Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

G.A.P. Kindervater, J.K. Lenstra, A.H.G. Rinnooy Kan

Perspectives on parallel computing

Department of Operations Research and System Theory Report OS-R8709 June

Perspectives on Parallel Computing

G.A.P. Kindervater, J.K. Lenstra
Centre for Mathematics and Computer Science, Amsterdam

A.H.G. Rinnooy Kan
Erasmus University, Rotterdam

Operations research is one of the areas that is likely to benefit from advances in parallel computing. We briefly review what has been achieved in recent years and try to sketch what may be expected in the near future. More realism in theoretical models of parallel computation and more uniformity in available architectures will be required. Formal techniques will have to be developed for the design and implementation of efficient parallel algorithms. Only then can parallelism fulfill its promise and considerably expand the range of effectiveness of operations research methods.

1980 Mathematics Subject Classification: 68M05, 68Q10, 90Bxx, 90Cxx.
Key Words & Phrases: operations research, parallelism, architectures, computations, computational models.
Note: This paper has been submitted for publication.

Over the last 40 years, computers have become faster by a steady series of improvements of their individual components, without fundamental changes in the concept as a whole. Operating speeds are now approaching their physical limits. In spite of all advances, there are still many problems which are unsolvable in reasonable time. The only way to achieve further speedups is through the use of a collection of processors that cooperate in the solution process. Technological developments have made it possible to actually construct parallel computers at moderate costs.

Architectures

Parallel computers built so far differ very much from each other, especially in their two main characteristics: processor capabilities and interprocessor communication. There is no general model which captures all parallel architectures. We can, however, distinguish several classes.

First, there is the class of *vector* machines. In these computers an arithmetic operation is split into a chain of small tasks. A processor performs a specific task and passes the result on to its neighbor. The computation is sped up by the pipelining of independent operations of the same type. Typical examples are the Cray-1, the Cyber-205, and the VP-100.

A second class contains the *single instruction multiple data* (SIMD) machines. At one point in time, the processors perform the same type of operation on local data. Usually, there is a large number (at least one thousand) of miniature processors, each with its own memory. They communicate through a fast fixed *interconnection network*. We mention the *cube connected* network (a hypercube with processors at the vertices and interconnections along the edges), and the *cube connected cycles* network (a hypercube with each of the processors replaced by a cyclicly connected set of processors, each of them having two cycle connections and one edge connection). In most cases, the interconnection network is a *two-dimensional mesh*: each processor is identified with a vertex on a square grid and connected to its horizontal and vertical neighbors. Examples are the Illiac-IV, the ICL/DAP, and the Goodyear/MPP.

Both classes are suitable for regular computations, where many operations of the same type

Report OS-R8709
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

have to be performed in a synchronized fashion. Vectorizing an existing program and adapting it for an SIMD machine are comparable tasks. While such an exercise may lead to an enormous speedup, it may also reveal the rigidity of these architectures. Apart from the restriction to one type of instruction at a time, the requirement that the data have to reside at a specific place often adds to the inflexibility of these machines.

The third class contains the *multiple instruction multiple data* (MIMD) machines. The only theoretical distinction with SIMD machines is that the processors can perform different types of operations at a time. In practice, there are a few (at most two hundred) powerful processors. They communicate through a shared memory or a slow interconnection network. MIMD machines usually operate in an asynchronous mode: a processor runs independently and waits only if information from other processors is needed. Examples of shared memory machines are the Denelcor/HEP, the Cray-XMP, and the New York Ultracomputer. (We note that a true shared memory, which is simultaneously accessed by the processors, is physically infeasible. Read and write instructions must be handled sequentially.) MIMD network computers use a variety of connection patterns; examples are the Crystal multicomputer in Madison (which uses a ring network), the Intel/iPSC (hypercube), the IBM/LCAP (master-slave), and the local area network of work stations in Boulder (ethernet).

Most of these systems are quite flexible. Due to their relative novelty, there is less experience with them than with vector and SIMD machines. In principle, however, MIMD architectures seem to be suitable for broad classes of problems.

