On parallel Turing machines with multi-head control units[*]

Thomas Worsch[†]
`http://liinwww.ira.uka.de/~worsch/`

Abstract

This paper deals with parallel Turing machines with multi-head control units on one or more tapes which can be considered as a generalization of cellular automata. We discuss the problem of finding an appropriate measure of space complexity. A definition is suggested which implies that the model is in the first machine class. It is shown that without loss of generality it suffices to consider only parallel Turing machines of certain normal forms.

1 Introduction

Hemmerling (1979) was probably the first to consider a model where several finite automata are working cooperatively on one common tape. It is known that this model can simulate cellular automata and vice versa in linear time and linear space simultaneously. Wiedermann (1984) considered these finite automata as TM control units and generalized the model to so-called *Parallel Turing Machines* (PTM) in the same way as one-tape one-head Turing machines have been generalized to Turing machines with possibly several tapes and several heads on each tape.

Later Wiedermann (1992) introduced the notion of weak parallel machines and showed that the parallel Turing machines are an example of this class of machines. This means that TM space and PTM pipeline period are polynomially related which is particularly interesting because on the other hand it seemed that TM space and PTM space were *not* linearly related and hence PTM were not in the first machine class. It will become clear in later sections that this has to do with the definition used for the space complexity of PTM. In definition 4.4 we will propose a different notion of space complexity for PTM which makes them a member of the first machine class (theorems 5.1 and 5.3).

Further information on a taxonomy for parallel machine models and their relations can be found in the papers of van Emde Boas (1990) and Wiedermann (1995a,b).

The rest of this paper is organized as follows: In section 2 we give the definitions of PTM and some related complexity measures with the exception of space complexity. The latter is considered in detail in section 4 because an adequate definition plays an important rôle for later results. In section 3 it is shown that it is often sufficient to consider only PTM of some special types. These results are used in section 5 where relations of PTM to sequential Turing machines are investigated.

In this paper we shall only consider deterministic machines with one-dimensional tapes.

2 Basic notions

In this section we introduce parallel Turing machines (essentially following Wiedermann (1984)) which consist of a finite constant number of tapes with several control units working on them cooperatively. Also three obvious complexity measures will be introduced, but *not* space complexity which is the topic of section 4. As computational tasks the complexities of

[*]This is technical report 11/96 of the Department of Informatics, University of Karlsruhe.

[†]Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler, Universität Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe, Germany.

which are to be measured the recognition of formal languages will be used (unless explicitly stated otherwise).

2.1 Definition. *A* ***parallel Turing machine*** *is specified by a tuple $P = (Q, F_+, F_-, (h_1, \dots, h_k), B, I, \delta, q_0, \Box, \llcorner\!\lrcorner)$. Q is the finite set of states, containing at least the initial state q_0. $F_+ \subset Q$ is the the set of accepting final states and $F_- \subset Q$ is the the set of rejecting final states with $F_+ \cap F_- = \emptyset$. $(h_1, \dots, h_k)$ is the type of the PTM P; k is its number of tapes, and each control unit has h_i heads on tape i for $1 \le i \le k$. B is the tape alphabet, containing the blank symbol $\Box$. $I \subset B$ is the input alphabet. The "no write" symbol $\llcorner\!\lrcorner$ is not an element of B, but it may be used in the specification of the transition function*[1]

$$\delta : Q \times B^{h_1} \times \cdots \times B^{h_k} \to \mathfrak{P}(Q \times B_{\llcorner\!\lrcorner}^{h_1} \times \cdots \times B_{\llcorner\!\lrcorner}^{h_k} \times D^{h_1} \times \cdots \times D^{h_k})$$

where $B_{\llcorner\!\lrcorner} = B \cup \{\llcorner\!\lrcorner\}$ and $D = \{-1, 0, 1\}$. For the interpretation of δ see definition 2.3 below.

The type $(h_1, \dots, h_k)$ of a (parallel) TM will sometimes be denoted as $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$. Instead of $\mathbb{T}_h \cdots \mathbb{T}_h$ (k times) we'll write $(\mathbb{T}_h)^k$, and if the number of heads and/or tapes can be chosen arbitrarily, notations like $(\mathbb{T}_*)^*$ will be used. We are only considering one-dimensional tapes in this paper.

2.2 Definition. *A* ***configuration*** *$c = (b_1, \dots, b_k, u)$ of a PTM is given by the inscriptions $b_i : \mathbb{Z} \to B$ of all tapes $(1 \le i \le k)$ and by a finite list $u = (u_1, \dots, u_m)$ where $u_{j_1} \ne u_{j_2}$ for $j_1 \ne j_2$ and each $u_j \in Q \times \mathbb{Z}^{h_1} \times \cdots \times \mathbb{Z}^{h_k}$ specifies the state of a control unit and the positions of all of its heads on their tapes.*

In the ***initial configuration*** *c_w for an input word $w \in I^+$ there is always only one control unit in the inital state with all heads positioned on cell 0 of the tapes, all of which are empty except the first one, on which the input is written into cells $0, \dots, |w| - 1$ symbol by symbol. Formally $c_w = (b_1, b_2, \dots, b_k, ((q_0, 0, 0, \dots, 0)))$ where $b_i(x) = \Box$ for all $2 \le i \le k$ and all $x \in \mathbb{Z}$, and $b_1(x) = w[x]$ for $1 \le x \le |w|$ and $b_1(x) = \Box$ otherwise.*

