PARALLEL TURING MACHINES

Juraj Wiedermann

RUU-CS-84-11
November 1984

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

PARALLEL TURING MACHINES

Juraj Wiedermann

Technical Report RUU-CS-84-11
November 1984

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
the Netherlands

PARALLEL TURING MACHINES*

Juraj Wiedermann
VUSEI-AR, Dubravska 3, 842 21 Bratislava
Czechoslovakia

Abstract:
A new model of parallel computation - a so called Parallel Turing Machine (PTM) - is proposed. It is shown that the PTM does not belong to the two machine classes suggested recently by van Emde Boas, i.e., the PTM belongs neither to the first machine class consisting of the machines which are polynomial-time and linear-space equivalent to the sequential Turing Machine, nor to the second machine class which consists of the machines which satisfy the parallel computation thesis. Further the notion of a pipelined PTM is introduced and the "period" is defined as a complexity measure suitable for evaluating the efficiency of pipelined computations. It is shown that to within a polynomial factor period on a PTM is equivalent to space on any first class machine, or to time on any second class machine. The close similarity of PTM to sequential Turing machines enables us to prove also the existence of universal PTM, which can simulate every other PTM in linear time. Finally it is shown that every one-tape PTM can be efficiently realized using systolic arrays.

1. Introduction.

There exist quite a number of different parallel computer

---

*This work has been supported by the Czechoslovak Research Project I-5-7/01 and was completed during a stay at the Department of Computer Science, University of Utrecht, the Netherlands (Fall 1984).

models. The most attractive feature of these machine models is their extreme time efficiency when compared to the usual sequential models: all of them satisfy a so called parallel computation thesis (see e.g. Cook, 1984) which states that time on a parallel machine is polynomially related to space on a sequential machine. Thus parallel machine models are, in fact, joined by the above thesis into a family, which has been recently introduced by van Emde Boas (1984) as the so called "second" machine class.

The unusual power of second machine class models, which yields exponential speedup in some cases, stems from their unique capability to activate an exponential number of processors in polynomial time. However, this phenomenon is realistic only for limited size problems, or for a limited number of processors, where, moreover, the speed of information transmission among them can be assumed to be infinitely fast. In all other cases, i.e., when the realistic asymptotic complexity is to be analysed, the realistic cost of communication, most notably the finite speed of information transmission and the non-empty volume of individual processors, must be taken into account.

By similar ideas Chazelle & Monier (1983) and Schorr (1983) proved that it is in fact impossible for a real parallel computation machine to be more than polynomially faster than a sequential machine, since the latter can simulate the former with only polynomial loss in time efficiency. Since this principal limitation of parallel computers is not reflected in members of second machine class, by its very definition, it is obvious that the question of an adequate realistic model of the universal parallel computer is still open.

In this paper a candidate for such a model - a so called Parallel Turing Machine (PTM) - is presented. In Section 2 we shall introduce the model of a PTM as a generalization of the usual Sequential Turing Machine (STM). After discussing some modifications of the PTM in Section 3, we will further investigate the relation of PTM to the first and the second machine class in Section 4. In particular we show that every PTM can be simulated by a STM in

polynomial time, which means that the PTM does not belong to the second machine class. Thus at most a polynomial speedup over the STM can be expected, which is in good agreement with basic physical laws. The class of machines which are polynomial-time and linear-space equivalent to the STM was introduced by van Emde Boas (1984) as the "first" machine class. We show that the PTM does not belong to the first machine class either, since it cannot be simulated by any STM in linear space. In Section 5 we first formalize the notion of pipelined PTM and we introduce the periode as a suitable complexity measure to evaluate the efficiency of pipelined computations. As a main result we show that for a pipelined PTM period supplies, to within a polynomial factor, as such computational power as space on any first class machine, or time on any second class machine. This fact can be seen as an analogue of the parallel computation thesis - a "pipelined computation thesis" - where the period plays the role of time.

Thus it appears that the PTM belongs to some new, "one-and-a-halfth" machine class, which lies somewhere between the first and the second machine class, with its member being characterized by the above pipelined computation thesis.

Finally, in Section 6, we show the existence of a universal, "programmable" PTM, which can simulate any other PTM in linear time, and in Section 7 we show that every one tape PTM can be efficiently implemented using systolic arrays.

2. Parallel Turing Machines - a basic model.

Informally, a PTM is a set of identical STMs cooperating on a single common tape. Moreover, the STMs which represent the individual processors of the parallel computer, can multiply themselves in the course of computation.

The PTM is formally defined similar to the single tape nondeterministic Turing Machine (see e.g. Aho, Hopcroft, Ullman, 1974). However, its behavior is defined quite differently.

DEFINITION 2.1.: A 1-dimensional 1-tape deterministic Parallel Turing Machine - or: a (1,1)-PTM for short - is a 6-tuple

$$M=(Q,T,I,\delta,q_o,q_f)$$

where

Q is the finite set of states,

T is the finite set of tape symbols,

I is the finite set of input symbols, $I \subseteq T$,

δ is the transition relation assigning to each element of

$$Q \times (T - \{ e \})$$ a subset of $$Q \times (T \times \{L,R,S\})$$,

q_o is the initial state,

q_f is the final state.

There are two distinguished symbols in T-I: b is the blank symbol, and e is the empty symbol. M works as a language acceptor as follows. At every moment of the computation the machine M has one or more active processors. Each processor has one tape head, and a copy of the finite state control information. Thus all processors follow the same transition relation. At any time each processor is in exactly one state from the set Q. The processor in state q_f is called passive; otherwise it is active. Different processors can be in different states. The tape is infinite to the right.

At the beginning of the computation the machine M has just one active processor, which is in initial state q_o and has its tape head at the left and of the tape.

A string of input symbols (a word) is written on the tape, one symbol per cell, beginning in the left most cell. All cells to the right of the cells containing the input string are blank.

