Scalability

# of Parallel Machines

Scalability is a common objective in the design of multiprocessors, judging by the number of parallel machine architectures touted as "scalable" or "large-scale." Such claims to scalability are hard to refute (or prove) because scalability has no commonly accepted, precise definition. There is, however, some consensus that as the size of a scalable machine is increased, a corresponding increase in performance is obtained. Furthermore, the increase in performance is related to communication patterns in applications programs and the communication infrastructure provided by the machine. This article presents a precise definition for scalability derived from these observations.

Why do we care about scalability? Primarily, once an architecture is shown to be scalable, then machines whose size varies over a wide range can use that same architecture. There is a caveat to this argument, however. We believe it is unlikely that practical machines differing in size by several orders of magnitude will have the same structure: there are size-related issues that are simply not relevant when building small machines. Nevertheless, we can use scalability to aid our *study* of large machines. For designers of new architectures, small experimental prototypes can demonstrate the viability of ideas in much larger machines. If a system can be shown to scale well for a certain class of algorithms, then measurements from simulations of those algorithms on relatively few processors can be used to predict the behavior of systems in which the number of processors is too large for simulation to be practical.

We define the scalability of a given architecture to be the fraction of the parallelism inherent in a given algorithm that can be exploited by any machine of that architecture as a function of problem size. For a given algorithm and problem size, we derive the inherent parallelism as the ratio of the serial execution time and the runtime on an ideal realization of a parallel random access machine (PRAM)[1]. For the same algorithm and problem size, we then derive the maximum speedup attainable by a machine of any size with the architecture of interest. (In this paper, machine size is stated in terms of p, the number of processors.) The maximum speedup, specified as a function of problem size, is called the *asymptotic speedup* of the architecture for the given algorithm. Thus the inherent parallelism of the algorithm is its asymptotic speedup on an ideal PRAM machine. Our measure of scalability follows as the ratio of the asymptotic speedups on the given architecture and on the PRAM.

As an example, let us compute the scalability of a hypercube machine on the problem of adding s numbers, ignoring overflows, where individual additions are structured as a balanced binary tree. We assume that individual binary additions complete in a single cycle and that single-word communications between directly connected processors also take one cycle. On an ideal PRAM realization, this addition can be done in log s steps. Similarly, an s-processor hypercube can achieve the minimum runtime of log s. Thus the scalability $\Psi(s)$ of a hypercube for the above problem is one. Intuitively, a scalability of one implies the hypercube communication structure allows full exploitation of the parallelism in-

Daniel Nussbaum and Anant Agarwal

herent in the given algorithm, regardless of problem size

Scalability is by no means the only consideration in the design of a parallel processor. Other factors such as processor efficiency, programmability, flexibility, and cost-effectiveness are also important in evaluating a machine design; in fact, when building a machine of a known size, scalability may not be a consideration at all.

Background

Notions of scalability are invariably tied to notions of speedup. Scalability might be defined as follows:

> A scalable architecture exhibits speedup linearly proportional to p, the number of processors employed.

We argue that this definition is deficient, using two commonly accepted notions of speedup—simple speedup and scaled speedup. Simple speedup keeps problem size fixed and scaled speedup allows problem size to grow with machine size [6].

If simple speedup is used, it is easy to show no architecture is scalable by the definition of scalability stated above. Amdahl's classic paper [1] argues that when problem size is fixed, speedup is bounded by the reciprocal of the serial fraction of the algorithm. More accurately, speedup is bounded by the ratio between the serial complexity for the algorithm and the length of the critical path through the dataflow graph for that algorithm. Flatt and Kennedy [4] are even more pessimistic, showing that as the number of processors is increased, the running time eventually hits a minimum, after which adding processors can only cause the algorithm to take longer to complete. Hard performance limits imply lack of scalability by the simple definition given above, rendering simple speedup inadequate.

Perhaps simple speedup is the wrong performance metric to use in a definition of scalability: we now try using scaled speedup instead. Gustafson [6] defines scaled speedup to allow the problem size (as measured by its serial complexity) to increase linearly with p, the number of processors, following the rationale that one often uses multiprocessors to run larger programs, not to run the same programs faster. Unfortunately, for most interesting algorithms, maximum achievable speedup does not increase as fast as serial complexity; so even using Gustafson's more liberal definition for speedup, no architecture is scalable by the simple definition given above. Furthermore, Flatt and Kennedy show that for any algorithm in which the overhead due to parallelism increases with p, the scaled speedup is necessarily less than linear, and that when the parallelism overhead is linear or worse with p, there is actually a hard upper bound on the achievable scaled speedup.