A configuration is a ***final*** *one, if and only if there is exactly one control unit and it is in a final state. A final configuration is* ***accepting*** *if and only if the state of the only control unit is an accepting one, otherwise it is a* ***rejecting*** *final configuration.*

2.3 Definition. *The* ***dynamics*** *of a PTM P, i.e. the partial mapping Δ describing one step of P, leading from a configuration $c = (b_1, \dots, b_k, u)$ to its successor configuration c' (if defined), can be described as consisting of four substeps:*

1. *Each control unit specified by $(q, p_{1,1}, \dots, p_{k,h_k})$ reads the symbols on the cells it is scanning with its heads. Formally $b_{i,j} = b_i(p_{i,j})$ for all $1 \le i \le k$ and all $1 \le j \le h_i$.*
2. *Each control unit in state q and having read symbols $b_{1,1}, \dots, b_{k,h_k}$ is replaced by the set $\delta(q, b_{1,1}, \dots, b_{k,h_k}) = \{v_1, \dots, v_l\}$ of new "control units" $v_j = (q_j, b'_{1,1}, \dots, b'_{k,h_k}, d_{1,1}, \dots, d_{k,h_k})$. If the set is empty, the old control unit is simply deleted.*
3. *Each new control unit tries to write $b'_{i,j}$ on cell $p_{i,j}$ if $b'_{i,j} \ne \llcorner\!\lrcorner$; otherwise the head does not write anything. If there are at least two heads somewhere in the configuration on the same cell trying to write different symbols onto it, the computation is illegal and its result is undefined. Otherwise the symbols are written on the tapes.*
4. *If no write conflict happened in the previous substep, each head "moves" to the cell $p'_{i,j} = p_{i,j} + d_{i,j}$ for all $1 \le i \le k$ and all $1 \le j \le h_i$ afterwards. This finally yields the new configuration c'.*

In the last paragraph it has to be understood that if two control units are in the same state and if their corresponding heads are positioned on the same tape cell, then they are considered to be only *one* control unit since they will behave identically and cannot be distinguished any more.

[1] $\mathfrak{P}(M)$ denotes the set of all subsets of M.

As usual one can define several measures for the resources needed for the recognition of a formal language. In the following it will always be assumed that all PTM involved reach a final configuration for every input. Hence in the next definition all functions are total.

2.4 Definition. *The* ***time complexity*** $t(n)$ *of a PTM is the maximal number of steps it needs for any input of length* n *to reach a final configuration.*

The ***tape complexity*** $r_t(n)$ *of a PTM is the maximal number of cells on one of the tapes which are used during a computation for any input of length* n*. A cell is used during a computation if in at least one configuration its inscription is not* $\square$ *or it is visited by a head of a control unit.*

The ***processor complexity*** $r_p(n)$ *of a PTM is the maximal number of control units which simultaneously exist in a configuration occuring during a computation for an input of length* n*.*

2.5 We shall for example write $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$–PTM–TAPE–PROC–TIME$(r_t, r_p, t)$ for the family of all languages which can be recognized by PTM of type $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$ which have time complexity $\leq t$, tape complexity $\leq r_t$ and processor complexity $\leq r_p$. (See also remark 4.5.) $\square$

2.6 A bound on the tape complexity always implies a bound on the processor complexity. For a given r_t there are at most r_t^h possibilities to chose different h-tuples of head positions for a control unit with $h = h_1 + \cdots + h_k$ heads. And each such h-tuple can be used by at most $|Q|$ control units. Hence always $r_p \leq |Q| \cdot r_t^h$. Since any "reasonable" PTM has to investigate all input symbols and therefore has $t(n) \geq n$ and since the number of used tape cells can only grow linearly with time, r_t and r_p are always bounded by polynomials in t. $\square$

One notices that we didn't introduce something called space complexity. It might be tempting to name tape complexity as space complexity. For example this is the approach of WIEDERMANN (1992). Later the present author suggested to use the sum of tape and processor complexity as the space complexity. But as should become clear from the considerations in the section following the next one, there are good reasons for preferring a more complicated definition.

But before we will show that for the proofs of many results it is sufficient to consider only PTM of some special types.

3 Normal forms for PTM

The aim of this section is to show that as far as the type $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$ of a PTM is concerned what usually is of importance is only the total number of heads per control unit but not how they are distributed onto tapes. More precisely as long as two PTM have the same number of heads per control unit they can simulate each other with a linear overhead of time, tape and processors, no matter how many tapes are involved.

The analogous of problem of reducing the number of heads per control unit is different. In general it is impossible without a nonlinear increase of tape, processor and time complexity. We will return to this point at the end of section 5.

3.1 Lemma. (Merging of tapes) *For each* $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$*–PTM* P *with* $k \geq 2$ *tapes there is a* $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_{k-1}+h_k}$*–PTM (with* $k-1$ *tapes)* P' *simulating each step of* P *in constant time and on linear tape with the same number of processors.*

3.2 Proof. We only describe the construction for the special case $k = 2$. The generalizations for larger k are obvious.

For each configuration $c = (b_1, b_2, u)$ of P we define the "corresponding" configuration $\overline{c} = (b', u')$ of P' such that there is a constant s with $\Delta_{P'}^s(\overline{c}) = \overline{\Delta_P(c)}$, the number of tape cells needed in $\overline{c}$ is linearly related to that needed in c, and s does not depend on c.

