What is Scalability?[1]

Mark D. Hill

Computer Sciences Department
1210 West Dayton St.
University of Wisconsin
Madison, Wisconsin 53706
markhill@cs.wisc.edu

ABSTRACT

Scalability is a frequently-claimed attribute of multiprocessor systems. While the basic notion is intuitive, scalability has no generally-accepted definition. For this reason, current use of the term adds more to marketing potential than technical insight.

In this paper, I first examine formal definitions of scalability, **but I fail to find a useful, rigorous definition of it.** I then question whether scalability is useful and conclude by challenging the technical community to either (1) rigorously define scalability or (2) stop using it to describe systems.

KEYWORDS: multiprocessor, parallel random access machine (PRAM), scalability and speedup.

1. Introduction

Even though *scalability* has no generally-accepted definition, it is often used without elaboration to describe multiprocessor systems. This practice imparts a positive feeling toward a system without providing any new information and therefore has no place in technical literature.

The attribute *scalability* can be made technically useful by rigorously defining it so that questions like the following can be answered:

- Is my architecture scalable?
- Are all architectures that avoid buses and broadcasts scalable?
- Can some architectures be more scalable than others?
- Does the intended workload affect an architecture's scalability?
- Does cost affect scalability?
- Is an architecture scalable with respect to a uniprocessor version of itself or a theoretical multiprocessor?
- Does a scalable architecture imply that all, some, or at least one implementation of it are scalable?
- When is an implementation scalable?
- How does one adjust for physical limitations (e.g., bounded fan-in/out and the speed of light)?
- Should a system builder design for scalability or speed?
- Who must consider scalability?

[1] The work is supported in part by the National Science Foundation's Presidential Young Investigator and Computer and Computation Research Programs under grants MIPS-8957278 and CCR-8902536, A.T.&T. Bell Laboratories, Cray Research, Digital Equipment Corporation, Texas Instruments, and the graduate school at the University of Wisconsin--Madison.

Below I try and fail to formalize scalability (Section 2), question whether scalability is important (Section 3), and conclude by asking others to define scalability (better than me) or to stop using it.

2. A Formal Definition of Scalability?

An intuitive notion of scalability is that it implies a favorable comparison between a larger version of some parallel system with either a sequential version of that same system or a theoretical parallel machine. If we choose to compare with a sequential version, it seems natural to define scalability with *speedup* [EZL89]. Let $time(n, x)$ be the time required by an n-processor system to execute a program to solve a of problem of size[2] x. The speedup on a problem of size x with n processors is the execution time on one processor divided by the time on n processors, or:

$$speedup(n, x) = \frac{time(1, x)}{time(n, x)}.$$

Related to speedup is *efficiency*, which is speedup divided by the number of processors, or:

$$efficiency(n, x) = \frac{speedup(n, x)}{n} = \frac{time(1, x)/n}{time(n, x)}.$$

In general, the best possible efficiency is one,[3] implying the best speedup is linear, $speedup(n, x) = n$. Clearly a system that achieves linear speedup matches an intuitive notion of scalability. Therefore, a restrictive definition of scalability is:

> *A system is scalable if $efficiency(n, x) = 1$ for all algorithms, number of processors n and problem sizes x.*

This definition is not useful, however, because it precludes any system from being called scalable. First, many parallel algorithms have a sequential (or at least not completely parallel) component, yielding poor efficiencies for a sufficiently large number of processors [Amd67].[4] Second, if problem size is held constant (as with benchmarks), efficiencies will be poor for sufficiently large systems, for example, when the number of processors nears or exceeds the problem size [FrM88]. Third, if problem size is increased for larger systems, then the rate of increase must be specified. Gustafson sparked considerable debate with his proposal that execution time rather than problem size be held constant for larger systems [Gus88, HeW89, Zho88]. Others propose to follow the lead of sequential complexity theory and examine only asymptotic efficiencies, requiring the number of processors to increase without bound and the problem size to grow at an even faster rate (i.e, $n \rightarrow \infty$ and $x/n \rightarrow \infty$) [AlG89].

