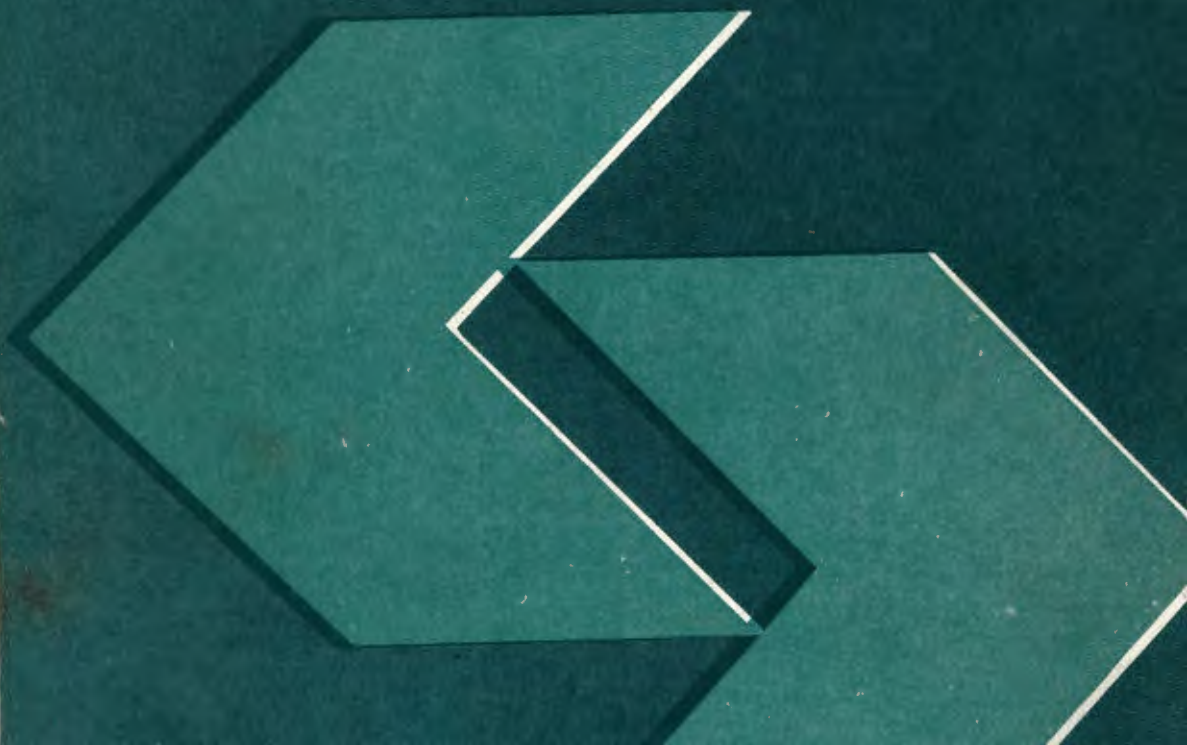


Polska
Akademia
Nauk
Instytut
Badań
Systemowych

Methodology and applications of decision support systems

Proceedings of the 3-rd
Polish-Finnish Symposium
Gdańsk-Sobieszewo, September 26-29, 1988

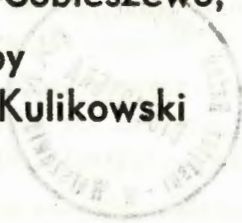
edited by
Roman Kulikowski





Methodology and applications of decision support systems

Proceedings of the 3-rd
Polish-Finnish Symposium
Gdańsk-Sobieszewo, September 26-29, 1988
edited by
Roman Kulikowski



Secretary of the Conference
dr. Andrzej Stachurski

Wykonano z gotowych oryginałów tekstowych
dostarczonych przez autorów



41267

ISBN 83-00-02543-X

PARALLEL ALGORITHMS FOR OPTIMIZATION

LEON SŁOMIŃSKI

Systems Research Institute, Polish Academy of Sciences
Newelska 6, 01-447 Warszawa, Poland

ABSTRACT

One of the major developments in computing in recent years has been the introduction of a variety of parallel computers and the development of algorithms that effectively utilize their capabilities. Significant opportunities exist for the utilization of parallelism in optimization. Algorithms and models that were intractable by the performance standards of von Neumann computers are becoming increasingly attractive in the parallel environment. New optimization algorithms are designed specially for parallel architectures. Our attention is focussed on discussing some directions and principles for designing, implementing and analyzing numerical optimization algorithms to be used on the commercially available vector and multiprocessor computers or those projected to appear in the nearest future.

