PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Optimally Universal Parallel Computers

L. G. Valiant

Phil. Trans. R. Soc. Lond. A 1988 **326**, 373-376
doi: 10.1098/rsta.1988.0093

Phil. Trans. R. Soc. Lond. A **326**, 373–376 (1988) [373]
Printed in Great Britain

Optimally universal parallel computers

By L. G. Valiant
Harvard University, Cambridge, Massachusetts 02138, *U.S.A.*

It is shown that any program written for the idealized shared-memory model of parallel computation can be simulated on a hypercube architecture with only constant factor inefficiency, provided that the original program has a certain amount of parallel slackness.

A key property of the von Neumann architecture for sequential computers is efficient universality. It can simulate arbitrary programs written in appropriate high-level languages in time proportional to that which they would take if special-purpose sequential machines were built for each of them. This makes possible standardized languages and transportable software, without which the level of pervasiveness that computers have now reached would be difficult to imagine.

The future of parallel computation may be strongly influenced by the extent to which efficient universality can be found and harnessed in that context also. To formulate such questions we denote a machine architecture by M, the class of programs to be simulated by U, and the efficiency function, the ratio of the total computational operation count of the original program to the total computational operation count of the simulation, by E. Thus a universality statement is of the form 'M can simulate any program P in U with efficiency at least E'. This is a quantative assertion expressed in terms of the relevant parameters such as the runtime of P and the number of processors used by M and by P. We shall say that *optimal* universality is achieved if the efficiency is a consant factor independent of the program P and of the number of processors.

Our task is to find appropriate choices of U and M. Some choices would seem to be overly optimistic in the light of current knowledge. For example, choosing U to be arbitrary sequential programs and M any parallel architecture parametrized by the number of processors seems hopeless because the only parallelization techniques known for general models of computation achieve minute factors of runtime reduction at enormous cost in processors. Similarly there seems to be little hope of choosing U and M in such a way that in the simulation locality in M can be exploited in any generality although, of course, locality is exploitable and beneficial for more limited applications.

The purpose of this note is to point out that a powerful optimal universality result for parallel computation is already largely implicit in existing results in theoretical computer science. Further details of this result and of the relevant background can be found elsewhere (Valiant 1989).

Our choice of M is a sparse network of communication lines with a universal sequential processor and a memory block at each node. As network topology we shall assume the binary n-dimensional cube (or hypercube) which is conceptually the simplest amongst those that are known to suffice.

Our choice of U are classes of programs written for synchronized shared-memory multiprocessors or PRAMs. This has proved a convenient language for describing parallel algorithms and is in widespread use for this purpose (Karp & Ramachandran 1989). The advantage of the shared-memory model as a language is that it assumes that communication and memory management are automated and hidden from the programmer, much as conventional high-level languages and virtual memory hardware hide storage allocation. We distinguish among different classes of PRAM in a vein similar to that of Borodin & Hopcroft (1985). The basic results are the same for all classes, but the constructions and algorithms differ.

Because the results concern comparisons with M, it is sufficient to say that the set of local instructions of each processor is the same in U as in M, without the need for further specification.

In a shared-memory system we ideally expect any number of processors to read or write into any word of any memory unit concurrently. (Note that for concurrent writing one needs some convention to resolve conflicts.) This is called the concurrent PRAM. We also distinguish two restrictions. An exclusive PRAM is one where concurrent 'reads' or 'writes' into the same word are forbidden. A seclusive PRAM is one where concurrent 'reads' or 'writes' by more than one processor into the same memory block is forbidden. We can enrich any one of these models by hypothesizing global operations other than reading or writing. For example, the products of all prefixes of a sequence of associative operations can be implemented efficiently on cubes and other networks, and are useful in parallel algorithms (Schwartz 1980; Blelloch 1988).

To achieve optimal efficiency we also need the extra condition of parallel slackness, that the PRAM program has larger parallelism than available in M. The results that can be achieved for the above varieties of PRAMs are of the following form.

THEOREM *Any time $T/\log_2 N$ algorithm on an $N \log_2 N$-processor PRAM can be simulated in time cT on an N-processor hypercube, where c is a constant independent of T and N.*

Optimality is achieved because the time-processor product TN is preserved in the simulation up to a constant factor. The simulation that achieves this assigns the functions of $\log_2 N$ distinct PRAM processors, as well as all their memory blocks, to each processor or node of the cube. To simulate one step of the $N \log_2 N$ processor PRAM each node of the cube has to perform $\log_2 N$ computational operations as well as up to $\log_2 N$ fetch and store operations to memories contained in other nodes. Because $O(\log_2 N)$ time is available the only problem is to arrange the data routing so that all the long distance messages, numbering up to $N \log_2 N$, get delivered to their destinations in $O(\log_2 N)$, steps.

The simplest case is that of seclusive PRAMs because this restriction ensures that the traffic pattern that has to be realized is such that at most $\log_2 N$ messages are issued from each node and at most $\log_2 N$ are to be received at each node. This corresponds to $\log_2 N$ simultaneous permutations being realized. Certain two phase algorithms for routing single permutations in $O(\log_2 N)$ time were given by Valiant & Brebner (1981) and Valiant (1982). That $\log_2 N$ permutations could be routed simultaneously in the same time bound is implicit in the work of Upfal (1984).

Note that these algorithms perform randomization, to avoid the danger of 'hot spots'. Nevertheless they provably deliver the last packet in the required time with overwhelming probability, independent of the permutation being realized. Thus the latency and hence the response time of the whole system can be tightly controlled.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

For exclusive PRAMs there is no bound on how many processors may access the same memory block simultaneously. To simulate these efficiently a successful approach is that of randomizing the address space by a pseudo-random hash function in the manner of Mehlhorn & Vishkin (1984). Karlin & Upfal (1986) showed how hashing can be combined with routing to achieve the required performance. Their result is for the butterfly, but is easily adapted to the hypercube. We note that the required performance is achieved only in an amortized sense because, in theory, the address space has to be rehashed, sometimes at considerable cost, but fortunately very rarely.