Computations

Parallel computers have provided a new playground for computational operations research. All kinds of algorithms have been implemented and tested on the parallel devices that happened to be available.

Most experience has been obtained with numerical algorithms and nonlinear optimization on vector and SIMD machines. Many of the parallel implementations are rather straightforward extensions of traditional sequential methods. However, some truly novel ideas have been developed, most notably in computational linear algebra, and substantial speedups have been realized.

In combinatorial optimization, vector and SIMD machines appear to perform well only as long as the computational process is regular, as is the case in dynamic programming. Many combinatorial algorithms call for more flexible architectures. A branch and bound method, for example, creates a search tree, the structure of which is not known in advance. The natural choice here is an MIMD environment, in which the processors independently explore different parts of the search tree and only communicate if the need occurs. Experience in this direction is limited but promising.

MIMD is similarly convenient for other computational approaches that are based on a successive partitioning of the solution space. One example is the sampling and clustering approach to global optimization.

Computational models

Traditional sequential computers are reasonably represented by models of computation such as the Turing machine and the random access machine. Sequential computers are exchangeable to the extent that the relative efficiency of algorithms is largely machine independent. These observations form the basis of a meaningful complexity theory and a prospering computational practice.

In parallel computing, realistic models are lacking and existing machines are by no means equivalent. We have already indicated that there are many different parallel architectures, which are suitable for very different types of algorithms. Not surprisingly, then, there is no single model of parallel computation that serves to represent reality. Indeed, a model that adequately reflects the actual burden of parallel computation and communication has to incorporate physical features of the computational environment that can be ignored in the sequential case.

A model that has received much attention in theoretical computer science is the parallel random access machine, or PRAM. It is a model for synchronized computations, and about the nicest that the algorithm designer could wish: it has an unlimited number of processors that communicate in unit time through a shared memory.

The PRAM solves all problems belonging to $\mathcal{PSPACE}$, and thereby all problems belonging to $\mathcal{NP}$, in polynomial time. Within the class $\mathcal{P}$ of problems solvable in polynomial sequential time, a further distinction is made. Some problems are solvable in parallel time bounded by a polynomial in just the logarithm of the problem size. Others are *$\mathcal{P}$-complete*, which implies that such a 'superfast' algorithm is unlikely to exist. As examples, we mention sorting as a 'very easy' problem and linear programming as a 'not so easy' one.

The PRAM is a very powerful model. It tells us a lot about the intrinsic parallelism in problems and algorithms. However, unbounded parallelism and unit-time communication are hardly realistic.

Prospects

One may try to cope with the idealistic nature of theoretical models of parallel computation by designing transformations to models that are closer to what we can expect in practice. Examples are the simulations of the PRAM by a network in which each processor has some constant number of connections, and of big networks by smaller ones. What is actually needed, however, is the investigation of severe restrictions on parallelism and communication. As a notable example, a robust theory for models with at most a linear number of processors that communicate over a bounded degree network could serve a very practical purpose.

The main obstacle for the breakthrough of parallel computing is not the lack of reasonable models but the chaos in the real world of architectures. A consensus will hopefully emerge on a single concept of a flexible MIMD computer. Given such a machine, the user should be able to define the type of parallelism he desires, by specifying the hierarchy and communication among his computational processes. The best way to realize this machine is by building it in software and analyzing its performance; at present, the hardware aspects are less relevant. Such a unified architecture requires a flexible set of tools and, in particular, a versatile programming language which (unlike present practice in parallel programming) does not bother the user with the internal structure of the machine. While vector and SIMD computers will probably lose ground as independent machines, they will always be useful as processors that speed up the individual processes in an MIMD configuration.