Goodman, Hill, and Woest [5] restrict the simple scalability definition given above to scalable algorithms. Parallelism in scalable algorithms grows at least linearly with the serial complexity. Furthermore, they relax the speedup requirement to $\frac{p}{\log p}$ for small networks, and $\frac{p}{\sqrt[3]{p}}$ for large networks, where p is the number of processors. This approach has two drawbacks: it excludes a large class of interesting algorithms, and it fails to distinguish between scalable algorithms that exhibit very different communication patterns. Using the definition presented by Goodman *et al.*, it is possible to devise scalable algorithms that make *most* interesting architectures appear unscalable.

The simple definition for scalability given above has problems both with respect to the meaning of speedup and with respect to differences in behavior between different algorithms. A good definition for scalability must take both of these issues into account.

Requirements

In our view, a useful definition of scalability has several requirements. First, it must express both the effects of the architecture's interconnection network and the communication patterns inherent to the algorithm. The communication behavior of an algorithm can be expressed in terms of properties of its communication graph such as the clustering of communication edges and the frequency of communication along those edges.

Second, the definition must measure an architecture's scalability with respect to a given algorithm. Because different algorithms can have widely different communication patterns, their performance can scale differently with machine size on a given architecture. Certain architectures might yield good performance only for algorithms whose communication patterns are well matched to the architecture's network structure. However, it may be possible to generalize the definition of scalability to classes of algorithms.

Third, we must carefully distinguish between *algorithmic scalability* and *architectural scalability*. Algorithmic scalability is related to the parallelism inherent in an algorithm, and can be measured through its speedup on an architecture with an idealized communication structure, such as a PRAM. By defining architectural scalability as the relative performance between the ideal and the real architectures, we will isolate those aspects of scalability arising from algorithmic behavior from those arising from machine design. This machine-oriented view of scalability is useful from the viewpoint of a computer architect, and is the basis for our definition in this article. However, the end-user might prefer a different notion of scalability that combines our concepts of algorithmic and architectural scalability.

Fourth, the effects of physical

[1]Specifically, we use an exclusive read, exclusive write (EREW) PRAM, which is an idealized model of a parallel computer that satisfies all nonconflicting memory accesses in one cycle and queues conflicting memory accesses to be satisfied one after another, each requiring one time unit [7].

constraints imposed by the three-dimensionality of space and fundamental limits on communication speeds have to be considered. This issue is taken up in the section called Physical Constraints.

Fifth, scalability should be an indicator of the best achievable performance, without regard to efficiency or cost-effectiveness. For algorithms whose parallelism overhead increases with p as p becomes large, increased speedup can only be obtained at the expense of lower efficiency [4], even when Gustafson's notion of scaled speedup is the speedup metric employed. Since efficiency is not the primary thrust of this article, we base our definition of scalability on *asymptotic speedup.* For a given algorithm, architecture and problem size, the asymptotic speedup is the best achievable speedup on a machine on *any* size. This definition keeps the problem size constant while allowing the number of processors to become arbitrarily large.

We will not consider cost-effectiveness in our definition. However, the cost often manifests itself indirectly through its impact on machine size. For example, architectures that make use of full-map directories [2] are commonly said to be nonscalable due to cost considerations. The full-map directory scheme requires memory at each node proportional to p, the number of processors. This has the effect of increasing node size (and consequently the communication time) as p increases, thus increasing overall machine cycle time, producing a corresponding decrease in scalability.

Finally, any attempt to define scalability ought to yield more information than a simple predicate. A simple predicate hides information as to how badly the architecture fails or how well it succeeds in being scalable. We want a definition that reflects behaviors of different algorithms in a concrete fashion.

With these ideas in mind, we present the following informal definition for scalability:

> For a given algorithm, an architecture's *scalability* is the ratio of the algorithm's asymptotic speedup when run on the architecture in question to its corresponding asymptotic speedup when run on an EREW PRAM, as a function of problem size.

Intuitively, this definition measures the fraction of the parallelism inherent to the given algorithm that can be realized by the architecture in question. In the next two sections, we formalize this definition.

Asymptotic Speedup

For a given architecture, algorithm, and problem size s, the asymptotic speedup $S(s)$ is the best speedup that can be attained, varying only p, the number of processors.