A computation step of M consists of the following activities, performed synchronously by every one of its active processors, in parallel:

- the processor determines the value of the transition relation δ for the particular configuration c represented by its state and by the symbol scanned by its head,
- if for a given configuration c the cardinality of $\delta(c)$ is greater

than 1, the processor multiplies itself. As many copies are created as there are elements in $\delta(c)$; each copy of the processor is added to M`s collection and acts further as an independent processor,

- in accordance with the value of $\delta(c)$ a processor does the following:
  - it enters a new state; if the new state is `final`, the processor becomes passive.
  - it rewrites the symbol scanned by its head. The empty symbol e in T-I occuring in the range of δ is interpreted as "write nothing". A simultaneous write into the same tape cell is allowed only if the processors are trying to write the same symbol; otherwise the computation is not legal and its result is undefined.
  - it moves its head by one position to the left (direction L), to the right (direction R), or keeps it stationary (direction S).

Passive processors do not perform any activity.

DEFINITION 2.2.: An input word $w \in I^*$ is accepted by M if and only if M, starting in state q_o, makes a sequence of moves in which every processor eventually enters the final state q_f, not necessarily in the same time. The language accepted by M is a set of words so accepted.

Schematically a (1,1)-PTM is shown in Fig. 2.1.

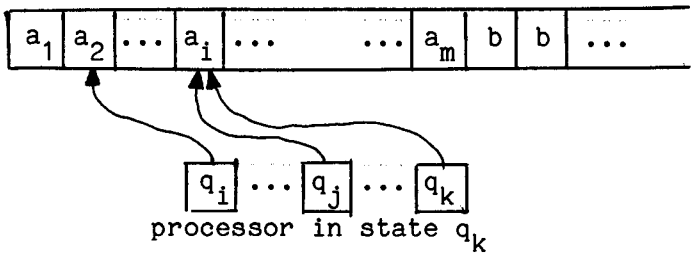

Fig. 2.1 A Parallel Turing Machine.

It is important to observe that the definition of PTM allows not only for the creation of the processors, but in a sense also for their disappearance: processors in the same state scanning the same tape cell are indistinguishable and efficiently act as one processor.

Note also that the processors of a PTM are controlled by the same "program", i.e., each processor decides independently, regardless of the other processors, about its next move. Communication among processors is achieved via message passing over the tape cells. Since this is the characteristic feature of distributed algorithms (van Leeuwen, 1983), the PTM can be seen as a model of a distributed computer.

The complexity measures for PTM are defined similarly as for sequential Turing Machine.

DEFINITION 2.3.:

(i) The parallel time PT(n) of a PTM M is the maximum number of parallel computation steps made by M in processing any input of length n, taken over all inputs of length n.

(ii) The space complexity PS(n) of M is the maximum distance from the left end of the tape which any tape head travels in processing any input of length n, taken over all inputs of length n.

(iii) The hardware complexity PH(n) is the maximal number of

distinguished processors, which are active at the same time, during the processing of any input of length n, taken over all inputs of length n.

One more complexity measure - the period of computation - will be defined in Section 5.

EXAMPLE 2.1.: the (1,1)-PTM M with the transition relation given in Fig. 2.2 recognizes palindromes on the alphabet {0,1}.

Symbol scanned	Current state	New symbol	New state	Head move	Comments
0	q_0	e	r_{00}	R	Starting processor: while not
			w_1	S	scanning blank, generate a new
1	q_0	e	r_{11}	R	working processor, remember
			w_1	S	in its state the first /and the last/
b	q_0	b	q_f	S	symbol scanned, and move right; prepare for waiting; otherwise halt in the accepting state
0 or 1	w_1	e	w_2	S	Starting processor: wait
0 or 1	w_2	e	w_3	S	steadily for 3 more steps,
0 or 1	w_3	e	q_0	R	then move right and enter the
b	w_3	b	q_f	S	initial state q_0 if nonblank is scanned; otherwise halt
0	r_{00}	0	r_{00}	R	Working processor: while not
1	r_{00}	1	r_{01}	R	arriving at b keep moving to
0	r_{11}	0	r_{10}	R	the right and remember in its
1	r_{11}	1	r_{11}	R	state the first and the last symbol scanned by the processor
0	r_{01}	1	r_{00}	R	Mnemonics: r_{ij} means "the first
1	r_{01}	1	r_{01}	R	symbol scanned is i, and the last
0	r_{10}	0	r_{10}	R	one is j"

Symbol scanned	Current state	New symbol	New state	Head move	Comments
1	r_{10}	1	r_{11}	R	
b	r_{00}	b	s	L	Working processor: scanning b
b	r_{11}	b	s	L	in state r_{00} or r_{11} move one
0 or 1	s	b	q_f	S	cell to the left and halt in the final state
b	r_{01}	b	t	L	Working processor: scanning b in
b	r_{10}	b	t	L	in state r_{01} or r_{10} move one
0 or 1	t	b	u	S	cell to the left and halt without accepting

Fig. 2.2.: The transition relation for PTM recognizing palindromes

The idea is to subsequently compare every symbol from the left half of the palindrome with the corresponding symbol from the right half and accept if any only if all the comparisons are successful. All the necessary comparisons are performed in the pipelined manner with only constant delays, as follows:

While not scanning the blank a single initial processor - called the starting processor from now on - keeps moving by one cell to the right in every fourth step; initially, and when arriving at a new nonblank cell, it issues a so called working processor, which keeps moving to the right at maximal speed. The working processor remembers in its state both the value of a symbol scanned at the moment of processor creation as well as that of a symbol scanned when leaving the last cell. For every working processor issued the following invariant is preserved: when arriving at the first blank the two symbols memorized in its state are exactly those which should equal each other, provided the input word is a palindrome indeed. This is initially true for the first working processor, since it starts from the first cell and the first blank occurs immediately to the right of the last input symbol. For the

subsequent working processors the invariant is restored by subsequently rewriting, with the help of the immediately preceeding working processors, the right end of the palindrome by blanks. Each working processor then halts in the legal accepting or in some illegal nonaccepting state, depending on the outcome of the respective symbol comparison, recorded in the processor state. The machine as a whole halts when the starting processor `meets`, in the middle of the palindrome, the first blank and no more processors are issued. For details see the comments in Fig. 2.2. Note the processor multiplying in state q_0 and the non-writing of the starting processor to avoid the possible write conflict /actually in state w_2 only/ with the returning working processor /in state s/ in the case of inputs of odd length. The time complexity of M is $PT(n)=2[n/2]+n+1$, the space complexity is $PS(n)=n+1$, and the hardware complexity is $PH(n)=[n/2]+1$.