P' uses the same tape alphabet as P and the tape inscription b' is defined as[2]

$$\forall x \in \mathbb{Z}\, \forall y \in \mathbb{Z}_2 :\ b'(2x+y) = b_y(x)$$

Symbols of adjacent tape cells in c are 2 tape cells apart in $\overline{c}$.

For each control unit of P in c there is exactly one control unit of P' in $\overline{c}$. If in c the head of a control unit is positioned on cell j of tape i the position of the corresponding head of the corresponding control unit in $\overline{c}$ is $2j+i$ (on the only tape).

Each step of P is simulated by 2 steps of P' which are needed to simulate the movement of heads onto cells carrying the symbols of adjacent tape cells in c.

Since two heads are visiting the same tape cell in c if and only if the corresponding heads are visiting the same tape cell in $\overline{c}$, it is easy to see that there are no more and no less write conflicts in P' than in P. ■

In the previous proof we have ignored the problem of what happens to inputs. In definition 2.2 it has been required that an input is provided symbol by symbol on *successive* tape squares, while for the previous simulation to work it is required that they stored on every other tape square. The additional time needed to rearrange the input symbols therefore has to be taken into account for all theorems using the above construction. This time of course depends on how many processors may be used for the rearrangement. For example:

3.3 Corollary. *For each $\mathbb{T}_{h_1}\cdots\mathbb{T}_{h_k}$–PTM P (with $k \geq 2$ tapes) there is a $\mathbb{T}_{h_1}\cdots\mathbb{T}_{h_{k-1}+h_k}$–PTM (with $k-1$ tapes) P' recognizing the same language as P with a linear overhead of time and tape and with the same number of processors, if one of the following (sufficient but not necessary) conditions holds:*

- *The time complexity r_t of P is $r_t \geq n^2$.*
- *The time complexity r_t of P is $r_t \geq n$ and its processor complexity r_p is $r_p \geq n$.*

The following result essentially is the reverse of lemma 3.1: If there are at least two heads of each control unit on one tape, those heads can be "separated" by splitting off additional tapes.

3.4 Lemma. (Separation of heads) *For each $\mathbb{T}_{h_1}\cdots\mathbb{T}_{h_{k-1}+h_k}$–PTM P (with $k-1$ tapes) there is a $\mathbb{T}_{h_1}\cdots\mathbb{T}_{h_k}$–PTM P' (with k tapes) simulating each step of P in constant time with a linear overhead of processors on the same amount of tape.*

3.5 Proof. The construction is described only for the transition $\mathbb{T}_2 \to \mathbb{T}_1\mathbb{T}_1$. It can be generalized to the other cases easily.

The idea is that every time P' wants to simulate one step of P both tapes of P' contain the same inscription, namely that of P and to make sure that this condition is also satisfied after the simulation of the step.

Figure 1 shows a configuration c of a $\mathbb{T}_2$–PTM and the corresponding configuration $\overline{c}$ of a $\mathbb{T}_1\mathbb{T}_1$–PTM. In $\overline{c}$ there are two types of contol units, which are called *simulation CUs* and *maintenance CUs*.

For each CU U present in c there is exactly one simulation CU $\overline{U}$ in $\overline{c}$. If $p_{1,1}$ and $p_{1,2}$ are the positions of the first and second head resp. of U on the only tape, then the positions of the heads of $\overline{U}$ are $\overline{p}_{1,1} := p_{1,1}$ on the first tape and $\overline{p}_{2,1} := p_{1,2}$ on the second.

For each tape cell x in c, which is visited by at least one head, there is a maintenance CU of $\overline{c}$ visiting tape cells x on both tapes.

Furthermore the description below will make use of so called *conflict CUs*. Their only purpose will be to enforce a write conflict if necessary. Therefore there are two types of conflict CUs, one always writing one fixed symbol with all of its heads and the other always writing another fixed symbol. If generated at all, conflict CUs will alsways be generated in pairs, one of each type, resulting in a write conflict.

[2] $\mathbb{Z}_2$ denotes $\{0,1\}$.

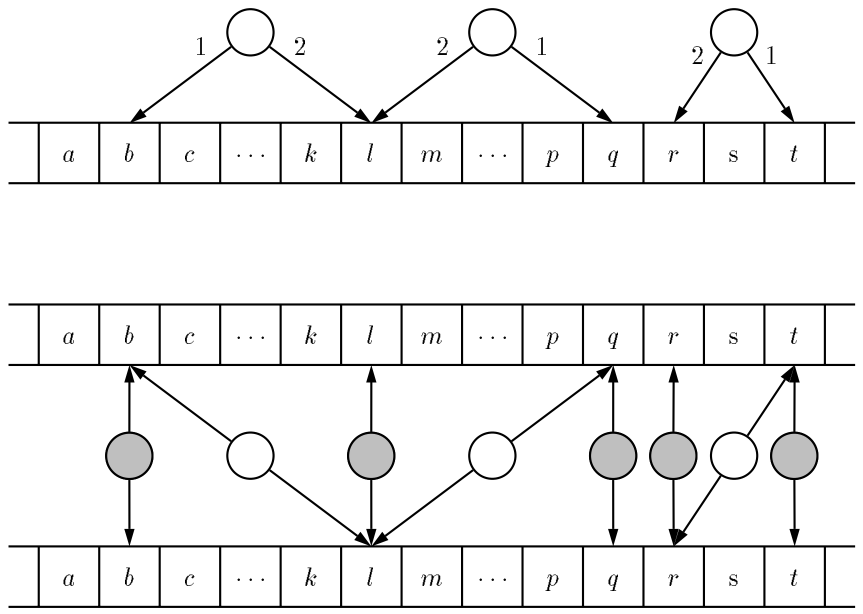

Figure 1: A $\mathbb{T}_2$–configuration (top) and its corresponding $\mathbb{T}_1\mathbb{T}_1$–configuration (bottom). The simulation CUs are white and the maintenance CUs are gray shaded.