Lastly, for concurrent PRAMs the switching functions of the cube have to be enhanced by a 'combining' capability. For example, when many read requests are sent to the same word these need to be coalesced as they meet and trace left of the tree of paths of the incoming requests. The content of the word read is broadcast back along the tree with replication at the branch points. An $O(\log_2 N)$ time sorting algorithm (Reif & Valiant 1987) is sufficient to convert a simulation of the exclusive PRAMs to one for the concurrent case (see Ullman 1984, p. 239).

The above results are proved for a 'store and forward' packet routing régime, with a queue for each directed network edge for storing the packets waiting to be sent. The methods of Ranade (1987) enable the queues to be replaced by finite buffers for the packets in all three cases.

The results quoted can be interpreted, in the first instance as providing an existence proof that the necessary routing algorithms exist, and with modest constant multipliers in the runtimes. They also suggest numerous algorithmic ideas for actual implementation, only some of which are currently analysable, and some experimentally testable in the sense of Valiant (1982). The trade-offs that have to be considered become especially complex if the machine is to be tuned for other modes of operation also, such as particular algorithms or processor farms. Note that we are assuming here that on-chip communication time does not have to be accounted for because it is dominated by off-chip communication. This tends to conceal the fact that as the PRAM increases in power, the costs in hardware and time (e.g. for combining networks) can both increase substantially.

It is natural to ask whether substantially more economical solutions to the universality problem exist also. In the hypercube the number of network edges grows faster than the number of nodes by a logarithmic factor. It is easy to argue, however, that this is unavoidable. Consider

$$\kappa = \frac{\text{time to transmit one word between adjacent nodes}}{\text{time to perform one basic computational operation}}.$$

In our formulation, we assume that the simulation has no locality and hence that a typical message traverses about d edges where d is the diameter of the network. Then in an N-processor machine in any one time unit there will be one N computational operations performed. If there is not to be a communication bottleneck there will have to be d times this many, or dN word transmissions performed in the same period. We conclude that in any universal simulation with no locality, in the system overall

$$\gamma = \frac{\text{number of communication lines}}{\text{number of processors}} \geqslant \kappa d.$$

It follows that as the machines scale up the investment in communication has to exceed that in computation by a factor of at least d (because κ will not decrease). As long as $\log_2 N$ is

essentially the smallest diameter that can be hoped for, γ has to grow as $\log_2 N$ at least, as it does for the hypercube.

In the theorem we have assumed that κ is a constant. Hence the main force of the result is that it guarantees in that case the existence of a universal machine with optimal γ (i.e. $\gamma = O(\log_2 N)$). The hypercube is not unique and we could have stated the result equally for the butterfly when just one of the $\log_2 N$ levels of nodes contains processors and the remainder only communications functions. The hypercube does stand out, however, in being desirable as a parallel architecture from other viewpoints also such as suitability for algorithms such as the FFT, for performing grid computations, and for allowing efficient off-line or non-distributed routing.

It is worth noting that in the definition of κ we may choose to think of a computational operation at any granularity that is appropriate. Hence in applications where there is only infrequent interprocessor communication in the PRAM the κ can be thought of as very small, and hence a machine with small γ is sufficient. Thus any parallel machine can be regarded as achieving universality over applications whose κ is small enough that the γ of the machine can support that level of communication.

The main observation of this paper is a universality result for parallel computation that is as efficient as the universality results that are implicitly exploited by all existing conventional sequential machines. We do not wish to underestimate the hardware costs of realizing such a concept. The costs, however, of building parallel machines that fail to realize any universality concept, is considerable also. The programmer would then appear to be doomed to program with explicit consideration for the structure of the particular machine on which his program will run.

I am grateful to the Programming Research Group, Oxford University, for their hospitality. This work was partly supported by grants NSF-DCR-86-00379, ONR-N00014-85-K-0445, and by the Science and Engineering Research Council.

References

Blelloch, G. 1988 *IEEE Trans. Comput.* (In the press.)
Borodin, A. & Hopcroft, J. E. 1985 *J. Comput. Syst. Sci.* **30**, 130–145.
Karlin, A. & Upfal, E. 1986 In *Proc. ACM Symp on Theory of Computing*, pp. 160–168. New York: Association of Computing Machinery.
Karp, R. M. & Ramachandran, V. L. 1989 In *Handbook of theoretical computer science* (ed. J. van Leeuwen). Amsterdam: North Holland. (In the press.)
Mehlhorn, K. & Vishkin, U. 1984 *Acta Informatica* **21**, 339–374.
Reif, J. H. & Valiant, L. G. 1987 *J. Ass. comput. Mach.* **34**, 60–76.
Ranade, A. G. 1987 In *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 185–194. Los Angeles: IEEE.
Schwartz, J. T. 1980 *ACM TOPLAS* **2**, 484–521.
Ullman, J. D. 1984 *Computational aspects of VLSI*. Rockville, Maryland: Computer Science Press.
Upfal, E. 1984 *J. Ass. comput. Mach.* **31**, 507–517.
Valiant, L. G. 1982 *SIAM Jl Comput.* **11**, 350–361.
Valiant, L. G. 1989 In *Handbook of theoretical computer science* (ed. J. van Leeuwen). Amsterdam: North Holland. (In the press.)
Valiant, L. G. & Brebner, G. J. 1981 In *Proc. ACM Symp. on Theory of Computing*, pp. 263–277. New York: Association of Computing Machinery.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES