8

Parallel Algorithms

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

CONTENTS

Parallel computing is receiving a rapidly increasing amount of attention. In theory, a collection of processors that operate in parallel can achieve substantial speedups. In practice, technological developments are leading to the actual construction of such devices at low cost. Given the inherent limitations of traditional sequential computers, these prospects turn out to be very stimulating for researchers interested in the design of computers and algorithms.

In this bibliography, we have tried to collect the literature on parallel computing that is relevant for the mathematics of operations research, in particular for the theory of combinatorial optimization. Its organization is as follows.

§1 is concerned with *machine models* designed for parallel computation.

Rather than including the complete literature on this topic, which could fill a sizeable bibliography by itself, we have only surveyed papers that are of general interest or that define models referred to in later sections. Many of the references in §4 and §5 mention machine models for which specific results have been obtained, and the reader who is interested in the characteristics of, say, an *SIMD machine with shared memory, simultaneous reads and no simultaneous writes* should consult §1.

§2 deals with the *complexity theory* of parallel computation. Beyond the basic distinction between *solvability in polynomial time* and *completeness for* $\mathcal{NP}$ in sequential computation, many concepts have been defined and analyzed that are relevant for parallel computing. Again, we have not aimed at a complete survey of this area, but important notions like *solvability in polylog parallel time* and *log space completeness for* $\mathcal{P}$ are introduced here.

§3 gives results for *numerical problems.* Problems like evaluating arithmetic expressions and recurrence relations, solving systems of linear equations and computing eigenvalues have been subjected to parallelization earlier and more extensively than combinatorial problems (*l'histoire se répète*: floating point arithmetic was well understood before anyone had heard of the traveling salesman). §§3.1−2 list references on those subjects, without much comment. §3.3 contains three papers on *nonlinear optimization,* an area in which parallel computing finds natural and potentially promising applications.

§4 reviews 51 papers on *elementary combinatorial problems*: typical subjects from *computer science* like finding the maximum, merging and sorting in §4.1, and problems from *algorithmic graph theory* like finding connected components, spanning trees and shortest paths in §4.2. In each case, the papers are grouped together according to the type of machinery involved, such as general parallel computers with a shared memory and specific fixed interconnection networks.

§5 finally discusses parallelism in *combinatorial optimization.* We have been able to find 18 papers on linear programming, maximum flow, knapsack, traveling salesman and scheduling problems and on dynamic programming and branch-and-bound methods. The formidable power of parallel computing in conjunction with the firm roots of combinatorial optimization in the theory of design and analysis of algorithms and computational complexity seems to hold great promise for a further development of this area in the very near future.

We are grateful to E.L. Lawler, J. van Leeuwen, F. Maffioli and G.L. Nemhauser, who brought many papers to our attention.

1. Models

1.1. *Classification and surveys*

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

Four classes of parallel computers are introduced:

(1) SISD: *single instruction stream—single data stream*; one instruction is performed at a time, on one set of data; e.g., $a+b$.
(2) SIMD: *single instruction stream—multiple data stream*; one type of instruction is performed at a time, possibly on different data; e.g., $a+b$ and $c+d$.
(3) MISD: *multiple instruction stream—single data stream*; different instructions on the same data can be performed at a time; e.g., $a+b$ and $a-b$.
(4) MIMD: *multiple instruction stream—multiple data stream*; different instructions on different data can be performed at a time; e.g., $a+b$ and $c-d$.

Beyond Flynn's classification scheme, it makes sense to subdivide the last class into *synchronized* machines, which wait for each other after each set of instructions and then perform the next set, and *asynchronous* machines, which run independently and wait only if information from other processors is needed. *Systolic* algorithms are highly synchronized processes: the processing elements act rhythmically on regular streams of data passing through the network. *Distributed* algorithms are typical asynchronous processes: the processors perform their own local algorithms and communicate by sending messages every now and then.

J.T. Schwartz (1980). Ultracomputers. *ACM Trans. Programming Languages and Systems 2,* 484-521.

Distinction is made between *paracomputers*, where the processors have simultaneous access to a *shared memory,* and *ultracomputers,* where each processor communicates directly with a fixed number of other processors through an *interconnection network.* Whereas paracomputers are primarily of theoretical interest, ultracomputers are more realistic and can be quite efficient at the same time.

Important in this bibliography (although not dealt with by Schwartz) is the way in which shared memory computers handle *read* and *write conflicts,* which occur when several processors try to read from or to write into the same memory location at the same time. If read [write] conflicts are (dis-)allowed, we speak of (*no*) *simultaneous reads* [*writes*].

G. Ausiello, P. Bertolazzi (1982). Parallel computer models: an introduction. *IBM Symp. Parallel Processing,* Rome, March 1982.

In this introduction to models for parallel computation, both theoretical and practical models are considered.

L.S. Haynes, R.L. Lau, D.P. Siewiorek, D.W. Mizell (1982). A survey of highly parallel computing. *IEEE Comput. 15.1,* 9-24.

A survey of the different types of practical parallel computer structures is given.

L.G. Valiant (1983). Parallel computation. J.W. de Bakker, J. van Leeuwen (eds.). *Foundations of Computer Science IV, Distributed Systems: Part 1, Algorithms and Complexity,* Mathematical Centre Tract 158, Centre for

Mathematics and Computer Science, Amsterdam, 35-48.

This review discusses characteristics of problems that make them amenable to fast parallel computation, as well as realistic computer architectures that are suitable for such computations.

U. Vishkin (1983). *Synchronous Parallel Computation - a Survey,* Preprint, Courant Institute, New York University.

A survey of theoretical models for parallel computation (for which existing algorithms are reviewed) and of their relation to realistic machines.

1.2. *Interconnection networks*

S.H. Unger (1958). A computer oriented toward spatial problems. *Proc. IRE 46,* 1744-1750.

Introduction of the two-dimensional *mesh connected* network. Each processor is identified with an ordered pair (i,j) $(i,j = 1,...,n)$ and processor (i,j) is connected to processors $(i \pm 1,j)$ and $(i,j \pm 1)$, provided they exist.

J.S. Squire, S.M. Palais (1963). Programming and design considerations of a highly parallel computer. *Proc. AFIPS Spring Joint Computer Conf. 23,* 395-400.

Description of the *cube connected* network. It can be seen as a hypercube with processors at the vertices and interconnections along the edges.

H.S. Stone (1971). Parallel processing with the perfect shuffle. *IEEE Trans. Comput. C-20,* 153-161.

A network with interconnections that imitate a *perfect shuffle* of a deck of cards.

J.L. Bentley, H.T. Kung (1979). A tree machine for searching problems. *Proc. 1979 Internat. Conf. Parallel Processing,* 257-266.

The interconnection pattern consists of two *binary trees* with common leaves.