We formally define the asymptotic speedup $S(s)$ for a given parallel algorithm as follows:

$$\begin{aligned} s &\equiv \text{problem size} \\ T_{seq}(s) &\equiv \Theta(\text{Serial Running Time}) \\ T_{par}(s) &\equiv \Theta(\text{Minimum Parallel Running Time}) \\ S(s) &\equiv \frac{T_{seq}(s)}{T_{par(s)}} \end{aligned}$$

The problem size s is the independent variable upon which all other metrics are based. Note that s is a function of the encoding of the input, and that a change in the units of s for a given problem changes only the form of the result. For example, in multiplying square matrices, we can use the number of elements n or the number of rows r as the problem size. The runtime for a straightforward matrix multiply algorithm is $n^{\frac{3}{2}}$ in the first case and r^3 in the second case. These two expressions are equivalent by the substitution $n = r^2$.

$T_{seq}(s)$ is the serial running time. Meaningful measurements of asymptotic speedup mandate the use of good serial algorithms, even if they differ in structure from the corresponding parallel algorithms. In practice, however, expedience may justify using the single processor running time of the parallel algorithm as an estimate of the sequential running time. Although we show in the next section that our choice of the serial algorithm does not affect scalability, it is still useful for calculating asymptotic speedup, which is an interesting metric in itself.

$T_{par}(s)$ is the minimum parallel running time for the given algorithm with problem size s. This time is calculated using as many processors as necessary to achieve the minimum runtime for the given problem size. We assume that issues of partitioning, placement and scheduling are addressed by either the machine itself (including its software system) or explicitly by the programmer. In other words, we consider solutions to these issues not to be part of the algorithm, but instead, part of the machine.

Finally, the *asymptotic speedup* $S(s)$ is defined to be the ratio between the serial running time $T_{seq}(s)$ and the minimum parallel running time $T_{par}(s)$.

In our definition of scalability, we are only interested in results obtained from solving large problems. Therefore, we express both $T_{seq}(s)$ and $T_{par}(s)$ using Θ notation. Θ is the "asymptotic limit" or the "dominant term" operator, defined more precisely as follows:

$$\Theta : \forall\, s, f(s), g(s) \in \mathrm{R}, f(s) = \Theta(g(s)) \Leftrightarrow \exists\, k_1, k_2, s_0 \in \mathrm{R}^+ : |s| > s_0 \Rightarrow |k_1 g(s)| \leq |f(s)| \leq |k_2 g(s)|$$

Use of the Θ operator enforces the requirement that the problem size s be large enough to display asymptotic behavior.

Scalability

We define the scalability $\Psi(s)$ of a machine for a given algorithm to be the ratio of the asymptotic speedups on the real machine and on the ideal realization of an EREW PRAM. The normalization with respect to an ideal machine reflects a focus on the communication structure of the given architecture and an attempt to separate architectural and algorithmic behavior. The asymptotic speedups for the ideal machine and for the given architecture are $S_I(s) = \dfrac{T_{par_I}(s)}{T_{seq}(s)}$ and

$S_R(s) = \frac{T_{par_R}(s)}{T_{seq}(s)}$ respectively. Thus,

$$\Psi(s) \equiv \frac{S_R(s)}{S_I(s)} = \frac{T_{par_I}(s)}{T_{par_R}(s)}$$

Intuitively, the larger the scalability, the better the performance that the given architecture can yield running the given algorithm.

We choose the EREW PRAM as our ideal architecture since its behavior resembles that of a real system when ignoring the effects of interconnection networks and caches. The asymptotic speedup $S_I(s)$ for an EREW PRAM is simply a measure of the parallelism inherent to the test algorithm.

This definition is valid for a specific architecture running a specific algorithm. While it may be possible to generalize our scalability definition to a class of architectures or a class of algorithms, we do not attempt to do so in this article.

Our definition yields more information than does a simple predicate. If a predicate is needed, one can easily be generated from the definition. An architecture could be said to be $f(s)$-scalable if its scalability is at least $f(s)$ for a given algorithm. For example, the Omega-network [8] is $\frac{1}{\log s}$-scalable for computing the parity of s bits, as shown in the next section.

An Example: Parity of *s* Bits

Table 1 gives running times, asymptotic speedups and scalabilities for various architectures running a parallel parity calculation. This calculation examines s bits, determining whether the number of bits set is even or odd using a balanced binary tree. In all cases, certain potentially relevant factors are ignored, such as partitioning overhead, scheduling overhead and program load time (the assumption is that the code and data are distributed to the appropriate processors before the program is run). The analysis in this section assumes unit communication time between directly connected communication nodes and unit computation time for a simple two-bit parity calculation. For the given algorithm, $T_{seq}(s) = s$, $T_{par_I}(s) = \log s$, and $S_I(s) = \frac{s}{\log s}$.