As to parallel algorithms, we have mentioned theoretical work for the easy problems and mainly experimental work for the hard ones. There is a definite need for a theoretical approach towards the design and analysis of parallel algorithms for the broad class of hard combinatorial and nonlinear optimization problems. One of the few things that have been done in this direction is the investigation of anomalies in parallel branch and bound; for example, adding a processor may slow down the algorithm, and conditions can be given under which this does not occur. For all the various kinds of search methods, the fundamental question is how the computational effort has to be distributed over the processors and how the communication has to be arranged so as to obtain a maximum speedup. Operations researchers are well positioned to model and solve this complicated design problem. In the words of Richard M. Karp: 'Even though you may never be able to go from exponential to polynomial, it's also clear that there is a tremendous scope for parallelism on those hard problems, and parallelism may really help us curb combinatorial explosions.'

Bibliographical notes

Beasley (1986) gives a literature survey on the use of supercomputers in operations research. Kindervater and Lenstra (1986) and Ribeiro (1987) present a more detailed review of parallelism in combinatorial optimization. A few relevant references in parallel computing are quoted below; many others can be found in these surveys.

Architectures are dealt with by Hockney and Jesshope (1981), Almasi (1985), and Dongarra and Duff (1985). Flynn (1966) invented the SIMD-MIMD classification.

Computations for parallel numerical mathematics are reviewed by Dongarra, Gustavson, and Karp (1984) and Ortega and Voigt (1985). Parallel nonlinear optimization algorithms are reported by Van Laarhoven (1985), Plummer, Lasdon, and Ahmed (1985), and Zenios and Mulvey (1985). A parallel implementation of a global optimization algorithm is given by Byrd, Dert, Rinnooy Kan and Schnabel (1986). For the parallelization of dynamic

programming and branch and bound, see Finkel and Manber (1985), Kindervater and Trienekens (1987), and Trienekens (1986). Lai and Sahni (1984), Lai and Sprague (1985, 1986), and Li and Wah (1986) investigate anomalies in parallel tree search; Boxma and Kindervater (1987) analyze the performance of such methods on a master-slave architecture.

Computational models, with emphasis on the PRAM, and the resulting complexity concepts are described by Johnson (1983); another review of this area is given by Cook (1981). Batcher (1968) and Ajtai, Komlós, and Szemerédi (1983) present polylog parallel sorting algorithms; Dobkin, Lipton, and Reiss (1979) proved $\mathcal{P}$-hardness of linear programming. For simulations of various models of parallel computation, see, for example, Alt, Hagerup, Mehlhorn, and Preparata (1986), Karlin and Upfal (1986), and Bodlaender (1986).

The quotation is from Karp's Turing Award interview (Frenkel 1986).

ACKNOWLEDGEMENTS

The authors gratefully acknowledge valuable suggestions from D.B. Shmoys. This research was partially supported by his NSF Presidential Young Investigator Award.

BIBLIOGRAPHY

M. AJTAI, J. KOMLÓS, E. SZEMERÉDI (1983). Sorting in $c \log n$ parallel steps. *Combinatorica 3*, 1-19.

G.S. ALMASI (1985). Overview of parallel processing. *Parallel Comput. 2*, 191-203.

H. ALT, T. HAGERUP, K. MEHLHORN, F.P. PREPARATA (1986). Deterministic simulation of idealized parallel computers on more realistic ones. J. GRUSKA, B. ROVAN, J. WIEDERMANN (eds.). *Mathematical Foundations of Computer Science 1986*, Lecture Notes in Computer Science 233, Springer, Berlin, 199-208.

K.E. BATCHER (1968). Sorting networks and their applications. *Proc. AFIPS Spring Joint Computer Conf. 32*, 307-314.

J.E. BEASLEY (1986). *Supercomputers and OR*, Manuscript, Department of Management Science, Imperial College of Science and Technology, London.

H.A. BODLAENDER (1986). *Distributed Computing: Structure and Complexity*, Ph.D. thesis, Department of Computer Science, University of Utrecht.

O.J. BOXMA, G.A.P. KINDERVATER (1987). *A Model for Analyzing a Class of Branch and Bound Algorithms on a Master-Slave Architecture*, Centre for Mathematics and Computer Science, Amsterdam, in preparation.