F.P. Preparata, J. Vuillemin (1981). The cube-connected cycles: a versatile network for parallel computation. *Comm. ACM 24,* 300-309.

The *cube connected cycles* network can be seen as a cube connected network with each processor replaced by a cyclicly connected series of processors. Each of them is connected to at most three others.

H.J. Siegel (1977). Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks. *IEEE Trans Comput. C-26,* 153-161.

H.J. Siegel (1979). A model of SIMD machines and a comparison of various interconnection networks. *IEEE Trans. Comput. C-28,* 907-917.

Both papers deal with the comparison of interconnection networks. Techniques for simulating one network by another are given.

Z. Galil, W.J. Paul (1983). An efficient general-purpose parallel computer. *J. Assoc. Comput. Mach. 30,* 360-387.

A universal parallel computer, which can simulate any reasonable parallel machine efficiently.

F.P. Preparata (1982). Algorithm design and VLSI architectures. *IBM Symp. Parallel Processing,* Rome, March 1982.

Outline of desirable features for VLSI implementable networks. Some specific interconnection networks are considered in detail.

2. Complexity

2.1. *Surveys*

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

This expository paper surveys machine models and complexity classes for parallel computations.

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

Section 2 of this edition is a brief review of the complexity theory of parallel computing.

A *parallel RAM* with an unbounded number of processors, shared memory, simultaneous reads and no simultaneous writes is introduced, for which the *parallel computation thesis* (see §2.3) holds: the class of languages it can recognize in polynomial *time* is precisely $\mathcal{P}$SPACE, the class of languages recognizable by a sequential machine in polynomial *space*. If only a polynomial number of processors is allowed, the class of languages recognizable in parallel polynomial time shrinks from $\mathcal{P}$SPACE to $\mathcal{P}$, the class of languages recognizable in sequential polynomial time.

Many problems can be solved in *polylog parallel time,* i.e., time that is polynomially bounded in the logarithm of problem size (with unbounded parallelism); see §§3–5 for examples. By the parallel computation thesis, these problems would form the class POLYLOGSPACE of problems solvable in polylog sequential space. Other problems have been shown to be *log space complete for* $\mathcal{P}$, i.e., (i) they belong to $\mathcal{P}$ and (ii) each problem in $\mathcal{P}$ can be reduced to any of them by a transformation using logarithmic work space; see §2.2 and §5.2 for examples. If any such problem can be solved in polylog space, then $\mathcal{P} \subseteq$ POLYLOGSPACE. Since this inclusion is not expected to be true, such problems are unlikely to be solvable in polylog space or in polylog parallel time.

New classes arise if *simultaneous resource bounds* (see §2.4) are imposed.

E.g., $\mathcal{NC}$ is the class of problems solvable in polylog parallel time using only a polynomial number of processors, and $\mathcal{SC}$ is the class of problems solvable in polynomial sequential time using polylog space. Research is oriented towards questions like $\mathcal{NC}=?\mathcal{SC}$.

2.2. *Log space completeness for* $\mathcal{P}$

S.A. Cook (1974). An observation on time-storage trade off. *J. Comput. System Sci. 9,* 308-316.

A *path system* is a quadruple $\mathcal{S} = <X,R,S,U>$, where X is a finite set of nodes, R is a three place incidence relation on X, $S \subset X$ is a set of source nodes, and $U \subset X$ is a set of terminal nodes. $\mathcal{S}$ is *solvable* if at least one node in S is contained in the least set A such that $U \subset A$ and such that, if $y,z \in A$ and $R(x,y,z)$ holds, then $x \in A$. Cook shows that each language of time complexity $T(n)$ is reducible in storage $\log(T(n))$ to the set of strings coding solvable path systems.

N.D. Jones, W.T. Laaser (1977). Complete problems for deterministic polynomial time. *Theoret. Comput. Sci. 3,* 105-117.

The *unit resolution problem* is the problem of determining whether a propositional formula in conjunctive normal form can be proved unsatisfiable by, roughly speaking, substituting the literals in unit clauses. This problem is shown to be log space complete for $\mathcal{P}$. Corollaries give similar results for other problems. For an application, see [Dobkin, Lipton & Reiss 1979] (§ 5.2(b)).

R.E. Ladner (1975). The circuit value problem is log space complete for $\mathcal{P}$. *SIGACT News 7.1,* 18-20.

The *circuit value* problem is the problem of determining the output of a circuit consisting of AND and NOT gates, given its input. This problem is shown to be log space complete for $\mathcal{P}$ by simulating Turing machines by combinatorial circuits.

L.M. Goldschlager (1977). The monotone and planar circuit value problems are log space complete for P. *SIGACT News 9.2,* 25-29.

A circuit is *monotone* if it consists of AND and OR gates; it is *planar* if it has a cross free planar embedding. The monotone and planar circuit value problems are shown to be log space complete for $\mathcal{P}$ by a log space transformation from the circuit value problem (see [Ladner 1975] above). For an application, see [Goldschlager, Shaw & Staples 1982] (§ 5.2(a)).

2.3. *Parallel time versus sequential space*

A.K. Chandra, D.C. Kozen, L.J. Stockmeyer (1981). Alternation. *J. Assoc. Comput. Mach. 28,* 114-133.

L.M. Goldschlager (1982). A universal connection pattern for parallel

computers. *J. Assoc. Comput. Mach. 29,* 1073-1086.

Statement of a hypothesis known as the *parallel computation thesis*: *time bounded parallel machines are polynomially related to space bounded sequential machines*; that is, for any function $T(n)$, the class of languages recognizable by a machine with unbounded parallelism in time $T(n)^{O(1)}$ (i.e., polynomial in $T(n)$) is equal to the class of languages recognizable by a sequential machine in space $T(n)^{O(1)}$. Evidence is given by proving the thesis for some well-behaved time bounds $T(n)$ on several parallel machine models.

J. Hartmanis, J. Simon (1974). On the power of multiplication in random access machines. *Proc. 15th Annual ACM Symp. Switching and Automata Theory,* 13-23.

V.R. Pratt, L.J. Stockmeyer (1976). A characterization of the power of vector machines. *J. Comput. System Sci. 12,* 198-221.

W.J. Savitch, M.J. Stimson (1979). Time bounded random access machines with parallel processing. *J. Assoc. Comput. Mach. 26,* 103-118.

S. Fortune, J. Wyllie (1978). Parallelism in random access machines. *Proc. 10th Annual ACM Symp. Theory of Computing,* 114-118.

A. Borodin (1977). On relating time and space to size and depth. *SIAM J. Comput. 6,* 733-744.

J.H. Reif (1982). On the power of probabilistic choice in synchronous parallel computations. M. Nielsen, E.M. Schmidt (eds.). *Proc. 9th Internat. Coll. Automata, Languages and Programming,* Lecture Notes in Computer Science 140, Springer, Berlin, 442-450.