While one can imagine a list of caveats to make the above definition more useful (but less elegant), its fundamental deficiency is that a definition of system scalability should exercise systems, not algorithms. One way to factor out the effect of algorithms is to define the efficiency of a real parallel system a second way: with respect to a theoretical parallel machine rather than with respect to a real sequential system. Let $time^*(n, x)$ be the time required by an n-processor theoretical machine to execute the same algorithm as used by the real machine to solve a problem of size x. An alternative definition of efficiency is:

$$efficiency^*(n, x) = \frac{time^*(n, x)}{time(n, x)}.$$

One candidate theoretical machine, at least for shared-memory MIMD multiprocessors, is a *parallel random access machine* (PRAM) [FoW78]. An n-processor PRAM has a single globally-addressable memory and n processors that operate on a lock-step read-memory, compute, write-memory cycle. PRAMs differ in whether they allow simultaneous reads of the same location (concurrent read (CR) versus exclusive read (ER)), whether they allow simultaneous writes of the same location (CW versus EW), and what value remains in a location written by simultaneous writers (COMMON — all simultaneous writes store the same value; ARBITRARY — any one of the values written may remain; and MINIMUM — the value written by the lowest numbered processor remains) [FiR84, Sni82].

A problem with using $efficiency^*(n, x)$ is that theoretical machines and real machines exist in different worlds, making comparisons between $time^*(n, x)$ and $time(n, x)$ difficult. Theoretical machines operate with hypothetical cycle times, occupy no space, and are not limited by technological constraints. Real machines, on the other hand, have cycle times determined by numerous practical factors, and have global communication limited by

[2] *Problem size* is the amount of memory necessary to specify the input to the problem. In serial complexity theory, it is usually denoted with n rather than x. We use x since n has already been used to denote the number of processors. Many notations for speedup omit mention of problem size, usually implying that it is large and held constant.

[3] Super-linear speedups occasionally occur, primarily due to interactions with a system's memory hierarchy.

[4] A recent paper [KaF90] even advocates using the lack of variability in experimental measures of a program's sequential fraction as an indicator of scalability.

the propagation of light and the bounded fan-in and fan-out of most real technologies [AlG89]. Implementing systems in three-space means global communication must eventually slow down with the cube-root of system size, while bounded fan-in and fan-out introduces a logarithmic factor. The multiplicative effect of these constraints can yield a second definition for scalability:

> *A system is scalable if $efficiency^*(n, x) = O([n^{1/3}\log n]^{-1})$ for all algorithms, number of processors n and problem sizes x.*

Another problem with using a theoretical machine is that one must be selected. Unfortunately, different theoretical machines have different asymptotic execution times for the same problem. For example, there exist problems for which a CREW (concurrent-read-exclusive-write) PRAM takes a factor of log n longer than a CRCW-COMMON PRAM, and other problems where the latter machine can take take log n longer than a CRCW-MINIMUM PRAM [FiR84]. Thus, the best the above definition allows is for an architecture to be called scalable or not scalable with respect to a particular, explicitly-mentioned theoretical machine.

A final problem with the above definition of scalability is that it is too much work (for systems architects and implementors). While it is time-consuming to compute $efficiency^*(n, x)$ for one algorithm, it is worse to do so for all algorithms. Perhaps it is more appropriate to call a machine scalable for a particular algorithm.

3. Is Scalability Important?

An implicit assumption of the previous section is that scalability is important and worth defining. Here I examine this assumption.

Scalability is important if it is useful. Whether it is useful depends on who is using it, what their purpose is, and how it is defined. Potential users are theoreticians, academic paper-writers, academic system-builders, industrial visionaries, industrial systems architects, industrial systems builders, product marketing, and customers.[5] Purposes for using scalability include gaining insight about mathematical models of large systems, designing new computer architectures, implementing new machines, marketing products,[6] and selecting new computers to purchase. Definitions of scalability range from considering performance relative to a PRAM to brainstorming about a somewhat larger system.

I view scalability, even without a rigorous definition, as being useful for providing insight. Contemplating large systems enable researchers to discover approaches not presently necessary or practical. Designers of the NYU Ultracomputer [GGK83], for example, invented *combining* for handling situations where the rate of requests to a single memory word increases with system size. While combining hardware is still not necessary (or practical?) in today's systems, the ideas have evolved to affect commercial systems (e.g., Sequent Symmetry [GrT90]).

I do not view scalability, especially asymptotic scalability, as being useful for selecting between design options or for describing an architecture or implementation. I see no reason to believe that the best software and hardware for an arbitrarily large number of processors is best for smaller systems. Engineering factors are important. The best interconnection network for ten or hundred processors may be a bus or 2-D torus, respectively. Does it matter whether either is asymptotically optimal?