3. Modifications of the basic model.

Like in the case of STM`s many modifications of PTM`s are possible.

First of all we will consider a multitape PTM, which is a generalization of a single tape PTM much in the same sense as a multitape STM is the generalization of a single tape STM.
The tapes of PTM can be one dimensional, as in the Definition 2.1 of the basic model, or in general they can be d-dimensional, for $d > 1$. The resulting model is then called a multidimensional multitape PTM. A d-dimensional k-tape PTM or STM will be denoted as (d,k)-PTM or (d,k)-STM, respectively, for $d \geq 1$, $k \geq 1$. We shall not define these models more formally, as the formalism is cumbersome and a straightforward generalization of notation for single tape one dimensional Turing machines. In what follows we will frequently use the PTM also as a transducer, i.e., as a device that computes a function on strings. For this purpose the arguments of the function at hand are encoded on a special read-only input tape, separated, if necessary,

by special markers. The result /the value of the function/ is written on a special write-only output tape. The corresponding device will be called an off-line PTM.

A separate input tape is useful not only for allowing computations with sublinear storage requirements, but also in the context of parallelism for allowing the efficient pipelining and overlapping in time of subsequent inputs and outputs, especially in the case of pipelined PTM (see Section 5).

The nondeterministic version of PTM will represent a further modification of PTM. It is defined similar to the basic, deterministic model of PTM from Definition 2.1, except for the transition relation, which reflects the fact that in the nondeterministic machine each processor has a finite number of choices for the next move. The nondeterministic PTM accepts its input if some sequence of choices of moves leads each processor to an accepting state.

In Wiedermann (1983) still some further modifications of the PTM have been investigated - namely the PTM with tree-like tapes and the PTM where the activity of processors is globally controlled. We will not pursue these topics here.

4. Basic Results.

In this section we will formulate the basic results concerning the relation of parallel to sequential Turing machines. Subsequently the activity of Turing machines will be described in the usual informal manner.

DEFINITION 4.1.: We will say that a Turing Machine M_1 simulates a Turing Machine M_2 of time complexity T(n) in linear (polynomial) time, if M_1 simulates M_2 and is of time complexity $O(T(n))$ ($O(T^k(n))$, for $k \geq 1$ fixed).

A similar definition will be used also for space complexity measure.

Now we can proceed to the description of the simulation of the STM on a PTM. First of all it is clear that a (1,k)-PTM is at least as powerful as the (1,k)-STM, since the latter machine is but a special case of the former one. Next we shall show that even the one-tape PTM is at least as powerful as the k-tape STM:

Theorem 4.1.: A(1,1)-PTM M_1 can simulate any (1,k)-STM M_2 in linear time and space.

Proof: The main idea of the simulation is to leave the heads of the simulating machine M_2 fixed and to appropriately shift its tapes instead. Moreover, all k-tapes of STM must be represented on a single tape of the PTM.

To do this we shall adapt the standard technique known from the theory of STM - namely the division of M_1`s tape into several tracks. Since several heads can attempt to rewrite the different tracks of the same cell simultaneously and independently, it is clear that we cannot straightforwardly use the trick of encoding the k-tuple, representing one cell divided into k-tracks, into one symbol. We must rather work with k-tuples explicitly all the time. Hence we shall divide the tape of M_1 into groups of k cells so that the i-th cell of the j-th group represents the j-th cell on i-th tape of M_2. The (equidistant) sequence of M_1`s cells representing the i-th tape of $M_2$ will be further called the i-th track of $M_2$. The activity of all STM heads is simulated by a single special head of $M_1$ which moves only within the first group of cells and remembers in its state the current state of STM`s control. Initially, and after finishing the simulation of one STM`s step the following invariant on $M_1$`s tape is preserved: k cells of the first group contain exactly those symbols scanned by the respective heads of M_2. This is achieved by shifting appropriately all the tracks by k cells

to the left, if the corresponding head of M_2 moves right and vice versa.

Which track, how and when is necessary to be shifted is determined by the aforementioned special head, which in k-steps scans its group and remembers the contents of each cell in its state. Knowing this information and the current state of M_2`s control it can decide now how to rewrite the corresponding cells and in which direction to shift the corresponding tracks, in accordance with $M_2$`s transition rules.

The action of rewritig is performed by the special head itself and for performing the shifts the power of PTM`s parallelism is evoked. For instance the left shift of the i-th track is performed in such a way that the special head just scanning the i-th cell generates two other workng heads /processors/: one of them is moving to the right, shifting every k-th /nonblank/ symbol it encounters k-places to the left, while the other working head is moving to the left, shifting every k-th /nonblank/ symbol k-places to the left as well.

The right shift is performed in the analogous manner. After arriving at the ends of the respective tracks the working heads enter the final state.

Now it should be obvious that although one shift of some track need not be finished yet, whenever the right cell of the M_2 is shifted into its proper position within the first group on M_1's tape, the next shift of the same track, in an arbitrary direction, can start. The working heads corresponding to the different shifts of the same track will never cross each other.

From the above description it follows that O(k) steps of PTM are necessary to simulate one step of STM and hence the whole simulation is performed in linear time.

It is also obvious, that the simulation runs in linear space. □

A similar theorem can be proved also for higher dimensional PTM's.

THEOREM 4.1. gives another way than that from Example 2.1 from Section 2 how to contruct a (1,1)-PTM recognizing palindromes - namely by a direct simulation of the corresponding (1,2)-STM, which works in linear time. Notice also that quadratic time is needed by a (1,1)-STM /Hopcroft, Ullman, 1969/ to recognize palindromes, which means that a one tape PTM is actually more powerful then a one tape STM.
In the next two assertions the basic relationship among the fundamental measures for PTM is described.