These papers further support the parallel computation thesis.

2.4. *Simultaneous resource bounds*

N. Pippenger (1979). On simultaneous resource bounds (preliminary version). *Proc. 20th Annual IEEE Symp. Foundations of Computer Science,* 307-311.

W.L. Ruzzo (1981). On uniform circuit complexity. *J. Comput. System Sci. 22,* 365-383.

P.W. Dymond, S.A. Cook (1980). Hardware complexity and parallel computation (preliminary version). *Proc. 21st Annual IEEE Symp. Foundation of Computer Science,* 360-372.

J.W. Hong (1980). On similarity and duality of computation (extended abstract). *Proc. 21st Annual IEEE Symp. Foundations of Computer Science,* 348-359.

These papers investigate an extended version of the parallel computation thesis, formulated as follows in [Dymond & Cook 1980]: (i) *parallel time and hardware requirements are simultaneously polynomially related to sequential (Turing machine) reversal and space requirements*; (ii) *parallel time and space requirements are polynomially related.*

3. Numerical Problems

3.1. *Evaluation of expressions and recurrence relations*

R. Brent, D. Kuck, K. Maruyama (1973). The parallel evaluation of arithmetic expressions without division. *IEEE Trans. Comput. C-22,* 532-534.
D. Kuck, Y. Muraoka (1974). Bounds on the parallel evaluation of arithmetic expressions using associativity and commutativity. *Acta Inform. 3,* 203-216.
R.P. Brent (1973). The parallel evaluation of arithmetic expressions in logarithmic time. J.F. Traub (ed.). *Complexity of Sequential and Parallel Numerical Algorithms,* Academic Press, New York, 83-102.
R.P. Brent (1974). The parallel evaluation of general arithmetic expressions. *J. Assoc. Comput. Mach. 21,* 201-206.
D.J. Kuck, K. Maruyama (1975). Time bounds on the parallel evaluation of arithmetic expressions. *SIAM J. Comput. 4,* 147-162.
D.E. Muller, F.P. Preparata (1976). Restructuring of arithmetic expressions for parallel evaluation. *J. Assoc. Comput. Mach. 23,* 534-543.
S. Winograd (1975). On the parallel evaluation of certain arithmetic expressions. *J. Assoc. Comput. Mach. 22,* 477-492.
I. Munro, M. Paterson (1973). Optimal algorithms for parallel polynomial evaluation. *J. Comput. System Sci. 7,* 189-198.
K. Maruyama (1973). On the parallel evaluation of polynomials. *IEEE Trans. Comput. C-22,* 2-5.
L. Hyafil (1979). On the parallel evaluation of multivariate polynomials. *SIAM J. Comput. 8,* 120-123.
L.G. Valiant (1980). Computing multivariate polynomials in parallel. *Inform. Process. Lett. 11,* 44-45.
L.G. Valiant, S. Skyum (1981). Fast parallel computation of polynomials using few processors. J. Gruska, M. Chytil (eds.). *Mathematical Foundations of Computer Science 1981,* Lecture Notes in Computer Science 118, Springer, Berlin, 132-139.

These twelve papers deal with the parallel evaluation of arithmetic expressions. The results differ with respect to the types of expressions considered (e.g., expressions with or without division, polynomials) and the transformations allowed (using associativity, commutativity, etc.). There is also a distinction between bounded and unbounded parallelism.

Of general importance is a lemma from [Brent 1974]: *if a computation can be performed in time t with q operations and sufficiently many processors that perform arithmetic operations in unit time, then it can be performed in time $t+(q-t)/p$ with p such processors.*

P.M. Kogge, H.S. Stone (1973). A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput. C-22,* 786-793.
P.M. Kogge (1974). Parallel solution of recurrence problems. *IBM J. Res.*

Develop. 18, 138-148.
S.-C. Chen, D.J. Kuck (1975). Time and parallel processor bounds for linear recurrence systems. *IEEE Trans. Comput. C-24,* 701-717.
H.T. Kung (1976). New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. *J. Assoc. Comput. Mach. 23,* 252-261.
L. Hyafil, H.T. Kung (1977). The complexity of parallel evaluation of linear recurrences. *J. Assoc. Comput. Mach. 24,* 513-521.
D.D. Gajski (1981). An algorithm for solving linear recurrence systems on parallel and pipelined machines. *IEEE Trans. Comput. C-30,* 190-206.
A.C. Greenberg, R.E. Ladner, M.S. Paterson, Z. Galil (1982). Efficient parallel algorithms for linear recurrence computation. *Inform. Process. Lett. 15,* 31-35.

These seven papers outline the results obtained on solving recurrence relations. Several types of such relations are attacked succesfully, although for the first-order recurrence problem p processors can achieve a speedup of at most $(2p+1)/3$ [Hyafil & Kung 1977].

3.2. *Numerical analysis and algebra*

D. Heller (1978). A survey of parallel algorithms in numerical linear algebra. *SIAM Rev. 20,* 740-777.

A survey of parallel techniques for problems in numerical linear algebra, such as the solution of systems of linear equations and the computation of eigenvalues, covering the literature up to 1977.

W.M. Gentleman (1978). Some complexity results for matrix computations on parallel processors. *J. Assoc. Comput. Mach. 25,* 112-115.
M.A. Franklin (1978). Parallel solution of ordinary differential equations. *IEEE Trans. Comput. C-27,* 413-420.
J.M. Lemme, J.R. Rice (1979). Speedup in parallel algorithms for adaptive quadrature. *J. Assoc. Comput. Mach. 26,* 65-71.
C.R. Jesshope (1980). The implementation of fast radix 2 transforms on array processors. *IEEE Trans. Comput. C-29,* 20-27.
O. Wing, J.W. Huang (1980). A computation model of parallel solution of linear equations. *IEEE Trans. Comput. C-29,* 632-638.
J.A.G. Jess, H.G.M. Kees (1982). A data stucture for parallel L/U decomposition. *IEEE Trans. Comput. C-31,* 231-239.
A. Borodin, J. Von Zur Gathen, J. Hopcroft (1982). Fast parallel matrix and GCD computations. *Inform. and Control 52,* 241-256.
D.J. Evans, R.C. Dunbar (1983). The parallel solution of triangular systems of equations. *IEEE Trans. Comput. C-32,* 201-204.
C.P. Arnold, M.I. Parr, M.B. Dewe (1983). An efficient parallel algorithm for the solution of large sparse linear matrix equations. *IEEE Trans. Comput. C-32,* 265-272.

J. Von Zur Gathen (1983). Parallel algorithms for algebraic problems. *Proc. 15th Annual ACM Symp. Theory of Computing,* 17-23.

More recent publications on a wide variety of problems in this very lively research area, which is, however, not of immediate interest to the theory of combinatorial optimization.

