Little's Law and High Performance Computing

David H. Bailey

RNR Technical Report RNR-XX-XXX
September 10, 1997

Abstract

This note discuses Little's law and relates the form cited in queuing theory with a form often cited in the field of high performance computing. A rigorous mathematical proof of Little's law is included.

Author's address: NAS Applications and Tools Group, NASA Ames Research Center, Moffett Field, CA 94035-1000; dbailey@nas.nasa.gov

Little's law is a well-known result of queuing theory. Suppose we have a single-server queue, with customers arriving, being served and then departing. Let N be the average number of waiting customers, A be the average arrival rate, and T be the average wait time. Then Little's law is the assertion that

$$N = AT$$

Little's law can be proved as follows (generalized from an argument in [1]). Let $f(t)$ be the cumulative number of customers that have arrived, and let $g(t)$ be the cumulative number of customers that have departed, as functions of time. In the discrete problems we normally deal with, both the domain and range of $f(t)$ and $g(t)$ are restricted to integer values, but they can be any Lebesgue-measurable, integrable functions. We will assume that there are no customers present at the beginning and at the end of a time interval $[0, T]$, so that $f(0) = g(0) = 0$, and $f(T) = g(T) = N$. Since the functions f and g are monotone increasing, we can define inverses $F(s)$ and $G(s)$ — in the case of discontinuities, merely join endpoints. Then by Fubini's theorem, the area between the curves f and g can be equivalently computed by integrating in either dimension:

$$\int_0^T [f(t) - g(t)]dt = \int_0^N [G(s) - F(s)]ds$$

so that

$$QT = DN$$

where Q is the average length of queue (averaged over time) and D is the average delay per customer (averaged over customers). Since the average arrival rate A (averaged over time) is N/T, we have

$$Q = DA$$

which is Little's law, in the form most often cited in queuing theory references.

In the context of a single processor computer system, one application of Little's law is as follows. Assume for point of discussion that a given computation deals exclusively with data in main memory — there is little or no cache or register re-use, and that there is a one-to-one ratio between floating-point operations and data fetches. Assume also that the system has a balanced design: the length of vector registers, in a vector system; the size of cache lines, in a cache-based system; or the maximum number of outstanding memory references, in a multi-threaded system, is comparable to the memory latency as measured in clock periods.

By Little's law, the average number of words in transit between memory and the processor is given by the product of the memory bandwidth and the memory latency (this assumes that the memory bandwidth is being fully utilized). Now if we assume a balanced design as above, then the number of words in transit at a given time is comparable to

the degree of concurrency inherent in the processor-memory design (i.e. length of vector registers, etc.). Thus we can write the (approximate) relation

$$C = BL$$

i.e., concurrency equals bandwidth times latency, where latency is measured in seconds. This form was originally due to Burton Smith of Tera [2]. Further, since we assume a one-to-one ratio between main memory bandwidth (measured in words/s) and performance (measured in flop/s), we could just as well write

$$C = PL$$

i.e. concurrency equals performance times latency. As an example, if the memory latency is 50 nanoseconds, and the clock period is two nanoseconds, then the processor-memory system must support roughly 25 outstanding memory references, or there is no way that a performance of 500 Mflop/s can be sustained.

This rule can now be extended to multiple processors, where by "concurrency" we mean the aggregate system concurrency (the product of the number of nodes times the concurrency inherent in a single node), and where by "bandwidth" or "performance" we mean the aggregate main memory bandwidth or aggregate system performance, respectively. This extension is clear in a distributed memory system, since we merely multiply each side of the above formulas by the number of processors. But it also can be seen to hold in a shared memory environment.

It is worth pointing out that the concurrency figures one obtains from the above formulas are minimum levels that must be exhibited in an algorithm that hopes to run with full efficiency on the system. In a real problem on a real system, with gaps and irregularities in memory operations, as well as (hopefully minor) single-threaded segments in the algorithm, it is probable that even higher levels of algorithm concurrency are required.

References

[1] Michael Molloy, *Fundamentals of Performance Modeling*, Macmillan, 1989, pg. 162–164.

[2] Burton Smith, "Computer Architectures", ARPA Principal Investigator meeting notes, July 11, 1995. `http://www.tera.com/arpa95/architecture.html`.