LEMMA 4.1.: For any (d,k)-PTM is holds that $PT(n)=\Omega(\sqrt[d]{PS(n)})$

Proof: It is obvious that in any direction the tape can be rewritten first in the linear time. Since the minimal "storage diameter" in some direction must be at least $\sqrt[d]{PS(n)}$ the result follows. □

LEMMA 4.2.: For any (d,k)-PTM it holds that $PH(n)=O(PS^{O(1)}(n))$

Proof: In the given space at most the polynomial number /with respect to the space size/ of processors, differing at least in their state or in the symbols scanned or in the positions of their heads, can be distinguished. □

The relationship between parallel and sequential Turing machines is described in the following theorem:

THEOREM 4.2.: A (d,k)-STM M_2 can simulate the (d,k)-PTM M_1 in polynomial time ad space.

Proof: For each parallel step of M_1 the M_2 adequately updates its tapes where the corresponding instantaneous description of PTM /i.e. the tape contents, states and head position of each processor/ is represented. Newly emerging processors are continuously numbered in order to keep track of which k-tuple of heads belongs to the same processor.

From Lemma 4.2. it follows that the length of the intantaneous description representation is of order $O(PS(n)+ PH(n).\log PH(n))= O(PS^{O(1)}(n))$, and as many updating passes over the tapes of STM are necessary as there are processors which the PTM possesses in that moment.

From Lemma 4.1. and 4.2. it follows that at most $PH(n)=O(PS^{O(1)}(n))=O(PT^{O(1)}(n))$ updating passes are necessary for simulating one parallel step of M_1, which in total still gives a polynomial time simulation. □

The results from Theorem 4.1. and 4.2. can be expressed in terms of relations between corresponding complexity classes. For that purpose we introduce the following notation:

DEFINITION 4.2.: Let C denote a family of computational models, RES a computational complexity measure. Then C-RES(F(n)) denotes class of languages accepted by some member of C within F(n) units of RES.

DEFINITION 4.3.: For any C and RES,

$$\text{C-POLY-RES} = \bigcup_{i\geq 1} \bigcup_{c\geq 0} \text{RES}\,(cn^i)$$

The next corollary follows from Theorem 4.1. and 4.2.:

COROLLARY 4.2.1.:

(1,k)-STM-POLY-TIME=(1,k)-PTM-POLY-TIME

(1,k)-STM-POLY-SPACE=(1,k)-PTM-POLY-SPACE

Thus the classes of problems solvable in polynomial time or space remain the same, irrespective of whether we use the STM or the PTM in their definition. Moreover, the previous result says that for the PTM the parallel computational thesis does not hold, since the time complexity required to solve any problem cannot be reduced more than by a polynomial factor by using PTM instead of STM. Nevertheless, as argued by Schorr /1983/, the polynomial speed up for parallel models is the best possible if the real, physical time consumed by the real physical device is considered.

It is interesting to try to put the PTM into the classification of computational models as introduced by van Emde Boas (1984). According to his classification (see Section 1), the PTM clearly does not belong to the second machine class since the parallel computational thesis is not satisfied.

Since the first machine class is defined by the machines which can simulate the STM in polynomial space and linear time, and vice versa, and in Theorem 4.2. we have achieved only polynomial-space simulation of a multitape PTM by the STM, it seems that the multitape PTM does not belong to the first machine class either. But to prove it we must show that in general the multitape PTM cannot be simulated by the STM in linear space. And this is indeed the case, as shown in the next theorem.

THEOREM 4.3.: There is a language L which can be accepted by a (1,2)-PTM in constant space, but cannot be accepted by any (1,k)-STM in less than $\Omega(\log \log n)$ space.

Proof: Let L be the set of palindromes to be recognized by a two-

tape off-line PTM (i.e., one of the two PTM tapes is read-only tape, containing the input string). A palindrome can be recognized by a PTM as follows: at the starting position, the single PTM remains stationery, but multiplies itself and sends the head of another processor to the right end of the palindrome. The second head of the moving processor remains stationary, scanning the first cell of the second tape.

When the moving head arrives at the end of the palindrome it activates the first processor through the first cell of the second tape that is scanned by the heads of both processors. From now on both heads on input tape start to proceed towards the opposite ends of the palindrome, at each step checking whether the same symbol is seen by both of them. For that purpose they communicate through the single common cell at second tape.

In the case of a STM it is known that the space of at least $\Omega(\log \log n)$ is needed to recognize L, since L is not a regular set (Hopcroft, Ullman, 1969). □

Thus the set of palindroms presents the language that separates multitape PTMs from the first machine class. Nevertheless, the situation is quite different in the case of one-tape PTM, as follows from the next theorem:

THEOREM 4.4.: A(d,1)-STM can simulate the (d,1)-PTM in polynomial time and linear space.

Proof: note that in the case of a one-tape PTM every processor has exactly one head. Thus, to keep track of the positions of PTM`s processors it is enough for STM to record, on a special track, for each of its tape cells only the states of simulated PTM`s processors which are currently scanning the cell at hand. Since the processors scanning the same cell can differ only in their states, their number

must be finite. The simulation itself can then be performed as in Theorem 4.2 in polynomial time and in linear space. □

From Theorems 4.4 and 4.1 it now follows that the one-tape PTM, although being a parallel machine, still is a member of the first machine class.

5. The power of pipeling.

The results from the previous section state that the amount of parallelism in PTM`s is polynomially time bounded and thus no spectacular effects of exponential speedup of sequential time can be expected. Nevertheless, in solving repeated instances of problems PTM`s are quite good, as will be seen from the following results.

First of all we shall formalize the notion of a pipelined PTM, which is capable of solving repeated problem instancies, and we define appropriate complexity measures for this case.