The tape alphabet B' of P' is $B' := B \cup ((B \cup \hat{B}) \times \mathfrak{P}(\{\leftarrow, \downarrow, \rightarrow\}))$ where $\hat{B}$ contains a copy $\hat{a}$ for each $a \in B$.

The simulation of one step of P is carried out by P' in six steps:

1. Each simulation CU $\overline{U}$ reads the symbols on the tape cells it is currently scanning with its heads. According to the transition function of P determines how U would behave in c: $\overline{U}$ stores in its memory the states of the new CUs replacing U in c (without already generating their simulating counterparts; this will happen in step 6), and also stores the movements of the heads of the new CUs. Furthermore $\overline{U}$ determines (separately for each of the tape cells visited by its heads) whether the new CUs

   (a) would cause a write conflict in c[3],

   (b) would not write anything at all, or

   (c) would consistently write a symbol a.

   Respectively $\overline{U}$

   (a) will generate a pair of conflict CUs,

   (b) will write $(a, \emptyset)$ (where a is the original cell symbol), or

   (c) will store the information that it has to write the copy $(\hat{a}, \emptyset)$ on the respective tape cell in the next step.

2. $\overline{U}$ will write $(\hat{a}, \emptyset)$ if it has to. This may lead to a write conflict, if another CU tries to write a different $(\hat{a}', \emptyset)$ which is okay because it means that there would be a write conflict in P, too.

3. The next three steps are used to add to each currently visited tape cell the information whether because of the simulation there will be a head moving from this cell to the left, whether there will be a head moving to the right, and whether there will be a head

[3]Note that these are not all write conflicts possible; the others will be "discovered" later.

staying on the same cell. One should note that it is necessary to do this in three steps in order to avoid write conflicts for which there is no corresponding write conflict in P.

First, if a simulation CU $\overline{U}$ will generate a new one which will move one of its heads from a tape cell x to the left, $\overline{U}$ adds $\leftarrow$ to the set of directions of the symbol on cell x.

4. Next, the analogous action is carried out for the opposite direction, adding $\rightarrow$ if a head will move to the right.

5. Finally the analogous action is carried out for heads staying in place, adding $\downarrow$ to the symbol.

6. Now the simulation can be completed:

   (a) Each simulating CU $\overline{U}$ replaces itself by a set of new simulating CUs in correspondence to the information stored in step 1 which do not write anything onto the tape (this has already been done also in step 1), but only move (if necessary) to neighboring tape cells.

   (b) Each maintenance CU checks whether its heads read *copies* of *different* symbols from B. If this is the case, in the original PTM P a write conflict occured, and P' therefore generates a pair of conflict CUs.

   Otherwise the maintenance CU deletes the information on head movements and generates a corresponding set of new maintenance CUs after copying newly written tape symbols from one tape to the other (replacing copies by originals).

The number of maintenance CUs and that of conflict CUs is bounded by twice the number of heads per original CU times the number of CUs in c. In both c and $\overline{c}$ the same amount of tape is needed. Hence the above construction satisfies the complexity requirements of the lemma. ■

By repeated applications of the above lemmata it is easy to see that the following holds (again not speaking about language recognition for which some additional time which might be necessary to rearrange the input).

3.6 Corollary. *If P and P' are PTM of types $(h_1, \ldots, h_k)$ and $(h'_1, \ldots, h'_{k'})$ respectively and $h_1 + \cdots + h_k = h'_1 + \cdots + h'_{k'}$ then each step of P can be simulated by P' in a constant number of steps and with a linear overhead of tape and processors.*

3.7 For example as long as one can afford to ignore an increase of time, tape and processor complexity by at most a constant factor, *it suffices to consider PTM with one tape or with control units which have ony one head on each tape.* □

4 Space complexity for PTM

What is wrong with calling the tape complexity of a PTM its space complexity?

This depends on the point of view of course. On one hand it is merely a definition and one may define whatever one wants to. On the other hand there are usually reasons for preferring certain definitions over others; for example one would probably have to face objections if one would call the number of tapes of a Turing machine its time complexity.

As far as space complexity is concerned, it has been discussed in some papers on other machine models that sometimes there are good reasons for considering something not so obvious as the space complexity.

We therefore begin with some general remarks. Their consequences for PTM are the topic of the subsequent subsection.

Some general remarks on space complexity

The problem of giving a definition of space complexity which is in some sense reasonable has already come up in the literature before for different models.

One example are RAM and the discussion on uniform versus logarithmic measures.

Another example, which perhaps is less known but more relevant for PTM, are the *storage modification machines* by SCHÖNHAGE (1980), also called *pointer machines* (PM) in more recent papers (LUGINBUHL AND LOUI, 1993). One of the first papers which explicitly discusses the problem of space complexity for PM is by VAN EMDE BOAS (1989).