R.H. BYRD, C.L. DERT, A.H.G. RINNOOY KAN, R.B. SCHNABEL (1986). *Concurrent Stochastic Methods for Global Optimization*, Report 8637/A, Econometric Institute, Erasmus University, Rotterdam.

S.A. COOK (1981). Towards a complexity theory of synchronous parallel computation. *Enseign. Math. (2) 27*, 99-124.

D. DOBKIN, R.J. LIPTON, S. REISS (1979). Linear programming is log-space hard for *P*. *Inform. Process. Lett. 8*, 96-97.

J.J. DONGARRA, I.S. DUFF (1985). *Advanced Architecture Computers*, Technical memorandum 57, Mathematics and Computer Science Division, Argonne National Laboratory.

J.J. DONGARRA, F.G. GUSTAVSON, A. KARP (1984). Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Rev. 26*, 91-112.

R. FINKEL, U. MANBER (1985). *DIB - a Distributed Implementation of Backtracking*, Technical report 588, Computer Sciences Department, University of Wisconsin, Madison.

M.J. FLYNN (1966). Very high-speed computing systems. *Proc. IEEE 54*, 1901-1909.

K.A. FRENKEL (1986). Complexity and parallel processing: an interview with Richard Karp. *Comm. ACM 19*, 112-117.

R.W. HOCKNEY, C.R. JESSHOPE (1981). *Parallel Computers: Architecture, Programming and Algorithms*, Hilger, Bristol.

D.S. JOHNSON (1983). The NP-completeness column: an ongoing guide; seventh edition. *J. Algorithms 4*, 189-203.

A.R. KARLIN, E. UPFAL (1986). Parallel hashing - an efficient implementation of shared memory (preliminary version). *Proc. 18th Annual ACM Symp. Theory of Computing*, 160-168.

G.A.P. KINDERVATER, J.K. LENSTRA (1986). *Parallel Computing in Combinatorial Optimization*, Report OS-R8614, Centre for Mathematics and Computer Science, Amsterdam.

G.A.P. KINDERVATER, H.W.J.M. TRIENEKENS (1987). Experiments with parallel algorithms for combinatorial problems, *European J. Oper. Res.*, to appear.

T.-H. LAI, S. SAHNI (1984). Anomalies in parallel branch-and-bound algorithms. *Comm. ACM 27*, 594-602.

T.-H. LAI, A. SPRAGUE (1985). Performance of parallel branch-and-bound algorithms. *IEEE Trans. Comput. C-34*, 962-964.

T.-H. LAI, A. SPRAGUE (1986). A note on anomalies in parallel branch-and-bound algorithms with one-to-one bounding functions. *Inform. Process. Lett. 23*, 119-122.

G.-J. LI, B.W. WAH (1986). Coping with anomalies in parallel branch-and-bound algorithms. *IEEE Trans. Comput. C-35*, 568-573.

J.M. ORTEGA, R.G. VOIGT (1985). Solution of partial differential equations on vector and parallel computers. *SIAM Rev. 27*, 149-240.

J. PLUMMER, L. LASDON, M. AHMED (1985). *Solving a Large Nonlinear Programming Problem on a Vector Processing Computer*, Working paper 85/86-3-1, Department of General Business, College of Business Administration, University of Texas, Austin.

C.C. RIBEIRO (1987). Parallel computer models and combinatorial algorithms. *Ann. Discrete Math. 31*, 325-364.

H.W.J.M. TRIENEKENS (1986). *Parallel Branch and Bound on an MIMD System*, Report 8640/A, Econometric Institute, Erasmus University, Rotterdam.

P.J.M. VAN LAARHOVEN (1985). Parallel variable metric algorithms for unconstrained optimization. *Math. Programming 33*, 68-81.

S.A. ZENIOS, J.M. MULVEY (1985). *Nonlinear Programming on Vector Supercomputers*, Report EES-85-13, Department of Civil Engineering, Princeton University.