DEFINITION 5.1. An off-line (1,k)-PTM M is called a pipelined (1,k)-PTM, recognizing the language $L \subseteq I^*$, if and only if

(i) M is a transducer equipped with a two-dimensional read-only tape, and a one-dimensional write-only output tape;

(ii) the words $w_i \in I, i=1,2,\ldots, m$, $m \geq 1$, are arranged on the input tape in the upper left corner in rows, i.e., the i-th input word is written in the i-th row of the input tape, aligned to the left;

(iii) if $w_i \in L$ then M prints 1 as its i-th output on the input tape; otherwise it prints 0, for $i=1,2,\ldots,m$.

(iv) M reads its input in order $w_1, w_2, \ldots$ and prints the corresponding outputs in the same order;

(v) the number of moves made by M between reading the first symbol of w_i and printing the i-th output depends only on the length

of w_i, for i=1,2,...,m;

(vi) after reading all of its m inputs and printing the m-th output the machine M halts.

DEFINITION 5.2.: We will say that the pipelined PTM M recognizing the language L recognizes the language L (or simply: computes) with period PP(n), if and only if for every sequence $w_1,w_2,\ldots,w_m,m\geq 1$ of input words of the same lenght n the machine M reads the first symbol of w_i after at most PP(n) moves after reading the first symbol of w_{i-1}, for i=2,3,...m, taken over all input words of length n.

DEFINITION 5.3.: A pipelined PTM M recognizes the language L in time PT(n), if and only if for every sequence $w_1,\ldots, w_m$, $m\geq 1$ of input words of the same length n the machine M prints i-th output after at most PT(n) steps after reading the first symbol of the i-th input word, for i=1,2,...,m, taken over all inputs words of length n.

DEFINITION 5.4.: A pipelined PTM M recognizes the language L in space PS(n) if and only if PS(n) is the maximum distance any read/write tape head travels in processing any sequence of inputs of length n, taken over all sequences and all inputs of length n.

COROLLARY 5.3.1.: A pipelined PTM of time complexity PT(n) and with period PP(n) can process any sequence of m inputs of length n in time

$$PT(n)+(m-1)PP(n).$$

The corollary claims in fact that in a long run, on a sufficiently long sequence of inputs, the effect of a possible high "starting" time can be amortized by the efficient processing of

subsequent inputs, provided there is an appreciable difference between the asymptotic growth of PT(n) and PP(n). The following example illustrates this fact.

EXAMPLE 5.1.: Consider the off-line PTM M_1 from Theorem 4.3., recognizing palindroms in time $PT_1(n)=O(n)$ and space $PS_1(n)=O(1)$. This machine can be converted into a pipelined PTM M_2, recognizing palindroms, by exchanging its one-dimensional input tape for a two-dimensional tape, and by providing it with one one-dimensional output tape. M_2 then works as follows: after performing one computational step, which is similar to that of M_1, the machine M_2 moves the entire rewritten part of its working tape one cell to the right, processor`s heads included. Then, in the next step, M_2 can start processing the next input, by creating a new processor with its input head scanning the first symbol of the next input in the next row of input tape, and in the same time M_2 can prolonge its computations started previously.

Obviously, the pipelined machine M_2 has the following characteristics: $PT_2(n)=O(n)$, $PS_2(n)=O(n)$, $PP_2(n)=O(1)$. Note that the machine M_2 can process a sequence of n inputs of length n in time $PT_2(n)+(n-1)PP_2(n)=O(n)$, while the same task would take time $O(n^2)$ on M_1. Thus, using pipelining, we have achieved "amortized efficiency" of order O(1) per input!

The construction of a pipelined PTM from a nonpipelined one, as in the previous example, can be generalized for any off-line PTM:

THEOREM 5.1.: A pipelined (1,1)-PTM M_1 can simulate a non-pipelined off-line (1,1)-PTM M_2 of time complexity $PT_2(n)$ and space complexity $PS_2(n)$ in time $PT_1(n)=O(PT_2(n))$, in space $PS_1(n)=O(PT_2(n))$, and with period $PP_1(n)=O(PS_2(n))$.

Proof: M_1 first performs a computation to mark $PS_2(n)$ cells on its

tape (we assume that $PS_2(n)$ is tape-constructable, cf Hopcroft & Ullman 1969.). Then, on the next input, it starts the pipelined computation: it straightforwardly simulates one step of M_2 and then shifts the entire rewritten part of its working tape one cell to the right. After $PS_2(n)$ shifts, when some specialized processor of M_1 passes the rightmost marked cell, it issues another processor P towards the left end of the tape. When P arrives at the end of the tape it initializes the next computation by reading the next row of input data. Note that at this time enough space for the next computation has been set free at the beginning of the working tape of M_1. □

Note that the penalty one has to pay when trading off "nonpipelined space" for "period", is the increase of space in the pipelined machine. It appears that a similar trade-off can be made also in the reverse direction.

DEFINITION 5.5.: A pipelined PTM M computing with period P(n) is called a uniform pipelined PTM, when, on a sufficiently long sequence of identical inputs, it starts cycling with period P(n).

THEOREM 5.2. Every uniform pipelined (1,1)-PTM M_1 computing with period $PP_1(n)$ can be simulated by an off-line nonpipelined nondeterministic (1,1)-PTM, M_2 in space $PS_2(n)=O(PP_1(n))$.

Proof: Consider the computation of M_1 on sufficiently long sequence of identical inputs. After $PT_1(n)$ moves the uniform machine M_1 is forced to cycle with the period $PP_1(n)$ due to the fact that it reads the same input data all over again.

The idea of the simulation is quite simple: the machine M_2 guesses any instantaneous description of the cycling machine M_1 and

deterministically simulates the actions of M_1 during one entire period. Then it verfies whether the same instantaneous description as the initial guessed one was riched again. The realization of this idea is complicated by the fact that we cannot afford to remember an entire instantaneous description at once, since in general it cannot be represented in space $O(PP_1(n))$.

Rather, we proceed as follows. Let cell(i) be the i-th cell of M_1`s working tape, and let c(i,t) denote the contens of cell(i) in time t, for $1 \leq i \leq PS_1(n)$, $0 \leq t \leq PP_1(n)$.

