Computer Tree – the Power of Parallel Computations

J. JAROSZ AND J. R. JAWOROWSKI*

Institute of Computer Science, The Stanislaw Staszic University of Mining and Metallurgy, 30–059 Cracow, Poland, al. Mickiewicza 30/A4

Computer Tree (CT) is the non-standard computer structure which consists of a large number of processing elements, which are connected so that they form a binary tree.

We have proved that every problem belonging to the polynomial-time hierarchy can be solved on CT in polynomial time. A 0 (n^3) algorithm for the maximal clique decision problem was presented, as an example of the real power of parallel computations on CT.

Received August 1984

1. INTRODUCTION

There exists a wide class of problems which can be characterised by exponential time complexity $0(2^{p(n)})$, when any problem belonging to it is solved on a deterministic sequential machine. Its proper subclass is the class NP,[6] which includes problems solved in polynomial time on non-deterministic Turing machines. The most important (and until now unsolved) question is, whether any problem solved on a non-deterministic Turing machine in polynomial time can be solved in polynomial time on a sequential deterministic one.

The implication of a problem being NP-complete[6] is that there is no fully polynomial time approximation scheme which solves the problem in a time bounded by a polynomial in the input length and the reciprocal of the prescribed degree of accuracy. Many problems in areas like deterministic scheduling, graph theory, routeing, data base, mathematical programming, automata and language theory, image processing, microprogram optimisation, etc. have been proved to be NP-complete.[6] The set of NP-complete problems therefore spans a wide spectrum of application areas.

Since the method of the time complexity reduction of many difficult problems on sequential machines is unknown, it is widely accepted that a further speed-up in the execution time of algorithms on computers can only be achieved by making available parallel computers, that can exploit the parallelism inherent in many algorithms. A huge number of different parallel computer architectures have been proposed and implemented. Four groups of architectures for executing parallel algorithms have been considered in the past: general-purpose vector and array processors,[10, 13, 1] crossbar systems,[18, 4] cluster-oriented structures[16] and systolic structures for special purposes.[11, 17]

We use the non-standard computer structure which consists of a large number of computers, which are connected so that they form a binary tree. The idea for the concept of computer trees[2, 3] was derived from the observation that the execution of recursive procedures on present-day computers of the von Neumann type has to be effected in an unnatural way. Instead of delegating different procedure calls to different processors the flow of procedure calls is artificially put into sequence. The parallelism inherent in many recursively formulated algorithms is thus destroyed. The concept of computer trees (CT) aims at exploiting this parallelism to reduce the computation time of a wide class of algorithms.

* To whom correspondence should be addressed.

Section 2 briefly describes the architecture of CT, its mode of operation and a suitable programming language (for a detailed description the reader is referred to Ref. 2). In Section 3 it is proved that any problem beonging to the polynomial-time hierarchy (introduced by Meyer and Stockmeyer[19]) can be solved on CT in polynomial time. In Section 4 the algorithm for the maxclique decision problem is given, for showing how the time complexity of problems (more complex than those belonging to the class NP) can be reduced using CT.

2. COMPUTER TREE

The hardware structure called computer tree has been introduced by Buchberger (see Fig. 1).[2, 3]

Every node of the tree denotes a microcomputer with its own CPU and its own storage. Every computer of the tree has access to its own storage and the storage of its left and right son. Accessing the left and right son, a certain type of address modification has to take place. A computer module in the tree, by means of variables of types X, X′ and X″ has access to its own storage, the storage of the left and right son, respectively.

Unlike other authors,[2, 12] we do not assume that part of the storage in every node is 'private', i.e. cannot be accessed by the father of the node.

Every computer of the tree has, to exchange the synchronising information, three sensor bits S, TL and TR. They may be set and reset only by its father, its left son and its right son, respectively. Sensor S is used to control the starting point of computation, and is set and reset by assigning the new value to VL or VR variable in the program of the father node. Sensors TL and TR are used for deciding whether a computation has terminated and may be set or reset using the U variable in the son program.

Test modules have been realized.[5, 7] In view of the development of VLSI technology the availability of 10000 and more modules in one tree is realistic.