In PM all information is stored as a directed edge-labeled graph. A program for a PM consists of instructions which allow to add nodes to the graph and to redirect edges from one target to another thereby modifying the graph. The nodes of such a graph cannot store any information, except that one node is designated as its "center". It is the *structure* of the graph in which all information is stored.

In early papers on PM (see e.g. HALPERN ET AL., 1986) the maximum number of nodes used during a computation is used as the space measure. But there are $\Theta(r^{dr})$ directed graphs with r nodes and degree at most d (which is a constant for each PM), and hence in the worst case one needs $\Theta(r \log r)$ bits[4] to write down the description of a configuration of a PM.

It turns out, that not only are there this many graphs, but they can really be used as configurations by an PM to store information:

4.1 Theorem. (VAN EMDE BOAS (1989), LUGINBUHL AND LOUI (1993))

- *Every multi-tape TM with space complexity s can be simulated by a PM with $\mathrm{O}(s/\log s)$ nodes in real time.*
- *Every PM with $\mathrm{O}(s/\log s)$ nodes can be simulated by a multi-tape TM with space complexity s in polynomial time.*

Let $r(n)$ denote the maximal number of nodes which are used for any graph occuring during computations of a given PM for inputs of size n. And let $C(n)$ denote the number of different PM configurations with at most $r(n)$ nodes. Then the above results say that it is also meaningful to consider $\log C(n) \in \mathrm{O}(r(n) \log r(n))$ as a space measure for PM. Consequently VAN EMDE BOAS (1989) distinguishes *uniform space measure* (the number of nodes) and *logarithmic space measure* (the logarithm of the number of configurations). LUGINBUHL AND LOUI (1993) avoid the term space measure completely and speak of *mass* and *capacity* instead.

4.2 In general it seems reasonable to us to use as a guideline the requirement that *the space complexity should measure the amount of information stored in configurations,* i.e. the number of bits needed to write down unique descriptions of configurations, at least approximately. □

Of course, there are potential problems with this formulation.

1. First of all, one has to have a notion of *configuration* and it has to be reasonable itself. It is by no means clear why this should always be the case. But it seems that at least this happens more often than one immediately has a "good" notion of space complexity. PM and also multi-head PTM are examples where only the former seems to be the case.

2. Now assume that some reasonable definition of configuration is given. Then the other problem is to find out, how much information is stored in a configuration. One approach which is used in many cases is the following: The model under consideration has at least one kind of "resource" and each configuration uses the resource to some extent. Limiting the amount of resources available to $r(n)$ also limits the number of configurations to $C(n) = f(r(n))$.

[4]Let log always denote the logarithm to base 2.

3. One might also have to think about the problem whether all of the $C(n)$ configurations counted can really occur during computations, and if they can whether the machine can really "extract" *all* the information.

For example in the case of TM the resource is the cells on the tapes and in the case of PM the resource is the nodes.

It should be noted, that even the usual definition of space complexity for one-tape one-head TM is a little bit more complicated than described above. Let $r_t(n)$ denote the maximal number of tape cells used during computations for inputs of size n, let Q denote the state set and B the tape alphabet. Then there are $C(n) = |B|^{r_t(n)} \cdot r_t(n) \cdot |Q|$ different configurations. Hence $\log C(n) = \log |Q| + \log |B| \cdot r_t(n) + \log r_t(n)$ which is different from $r_t(n)$.

Since the "usual" point of view is that one may choose B and Q arbitrarily, and since $\log C(n) \in \Theta(r_t(n))$ it is reasonable to define the space complexity of one-tape one-head TM to be r_t.

Observe also, that because of this, $r_t(n)$ turns out to be compressible by constant factors without any slow-down or other "disadvantages".

Space complexity for PTM

After the discussions in the previous subsection it probably doesn't come as a big surprise, that we will take into account the tape complexity and the processor complexity when suggesting a definition of space complexity of PTM.

4.3 Let P be a PTM of type $\mathbb{T}_{h_1} \cdots \mathbb{T}_{h_k}$ with $h = h_1 + \cdots + h_k$, tape complexity r_t and processor complexity r_p. How many configurations are there respecting these resource limits?

There are $(\sum_{i=1}^{r_t} |B|^i)^k = \Theta(|B|^{kr_t})$ different tape inscriptions.

There are $|Q| r_t^h$ different possibilities to choose a state and positions $x = (x_1, \ldots, x_h)$ for the h heads of a control unit (see 2.6). From these $i \leq r_p$ have to be chosen. That makes $\sum_{i=1}^{r_p} \binom{|Q|r_t^h}{i}$ possibilities. This sum can be bounded by $\binom{|Q|r_t^h}{r_p}$ from below and by $r_p \binom{|Q|r_t^h}{r_p}$ from above. (At least if $r_p < \frac{1}{2}|Q|r_t^h$; the other case is trivial.)

This gives a total of approximately

$$C = \Theta\left((|B|^k)^{r_t} \cdot \binom{|Q|r_t^h}{r_p} \right) \mathrm{O}(r_p)$$

different configurations. Taking logarithms we can ignore the $\mathrm{O}(r_p)$, and because of $\frac{(x-y)^y}{y^y} \leq \binom{x}{y} \leq \frac{x^y}{y!} \leq \frac{x^y e^y}{y^y}$ it follows that

$$\log C \in \Theta\left(r_t + r_p + r_p \log \frac{|Q|r_t^h}{r_p} \right)$$

□

This motivates:

4.4 Definition. *The* ***space complexity*** *of a PTM P is*

$$s(n) = r_t(n) + r_p(n) + r_p(n) \log \frac{|Q|r_t^h(n)}{r_p(n)}$$

where $r_t(n)$ is the tape complexity of P and $r_p(n)$ its processor complexity.