3.3. *Nonlinear optimization*

J.J. McKeown (1980). Aspects of parallel computation in numerical optimization. F. Archetti, M. Cugiani (eds.). *Numerical Techniques for Stochastic Systems,* North-Holland, Amsterdam, 297-327.

Global optimization algorithms are adapted for SIMD and MIMD computers. Parallelization is only considered at a high level, e.g. concerning the number of parallel function evaluations and local optimizations.

M.A. Franklin, N.L. Soong (1981). One-dimensional optimization on multiprocessor systems. *IEEE Trans. Comput. C-30,* 61-66.

The trade-off between two strategies for optimizing one-dimensional functions on MIMD systems is analyzed. The first strategy evaluates the function in parallel, the second one computes several function values at a time.

L.C.W. Dixon, K.D. Patel (1982). The place of parallel computation in numerical optimisation; VI parallel algorithms for nonlinear optimisation. *IBM Symp. Parallel Processing,* Rome, March 1982.

The modified Newton algorithm for nonlinear programming is parallelized, and results of an implementation on the ICL/DAP SIMD computer are presented.

4. Combinatorics

4.1. *Sorting and related problems*

(a) *sorting networks*

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

Networks are presented that sort n keys in $O(\log^2 n)$ time using $O(n \log^2 n)$ comparison elements. They are based on the principle of iterated merging. One network uses bitonic sequences, the other merges two ordered lists by first merging the odd and even numbered keys from both lists separately and then comparing the results.

D.E. Muller, F.P. Preparata (1975). Bounds to complexities of networks for sorting and for switching. *J. Assoc. Comput. Mach. 22,* 195-201.

A network with $O(n^2)$ elements for sorting n numbers in $O(\log n)$ time,

based on enumeration sort.

M. Ajtai, J. Komlós, E. Szemerédi (1983). An $O(n \log n)$ sorting network. *Proc. 15th Annual ACM Symp. Theory of Computing,* 1-9.

A network with only $O(n \log n)$ comparison elements for sorting n numbers in $O(\log n)$ time. The basis is a network with $O(n)$ elements that splits the set of numbers in a lower and an upper half in constant time with only few errors.

(b) *shared memory computers: merging*

F. Gavril (1975). Merging with parallel processors. *Comm. ACM 18,* 588-591.

Merging two ordered sets using a small number of processors. The algorithm first splits the sets in an appropriate way and then merges the smaller parts in parallel. The processors merge the subsets sequentially.

R.H. Barlow, D.J. Evans, J. Shanehchi (1981). A parallel merging algorithm. *Inform. Process. Lett. 13,* 103-106.

Merging k sorted lists using $p \leqslant k$ processors. From one list $k-1$ elements are chosen and their place in the other lists is determined. The processors then merge the sublists obtained sequentially. The behavior of the algorithm strongly depends on the input.

(c) *shared memory computers: sorting*

S. Even (1974). Parallelism in tape-sorting. *Comm. ACM 17,* 202-204.

Synchronized MIMD.

Sorting algorithms using merge sort. They have an optimial speedup as long as the number of processors is small relative to input size.

S. Todd (1978). Algorithm and hardware for a merge sort using multiple processors. *IBM J. Res. Develop. 22,* 509-517.

Synchronized MIMD.

A parallel version of the straight merge sort algorithm. It runs in $O(n)$ time using $\log n$ processors and can be implemented in hardware.

L.G. Valiant (1975). Parallelism in comparison problems. *SIAM J. Comput. 4,* 348-355.

Valiant explores the parallelism in problems like finding the maximum, merging and sorting. If only comparisons are counted and the overhead is neglected and if the input size n is not less than the number p of processors, speedups of $\Omega(p/\log \log p)$ can be achieved. For example, $n/2$ processors can sort n keys in $O(\log n \log \log n)$ steps.

D.S. Hirschberg (1978). Fast parallel sorting algorithms. *Comm. ACM 21,*

657-661.

SIMD, shared memory, simultaneous reads, no similtaneous writes.

An algorithm to sort n keys in $O(k \log n)$ time using $n^{1+1/k}$ processors. It employs the result from [Gavril 1975] (see §4.1(b)) and a parallel bucket sort routine.

F.P. Preparata (1978). New parallel-sorting schemes. *IEEE Trans. Comput. C-27,* 669-673.

SIMD, shared memory, (no) simultaneous reads, no simultaneous writes.

Two algorithms are described to sort n numbers with enumeration sort. The first one allows read conflicts, uses the merging scheme from [Valiant 1975] (see above) and runs in $O(\log n)$ time on $n \log n$ processors, disregarding some of the overheads. The second one disallows read conflicts, uses the odd-even merging scheme from [Batcher 1968] (see §4.1(a)) and runs in $O(k \log n)$ time on $n^{1+1/k}$ processors.

Y. Shiloach, U. Vishkin (1981). Finding the maximum, merging, and sorting in a parallel computation model. *J. Algorithms 2,* 88-102.

Synchronized MIMD, shared memory, simultaneous reads, simultaneous writes provided the same value is written.

The maximum finding algorithm from [Valiant 1975] (see above) is implemented so as to achieve the same time bound while counting the overheads. Further, a merge sort algorithm is given, free of write conflicts and having the same time and processor complexities as those from [Hirschberg 1978] and [Preparata 1978] (see above).

R. Reischuk (1981). A fast probabilistic parallel sorting algorithm. *Proc. 22nd Annual IEEE Symp. Foundations of Computer Science,* 212-219.

Synchronized MIMD, shared memory, simultaneous reads, no simultaneous writes.

An algorithm to sort n keys in $O(\log n)$ average time using n processors. The set of keys is partitioned into $\lfloor \sqrt{n} \rfloor + 1$ groups, which have size $O(\sqrt{n} \log n)$ with probability close to 1, and next the groups are sorted separately.

A. Borodin, J.E. Hopcroft (1982). Routing, merging and sorting on parallel models of computation; extended abstract. *Proc. 14th Annual ACM Symp. Theory of Computing,* 338-344.

In fixed connection networks with indegrees d, oblivious routing strategies require $\Omega(\sqrt{n}/d^{3/2})$ time. On a synchronized MIMD computer with shared memory, simultaneous reads but no simultaneous writes, the merging and sorting schemes from [Valiant 1975] (see above) are implemented such that the running time is of the same order as the number of comparison steps.

C.P. Kruskal (1982). Results in parallel searching, merging and sorting

(summary). *Proc. 1982 Internat. Conf. Parallel Processing,* 196-198.

Synchronized MIMD, shared memory, simultaneous reads, no simultaneous writes.

Improvements on the results from [Valiant 1975] (see above). E.g., a merge sort algorithm is given that sorts n keys in $O(\log n \log\log n / \log\log\log n)$ time using n processors.