The basic principle of the present approach is language-independent. We use, for the computer tree to be programmed, a PASCAL-like language called PL/CT[8] with additional sensor instructions:

when ⟨condition⟩ **wait**; (with semantics *lab*: **if** ⟨condition⟩ **then goto** *lab*;)

$U := 1$; $U := 0$; (set the TL or TR sensor of father, depending on whether the module is the left or right son of its father)

$VL := 1$; $VL := 0$;

$VR := 1$; $VR := 0$; (set and reset the sensor S in the left and right son, respectively)

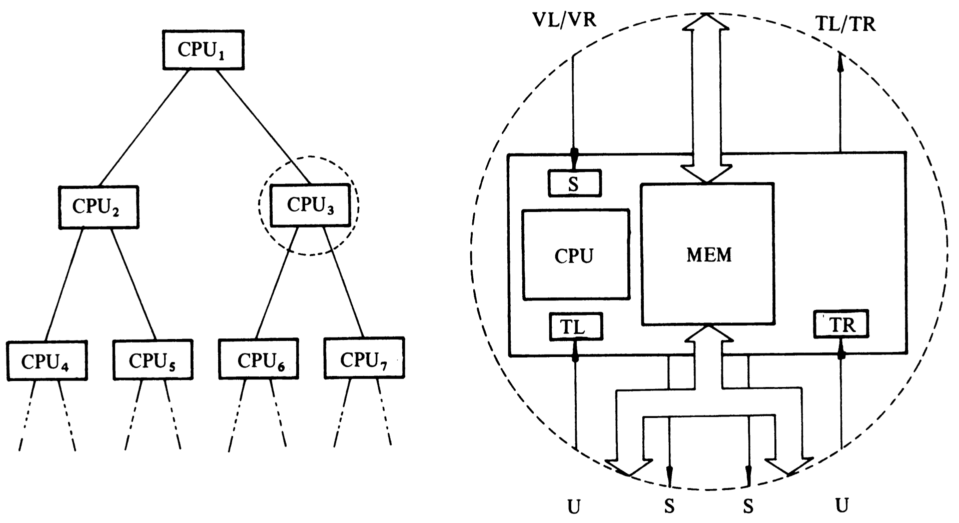

Figure 1. The architecture of a Computer Tree

The data input instruction *read*(*var*) initializes this same location name *var* in all the tree. The special mechanism was used to preset the memory and sensors to zero throughout the tree before the program loading process. We assume the latter, that the same program is at first loaded into all processors, and then the 'root' processor is started by external setting of sensor S.

3. COMPUTING POWER OF CT

CT is designed to solve in polynomial time difficult problems, i.e. problems with exponential time complexity. The polynomial time algorithms for problems belonging to the class NP-com, which may be used in the same way for solving the complementary problems from the class co-NP[2, 12] are known.

Buchberger[2] has presented the O(n) algorithm for the tautology problem (TAU), formulated as follows:

Having the logical expression given, check if it yields truth value 1 for all possible assignments of the truth values 1 and 0 to variables.

It is obvious that TAU$\in$ co-NP.[6] In fact this same algorithm on CT can be used to solve the problem $\overline{\text{TAU}}\in$ NP-com. Let P_{CT} denote the class of all problems solved in polynomial time using CT. The known algorithms for CT[2, 12] suggest that (NP $\cup$ co-NP) $\subset P_{\text{CT}}$, i.e. that P_{CT} includes problems more complex than those from NP and co-NP classes.

In 1972 the infinite polynomial-time hierarchy of the classes of language was introduced by Meyer and Stockmeyer.[19] Language classes Δ_k^p, Σ_k^p, π_k^p have been

$$k = 0: \Delta_0^p = \Sigma_0^p = \pi_0^p = P$$

$$k > 0: \Delta_{k+1}^p = P^{\Sigma_k^p}$$

$$\Sigma_{k+1}^p = \text{NP}^{\Sigma_k^p}$$

$$\pi_{k+1}^p = \text{co-}\Sigma_{k+1}^p$$

where

$$\text{P}^{\text{Y}} = \{\text{L}: \exists \text{L}' \in \text{Y} \text{ and } \text{L}\alpha_{\text{T}}\text{L}'\}$$

$$\text{NP}^{\text{Y}} = \{\text{L}: \exists \text{L}' \in \text{Y} \text{ and } \text{L}\alpha_{\text{NT}}\text{L}'\}$$

α_{T} is the polynomial deterministic reducibility relation defined using the oracle-deterministic Turing machine. The relation Lα_{T}L$'$ is fulfilled iff the recognising process