On the real architectures, the parity algorithm's performance is limited by network diameter. For example, on a one-dimensional mesh, the network diameter is proportional to the number of processors p, yielding a total parallel running time of $\frac{s}{p} + p$. The optimal partition of the problem for such a machine uses $p = \sqrt{s}$ processors, so that each processor performs the parity computation on $\sqrt{s}$ bits locally. This partition gives the best match between computation costs and communication costs, with parallel running time $T_{par_R}(s) = s^{\frac{1}{2}}$, real asymptotic speedup $S_R(s) = s^{\frac{1}{2}}$ and scalability $\Psi(s) = \frac{\log s}{s^{\frac{1}{2}}}$. The other two mesh networks examined use similar partitions to match their own communication structures with their computational loads, yielding results of a similar form. Not surprisingly, $\Psi(s)$ increases as the communication latency decreases in networks with smaller diameters.

The hypercube and the Omega-network provide richer communication structures (and lower diameters) than meshes of lower dimensionality. The hypercube does as well as a PRAM for this algorithm, yielding $\Psi(s) = 1$. The Omega-network does not exploit locality: communication with all processors takes the same amount of time. This loss of locality hurts its performance when compared to the hypercube, but its lower diameter gives it better scalability than any of the meshes. Although performance is limited by network diameter for the parity algorithm, for many other algorithms, network bandwidth is the performance-limiting factor.

The above analysis assumed unit communication time between directly connected communication nodes. We now examine the validity of this assumption and propose a modified definition of scalability that incorporates the effects of physical constraints.

TABLE 1

Scalabilities of various direct [9] and indirect [10] network-based architectures. For the parity calculation,

$T_{seq}(S) = S$, $T_{par_I}(S) = \log S$, and $S_I(S) = \frac{S}{\log S}$.

	1-d mesh (direct)	2-d mesh (direct)	3-d mesh (direct)	Hypercube (direct)	Omega (indirect)
$T_{par_R}(S)$	$S^{\frac{1}{2}}$	$S^{\frac{1}{3}}$	$S^{\frac{1}{4}}$	$\log S$	$\log^2 S$
$S_R(S)$	$S^{\frac{1}{2}}$	$S^{\frac{2}{3}}$	$S^{\frac{3}{4}}$	$\frac{S}{\log S}$	$\frac{S}{\log^2 S}$
$\Psi(S)$	$\frac{\log S}{S^{\frac{1}{2}}}$	$\frac{\log S}{S^{\frac{1}{3}}}$	$\frac{\log S}{S^{\frac{1}{4}}}$	1	$\frac{1}{\log S}$

Physical Constraints

Physical constraints impose limitations on the communication speed between nodes, making our assumptions of unit communication time between directly connected nodes unrealistic. When a network with more than three dimensions, for example, a hypercube, is embedded in three-dimensional space, wires in some dimensions of the network will be much longer than wires in other dimensions. If the network clock cycle is related to the length of the longest wire (assuming a synchronous system with a single bit on a given wire at a given time), networks with higher dimensionality will have a slower clock, and will therefore yield larger values for $T_{par_R}(s)$ and correspondingly lower values for $\Psi(s)$. For example, the length of the longest wire in a naive embedding of a p-processor hypercube into three-dimensional space is $\Theta(\sqrt[3]{p})$. (A better embedding exists but we use the naive embedding here for simplicity.)

The notion of *three-space scalability* can now be defined to account for the increase in communication costs due to physical constraints. If $S_{R^3}(s)$ reflects the asymptotic speedup on a real architecture implemented in three-dimensional space,

$$\Psi^3(s) \equiv \frac{S_{R^3}(s)}{S_I(s)} = \frac{T_{par_I}(s)}{T_{par_{R^3}}(s)}$$

We now apply this modified definition to the parity algorithm running on a hypercube. The slower network clock constrained by the longest wire in the network ($\Theta(\sqrt[3]{p})$) produces a corresponding increase in the communication time between every pair of directly connected nodes. Therefore, for the given algorithm, the communication cost in a hypercube implemented in three-dimensional space is worse than that of a three-dimensional mesh. On p processors, the runtime of an s-bit parity computation is $\frac{s}{p} + \sqrt[3]{p} \log p$. The runtime for this computation is minimized when $p = \left(\frac{s}{\log s}\right)^{\frac{3}{4}}$ processors, yielding $T_{par_R}(s) = s^{\frac{1}{4}} \log^{\frac{3}{4}} s$ and $S_R(s) = \frac{s^{\frac{3}{4}}}{\log^{\frac{3}{4}} s}$. The resulting scalability of the hypercube $\Psi^3(s) = \frac{\log^{\frac{1}{4}} s}{s^{\frac{1}{4}}}$ is lower than the scalability of a three-dimensional mesh by a factor of $\log^{\frac{3}{4}} s$.