This definition corresponds to the fact that a PTM can store information on the tape, in the states of the control units and in the structure of the graph built by the connections between tape cells and control units via the read/write heads.

We could not drop the factor $|Q|$ in the above definition because otherwise the logarithm might become negative.

As in the definition of space complexity of sequential TM we have ignored constant factors. But unfortunately, for PTM it is *not* known how this can be compensated for by a kind of compression. It is possible to reduce the tape complexity, but it is not known whether it is also possible for processor complexity without significantly increasing the time complexity.

As a matter of fact, if there is such a result it has to exclude at least the case of PTM with only one tape and two processors with one head each. It is known that in this case a reduction of the processor complexity from 2 to 1 (i.e. by only a constant summand) must increase the time complexity by a factor of $\log n$ (see Worsch, 1991).

4.5 To avoid problems arising from the facts that we have arbitrarily ignored any constant factors in the above definition but on the other hand don't know any space compressability results, we will be careful only to make statements about PTM with a space complexity in $\Theta(s)$ (and not exactly s). □

4.6 Note that for example for $r_p \in \mathrm{O}(r_t/\log r_t)$ we have $s \in \Theta(r_t)$. If for some r both $r_t, r_p \in \Theta(r)$ then $s \in \Theta(r \log r)$ and if $r \in \Theta(s/\log s)$ then the space complexity is in fact $\Theta(s)$. □

5 Relations of PTM to sequential TM

5.1 Theorem.

$$\begin{aligned}\mathbb{T}_1\text{–}\textsc{Tm}\text{–}\textsc{Spc}\text{–}\textsc{Time}(s,t) &\subseteq \mathbb{T}_2\text{–}\textsc{Ptm}\text{–}\textsc{Tape}\text{–}\textsc{Proc}\text{–}\textsc{Time}(\Theta(s/\log s), \Theta(s/\log s), \Theta(t)) \\ &\subseteq \mathbb{T}_2\text{–}\textsc{Ptm}\text{–}\textsc{Spc}\text{–}\textsc{Time}(\Theta(s), \Theta(t))\end{aligned}$$

The second inclusion is already obvious because of 4.6. As one can see from the first inclusion, it is possible to store most of the information about the TM configurations "in the structure of the PTM graph" (and neither on the tape nor in the control units of the PTM).

5.2 Proof. One can use a construction almost identical to the one given by Luginbuhl and Loui (1993) (LL for short) in the proof of their theorem 4.1. In fact since we are only interested in a linear time and not in a real time simulation the first part of that proof is sufficient.

We therefore restrict ourselves to a few remarks:

- LL describe a pointer machine simulating a Turing machine while here a PTM has to be constructed.
- LL construct a directed graph with outdegree 3. In the PTM the nodes can be represented by tape cells and the outgoing arcs to other tape cells by control units with 2 heads; head 1 points to the source of the arc and head 2 to its destination.
- One has to be careful about the fact, that in PTM the heads can only move from one tape cell to an adjacent one while in pointer machines there are instructions which can make an arc jumping to an arbitrary other node. Fortunately a close inspection shows that every part of the contruction given by LL can be carried over to PTM.

■

We continue by describing the reverse simulation of PTM by TM.

5.3 Theorem. *For all $h \in \mathbb{N}$ holds:*

$$(\mathbb{T}_1)^h\text{–}\textsc{Ptm}\text{–}\textsc{Spc}\text{–}\textsc{Time}(s,t) \subseteq \mathbb{T}_1\text{–}\textsc{Tm}\text{–}\textsc{Spc}\text{–}\textsc{Time}(\Theta(s), \mathrm{Pol}(t))$$

where the degree of the polynomial depends on h.

Here we are not interested in minimizing the degree of the polynomial bounding the overhead of the time complexity. Instead what we do want is a simulation which preserves the space complexity. Therefore in general it is *not* possible for the TM to store for each control unit the positions of all heads in binary. This needs $r_p \log r_t$ space which might be significantly more than the space complexity of the simulated PTM (for example if $r_p \in \Theta(r_t^h)$).

5.4 Proof. Once again we first consider the case $h = 2$. The generalization of the construction to more heads is straightforward.

The states of the control units and the positions of their heads in a configuration of a $(\mathbb{T}_1)^2$-PTM P with tape complexity r_t (for some input size n) can be depicted as a square S of size $r_t \times r_t$. The entry $S_{ij} \in \mathfrak{P}(Q)$ in row i $(1 \leq i \leq s)$ and column j $(1 \leq j \leq s)$ is the set of all states of control units which have their heads on cell i of the first tape and on cell j on the second tape; see figure 2 for an example.