KEY WORDS: parallel computing, combinatorial algorithms, efficiency of parallel algorithms.

INTRODUCTION

The impact of high performance computers on large scale scientific computing has infiltrated the field of numerical optimization and its applications in operations research, management sciences and engineering. Parallelism influences the design of new algorithms that hold great promise for solving very large problems in significant areas of applications.

In this article we concentrate our attention on discussing some new ideas in designing, implementing and evaluating parallel optimization algorithms, in particular - algorithms for combinatorial optimization problems defined on graphs and discrete sets. We take into account here existing vector and parallel computers as well as machines of the nearest future (systolic

arrays, cellular automata, neural networks).

Section 1 is a short introduction into vector and multiprocessor terminology and taxonomy. The importance of intercommunication features and I/O issues in parallelism are stressed.

Section 2 treats general methods that have been worked out for constructing optimization algorithms in parallel environments. The relations between the time complexity results gained from theoretical analysis and results of experiments obtained by implementations based on simulated multiprocessor structures are highlighted.

1. PARALLELISM IN COMPUTING. PARALLEL MODELS AND ARCHITECTURES

Parallelism is a set of techniques that introduce concurrency in computer systems, enabling several interconnected units to be simultaneously active in order to decrease the time necessary for solving our problem. So, by parallel computer, informally, we mean any computer capable of performing two or more logically connected computations simultaneously. Algorithms that are designed to do simultaneous computation by exploiting the parallelism inherent in a given problem will be called - parallel algorithms.

By now a number of different architectures of parallel computers have emerged, all based on some notion of how computations are to proceed and of how components in the architecture interact. A summary (Le,85) of the correspondences for present day architectures is given in Table 1.

There is no general model which captures all parallel architectures. We can, however, distinguish several classes of parallel computers. First, there is the class of pipelined machines

Table 1

Model of computation	Corresponding architecture
1. Sequential control on scalar data	1a. von Neumann - type computer 1b. Multiplication CPU 1c. Pipelined computer
2. Sequential control on vector data	2a. Vector computers 2b. Array processors
3. Independent communicating processors	3a. Shared memory multiprocessors 3b. Ultracomputers 3c. Networks of small machines
4. Functional and data-driven computation	4a. Reduction machines 4b. Data flow 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 neighbour. Inputs and results are streamed through a pipeline in such a way that successive tasks are overlapped, and the overall saving in execution time in a pipelined model is of $O(K)$, where K is the number of stages in the pipeline.

Vector computers are a subclass of pipeline machines that use this technique to rapidly compute arithmetic and logic operations on vectors of data.

A second class contains the single instruction stream multiple data stream (SIMD) machines. At one point in time, the processors perform the same type of operations on local data. Usually, there is a large number of miniature processors each with

its own memory. We shall consider systolic arrays as a subclass of SIMD type machines. A systolic array is a collection of synchronized, special-purpose, rudimentary processors with a fixed interconnection network. The function of processors and the type of interconnection scheme depend upon the problem being solved. The simplicity of the processors and the uniformity of the processor interconnection allow large systolic arrays to be implemented on a single chip by using VLSI technology.

Cellular automata (e.g. CAM 6 and CAM 7 from MIT, with the power of 200 GIPS) and neural network machines also fall in the SIMD category of parallel architectures.

A third class of the parallel machines, the largest one, contains the multiple instruction stream multiple data stream (MIMD) computers. More frequent, up to now, are control-flow multiprocessor architectures (synchronous or asynchronous) in which the operations are executed in a predetermined order by a control convention used by the programmer. Data-flow (e.g. Manchester Data-Flow prototype) and reduction (data-driven) machines have been proposed and designed. SIMD and control-flow MIMD machines can be categorized according to the way in which the processors intercommunicate.