The above analysis takes increased wire length into account, but ignores the effect of wire thickness. If nonzero wire thickness is also considered, the scalability of the hypercube is further diminished.

Summary

This article proposes a definition of scalability based on the communication patterns in parallel algorithms and the communication structures provided by parallel architectures. Intuitively, the scalability of an architecture measures the fraction of the parallelism inherent to a given algorithm that can be exploited by any machine with that architecture. We define scalability $\Psi(s)$ of an architecture for a given algorithm with problem size s as the ratio of the algorithm's asymptotic speedup on the architecture in question and its corresponding asymptotic speedup on an EREW PRAM. The asymptotic speedup for the given algorithm and architecture, specified as a function of problem size, is the maximum speedup attainable by any machine of the given architecture, regardless of the number of processors employed.

Acknowledgments

The research reported here has benefited greatly from discussions with Tom Leighton. Useful comments from Steve Ward and Mark Hill are also gratefully acknowledged. ■

References

1. Amdahl, G.M. Validity of the single-processor approach to achieving large scale computing capabilities. In *AIFPS Conference Proceedings* (Apr. 1967).
2. Censier, L.M. and Feautrier, P. A new solution to coherence problems in multicache systems. *IEEE Trans. Comput. C-27,* 12 (Dec. 1978), 1112–1118.
3. Eager, D.L., Zahorjan, J., and Lazowska, E.D. Speedup versus efficiency in parallel systems. *IEEE Trans. Comput. 38,* 3 (Mar. 1989).
4. Flatt, H.P. and Kennedy, K. Performance of parallel processors. *Parallel Comput. 31,* 1989. 1–20.
5. Goodman, J.R., Hill, M.D., and Woest, P.J. Scalability and its application to multicube. Department of Computer Sciences, University of Wisconsin-Madison, 1988.
6. Gustafson, J.L. Reevaluating Amdahl's Law. *Commun. ACM 31,* 5 (May 1988), 532–533.
7. Karp, R.M. and Ramachandran, V. Parallel algorithms for shared-memory machines. Tech. Rep. 408, University of California at Berkeley, Fall 1988.
8. Lawrie, D.H. Access and alignment of data in an array processor. *IEEE Trans. Comput. C-24,* 12 (Dec. 1975), 1145–1155.
9. Seitz, C.L. Concurrent VLSI Architectures. *IEEE Trans. Comput. C-33,* 12 (Dec. 1984).
10. Siegel, J. *Interconnection Networks for Large-Scale Parallel Processing.* McGraw-Hill, 1990. Second Edition.

CR Categories and Subject Descriptors: C.1.2 [**Multiple Data Stream Architectures**]:—*Parallel processors, interconnection architectures;* C.4 [**Performance of Systems**]:—*Performance attributes*

General Terms: Algorithms, Design, Experimentation, Performance, Theory

About the Authors:

DANIEL NUSSBAUM has been doing graduate research with the Laboratory for Computer Science at the Massachusetts Institute of Technology in Cambridge, Mass. since 1983. He is currently pursuing a Ph.D. His primary research interests include parallel architectures, languages, algorithms and operating systems.

ANANT AGARWAL has been with the Laboratory for Computer Science at the Massachusetts Institute of Technology since January 1988. He is an assistant professor of electrical engineering and computer science. At Stanford, he participated in the MIPS and MIPS-X projects. His current research interests include the design of scalable multiprocessor systems, VLSI processors, parallel-processing software, and performance evaluation.

Authors' Present Address: Daniel Nussbaum and Anant Agarwal are at the Laboratory for Computer Science, Room NE43-629 Mass. Inst. of Tech., 545 Tech. Square, Cambridge, MA 02139. Email: Nussbaum—dann@masala.lcs.mit.edu, Agarwal—agarwal@masala.lcs.mit.edu

This research was funded by DARPA contract #N00014-87-K-0825