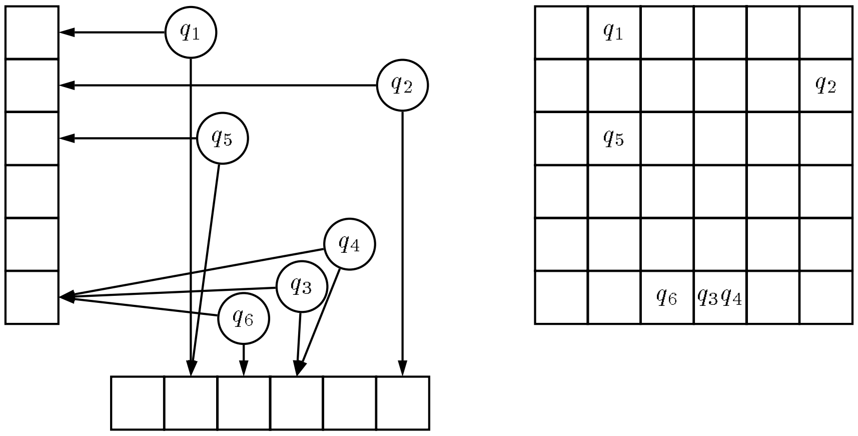

Figure 2: Transforming the "states part" of a $(\mathbb{T}_1)^2$-PTM configuration into a square.

Such a square S will be encoded as a string $\hat{S}$ which is the concatenation of the encodings of the rows of S.

A row of S is encoded as follows: If all entries S_{ij} of a row are the empty set, then it is encoded as the two letter string `[ ]`. If for the i-th row $j_1, \ldots, j_f$ are all columns such that $S_{ij_g} \neq \emptyset$ for $1 \leq g \leq f$, then it is encoded as the word `[`$\mathrm{bin}(j_1 - 1) S_{ij_1} \mathrm{bin}(j_2 - j_1 - 1) S_{ij_2} \cdots \mathrm{bin}(j_f - j_{f-1} - 1) S_{ij_f} \mathrm{bin}(s - j_f)$`]` where $\mathrm{bin}(x)$ is the binary representation of x without leading zeroes (except for $x = 0$) and the S_{ij} are encoded by simply writing its states side by side. In other words the non empty sets of states are separated by the binary representations of the numbers of empty entries in between them.

For example the states from the configuration in figure 2 would be encoded as the word $\hat{S} =$ `[1`q_1`100][101`q_2`0][1`q_5`100][ ][ ][10`q_6`0`q_3q_4`10]`.

It remains to show

1. that a $\mathbb{T}_1$–TM T can use this encoding of PTM configurations to simulate a PTM P in polynomial time and
2. that this kind of encoding needs $\mathrm{O}(r_t + r_p + r_p \log \frac{|Q| r_t^h}{r_p})$ TM space.

Assuming that the second claim has already been shown it is straightforward to verify the first one: T stores $\hat{S}$ and the tape inscriptions of P. For one step of P it simulates one control unit after the other constructing incrementally (on separate tracks) the new tape inscriptions and the encoding $\hat{S}'$ of the new "square of states". To simulate the work of one control unit T needs to find out the tape symbols currently read and then accroding to δ_P update the preliminary version of $\hat{S}'$. Obviously both can be done in a time polynomial in the length of $\hat{S}$ which itself is polynomial in r_t and r_p; these in turn are polynomial in the time complexity of P.

It finally needs to be shown that the length of $\hat{S}$ really is in $\mathrm{O}(r_t + r_p + r_p \log \frac{|Q|r_t^h}{r_p})$.

First of all it is obvious that the number of occurrences of [and] symbols is in $\Theta(r_t)$ and that the number of occurrences of $q \in Q$ is in $\Theta(r_p)$.

It remains to estimate the total length of all maximal subwords $v \in \{0,1\}^+$ of $\hat{S}$. The number of these words is bounded by $\mathrm{O}(r_p)$, so it suffices to show that in the worst case (i.e. in the case of configurations which for fixed r_t and r_p have the longest encodings) there are $\Theta(r_p)$ words the lengths of which are in $\mathrm{O}(\log \frac{|Q|r_t^2}{r_p})$.

To this end we determine the worst case length of a different word $\tilde{S}$ corresponding to the same configuration. From its definition it will be clear that $\frac{1}{2}|\hat{s}| \leq |\tilde{s}| \leq 2|\hat{s}|$ where $\hat{s}$ is the length of the word one obtains by deleting all subwords [ ] from $\hat{S}$ and $\tilde{s} = |\tilde{S}|$. Hence it is enough to consider $\tilde{S}$ instead of $\hat{S}$ in order to deduce the desired upper bound. To construct $\tilde{S}$ first concatenate all rows of S which contain at least one non empty entry. This gives one long row R which is then encoded as describe above. In the example of figure 2 one gets $\tilde{S} =$ [1$q_1$1001$q_2$1$q_5$110$q_6$0$q_3q_4$10].

In the following when we speak of a $\emptyset$-sequence what we mean is a subsequence of R which consists of empty sets only and which is maximal in this respect. The lengths of the $\emptyset$-sequences are the binary numbers occuring in $\tilde{S}$.