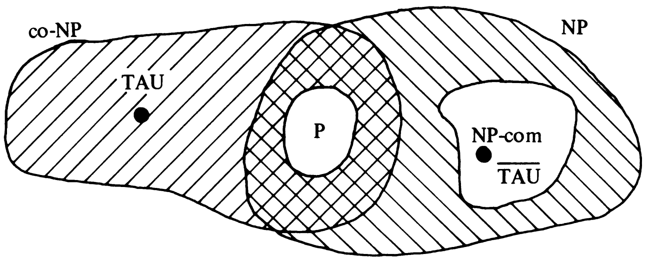

Figure 2. The map of NP and co-NP classes, after assumption that P $\neq$ NP and NP $\neq$ co-NP

of L is performed in polynomial time, and the Turing machine recognising L uses at least once, as the 'subprogram', the Turing machine which recognises the language L$'$. We assume that the recognising process of L$'$ is performed in exactly one step from the viewpoint of the machine which recognises L. α_{NT} is the polynomial nondeterministic reducibility relation and is defined similarly to α_{T} using the oracle-nondeterministic Turing machine.

Let

$$PH = \bigcup_{j=0}^{\infty} \Sigma_j^p$$

be the class of all languages in polynomial-time hierarchy. The important problem is in the verification of the following proposition.

Proposition

If $L \in PH$, then $L \in P_{\text{CT}}$, i.e. for every problem belonging to the polynomial-time hierarchy there exists a polynomial time algorithm when solved on CT.

Proof

Since the proof method is well known (see Ref. 6) we only give a sketch of the proof. Let $L \in \Sigma_k^p$, $k > 0$, and let p, q be polynomials such that every computation *CT* on an input of length n has length $p(n)$ (or $q(n)$).

(i) $k = 1$

It is obvious that $\overline{\text{TAU}} \in NP$-com[6] and $\overline{\text{TAU}} \in P_{\text{CT}}$. Then $L \in NP\text{-com} \Rightarrow L \in P_{\text{CT}}$ because

$$L \in NP\text{-com} \Leftrightarrow L\,\alpha\,\text{TAU} \;\&\; \text{TAU}\,\alpha\,L,$$

where α is the polynomial reducibility as defined in Ref.

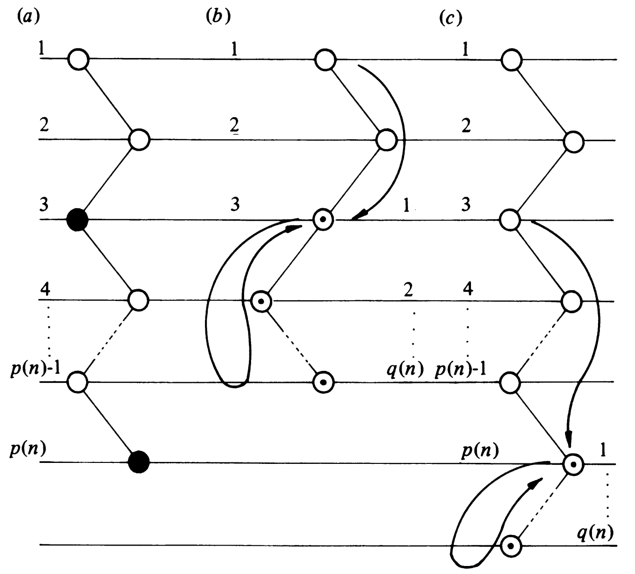

Figure 3. The computation strategy on CT. (*a*) The worst-case path of computing $L \in \Sigma_2^p$ (after assumption that L' is computed in one step) – L' is executed in black nodes. (*b*) First call of L' on CT – the computational wave for L' is generated, which distributes from level 3 downward until the level $q(n)+2$ is reached (worst case) and then contracts again upwards. (*c*) Second call of L' – worst-case level is bounded by $p(n)+q(n)-1$.

6. Let $L \in \Sigma_1^p = NP$. Then exist $L' \in NP$-com and $L \alpha L'$. This implies

$$L \in \Sigma_1^p \Rightarrow L \in P_{\mathrm{CT}}.$$

(ii) $k > 1$

Let

$$L \in \Sigma_k^p = \{L_0 : \exists L' \in \Sigma_{k-1}^p \ \&\ L_0 \alpha_{\mathrm{NT}} L'\}.$$

We assume that L' is computed in exactly one step from the viewoint of the program which computes L. Then the time complexity of L is $p(n)$. The last follows from (i), because L can be treated as a problem belonging to NP. The 'subprogram' which computes L' can be called no more than $p(n)$ times. Let every computation of L' have length bounded by $q(n)$ when solved on CT. Then the time complexity of L when solved on CT is $p(n) \cdot q(n)$. An example of computation strategy for solving L on CT is shown on Fig. 3.

