The Efficiency of Parallel Processing

by B. L. Buzbee

Parallel processing, or the application of several processors to a single task, is an old idea with a relatively large literature. The advent of very large-scale integrated technology has made testing the idea feasible, and the fact that single-processor systems are approaching their maximum performance level has made it necessary. We shall show, however, that successful use of parallel processing imposes stringent performance requirements on algorithms, software, and architecture.

The so-called asynchronous systems that use a few tightly coupled high-speed processors are a natural evolution from high-speed single-processor systems, Indeed, systems with two to four processors will soon be available (for example, the Cray X-MP, the Cray-2, and the Control Data System 2XX). Systems with eight to sixteen processors are likely by the early 1990s. What are the prospects of using the parallelism in such systems to achieve high speed in the execution of a single application? Early attempts with vector processing have shown that plunging forward without a precise understanding of the factors involved can lead to disastrous results. Such understanding will be even more critical for systems now contemplated that may use up to a thousand processors.

The key issue in the parallel processing of a single application is the speedup achieved, especially its dependence on the number of processors used. We define speedup *(S)* as the factor by which the execution time for the application changes: that is,

$$S = \frac{\text{execution time for one processor}}{\text{execution time for } p \text{ processors}}$$

To estimate the speedup of a tightly coupled system on a single application, we use a model of parallel computation introduced by Ware. We define a as the fraction of work in the application that can be processed in parallel. Then we make a simplifying assumption of a two-state machine; that is, at any instant either all *p* processors are operating or only one processor is operating. If we normalize the execution time for one processor to unity, then

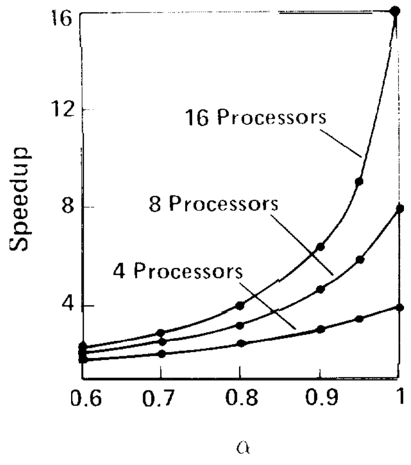
$$S(p,a) = \frac{1}{(1-a) + a/p}$$

Note that the first term in the denominator is the execution time devoted to that part of the application that *cannot* be processed in parallel, and the second term is the time for that part that can be processed in parallel. How does speedup vary with *a?* In particular. what is this relationship for *a* = 1. the ideal limit of complete parallelization? Differentiating *S,* we find that

$$\left.\frac{\partial S(p,\alpha)}{\partial \alpha}\right|_{\alpha=1} = p^2 - p \ .$$

The accompanying figure shows the Ware model of speedup as a function of a for a 4-processor, an 8-processor, and a 16-processor system. The quadratic dependence of the derivative on *p* results in low speedup for a less than 0.9. Consequently, to achieve *significant* speedup, we must have highly parallel algorithms. It is by no means evident that algorithms in current use on single-processor machines contain the requisite parallelism. and research will be required to find suitable replacements for those that do not. Further, the highly parallel algorithms available must be implemented with care. For example, it is not sufficient to look at just those portions of the application amenable to parallelism because *a* is determined by the entire application. For a close to 1. changes in those few portions less amenable to parallelism will cause small changes in a, but the quadratic behavior of the derivative will translate those small changes in a into large changes in speedup.

Those who have experience with vector processors will note a striking similarity between the Ware curves and plots of vector processor performance versus the fraction of vectorizable computation. This similarity is due to the assumption in the Ware model of a two-state machine since a vector processor can also be viewed in that manner. In one state it is a relatively slow, general-purpose machine. and in the other state it is capable of high performance on vector operations,



Speedup as a function of parallelism (a) and number of processors.

Ware's model is inadequate in that it assumes that the instruction stream executed on a parallel system is the same as that executed on a single processor. Seldom is this the case because multiple-processor systems usually require execution of instructions dealing with synchronization of the processes and communication between processors. Further, parallel algorithms may inherently require additional instructions. To correct for this inadequacy, we add a term, $\sigma(p)$, to the execution time for parallel implementation that is at best nonnegative and usually monotonically increasing with *p*. Actually, σ is a function not only of *p* but of the algorithm. the architecture, and even of α. Let $S(p,\alpha,\sigma)$ denote speedup for this modified model. Then

$$S(p,\alpha,\sigma) = \frac{1}{(1-\alpha) + \alpha/p + \sigma(p)} \ .$$

If the application can be put completely in parallel form, then

$$S(p,\alpha,\sigma)\Big|_{\alpha=1} = \frac{p}{1 + p\sigma(p)} \ .$$

In other words, the maximum speedup of a real system is less than the number of processors *p,* and it may be significantly less. Also note that, whatever the value of *a, S* will have a maximum for sufficiently large *p* because α/p becomes insignificant while $\sigma(p)$ continues to increase,

Thus the research challenge in parallel processing involves finding algorithms, programming languages, and parallel architectures that, when used as a system, yield a large amount of work processed in parallel (large *a)* at the expense of a minimum number of additional instructions (small σ). ■