Let l be the length of a longest $\emptyset$-sequence in R and k be the length of a shortest $\emptyset$-sequence. Let $d = \lfloor (l-k)/2 \rfloor$. Assume that $l \geq 7k$; hence $d \geq 3k$. Construct a new long row R' by deleting d empty sets in a longest $\emptyset$-sequence L of R and inserting them in a shortest $\emptyset$-sequence K. Let $\tilde{s}'$ be the length of the encoding of R'. Call the lengths of the resulting $\emptyset$-sequences l' and k'. Then $l' \geq \frac{l}{2}$ and $k' \geq k + 3k \geq 4k$ and therefore

$$\tilde{s}' - \tilde{s} = |\operatorname{bin}(l')| - |\operatorname{bin}(l)| + |\operatorname{bin}(k')| - |\operatorname{bin}(k)| \geq -1 + 2 = 1.$$

Similarly one finds that splitting one long $\emptyset$-sequence of length l in to two subsequences of lengths $\lfloor \frac{l-1}{2} \rfloor$ and $\lceil \frac{l-1}{2} \rceil$ by making a middle set non empty increases the length of the encoding.

This means that of all encodings with the same r_t and r_p those with the longest encodings have as many $\emptyset$-sequences as possible satisfying the property $l < 7k$. Consequently in an encoding of maximal length (for given r_t and r_p) no $\emptyset$-sequence is longer than $\mathrm{O}(\frac{r_t^2}{r_p})$ which in an encoding needs $\mathrm{O}(\log \frac{|Q|r_t^2}{r_p})$ space.

■

From theorems 5.1 and 5.3 immediately follows:

5.5 Corollary.

$$\text{PTM–SPC–TIME}(\Theta(s), \mathrm{Pol}(t)) = \text{TM–SPC–TIME}(\Theta(s), \mathrm{Pol}(t))$$

5.6 Corollary. *Parallel Turing machines are in the first machine class.*

This is in contrast with a statement of WIEDERMANN (1992) the reason of course being that the space measures involved are different.

We conclude this section with a remark on the reduction of the number of heads per control unit in PTM. It has been shown that the space complexities of sequential and parallel Turing machines are linearly related. Therefore the space hierarchy theorems for TM carry over to the PTM. Consider for example PTM P with h-heads control units, $h \geq 2$, some tape complexity r_t (which is always greater or equal n) and processor complexity $r_p \in \Theta(r_t^h)$; then the space complexity also is $s \in \Theta(r_t^h)$. Because of the space hierarchies it is therefore impossible to simulate P by another PTM P' the control units of which have only $h-1$ heads on the same amount of tape because there the space complexity is at most $s \in \Theta(r_t^{h-1})$.

Hence in general a reduction of the number of heads is impossible without increasing tape and/or processor complexity. There are special cases, for example if $r_p \in \Theta(r_t)$, in which it is possible to simulate a PTM with h-heads control units by a PTM with $(h-1)$-heads control units with a linear overhead in tape, processor and time complexity. But a general treatment of this problem is still beyond our knowledge.

6 Conclusion

In this paper parallel Turing machines have been investigated. Emphasis has been put on a certain definition of space complexity. Using this definition it could be shown that PTM are in the first machine class. It has been shown that the number of tapes of a PTM is not important: All PTM with the same total number of heads per control unit can simulate each other with linear overheads in time, tape and processors. Reducing the number of heads sometimes requires a non linear increase of the tape complexity.

Some problems remain open, concerning even some very basic questions one is used to know the answers for in the case e.g. of (sequential) Turing machines: It is not known how to reduce the number of processors by a constant factor without increasing the time complexity by more than a constant factor. The same is true for the space complexity. And it is not even known how to speed up a PTM by a constant factor.

Acknowledgement

The author gratefully acknowledges the inspiring atmosphere and stimulating discussions with Jiří Wiedermann and his group during a stay at the Academy of Sciences of the Czech Republic in March 1996.

References

Halpern, J. Y., Loui, M. C., Meyer, A. R., Weise, D. (1986): On time versus space III. *Mathematical Systems Theory*, **19**, 13–28.

Hemmerling, A. (1979): Systeme von Turing-Automaten und Zellularräume auf rahmbaren Pseudomustermengen. *Journal of Information Processing and Cybernetics EIK*, **15**, 47–72.

Luginbuhl, D. R., Loui, M. C. (1993): Hierarchies and space measures for pointer machines. *Information and Computation*, **104**, 253–270.

Schönhage, A. (1980): Storage modification machines. *SIAM Journal on Computing*, **9**, 490–508.

van Emde Boas, P. (1989): Space measures for storage modification machines. *Information Processing Letters*, **30**, 103–110.

van Emde Boas, P. (1990): *Machine Models and Simulations*, vol. A of *Handbook of Theoretical Computer Science*, chap. 1, pp. 1–66. Elsevier Sceince Publishers, Amsterdam.

Wiedermann, J. (1984): Parallel Turing machines. Tech. Rep. RUU-CS-84-11, University Utrecht, Utrecht.

Wiedermann, J. (1992): Weak parallel machines: a new class of physically feasible parallel machine models. In: I. M. Havel, V. Koubek (eds.), *MFCS '92, 17th International Symposium Mathematical Foundations of Computer Science*, vol. 629 of *LNCS*, pp. 95–111, Springer.

Wiedermann, J. (1995a): Parallel machine models: How they are and where are they going. In: *SOFSEM'95: Theory and Practice of Informatics*, vol. 1012 of *LNCS*, pp. 1–30, Springer-Verlag.

Wiedermann, J. (1995b): Quo vadetis, parallel machine models. In: J. van Leeuwen (ed.), *Computer Science Today*, vol. 1000 of *LNCS*, pp. 101–114, Springer-Verlag.

Worsch, Th. (1991): *Komplexitätstheoretische Untersuchungen an myopischen Polyautomaten*. Dissertation, Technische Universität Braunschweig.