The contents c(i,t) are represented on M_2`s working tape. The initial contents c(i,0) for any i can be guessed by M_2; the contents c(i,t) for t>0 depend only on the actions of processors which have their heads on cell(i-1), cell(i), and cell(i+1) in time t-1, and the other heads elsewhere on input and output tape.

Let the tape configuration of a processor in a given time be determined by its state, the symbols scanned by each of its heads, and positions of its heads on all tapes, at that time. Then, the contents c(i,t) for t>0 can be deterministically computed once we know the contents of neighbouring cells in the previous step, together with the tape configurations of processors scanning at this time the cells at hand. Note that there can be more than $PP_2(n)$ tape configurations pertinent to c(i,t) and thus they cannot be directly represented on M_2`s working tape. To save space, the tape configuration of each processor P_1 of M_1 scanning cell(i) at time t is represented by a processor P_2 of M_2 scanning c(i,t) and otherwise being in the same tape configuration like the simulated processor, i.e., when P_1 scans the j-th symbol of some input word, $1 \leq j \leq n$, then P_2 scans also the j-th symbol of its (single) input word. The heads on output tape of M_1 need not be considered at all, since they are write only heads and cannot influence the course of computation.

With these ideas in mind we can now finally advance to the description of the simulation.

Simulating PTM M_2 subsequently, for i=1,2,...,p, guesses the contents c(i,0) and processors scanning at time t=0 the cell(i). Then it nondeterministically sets the heads of these processors on

the input tape, and their states in order that they correspond to tape configurations of simulated processors. From this time on, these processors enter into waiting state, remembering in it, however, their original state before entering into waiting state. Starting from this situation M_2 then deterministically computes those contents c(i,t) which are necessary to compute c(i,p), where $p=PP_2(n)$.

To compute any c(j,t), for $1 \leq j \leq PS_2(n)$, $1 \leq t \leq p$, it is enough for M_2 to compute the values along two diagonals (see Fig.6.1), and remember them together with corresponding tape configurations.

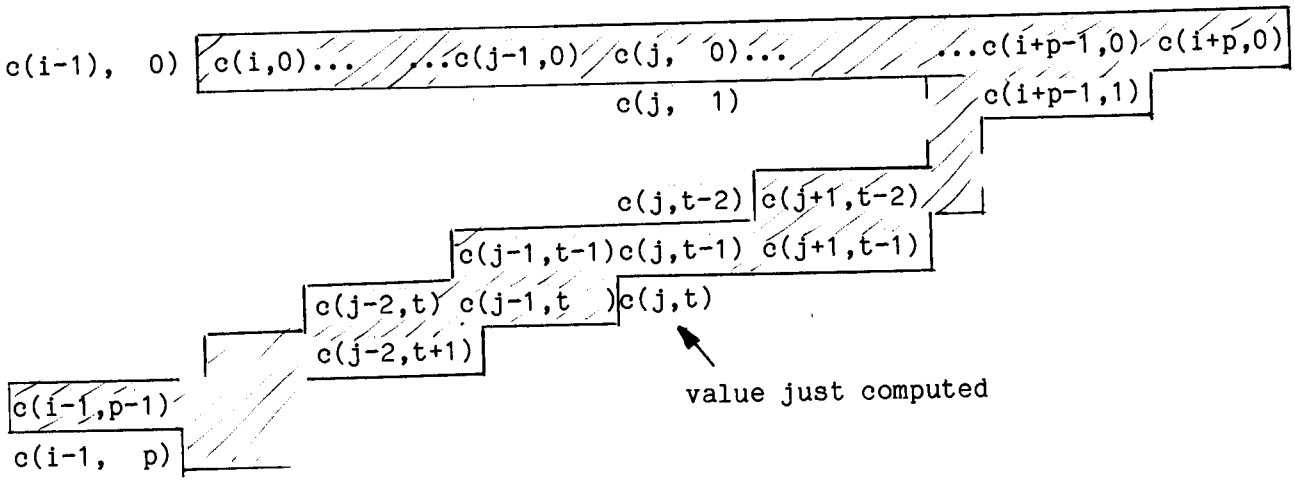

Fig.6.1. The organization of M_2`s computations.

Thus, to compute c(j,t), the machine M_2 activates another copies of processors, waiting over c(j-t,t-1), c(j,t-1), and c(j+1,t-1), and let them perform one computational step. Then the processors over c(j,t) are set into waiting state, the value c(j-1,t-1) can be "forgotten" and the corresponding processors are set into final state, and the process repeat itself for c(j-1,t+1).

After computing c(i,p) the machine M_2 has to verify whether c(i,0)=c(i,p) and whether tape configurations of corresponding

processors match.

The first condition is easily verified, providing M_2 has remembered the initial value c(i,0).

To verify the second condition, the machine M_2 activates, in groups, the processors waiting over c(i,0); one group is created by the processors having remembered in its waiting state the same state they were in before entering the waiting state. Then M_2 lets the input heads of these processors, as well as the heads of corresponding processors over c(i,p), move towards the left end of its input tape. When the heads of the processors over c(i,0) and c(i,p) have been scanning the same cells on input tape, then they must arrive at the same time at the tape end. This condition can be checked with the help of some distinguished symbol at the end of the tape.

When all the groups of processors have been successfully verified, the machine M_2 can start the next simulation and verification phase, corresponding to the cell(i+1). M_2 halts after a successful verification of the computation over the last cell of M_1 and accepts its input if and only if it has simulated during its computation such a move of M_1 in which some processor of M_1 attempted to write 1 on the output tape.