M. Aigner (1982). Parallel complexity of sorting problems. *J. Algorithms 3,* 79-88.

Synchronized MIMD, no simultaneous reads, no simultaneous writes.

Lower and upper bounds are given on the number of comparison steps needed for selection, merging and sorting.

(d) *shared memory computers: convex hull*

D. Nath, S.N. Maheshwari, P.C.P. Bhatt (1981). Parallel algorithms for the convex hull problem in two dimensions. W. Händler (ed.). *CONPAR 81,* Lecture Notes in Computer Science 111, Springer, Berlin, 358-372.

SIMD, shared memory, (no) simultaneous reads, no simultaneous writes.

If read conflicts are allowed, the convex hull of n points in the plane can be found in $O((n/p)\log n + \log p \log n)$ time using $p \leqslant n$ processors and in $O(k \log n)$ time using $n^{1+1/k}$ processors ($k \leqslant \log n$). If read conflicts are disallowed, the same bounds still hold. After an initial sort of the points on one of the coordinates, the algorithms use a divide and conquer strategy.

(e) *mesh connected networks: permuting and sorting*

S.E. Orcutt (1976). Implementation of permutation functions in Illiac IV-type computers. *IEEE Trans. Comput. C-25,* 929-936.

SIMD, $n \times n$ mesh connected network.

This implementation of the bitonic sort from [Batcher 1968] (see §4.1(a)) performs a permutation of the n^2 elements in $O(n \log n)$ time.

C.D. Thompson, H.T. Kung (1977). Sorting on a mesh-connected computer. *Comm. ACM 20,* 263-271.

SIMD, $n \times n$ mesh connected network.

Sorting n^2 elements in snake-like (or shuffled) row-major order in $O(n)$ time, based on the odd-even (bitonic) sort from [Batcher 1968] (see §4.1(a)).

D. Nassimi, S. Sahni (1979). Bitonic sort on a mesh-connected parallel computer. *IEEE Trans. Comput. C-28,* 2-7.

SIMD, $n \times n$ mesh connected network.

Sorting n^2 elements in row-major order in $O(n)$ time by an adaptation of the bitonic sort from [Batcher 1968] (see §4.1(a)) different than that from [Thompson & Kung 1977] (see above).

D. Nassimi, S. Sahni (1980). An optimal routing algorithm for mesh-connected parallel computers. *J. Assoc. Comput. Mach. 27,* 6-29.

SIMD, k-dimensional mesh connected network ($k \geqslant 2$).

An algorithm for permuting data, which is optimal in the sense that it uses the minimum number of unit distance routing steps.

M. Kumar, D.S. Hirschberg (1983). An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes. *IEEE Trans. Comput. C-32,* 254-264.

SIMD, $n \times n$ mesh connected network.

Another algorithm for sorting n^2 elements in $O(n)$ time based on [Batcher 1968] (see §4.1(a)).

H.-W. Lang, M. Schimmler, H. Schmeck, H. Schröder (1983). A fast sorting algorithm for VLSI. J. Díaz (ed.). *Proc. 10th Internat. Coll. Automata, Languages and Programming,* Lecture Notes in Computer Science 154, Springer, Berlin, 408-419.

Synchronized MIMD, $n \times n$ mesh connected network.

An algorithm for sorting n^2 elements in $O(n)$ time, based on odd-even transposition sort. A systolic version is presented that runs in $O(n)$ time using $O(n^2)$ cells.

(f) *other interconnection networks: permuting and sorting*

G. Baudet, D. Stevenson (1978). Optimal sorting algortihms for parallel computers. *IEEE Trans. Comput. C-27,* 84-87.

SIMD, (i) linearly connected network, (ii) mesh connected network, (iii) perfect shuffle network.

As long as the number of processors stays small compared to the number of keys, odd-even transposition sort has an optimal speedup on (i) and the methods from [Batcher 1968] (see §4.1(a)) on (ii) and (iii).

D. Nassimi, S. Sahni (1982). Parallel permutation and sorting algorithms and a new generalized connection network. *J. Assoc. Comput. Mach. 29,* 642-667.

SIMD, (i) cube connected network, (ii) perfect shuffle network.

Sorting algorithms similar to that from [Preparata 1978] (see §4.1(c)) are given for networks (i) and (ii); sorting n elements requires $O(k \log n)$ time using $n^{1+1/k}$ processors ($k \leqslant \log n$). Further, permutation algorithms that are faster by a constant factor are given for these machines.

L.G. Valiant (1982). A scheme for fast parallel communication. *SIAM J. Comput. 11,* 350-361.

(Synchronized) MIMD, cube connected network.

Description of a randomized two-phase algorithm that performs permutations on an n-node cube connected network in $O(\log n)$ time with probability

close to 1. In the first phase each packet is sent to a randomly chosen node, in the second phase the packets find their way to their destination.

J.H. Reif, L.G. Valiant (1983). A logarithmic time sort for linear size networks. *Proc. 15th Annual ACM Symp. Theory of Computing,* 10-16.

(Synchronized) MIMD, cube connected cycles network.

A randomized algorithm for sorting n keys on an n-node cube connected cycles network in $O(\alpha \log n)$ time with probability at least $1-n^{\alpha}$, using ideas from [Valiant 1982] (see above).

(g) *interconnection networks: data transmission*

D. Nassimi, S. Sahni (1981). Data broadcasting in SIMD computers. *IEEE Trans. Comput. C-30,* 101-106.

SIMD, (i) mesh connected network, (ii) cube connected network, (iii) perfect shuffle network.

Algorithms for data transmission, with particular attention for read and write conflicts.

L.G. Valiant, G.J. Brebner (1981). Universal schemes for parallel communication. *Proc. 13th Annual ACM Symp. Theory of Computing,* 263-277.

Description of a randomized data transmission algorithm based on ideas from [Valiant 1982] (see §4.1(f)). With probability close to 1, it runs in $O(\log n)$ time on networks like the n-cube.

4.2. *Graph theory*

Many parallel algorithms have been developed for problems on graphs, such as finding connected components, transitive closures, spanning trees and shortest paths. Throughout this subsection, graphs (digraphs) have n vertices and m edges (arcs).

(a) *shared memory computers*

E. Reghbati (Arjomandi), D.G. Corneil (1978). Parallel computations in graph theory. *SIAM J. Comput. 7,* 230-237.

SIMD, shared memory, simultaneous reads, no simultaneous writes.

Finding the connected components of a graph and the weakly and strongly connected components of a digraph has the same time complexity as finding the transitive closure and therefore requires $O(\log^2 n)$ time using n^3 processors. Of three different bounded parallel graph search techniques, namely k-depth, breadth-depth and breadth-first search, the last one achieves a bound close to optimal if $m = \Theta(n^2)$.

D.M. Eckstein, D.A. Alton (1977). *Parallel Searching of Non-Sparse Graphs,*

Technical report 77-02, Department of Computer Science, University of Iowa, Iowa City.

