's Law is a simple and valuable indication of the potential for speedup in a serial application.

Additional Resources

Intel® Software Network Parallel Programming Community

John L. Gustafson. "Reevaluating Amdahl's Law." *Communications of the ACM,* Vol. 31, pp. 532-533, 1988.

Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP.* McGraw-Hill, 2004.