Note that no more than $O(PP_2(n))$ space on the working tape of M_2 is ever needed to remember the necessary information for performing each simulation and verification phase; corresponding entries, which are to be remembered during one phase, are pictured in the shaded area in Fig. 6.1. □

For pipelined PTM we can define the complexity class PTM-POLY-PERIOD similar to Definition 4.3., and by the previous two theorems we can prove the following important corollary, concerning the relation of STM`s and PTM`s:

COROLLARY 5.2.1.:

(1,k)-STM-POLY-SPACE=(1,1)-PTM-POLY-PERIOD

Proof: Returning to the proof of Theorem 5.1 we see that it remains valid also for the case of off-line (1,k)-STM instead of off-line (1,1)-PTM M_2, since according to Theorem 4.1 the former machine can be simulated by M_2 in linear space. This proves that (1,k)-STM-POLY-SPACE $\subseteq$ (1,1)-PTM-POLY-PERIOD.

To see the opposite inclusion, consider the off-line nondeterministic (1,1)-PTM M_2 from Theorem 5.2, of space complexity $PS_2(n)$, that simulates some pipelined (1,1)-PTM M_1 with polynomial periode $PP_1(n)$. Then M_2 is of polynomial space complexity $PS_2(n)=O(PP_1(n))$ even in the case when its input tape is included in space complexity estimation. According to Theorem 4.2, M_2 then can be simulated by a nondeterministic (1,2)-STM in polynomial space, which in turn can be simulated by a deterministic (1,2)-STM in polynomial space, due to Savitch's theorem (see e.g. Aho, Hopcroft, Ullman, 1974). □

Thus, to within a polynomial factor space for (1,k)-STM is equally powerful computatinal resource as period for pipelined (1,1)-PTM.

The previous corollary can be rephrased also in terms of the classification suggested by van Emde Boas (1984) to obtain a final characterization of (1,1)-PTM, relating it to both the first machine class C_1 and the second machine class C_2:

COROLLARY 5.2.2.:

$$C_1\text{-POLY-SPACE}=(1,1)\text{-PTM-POLY-PERIOD}=C_2\text{-POLY-TIME}$$

Proof: since (1,k)-STM is from C_1, we have (1,k)-STM-POLY-SPACE= C_1-POLY-SPACE, and using Corollary 5.2.1 we get the first equality. The second equality follows directly from the parallel computation thesis, which asserts that C_1-POLY-SPACE= C_2-POLY-TIME. (see Section 1). □

It is interesting to observe that the relation from Corollary 5.2.2 can be seen as a weak analogue of the parallel computation thesis - the difference being that this time period, rather than parallel computation time, is polynomially related to sequential space. The relation can be appropriately called the "pipelined computation thesis" and, indeed, can be used to characterize the class of machines to which the off-line, or pipelined (1,1)-PTM belongs. In view of the results from Section 4, stating that multitape PTM`s are members neither of the first, nore of the second machine class, it appears that a new, one-and-a-halfth machine class $C_{1.5}$, emerges, which is neatly related to the classes C_1 and C_2 by the relation.

$$C_1\text{-POLY-SPACE}=C_{1.5}\text{-POLY-PERIOD}=C_2\text{-POLY-TIME},$$

with pipelined (1,1)-PTM presently being the single known member of $C_{1.5}$.

Let us return once more to the problem of efficient periode simulation of (1,k)-STM by pipelined PTM. We shall show that adding one more space dimension to the simulating PTM leads to the period optimal simulation.

THEOREM 5.3. A pipelined (2,1)-PTM M_1 can simulate any off-line (1,k)-STM M_2 of time complexity $ST_2(n)$ and space complexity $SS_2(n)$ in time $PT_1(n)=O(ST_2(n))$, in space $PS_1(n)=O(ST_1(n).SS_1(n))$, and with period $PP_1(n)= O(1)$.

Proof: Similar to Theorem 4.1 we can construct the off-line (1,1)-PTM M_3 that simulates M_1 in linear time and linear space, and simulate the machine M_3 instead of M_1, by the machine M_2. The simulation then is similar to that of the Theorem 5.1., but instead of moving the whole "computation", belonging to one instance of the input, to the right, it is moved "down" (i.e. into the second dimension) on the two-dimensional working tape of M_2 - so that the next computation can start immediately. □

6. A Universal Parallel Turing Machine.

The theory of PTM`s can benefit from the known theory of STM`s, since most of the results of the latter theory can be generalized immediately. For instance, like in the case of STM`s one can show that there exists a universal, "programmable" PTM, capable to simulate any other suitably encoded PTM, without any substantial loss of efficiency. We will state the corresponding result for one tape PTM`s, since, as we shall see in Section 7, these are of special interest.

THEOREM 6.1.: For any $d \geq 1$ there exists a universal $(d,1)$-PTM M_1 which can simulate every other $(d,1)$-PTM M_2 in linear time and in linear space.

Proof: The machine M_1 simulates the other machine by keeping and appropriately updating in parallel the instantaneous descriptions of M_2. The instantaneous descriptions are represented in the same way as in the simulation from Theorem 4.4, which ensures a linear space simulation.

The tape of M_1 is further divided into regions of equal size. Each region takes the form of the smallest d-dimensional cube in which the standard encoding of M_2 (see e.g. Hopcroft, Ullman, 1969) can be represented. In each of these regions a single copy of standard encoding of M_2 is written on a special track (which is to be understood similarly as in Theorem 4.1), and further there resides a single processor of M_1 whose task is to appropriately update, within its region, the instantaneous description of M_2.

To simulate one parallel step of M_2 each processor of M_1 must sequentially, one by one, simulate the actions of M_2`s processors currently being in its region. Therefore, every processor of M_1, systematically for each cell within its region and for each processor of M_2 scanning that cell, searches its copy of the standard encoding of M_2 to find the value(s) of the transition relation

pertinent to the state of the simulated processor and the tape contents scanned. Then the processor updates the corresponding part of the instantaneous description accordingly and starts to simulate the action of next processor of M_2.

The simulation of one step of M_1 is completed after the simulation of actions of all processors of M_2, in parallel within all regions, was finished.