Synchronized MIMD, shared memory, simultaneous reads, no simultaneous writes.

Depth-first search and breadth-first search of a graph can be performed in $O(n+m/p)$ time using p processors. These algorithms are essentially optimal if $n = O(m)$.

D.S. Hirschberg, A.K. Chandra, D.V. Sarwate (1979). Computing connected components on parallel computers. *Comm. ACM 22,* 461-464.

SIMD, shared memory, simultaneous reads, no simultaneous writes.

The connected components of a graph and hence the transitive closure of an $n \times n$ symmetric Boolean matrix can be obtained in $O(\log^2 n)$ time using n^2 processors and even using $n\lceil n/\log n\rceil$ processors. The connected components are built up by merging smaller parts together.

C. Savage, J. Ja'Ja' (1981). Fast, efficient parallel algorithms for some graph problems. *SIAM J. Comput. 10,* 682-691.

SIMD, shared memory, simultaneous reads, no simultaneous writes.

Algorithms are presented for finding the connected and biconnected components, the bridges and a minimum spanning tree of a graph in $O(\log^2 n)$ time. The number of processors used is small enough to make the parallel implementations relatively efficient.

D. Nath, S.N. Maheshwari (1982). Parallel algorithms for the connected components and minimal spanning tree problems. *Inform. Process. Lett. 14,* 7-11.

SIMD, (i) shared memory without simultaneous reads and simultaneous writes, (ii) perfect shuffle network, (iii) orthogonal trees network.

The algorithm from [Hirschberg, Chandra & Sarwate 1979] (see above) is modified such that it finds the connected components of a graph or a minimum spanning tree in a weighted connected graph in $O(\log^2 n)$ time using n^2 $(n\lceil n/\log n\rceil)$ processors on a shared memory model *without* read (and write) conflicts. Implementations on networks (ii) and (iii) require $O(\log^2 n \log\log n)$ time.

F.Y. Chin, J. Lam, I-N. Chen (1982). Efficient parallel algorithms for some graph problems. *Comm. ACM 25,* 659-665.

SIMD, shared memory, simultaneous reads, no simultaneous writes.

The algorithm from [Hirschberg, Chandra & Sarwate 1979] (see above) is modified to run in $O(n^2/p + \log^2 n)$ time using p processors $(p \geqslant n)$, i.e., in $O(\log^2 n)$ time for $p = n\lceil n/\log^2 n\rceil$. Slight adaptations of the algorithm find the weakly connected components of a digraph, a spanning forest and a minimum spanning tree.

Y. Shiloach, U. Vishkin (1982). An $O(\log n)$ parallel connectivity algorithm.

J. Algorithms 3, 57-67.

Synchronized MIMD, shared memory, simultaneous reads, simultaneous writes (one (unknown) processor succeeds).

In this strong model, the connected components of a graph can be found in only $O(\log n)$ time using $n+2m$ processors.

J. Ja'Ja', J. Simon (1982). Parallel algorithms in graph theory: planarity testing. *SIAM J. Comput. 11,* 314-328.

Synchronized MIMD, shared memory, simultaneous reads, no simultaneous writes.

Finding the connected components and planarity testing can be done in $O(\log^2 n)$ time using a polynomial number of processors.

L. Kučera (1982). Parallel computation and conflicts in memory access. *Inform. Process. Lett. 14,* 93-96.

SIMD, shared memory, simultaneous reads, simultaneous writes.

In models where simultaneous writes are allowed under certain conditions, the connected components of a graph can be found in only $O(\log n)$ time using a polynomial number of processors. Similar results hold for other graph problems such as finding a minimum spanning tree.

N. Deo, C.Y. Pang, R.E. Lord (1980). Two parallel algorithms for shortest path problems. *Proc. 1980 Internat. Conf. Parallel Processing,* 244-253.

MIMD, shared memory, simultaneous reads, no simultaneous writes.

The Moore/Pape-D'Esopo algorithm for finding all shortest paths from one vertex and the Floyd-Warshall algorithm for finding the shortest paths between all pairs of vertices are implemented on the Heterogeneous Element Processor, an MIMD machine.

N. Deo, Y.B. Yoo (1981). Parallel algorithms for the minimum spanning tree problem; summary. *Proc. 1981 Internat. Conf. Parallel Processing,* 188-189.

MIMD, shared memory, simultaneous reads, no simultaneous writes.

Three minimum spanning tree algorithms are considered: Prim-Dijkstra, requiring $O(n^2/p+np)$ time on p processors ($p \leqslant n$); Kruskal, achieving no speedup; and Sollin, requiring $O((n^2/p)\log n)$ time on p processors ($p \leqslant n$).

J.A. Wisniewski, A.H. Sameh (1982). Parallel algorithms for network routing problems and recurrences. *SIAM J. Algebraic Discrete Meth. 3,* 379-394.

The single source shortest path problem can be stated as solving systems of the form $x = Ax+b$ in the regular algebra of Carré. Solution methods, mostly known from linear algebra, are parallelized.

(b) *interconnection networks*

K.N. Levitt, W.H. Kautz (1972). Cellular arrays for the solution of graph

problems. *Comm. ACM 15,* 789-801.

Synchronized MIMD, two-dimensional mesh connected network.

The Floyd-Warshall shortest path algorithm, Kruskal's minimum spanning tree algorithm and other graph theoretical algorithms are implemented on special purpose hardware, buildable using (V)LSI technology.

L.J. Guibas, H.T. Kung, C.D. Thompson (1979). Direct VLSI implementation of combinatorial algorithms. *Caltech Conf. VLSI,* 509-525.

Synchronized MIMD, $n \times n$ mesh connected network.

For the transitive closure problem and a special class of dynamic programming problems, algorithms are designed that run in $O(n)$ time. They are suitable for VLSI implementation.

F.L. Van Scoy (1980). The parallel recognition of classes of graphs. *IEEE Trans. Comput. C-29,* 563-570.

Synchronized MIMD, $n \times n$ mesh connected network.

On this network, Warshall's transitive closure algorithm is implemented to run in $O(n)$ time.

D. Nassimi, S. Sahni (1980). Finding connected components and connected ones on a mesh-connected parallel computer. *SIAM J. Comput. 9,* 744-757.

SIMD, k-dimensional mesh connected network.

Consider a graph with $n = l^k$ vertices, none of which has degree more than d. Following [Hirschberg, Chandra & Sarwate 1979] (see §4.2(a)), the connected components can be found in $O(k^3(k+d)l \log l)$ time on a k-dimensional mesh connected network with l^k processing elements. For $d = 2$, the algorithm is modified to run in $O(k^4 l)$ time. The connected ones problem, a connected connectivity problem, requires $O(k^6 l)$ time.