We have proved that every problem $L \in \Sigma_k^p$ can be solved on CT in polynomial time if problems belonging to Σ_{k-1}^p can be computed in polynomial time on CT.

(iii)

From (i) and (ii) it follows, by induction over k, that

$$L \in \Sigma_k^p \Rightarrow L \in P_{\mathrm{CT}}, \quad k > 0.$$

For every $L \in \mathrm{PH}$ there exist m such that $L \in \Sigma_m^p$ and $L \notin \Sigma_{m-1}^p$, and finally the following holds:

$$L \in \mathrm{PH} \Rightarrow L \in P_{\mathrm{CT}}.$$

4. AN EXAMPLE – THE 0(n^3) MAXCLIQUE ALGORITHM ON CT

For illustrating the computation strategy and the real power of parallel computing on CT we select the maxclique decision problem (CMS). CMS is defined in the following terms:

Let $G(V, E)$ be an undirected graph, with vertex set V, edge set E, and $|V| = n$. Have the maximal clique of G exactly k vertexes?

Legget[6] has shown that CMS $\notin$ (NP $\cup$ co-NP). The algorithm presented as Fig. 4 has $0(n^3)$ time bound and uses $0(n)$ levels of processors. In the first search (SEARCH = 0) the algorithm checks whether there exist in the graph G the clique with k vertexes. In line 86 the following check is made. If the next-level processor number is less than or equal to the number of possible k-vertex subgraphs of G, then values 2·PROCNUMB and 2·PROCNUMB + 1 are sent respectively to the left and right son, as their own processor numbers. Both sons are started by setting the VL and VR variables to value 1.

In line 99 the procedure COMB is called to set the vector VERTXCOMB to values of the PROCNUMB-th combination (in lexicographical order) of k-vertex subgraph of G. In this way in every processor with number less than or equal to MAXNODE the unique subgraph of G is processed, and in addition all possible subgraphs of G with k-vertexes are processed. The loop beginning at line 50 in procedure COMB is executed exactly k times and the loop beginning at line 54 no more than $n-k$ times. Therefore the time complexity of COMB is $0(n^2)$.

In line 100, by calling the procedure CLIQUE a check is made on whether or not all vertexes of selected subgraph are connected by the edge. If they are, the value of MAXCLQ is set to 1. If not, in line 108 the processor waits for the ending of computations by its left and right son, and sets the result MAXCLQ as the logical sum of the results of its sons. The assignment instruction in line 32 is executed exactly k^2 times. Thus the time complexity of procedure CLIQUE is $0(n^2)$ too.

At line 116 every processor in the tree, except the 'root' processor, informs its father about ending computations by assigning the value 1 to sensor variable U. Then it is waiting for the reset of its own S sensor by its father (line 120).

When the first search is completed and the k-vertex clique is found (line 131) the 'root' processor performs the second search (SEARCH = 1) for checking the non-existence of the $k+1$-vertex clique, otherwise it stops with result MAXCLQ = 0.

In each search the computational wave is generated in the tree by the 'root' processor. At first the wave distributes downward until the needed level is reached, and then contracts upward. The search number is in fact bounded by 2 and the needed level of the tree is less than or equal to $n-1$. At any tree level the computation time is bounded by $0(n^2)$, therefore the time complexity of the present algorithm is $0(n^3)$. This algorithm was successfully run using the instruction level computer tree simulator.[9]