The first category is formed by the so-called switched systems in which there is an identifiable and separate unit that connects a number of processors and memory modules. Shared memory architectures are in this class. In shared memory machines a number of processors are connected via the switch unit to a number of independent memory units. In the most restrictive shared memory (SM) model no two processors may have access to

the same memory location during the same instruction. The SM model which allows any number of processors to address the same location for reading but not for writing, is usually designated by SM-R. The SM-RW model allows any number of processors to address the same location during both read and write instructions.

The second category is that of network systems. In these architectures a number of processors are connected through an interconnection network with a particular topology. Interconnection network can be viewed as a graph with vertices corresponding to processors and edges (undirected) or arcs (directed) to make connection. Two parameters of the graph are important in this context: the maximum vertex degree d_1 , which should be bounded by a constant on grounds of practical feasibility, and the maximum path length d_2 which should grow at most logarithmically in the number of processors, to ensure fast communication. The most popular interconnection patterns are:

1. Two-dimensional mesh connected network ($d_1 = 4$, $d_2 = O(\sqrt{p})$), where p - the number of processors.
2. Cube connected network ($d_1 = d_2 = d = \log_2 p$).
3. Perfect shuffle network ($d_1 = 3$, $d_2 = O(\log_2 p)$).
4. Binary tree network ($d_1 = 3$, $d_2 = O(\log_2 p)$).

All these networks can simulate each other in polynomial time and space.

Commercial and laboratory examples of the parallel architectures, we have listed, are as follows:

- Supercomputers and minisupercomputers (pipelined and vectorized): Cray 1 from Cray Research, Cyber 205 from Control Data, Cedar from the Univ. of Illinois, S-810 from Hitachi, Facom VP

from Fujitsu, FX/8 from Alliant, C1 from Convex, Sequent Balance 21000 and SCS-40 from Sequent Balance.

- Shared memory multi-CPU supercomputers: Cray XMP (4CFU's), Cray 2 and 3, RP-3 from IBM, HEP from Denelcor, ETA-10 from ETA.

- SIMD computers: DAP from ICL (4096 processors, mesh with wrapped around topology), ILLIAC IV from the Univ. of Illinois (64 processors, toroidal two-dimensional mesh), BSP from Borroughs.

- MIMD computers: Cyberplus from CDC (ring interconnection), MPP from Goodyear (two-dimensional mesh with wrapped around), Transputer networks from INMOS, Local Area Networks of microcomputers and workstations (Univ. of Colorado, Univ. of Wisconsin).

- Hypercube computers: System 14 from Ametek, iPSC from Intel, NCube (1024 processors) from NCube Corp., Connection Machine 1 and 2 from MIT and Thinking Machine Co., Hypercube ($p = 2^6, 2^7$) from Caltech.

A crucial parameter characterizing the way we are thinking about parallel machines is the ratio of the communication time for a piece of data to the time required for the operation to be performed on it (τ -ratio). From this point of view C.G. Bell (Be,87) distinguishes three classes of parallel computers:

1. Message passing and multiple processors operating simultaneously: 1a. The workstation clusters (50-100 stations), $\tau = 10^3 - 10^6$. - 1b. Multicomputers ($32 - 10^3$ parallel processors) $\tau \approx 2 - 10^3$ (e.g. - hypercubes). - 1c. Multiprocessors with shared memory (e.g. - Cray XMP/4, ϕ), $\tau = 1 - 10^2$.

2. Computers with micromultitasking of a single process which is also done on N processors with SM, (newer Cray, new mainframes, minisupera), $\tau \approx 1/N$.

3. Computers with massive parallelism (10^3 - 10^6 data streams, controlled by a single instruction stream, examples: CM-1,2, GF-11, systolic and neural arrays), $\tau < 1$.

It is projected that machines of the third class will dominate parallel architectures in the nearest 10-15 years.

Serious limits come from I/O communication devices, as Table 2 shows (Ric, 87):

Table 2

Facility	Peak rate	Effective rate
Telephone	300	300
ARPA net	57K	20K
ETHERNET	10M	1,5M
CYBER-205 Channel	200M	100M

2. PARALLEL OPTIMIZATION

It is well recognized that many optimization problems are very expensive for one or more of the following reasons: the size of the problem (number of variables and constraints) is large; the objective function or constraints are expensive to evaluate; many iterations or function evaluations are required to solve the problem; a large number of variants of the same problem must be solved.

The only way to achieve considerable speedups and to decrease the costs of solving optimization problems is through the use of a collection of processors that cooperate in the solution process.