M.J. Atallah, S.R. Kosaraju (1982). Graph problems on a mesh-connected processor array (preliminary version). *Proc. 14th Annual ACM Symp. Theory of Computing,* 345-353.

SIMD, $n \times n$ mesh connected network.

On this network, the bridges, the articulation points, the length of a shortest cycle and a minimum spanning tree of a graph are found in $O(n)$ time.

S.E. Hambrusch (1983). VLSI algorithms for the connected component problem. *SIAM J. Comput. 12,* 354-365.

Synchronized MIMD, k-dimensional mesh connected network.

Algorithms for finding connected components in $O(n^{1+1/k})$ time on a k-dimensional mesh connected network with n processors, suitable for VLSI implementation.

J.L. Bentley (1980). A parallel algorithm for constructing minimum spanning trees. *J. Algorithms 1,* 51-59.

Synchronized MIMD, tree structured network.

A parallel version of the Prim-Dijkstra minimum spanning tree algorithm, requiring $O(n \log n)$ time on a tree structured machine consisting of $O(n/\log n)$ processors.

E. Dekel, D. Nassimi, S. Sahni (1981). Parallel matrix and graph algorithms. *SIAM J. Comput. 10,* 657-675.

SIMD, (i) perfect shuffle network, (ii) cube connected network.

On both networks, two $n \times n$ matrices can be multiplied in $O(n/m + \log n)$ time using mn^2 processors and in $O(n^2/m + m(n/m)^{2.61})$ time using m^2 processors $(1 \leqslant m \leqslant n)$. These algorithms are applied to solve several problems on graphs, e.g., shortest path problems.

(c) *distributed algorithms*

R.G. Gallager, P.A. Humblet (1979). *Minimum Weight Spanning Trees*, Technical report LIDS-P-906, Massachusetts Institute of Technology, Cambridge.

MIMD, full interconnection.

This distributed minimum spanning tree algorithm is an asynchronous implementation of Sollin's method and requires $O(n \log n)$ time using n processors.

P.A. Humblet (1981). *A Distributed Algorithm for Minimum Weight Directed Spanning Trees*, Technical report LIDS-P-1149, Massachusetts Institute of Technology, Cambridge.

MIMD, full interconnection.

This algorithm for finding n minimum spanning arborescences, one rooted at each vertex, parallelizes the Chu-Liu/Edmonds/Bock algorithm and requires $O(n^2)$ time using n processors.

K.M. Chandy, J. Misra (1982). Distributed computation on graphs: shortest path algorithms. *Comm. ACM 25,* 833-837.

MIMD, full interconnection.

A distributed version of Ford's algorithm for finding all shortest paths from a single vertex, terminating properly if negative cycles occur.

5. Combinatorial Optimization

5.1. *Well-solvable problems: polylog parallel algorithms*

(a) *sequencing and scheduling*

The machine scheduling problems that have been subjected to parallelization will be indicated below by means of the concise notation of Graham, Lawler, Lenstra & Rinnooy Kan (*Ann. Discrete Math. 5* (1979), 287-326).

E. Dekel, S. Sahni (1983A). Binary trees and parallel scheduling algorithms. *IEEE Trans. Comput. C-32,* 307-315.

SIMD, shared memory, no simultaneous reads, no simultaneous writes.

Binary trees turn out to be useful for all sorts of parallel computations. E.g., the partial sums of a series of n numbers can be computed in $O(\log n)$ time using $O(n/\log n)$ processors

scheduling problems considered	*sequential time*	*parallel time*	*#processors*
$P \mid p_j = 1, r_j \mid L_{\max}$	$O(n \log n)$	$O(\log^2 n)$	$O(n)$
$1 \mid pmtn, r_j \mid L_{\max}$	$O(n \log n)$	$O(\log^2 n)$	$O(n)$
$1 \mid prec, p_j = 1, r_j \mid L_{\max}$	$O(n^2)$	$O(\log^2 n)$	$O(n^2/\log n)$
$1 \mid prec, pmtn, r_j \mid L_{\max}$	$O(n^2)$	$O(\log^2 n)$	$O(n^2/\log n)$
$1 \mid \mid \Sigma U_j$	$O(n \log n)$	$O(\log^2 n)$	$O(n)$
$1 \mid p_j = 1 \mid \Sigma w_j U_j$	$O(n \log n)$	$O(\log^2 n)$	$O(n)$

E. Dekel, S. Sahni (1981). *A Parallel Matching Algorithm for Convex Bipartite Graphs and Applications to Scheduling,* Technical report 81-3, Computer Science Department, University of Minnesota, Minneapolis.

E. Dekel, S. Sahni (1982). A parallel matching algorithm for convex bipartite graphs. *Proc. 1982 Internat. Conf. Parallel Processing,* 178-184.

SIMD, shared memory, no simultaneous reads, no simultaneous writes.

A bipartite graph with vertex sets $V = \{v_1,...,v_n\}$, $W = \{w_1,...,w_m\}$ and edge set E is *convex* if $\forall v_j \in V\ \exists l(j), u(j)$: $(v_j, w_i) \in E \Leftrightarrow l(j) \leqslant i \leqslant u(j)$. The binary tree method provides the basis of an algorithm for finding a maximum matching in such a graph in $O(\log^2 n)$ time using $O(n)$ processors.

scheduling problems considered	*sequential time*	*parallel time*	*#processors*
$1 \mid p_j = 1, r_j \mid f_{\max}$	$O(n^2 \log n)$	$O(\log^3 n)$	$O(n^2/\log^2 n)$
$1 \mid p_j = 1, r_j \mid \Sigma w_j U_j$	$O(n^2)$	$O(\log^2 n)$	$O(n^2/\log n)$
$P2 \mid pmtn, p_j = 1, prec \mid C_{\max}$	$O(n^2)$	$O(\log^2 n)$	$O(n^3/\log n)$

E. Dekel, S. Sahni (1983B). Parallel scheduling algorithms. *Oper. Res. 31,* 24-49.

SIMD, shared memory, no simultaneous reads, no simultaneous writes.

The developed algorithms rely on parallel sorting and the parallel computation of partial sums.

scheduling problems considered	*sequential time*	*parallel time*	*#processors*	
$P \mid pmtn \mid C_{\max}$	$O(n)$	$O(\log n)$	$O(n/\log n)$	
$Q \mid \mid \Sigma C_j$	$O(n \log mn)$	$O(\log mn)$	$O(m^2 n^2)$	
$1 \mid p_j = 1 \mid \Sigma U_j$	$O(n)$	$O(\log n)$	$O(n^2)$	
$1 \mid p_j = 1 \mid \Sigma w_j U_j$	$O(n \log n)$	$O(\log n)$	$O(n^2)$	[1]
$1 \mid r_j \mid \max_j \{\max\{e(E_j), f(T_j)\}\}$	$O(n \log n)$	$O(\log n)$	$O(n^2)$	[2]
$P \mid r_j, C_j = r_j + p_j \mid m$	$O(n \log n)$	$O(\log n)$	$O(n^2)$	[3]

[1] See [Dekel & Sahni 1983A] above for a different algorithm.