5. CONCLUSIONS

A concept for a multi-microprocessor system was presented by which the time complexity of a wide class of algorithms could be converted into hardware complexity. We have proved that every problem belonging to the Meyer and Stockmeyer polynomial-time hierarchy can be solved on CT in polynomial time. We believe that the Schorr thesis[14] that 'the physical time

```
0020 (*                                                                                  *)
0020 (*                   MAXCLIQUE DECISION PROBLEM ON THE COMPUTER TREE                *)
0020 (*                                                                                  *)
0020
0020 VAR N,                    (*  CARDINALITY OF THE GRAPH VERTEX SET                   *)
0023       K,                  (*  CARDINALITY OF THE CLIQUE VERTEX SET                  *)
0023       MAXNODE,            (*  NUMBER OF THE K-VERTEX SUBGRAPHS OF GRAPH             *)
0023       PROCNUMB            (*  PROCESSOR NUMBER IN THE TREE                          *)
0023       SEARCH,             (*  SEARCH NUMBER                                         *)
0023       MAXCLQ,             (*  RESULT VARIABLE: 1 - IFF GRAPH CONTAINS               *)
0023                           (*  MAXIMAL CLIQUE WITH K VERTEXES                        *)
0023       GRAPH[1..64,1..64], (*  GRAPH ADJACENCY MATRIX                                *)
0023       VERTXCOMB[1..64],   (*  VERTEX COMBINATIONS VECTOR                            *)
0023       Q,L,I,J;            (*  AUXILIARY VARIABLES                                   *)
0023
0023(***************************************************************************************)
0023
0023 PROCEDURE CLIQUE (K; VAR MAXCLQ, GRAPH, VERTXCOMB);
0023
0023                           (*  PROCEDURE CHECKS WHETHER THE K-VERTEX                 *)
0023                           (*  SUBGRAPH OF GRAPH IS A CLIQUE OR NOT                  *)
0023
0023   VAR I, J;
0026
0026   BEGIN (* CLIQUE *)
0035     I: = 0;
0051     REPEAT
0051       I: = I+1;
0100       J: = I;
0120       REPEAT
0120         J: = J+1;
0147         MAXCLQ: = GRAPH[VERTXCOMB[I], VERTXCOMB [J]]
0274       UNTIL NOT MAXCLQ OR (K = J)
0355     UNTIL NOT MAXCLQ OR (K-1 = I)
0436   END (* CLIQUE *)
0452
0452(***************************************************************************************)
0452
0452 PROCEDURE COMB (N, K, MAXNODE, PROCNUMB; VAR VERTXCOMB);
0452
0452                           (*  THE PROCEDURE SETS THE VECTOR VERTXCOMB TO            *)
0452                           (*  THE VALUE OF THE PROCNUMB-TH COMBINATION              *)
0452                           (*  (IN LEXICOGRAPHICAL ORDER) OF LENGTH K                *)
0452
0452   VAR P, R;
0455
0455   BEGIN (* COMB *)
0464     P: = 0;
0500     R: = 1;
0514     REPEAT
0514       MAXNODE: = MAXNODE*K/N;
0562       N: = N-1;
0611       K: = K-1;
0640       WHILE PROCNUMB > MAXNODE DO
0674         BEGIN
0674           PROCNUMB: = PROCNUMB-MAXNODE;
0727           MAXNODE: = MAXNODE*(N-K)/N;
1010           N: = N-1;
1037           P: = P+1
1053         END;
1071       VERTXCOMB[R]: = R+P;
1152       R: = R+1
1166     UNTIL R > K
1211   END (* COMB *);
1240
1240(***************************************************************************************)
1240
1240
1240 BEGIN (* MAIN PROGRAM *)
1247 WHILE NOT Q DO
1270   BEGIN
```

```
1270     WHEN NOT S WAIT;
1306     IF PROCNUMB = 0
1322       THEN
1330         BEGIN
1336           PROCNUMB: = 1;
1352           READ (N, K, MAXNODE);
1413           READ (I, J);
1441           WHILE I < > 0 DO
1474             BEGIN
1474               GRAPH [I, J]: = 1;
1567               READ (I, J);
1615             END;
1620         END;
1620     L: = PROCNUMB+PROCNUMB;
1653     IF L < = MAXNODE
1663       THEN
1704         BEGIN
1712           IF L < MAXNODE
1722             THEN
1740               PROCNUMB″: = L+1
1762             ELSE
1775               PROCNUMB″: = L;
2020           PROCNUMB′: = L;
2040           VL: = 1;
2052           VR: = 1
2056         END;
2064
2064   CALL COMB (N, K, MAXNODE, PROCNUMB, VERTXCOMB);
2101   CALL CLIQUE (K, MAXCLQ, GRAPH, VERTXCOMB);
2116
2116   IF L < = MAXNODE
2126     THEN
2147       BEGIN
2155         IF NOT MAXCLQ
2155           THEN
2170             BEGIN
2176               WHEN NOT (TL AND TR) WAIT;
2224               IF MAXCLQ′ OR MAXCLQ″
2234                 THEN
2247                   MAXCLQ: = 1
2261             END;
2271         VL: = 0;
2303         VR: = 0;
2315       END;
2315   IF PROCNUMB < > 1
2331     THEN
2342       BEGIN
2350         U: = 1;
2362         WHEN S WAIT;
2375         U: = 0
2401       END
2407     ELSE
2407       IF SEARCH
2412         THEN
2422           BEGIN
2430             MAXCLQ: = 1 – MAXCLQ;
2457             Q: = 0
2463           END
2473         ELSE
2473           IF MAXCLQ
2476             THEN
2506               BEGIN
2514                 SEARCH: = 1;
2530                 MAXNODE: = MAXNODE* (N – K)/(K + 1);
2620                 K: = K + 1;
2647               END
2647             ELSE
2647               Q: = 1;
2666  END;      (* WHILE *)
2671 WRITE (MAXCLQ)
2704 END . (* MAIN PROGRAM *)
```