Systems architects probably want to design a system to work well over a size range of ten to twenty times, while implementors should be concerned with a smaller range, say two to four times. Thus, a company designing a new architecture for which initial, low-end systems that will have four processors may wish to consider ramifications for 80-processor systems when making architectural decisions, but should probably implement the first system with a bus. Furthermore, systems implementors should consider using ugly, unscalable things (e.g., buses and broadcast) if such things simplify the design, reduce system cost, and improve performance.

4. Conclusions

In this paper, I examined aspects of scalability, but did not find a useful, rigorous definition of it. Without such a definition, I assert that calling a system "scalable" is about as useful as calling it "modern". I encourage the technical community to either rigorously define scalability or stop using it to describe systems.

[5] These categories are not mutually exclusive.

[6] One company even calls its instruction set architecture "scalable".

5. Acknowledgments

The ideas in this paper evolved from ideas appearing in [GHW89] through discussions with Jim Goodman, Mary Vernon, Phil Woest and others and through feedback from a talk I gave at the *Scalable Shared-Memory Architectures* workshop in Seattle on May 26, 1990. I would like to thank Michel Dubois and Shreekant Thakkar for organizing the workshop. Finally, I wish to thank Sarita Adve, Anant Agarwal, Sue Dentinger, Rajiv Jauhari, Ross Johnson, Richard Kessler, James Larus, Dan Nussbaum, Diana Stone, and Mary Vernon for reading and improving drafts of this paper.

6. References

[AlG89] G. S. ALMASI and A. GOTTLIEB, *Highly Parallel Computing*, Benjamin / Cummings Publishing Company, Inc., Redwood City, CA, (1989).

[Amd67] G. M. AMDAHL, Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities, *AFIPS Conference Proceedings*(April 1967), 483-485.

[EZL89] D. L. EAGER, J. ZAHORJAN and E. D. LAZOWSKA, Speedup Versus Efficiency in Parallel Systems, *IEEE Trans. on Computers*, C-38, 3 (March 1989), 408-423.

[FiR84] F. E. FICH and P. L. RAGDE, Relations Between Concurrent-Write Models of Parallel Computation, *Proc. Principals of Distributed Computing*(August 1984), 179-190.

[FoW78] S. FORTUNE and J. WYLLIE, Parallelism in Random Access Machines, *Proc. Tenth ACM Symposium on Theory of Computing*(1978), 114-118.

[FrM88] R. S. FRANCIS and I. D. MATHIESON, A Benchmark Parallel Sort for Shared Memory Multiprocessors, *IEEE Transactions on Computers*, 12 (December 1988), 1619-1626.

[GHW89] J. R. GOODMAN, M. D. HILL and P. J. WOEST, Scalability and Its Application to Multicube, Computer Sciences Technical Report #835, Univ. of Wisconsin (March 1989).

[GGK83] A. GOTTLIEB, R. GRISHMAN, C. P. KRUSKAL, K. P. MCAULIFFE, L. RUDOLPH and M. SNIR, The NYU Ultracomputer--Designing an MIMD Shared Memory Parallel Computer, *IEEE Trans. on Computers*, C-32, 2 (February 1983), 175-189.

[GrT90] G. GRAUNKE and S. THAKKAR, Synchronization Algorithms for Shared-Memory Multiprocessors, *IEEE Computer* , 23 , 6 (June 1990), 60-69.

[Gus88] J. L. GUSTAFSON, Reevaluating Amdahl's Law, *Communications of the ACM*, 31, 5 (May 1988), 532-533.

[HeW89] M. HEATH and P. WORLEY, Once Again, Amdahl's Law, *Communications of the ACM*, 32, 2 (February 1989), 262-264.

[KaF90] A. H. KARP and H. P. FLATT, Measuring Parallel Processor Performance, *Communications of the ACM*, 33, 5 (May 1990), 539-543.

[Sni82] M. SNIR, On Parallel Search, *Proc. Principals of Distributed Computing* (August 1982), 242-253.

[Zho88] X. ZHOU, Bridging the Gap Between Amdahl's Law and Sandia Laboratory's Result, *Communications of the ACM*, 31, 8 (August 1988), 1014-1016.