[2] $E_j = \max\{0, r_j - (C_j - p_j)\}$ is the *earliness* of job j; e & f are nondecreasing functions with $e(0) = f(0) = 0$.

[3] The *channel assignment problem*: minimize the number of identical parallel

machines needed to process a set of jobs with fixed starting times.

(b) *miscellaneous*

N. Megiddo (1983). Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach. 30,* 852-865.

The efficiency of serial algorithms for one problem may be improved by exploiting the parallelism in other problems. E.g., Valiant's and Preparata's parallel sorting algorithms (see §4.1(c)) turn out to be useful for cost-effective resource allocation and parallel all-pairs shortest path algorithms for the minimum ratio cycle problem. Other examples are given for scheduling and spanning tree problems.

5.2. *Well-solvable problems, log space complete for* $\mathcal{P}$

(a) *maximum flow*

L.M. Goldschlager, R.A. Shaw, J. Staples (1982). The maximum flow problem is log space complete for P. *Theoret. Comput. Sci. 21,* 105-111.

The result stated in the title is obtained through a log space transformation from the monotone circuit value problem (see [Goldschlager 1977], §2.2).

Y. Shiloach, U. Vishkin (1982). An $O(n^2 \log n)$ parallel MAX-FLOW algorithm. *J. Algorithms 3,* 128-146.

Synchronized MIMD, shared memory, simultaneous reads, simultaneous writes provided the same value is written.

A p-processor system is developed that solves the maximum flow problem on an n-vertex network in $O(n^3(\log n)/p)$ time, for $p \leqslant n$. The algorithm is closely related to the sequential methods due to Dinic and Karzanov, that use layered networks.

D.B. Johnson, S.M. Venkatesan (1982). Parallel algorithms for minimum cuts and maximum flows in planar networks (preliminary version). *Proc. 23rd Annual IEEE Symp. Foundations of Computer Science,* 244-254.

Synchronized MIMD, shared memory, simultaneous reads, no simultaneous writes.

Computing the maximum flow in planar directed n-vertex networks requires $O(\log^3 n)$ time using $O(n^4)$ processors and $O(\log^2 n)$ time using $O(n^6)$ processors; in planar undirected networks, $O(\log^2 n)$ time and $O(n^4)$ processors suffice. The results are based on the fact that the minimum cut capacity in a network N is equal to the length of a minimum forward cut cycle in a network related to the dual network of N.

(b) *linear programming*

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

In conjunction with Khachian's algorithm, this implies that linear programming is log space complete for $\mathcal{P}$. The log space transformation starts from the unit resolution problem (see [Jones & Laaser 1974], §2.2).

B. Kamdoum (1982). Speeding up the primal simplex algorithm on parallel computer. *SIGMAP Newsletter 31, 19-23.*

Each pivot step of the simplex method can be executed p times faster when p processors are available and p is small compared to the number n of variables and the number m of constraints.

N. Megiddo (1982). *Poly-log Parallel Algorithms for LP with an Application to Exploding Flying Objects,* Unpublished manuscript.

Megiddo has previously shown that linear programs can be solved by an $O(m)$ sequential algorithm when n is fixed *(J. Assoc. Comput. Mach.* (to appear)). A parallel implementation of this method runs in $O(\log^n m)$ time. Improvements lead to an algorithm requiring $O(\log^{n-1} m \log\log m)$ time and a probabilistic algorithm requiring $O(\log m (\log\log m)^{n-2})$ expected time on a parallel RAM model. An interesting application to warfare is presented.

5.3. *$\mathcal{NP}$-hard problems and enumerative methods*

(a) knapsack

A.C.-C. Yao (1982). On parallel computation for the knapsack problem. *J. Assoc. Comput. Mach. 29,* 898-903.

In parallel computation models with real arithmetic, solution of the knapsack problem with n real inputs requires an exponential number of processors if a running time of at most $\sqrt{n}/2$ is to be achieved.

(b) *traveling salesman*

E.A. Pruul (1975). *Parallel Processing and a Branch-and-Bound Algorithm,* M.Sc. thesis, Cornell University, Ithaca, NY.

MIMD, shared memory.

A p-processor implementation of the subtour elimination algorithm for the traveling salesman problem with n cities is developed and simulated on a sequential machine. For small p and n, the simulated parallel algorithm runs faster than the traditional serial method.

M.J. Quinn, N. Deo (1983). *A Parallel Approximate Algorithm for the Euclidean Traveling Salesman Problem,* Report CS-83-105, Computer Science

Department, Washington State University, Pullman.

MIMD, shared memory, simultaneous reads, no simultaneous writes.

The farthest-insertion heuristic for the Euclidean traveling salesman problem is implemented to run on the Heterogeneous Element Processor (see [Deo, Pang & Lord 1980], §4.2(a)) with p processors in $O(n^2/p + np)$ time.

(c) *dynamic programming*

J. Casti, M. Richardson, R. Larson (1973). Dynamic programming and parallel computers. *J. Optim. Theory Appl. 12,* 423-438.

Finite-stage dynamic programming procedures allow a natural parallelization. E.g., at each stage, the various states can be dealt with simultaneously by different processors.

D. Al-Dabass (1980). Two methods for the solution of the dynamic programming algorithm on a multiprocessor cluster. *Optimal Control Appl. Methods 1,* 227-238.

The efficieny of the algorithms developed in the previous paper is analyzed on a master-slave architecture.

P. Bertolazzi, M. Pirozzi (undated). *Parallel Algorithms for Dynamic Programming Algorithms,* Unpublished manuscript.

After a review of the methods proposed in the above two papers, for two classes of problems an implementation on a special configuration is shown to reduce computational complexity.

(d) *branch-and-bound*

O.I. El-Dessouki, W.H. Huen (1980). Distributed enumeration on between computers. *IEEE Trans. Comput. C-29,* 818-825.

MIMD, full interconection.

Note. In the title, read 'network' for 'between'.

A distributed branch-and-bound algorithm. Each processor determines by itself which part of the tree it searches for an optimal solution.

F.W. Burton, G.P. McKeown, V.J. Rayward-Smith, M.R. Sleep (1982). Parallel processing and combinatorial optimization. L.B. Wilson, C.S. Edwards, V.J. Rayward-Smith (eds.). *Combinatorial Optimization III,* University of Stirling, 19-36.

MIMD, r-ary n-cube.

Distributed branch-and-bound algorithms are considered for execution on the r-ary n-cube, a processor network that can be built using VLSI techniques. Each processor is invoked by one of its neighbors.