Since the size of the standard encoding of M_2 is bounded, so is the size of each region on M_1`s tape, and hence each processor of $M_1$ has to simulate sequentially only a bounded number of $M_2$`s processors. It is thus clear that the actions of all M_2`s processors in one parallel step of M_2 are simulated in a bounded number of parallel steps of M_1 which leads to a linear time simulation. □

Combining Theorem 6.1 and Theorem 5.3 yields the following construction of an interesting pipelined PTM:

COROLLARY 6.1.1.: There exists a single universal pipelined (2,1)-PTM which can simulate any sequence of ST(n) time bounded and SS(n) space bounded (1,k)-STM computations, in time O(ST(n)), space O(ST(n)), and with constant period, providing that the standard encoding of the corresponding (1,k)-STM is a part of each input.

The requirement concerning the boundedness of computational resources can be realized, as usual in practice, by "aborting" computations whenever they pass over the prescribed time or space limits. The results of all computations are printed with constant period only after the common time bound expires.

7. One-tape PTM`s and systolic computations.

In previous sections we have seen that the PTM possesses many properties which make it interesting from the theoretical point of view. Now we will show that the PTM presents an interesting model of parallel computations in a practical sense also, since it appears that its simplest version - the one-tape machine - can be realized efficiently using systolic arrays (see e.g. Kung, 1979).

THEOREM 7.1.: A d-dimensional orthogonal systolic array can simulate a (d,1)-PTM in linear time and area.

Proof: The simulating systolic array consists of O(SS(n)) identical processors, each processor corresponding to one tape cell, connected with neighbouring processors. In each processor a tape symbol in one memory location can be stored, and, moreover, there is also a boolean state vector of length $|Q|$.
For each processor the simulation preserves the following invariant in each step: the tape symbol of the corresponding cell is recorded in the corresponding processor`s memory location, and if the cell is scanned by a tape head in state $q_i$ then i-th bit of the state vector is set to 1, otherwise to 0. The moves and multiplication of the PTM`s processors are simulated by updating the corresponding tape contents and bits of state vectors in the appropriate neighbouring systolic array processors. The unification of several heads, scanning the same cell in the same state, is done automatically, since a set, rather than the multiset of states is stored in the state vector. □

In principle, the reverse of Theorem 7.1 can be formulated also; the difficulty lies in the fact that the theory of systolic system computations is not sufficiently formalized for our purposes. But it is intuitively clear that once a suitable algorithmic

description of a systolic system is given, we can first "generate" and initialize the system on our PTM, and then start with the computation with no extra required resources. Along these lines Gruska (1984) has recently given an interesting characterization of (1,1)-PTM`s by proving that these machines are equivalent to the homogeneous systolic trellis automata. From these consideration it is also clear that a number of systolic algorithms can be straightforwardly implemented on (1,1)- or (2,1)-PTM /e.g. pattern matching, sorting, matrix multiplication etc./, with the corresponding complexity results, and vice versa /e.g. palindrome recognition from Example 2.1/. A similar assertion is true also for more general VLSI circuits, provided again that the geometrical layout of the circuit is "PTM constructible", and the linear complexity communication measure is used /see e.g. Chazelle & Monier, 1983/. As far as the circuit simulation of PTM`s with several /two or more, including input and output/ tapes concerns, we do not know any realistic and efficient way to realize unbounded, time varying fan-in or fan-out, appearing when the unbounded number of PTM`s processors overlap with their heads on one tape, but not on the other tape.
Note that such a situation cannot emerge on one tape machines and this is the reason why they can be realized efficiently on circuits. The situation concerning time-varying and unbounded fan-in and fan-out is slightly better when only off-line or pipelined one-tape PTM with a separate input and output tape is considered, since in many practical situations no completely data dependent reading is necessary. In these cases various analogues of oblivious input schedules, widely used in /the theory of/ VLSI parallel processing, can be modeled by PTM, and the resulting machine subsequently realized by the corresponding circuit.

8.Conclusion.

A new model of a universal parallel computer - the Parallel Turing Machine - has been proposed. The model appears to be a useful

theoretical device for studying the asymptotic power of parallelism, since it takes the true physical nature of parallel computations into account, and yet allows for a simple, elegant and formal mathematical treatment within the framework of the theory of automata. There is also a theoretical evidence that the PTM represents a new class of machine models, which could provide a suitable theoretical tool for studying the pipelined computations. Moreover, simple but still powerful versions of the models can be realized efficiently with the help of systolic arrays.

Acknowledgement: I would like to thank Jan van Leeuwen and Peter van Emde Boas for their careful reading and helpful criticism of an earlier version of the manuscript.

REFERENCES

Aho,A.V.-Hopcroft,J.E.-Ullman,J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974

Chazelle,B.-Monier,L.: Unbounded Hardware is Equivalent to Deterministic Turing Machine. Theoretical Computer Science 24 /1983/, pp. 123-130

Cook,S.A.: Towards a complexity theory of synchronous parallel computations. L`Enseignement Mathematique, IIe Serie, Tome XXVII-Fascicule 1-2, pp. 99-124, 1981

Gruska,J.: Systolic automata-power, characterization, nonhomogeneity, in: M.Chytil (Editor), Mathematical Foundations of Computer Science 1984, Springer Lecture Notes in Computer Science Vol.176, 1984

Hopcroft,J.E.-Ullman,J.D.: Formal Language and Their Relation to Automata. Addison-Wesley, Reading, Mass., 1969

Schorr,A.: Physical Parallel Devices Are Not Much Faster Than Sequential Ones. Information Processing Letters, Vol. 17, 1983, pp. 103-106

Van Emde Boas,P.: The Second Machine Class, Models of Parallelism, in: J. van Leeuwen, J.K. Lenstra and A.H.G.Rinnoy Kan (eds.). Parallel computers and computations. CWI Syllabus Centre for Math. and CS., Amsterdam, (to appear), 1984

Van Leeuwen,J.: Distributed Computing. In: J.W. de Bakker and J.van Leeuwen (eds.), "Foundations of Computer Science IV", Mathematical Centre Tracts Vol.158, Amsterdam, pp.1-34

Wiedermann,J.: Parallel Turing Machines - a preliminary report. Unpublished manuscript, Computing Research Centre, Bratislava, 1983