Efficient utilization of vector and parallel machines requires new organization of the computational processes, new

ideas and new theoretical tools are necessary for designing, analysing and evaluating parallel algorithms.

In the theoretical complexity analysis of parallel algorithms an important role is played by the Parallel Random Access Machine (PRAM) model, which is defined as follows: A synchronized machine with an unbounded number of processors and a shared memory which allows simultaneous reads from the same memory location but disallows simultaneous writes into the same memory location. The computation starts with one processor activated; at any step an active processor can do a standard operation or activate another processor; computation stops when the initial processor halts.

There is a nice result (Br, 74) that relates the unbounded and bounded models of parallel computation. Assume a computation consisting of a total of b operations can be carried out in time t using an unbounded number of processors. The computation can be carried out with p processors in approximately $t + (b - t)/p$ steps.

One is usually interested in knowing how many times faster his problem can be solved on the parallel computer as compared with the serial machine. The parameter which estimates this quantity is the speedup:

$$\text{SPEEDUP} = \frac{\text{TIME COMPLEXITY OF THE BEST KNOWN SERIAL ALGORITHM}}{\text{TIME COMPLEXITY OF THE PARALLEL ALGORITHM}}$$

Also the exploitation efficiency of the processors in the computation process is an important measure for estimating the quality of a parallel process:

$$\text{EFFICIENCY OF PROCESSORS UTILIZATION (EPD)} = \frac{\text{SPEEDUP}}{\text{NUMBER OF PROCESSORS}}$$

Work in the area of parallel numerical algorithms is proceeding in two major directions. First, specialization of existing algorithms for parallel computers are proposed and refined. Decomposition and straightforward parallelization methods hold a prominent position in this activity. Second, new algorithms are being proposed, whose characteristics make them suitable for parallelism. In (Mi, 84) five groups of methods for creating parallel algorithms are distinguished:

- Restructuring of a given algorithm as an algorithm numerically equivalent which contains a greater degree of parallelism.

- Splitting of a given problem into subproblems that are solved in parallel; the solution of the whole problem is obtained by combining the solutions of the subproblems.

- The "divide at impera" (divide and conquer) technique: a task of size s is split into two independent tasks of identical complexity but of size $s/2$, and this splitting can be continued recursively.

- Vectorization of the internally serial algorithm; a direct serial algorithm is converted to an iterative method which rapidly converges to the solution under certain conditions.

- Asynchronous parallel implementation of a serial strictly synchronized algorithm.

In (Sl,88) parallel algorithms for the minimax tree problem in a rooted weighted digraph of n vertices are presented and analysed. The algorithms are proposed using three theoretical models of parallel computers: SIMD-SM-CREW (Shared Memory-Concurrent Read Exclusive Write) model, Systolic Array model, and Double - binary tree machine. Table 3 presents the complexities

and number of processors for the serial algorithms and their parallel counterparts that solve the mentioned problem.

Table 3

Sequential algorithms		Parallel algorithms	
Algorithm	Complexity	Complexity Number of processors	Algorithm and computer model
1. Camerini's threshold algorithm	$O(n^2 \log n)$	$O(n \log^2 n)$ $O(n^2)$	1. Parallel threshold algorithm. SIMD-SM-CREW
2. Dijkstra's minimax path one-to-all algorithm	$O(n^2)$	$O(n \log n)$ $O(n)$	2. Dijkstra's parallel one-to-all algorithm. DOUBLE-BINARY TREE
3. All-to-all minimax paths algorithm by matrix multiplication	$O(n^3)$	$O(n \log^2 n)$ $O(n^3)$	3. Parallel all-to-all minimax paths algorithm by matrix multiplication. SIMD-SM-CREW
4. Floyd-Warshall's all-to-all minimax paths algorithm	$O(n^3)$	$O(n)$ $O(n^2)$	4. Floyd-Warshall's all-to-all parallel algorithm. SYSTOLIC ARRAY

Relevant speedups and processor utilization efficiencies are summarized in Table 4.