Figure 4. CMS program on CT (listing).

complexity required to solve any problem is not reduced by more than a polynomial factor by using parallel models of computation instead of sequential ones' is wrong.

REFERENCES

1. J. C. Brown, Parallel architectures for computer systems. *Physics Today* **5**, 28–35 (1984).
2. B. Buchberger, Computer-Trees and their programming, *4th. Colloquillum on Trees in Algebra and Programming*, *University of Lille* (1978).
3. B. Buchberger, J. Fegerl and F. Lichtenberger, Computer-trees: a concept for parallel processing. *Microprocessors and Microsystems*, **3** (6), 244–248 (1979).
4. B. Buchberger, Components for restructurable multi-microprocessor systems of arbitrary topology. In *Mini and Microcomputers and Their Application*, pp. 67–71. Acta Press (1983).
5. B. Buchberger and J. Fegerl, *A Universal Module for the Hardware Implementation of Recursion*, Bericht Nr. 106, Institut Für Mathematik, Universität Linz.
6. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco (1979).
7. J. R. Jaworowski and J. M. Zaczek, Implementation of the Computer Tree Processor, Preliminary Technical Report CT-1, Institute of Computer Science, the Stanislaw Staszic University of Mining and Metallurgy, Cracow (1982) (in Polish).
8. J. R. Jaworowski, Software for CT: A Programming Language PL/CT, Preliminary Technical Report CT-2 (1983) (in Polish).
9. J. R. Jaworowski and J. M. Zaczek, CTS – Computer Tree instruction level simulator. Preliminary Technical Report CT-3 (1983) (in Polish).
10. R. Kober and C. Kuznia, SMS-201 – A powerful parallel processor with 128 microprocessors. *Euromicro Journal* **5**, 48–52 (1979).
11. H. T. Kung, *Why systolic architectures*? Technical Report CMU-CS-81-148, Carnegie-Mellon University, Pittsburgh (1981).
12. F. Lichtenberger, Speeding up algorithms on graphs by using computer trees. In *Graphs, Data, Structures, Algorithms*, edited M. Nagl and H. J. Scheider, pp. 65–79. Applications of Computer Science 13, Hauser, Munich (1979).
13. G. J. Lipovski, The Banyan switch in TRAC – the Texas reconfigurable array computer. *Distribution processing Technology Committee Newsletter* **6** (SI-1), 13–26 (1984).
14. A. Schorr, Physical parallel devices are not much faster than sequential ones. *Information Processing Letters*, **17**, 3, 103–106 (1983).
15. L. Snyder, Overview of the CHiP Computer. In *VLSI 81*, edited J. P. Gray, pp. 237–246, London, Academic Press (1981).
16. R. J. Swan S. H. Fuller and D. P. Siewiorek, Cm* – a modular multimicroprocessor. *AFIPS, Proc. National Computer Conference* **46**, 637–644, (1977).
17. B. W. Wah and Y. W. Ma, The architecture of MANIP – a parallel computer system for solving NP-complete problems, *AFIPS, Proc. National Computer Conference* **50**, 149–161 (1981).
18. W. A. Wulf and C. G. Bell, C.mmp – A multi-microprocessor, *AFIPS, PROC. Fall Joint Computer Conference* **41**, 765–777 (1972).
19. C. Stockmeyer, The polynomial-time hierarchy. *Theoretical Computer Science*, **3**, 1–22 (1977).