Table 4

Algorithms from Table 3				
	1	2	3	4
SPEEDUP	$O(n/\log n)$	$O(n/\log n)$	$O(n^3/\log^2 n)$	$O(n^2)$
EFU	$O((n \log n)^{-1})$	$O((\log n)^{-1})$	$O((\log^2 n)^{-1})$	$O(1)$

The discussion we have conducted illustrates one way of thinking about parallel algorithms, namely the way we have to proceed toward the parallel algorithm, starting with its serial prototype and utilizing one of the many theoretical models of parallel computation. Then, the time complexities, speedups and efficiencies are evaluated (see Tables 3 and 4).

In the case of the second way, preferences are given to experimentation on existing computers. The main difficulty encountered on this way is the inadequacy between the highly efficient algorithm we designed and limitations attributed to the implementation facilities offered by a particular computer. We may either emulate our algorithm on such a machine or restructure it, which is equivalent to a redesigning process. However, the overall complexity of the algorithm and the algorithm itself may be completely changed.

Having this in mind many researchers prefer simulation methods for research and study purposes. We have adopted this approach in the investigations currently underway. One of the simulation tools we are using is the Jupiter-86 simulator (Ju, 86). The simulator is written in PASCAL and runs under the DOS operating system on a PC/AT microcomputer.

Codes are written in PARLAN, which is a PASCAL-like language.

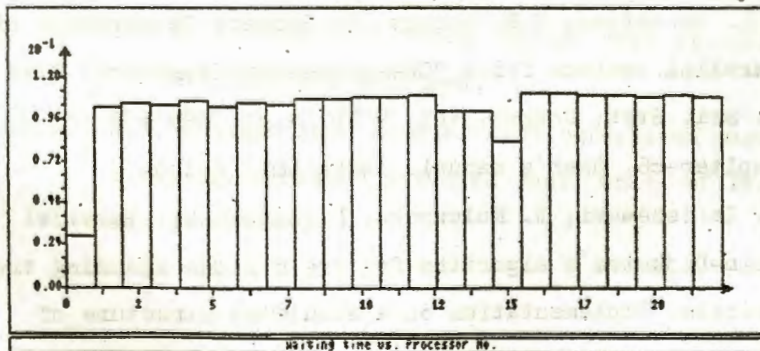
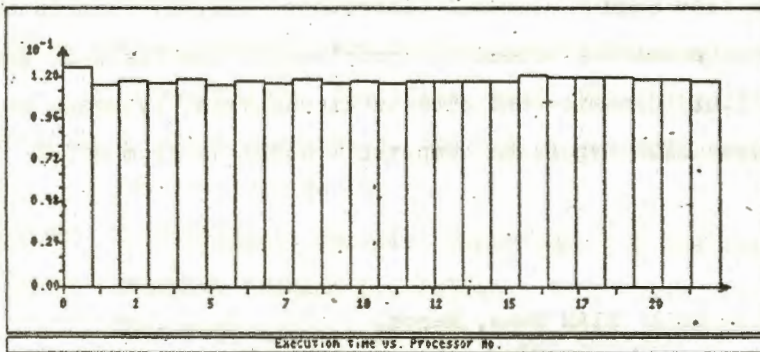
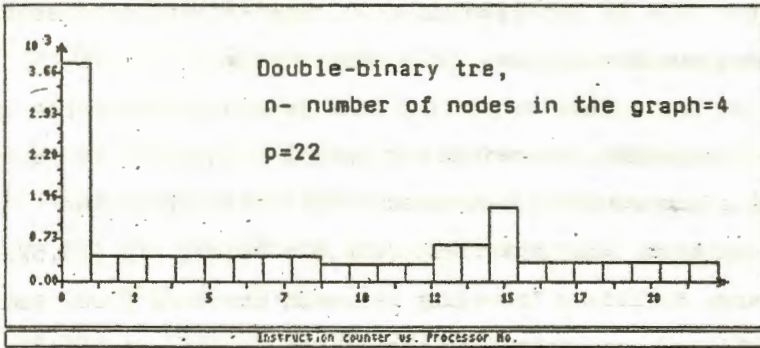
The Jupiter-86 is a simulator with strictly limited possibilities for multivariate experimental purposes. However, it is a very useful tool for simulating regular multiprocessor structures (mesh, ring, cube tree, etc.), and for solving small demonstrative problems. A serious disadvantage of this simulator is its inability to make precise measurements of computation and processing times.

A typical plot produced as a result of the program run is given in Figure 1.

So far we have implemented three algorithms with the Jupiter-86: Prim-Dijkstra's algorithm for the minimum (minimax) spanning tree problem in a graph, (KKS,88); Dijkstra's algorithm for the one-to-all shortest path problem in a digraph, (SG,88) and an algorithm for the b-matching problem in a tree, (Ku,88). The first two algorithms are implemented on a structure known as a double-binary tree computer (provides pipelining), the third algorithm is implemented on a one-dimensional mesh and on a general tree multiprocessor machine.

Further combinatorial algorithms and multiprocessor structures are under intensive study: matrix multiplication, all-to-all shortest path, testing connectivity and spanning tree problems are being tested on hypercube, perfect-shuffle and orthogonal tree multiprocessors.

In contrast to the widely used approach which in the complexity analysis neglects the time spent for imputing and outputting data to and from a multiprocessor structure, we do consider these phenomena. We demonstrate (KKS,88; Ku,88; SG,88) that under this assumption the time complexity of an algorithm im-



plemented on tree machines is the same as for the corresponding serial algorithm. The tree structure shows higher efficiency in the case when data is generated in $O(1)$ time in each processor, which is not true for the data in a matrix form.

Finally we would like to mention some interesting reports on successful application of vector and parallel machines for solving real-life and randomly generated optimization problems (linear, nonlinear, combinatorial) (Rib,87; Ze,87). In (PR,87) two very large Euclidean Traveling Salesman problems (1002 and 2392 cities) have been solved to optimality on the pipelined CDC Cyber 205 computer.

The team from Sandia National Laboratory (GMB,88) reports on solving three practical scientific problems in the field of wave mechanics, fluid dynamics and structural analysis, by means of a 1024-processor MIMD hypercube computer - NCUBE/10 from NCUBE Corp.

REFERENCES

- (Be,87) C.G. Bell: SIAM News, March.
- (Br,74) R.P. Brent: The parallel evaluation of general algorithmic expressions. J. ACM Vol. 21 pp. 201-206.
- (GMB,88) J.L. Gustafson, G.R. Montry, R. Benner: Development of parallel methods for a 1024-processor Hypercube. SIAM J. Sci. Stat. Comput. Vol. 9 No. 4 pp. 609-638.
- (Ju,86) Jupiter-86. User's manual. Abaks Lmt. Kraków.
- (KKS,88) I. Kaliszewski, D. Kulczycka, L. Słomiński: Parallel Prim-Dijkstra's algorithm for the minimum spanning tree problem. Implementation on a simulated structure of double-binary tree type (in Polish)..ZPM-5-A1530/88
IBS Warszawa.

- (Ku,88) D. Kulczycka: Parallel algorithms for b-matching in trees. Simulated implementation with the system Jupiter. ZPM-6/A1530/88 IBS Warszawa.
- (Le,85) J. van Leeuwen: Parallel computers and algorithms. In: Parallel computers and computations. Eds. J. v. Leeuwen and J.K. Lenstra. CWI Syllabus 9, Amsterdam pp. 1-32.
- (Mi,85) J. Miklosko, V.E. Kotov (eds.): Algorithms, software and hardware of parallel computers. Springer Vlg. and Veda pp. 27-28.
- (PR,88) M. Padberg, G. Rinaldi: Optimization of a 532-city symmetric traveling salesman problem by branch and cut. Operat. Res. Letters Vol. 6 No. 1 pp. 1-7.
- (Rib,87) C.C. Ribeiro: Parallel computer models and combinatorial algorithms. Ann. Disc. Math. Vol. 31 pp. 325-364.
- (Ric,87) J. Rice: Barriers to the use of supercomputers. SIAM News Vol. 20 No. 5.
- (Sl,88) L. Słomiński: Parallel algorithms for the minimax arborescence problem (in Polish). Zesz. Nauk. Politechn. Śląskiej, Automatyka No.94 pp. 287-301.
- (SG,88) L. Słomiński, H. Gondek: Simulated implementation Dijkstra's algorithm for one-to-all shortest path problem in a digraph on double-binary tree structure. ZPM-37/A1530/88 IBS Warszawa.
- (Ze,87) S.A. Zenios: An annotated bibliography on parallel optimization. Decision Sciences Dept. Univ. of Pennsylvania. Report 87-12-04.



IBS

41267