Institute of Computer Science
Academy of Sciences of the Czech Republic

Computing by Broadcasting: Alternation in Disguise (Extended Version)

Jiří Wiedermann and Dana Pardubská

Technical report No. 975

September 2006

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +4202 6605 3520, fax: (+4202) 86 585 789,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Computing by Broadcasting: Alternation in Disguise (Extended Version)[1][2]

Jiří Wiedermann[3] and Dana Pardubská[4]

Technical report No. 975

September 2006

Abstract:

Wireless parallel Turing machine (WPTM) is a new computational model introduced recently by the authors. Its design has been inspired by wireless mobile networking. The WPTM is an entirely deterministic model of parallel computations where the processes communicate wirelessly via dynamically tuned broadcasting channels. We show a tight connection between the computations of WPTMs and those of the classical alternating Turing machines. Namely, we prove that time-space simultaneously bounded computations on WPTMs correspond exactly to similarly bounded computations of alternating Turing machines. This result leads not only to an alternate characterization of alternation, but it also offers an alternate view of bounded uniform circuit families, such as NC^i and NC.

Keywords:
wireless parallel Turing machine; alternating Turing machine; time and space complexity; circuit complexity

[1]The research was carried out within the institutional research plan AV0Z10300504 and partially supported by grant No. 1ET100300517

[2]The research was partially supported by grant VEGA 1/3106/06

[3]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic

[4]Department of Computer Science, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia, pardubska@fmph.uniba.sk

1 Introduction

Inspired by the recent trends in wireless communication technologies and by the models of alternating and synchronous Turing machines (cf. [1], [2], [3], [5]) the authors have recently designed a new model of parallel computing, called Wireless Parallel Turing Machine (WPTM) [6]. The initial motivation was the investigation of the computational power and efficiency of a device consisting of a dynamic set of processors communicating one with each other via radio, on different channels. The first results have shown that despite their different architecture, the WPTMs are computationally related to the classical parallel computational model, viz. the alternating Turing machines (ATMs). Under the self-explaining notation and reasonable assumptions on $T(n)$, in [6] it has been established that $\mathsf{ATM}-\mathsf{TIME}(\mathsf{T}(\mathsf{n})) \subseteq \mathsf{WPTM}-\mathsf{TIME}(\mathsf{T}(\mathsf{n}))$ and $\mathsf{WPTM}-\mathsf{TIME}(\mathsf{T}(\mathsf{n})) \subseteq \mathsf{ATM}-\mathsf{TIME}(\mathsf{T}^2(\mathsf{n}))$, i.e., a polynomial-time equivalence of both models: $\mathsf{WPTM}-\mathsf{PTIME} = \mathsf{ATM}-\mathsf{PTIME}$. Both respective simulations work in space of size $O(T(n))$.

In the present paper we continue studies of the computational relations between WPTMs and ATMs. Our new simulations present an ultimate improvement of the previous results — namely mutual simulations of both models simultaneously preserving their time and space complexities. This is quite a surprising result taking into account completely different architectures of both models and the fact that among all universal models of parallel computing the models are only known which are polynomially, but not linearly, time- and space-equivalent to ATMs. These new simulations use a number of interesting techniques which differ quite substantially from the simulations described in the original paper and are interesting by themselves because they show how alternation can be efficiently realized by broadcasting, and vice versa. The new results have shifted the motivations of our investigations: now the primary goal appears to be to obtain an alternative computational characterization of alternation, rather than capturing the power of wireless mobile computing. In terms of WPTMs, the new results not only bring a new characterization of the computational power of alternation, but also that of the related complexity classes which are sensitive to asymptotical increase of complexity bounds, such as bounded uniform circuit families.

Note that, to some extent, the WPTMs represent a more realistic model of parallel computing than ATMs. Namely, the WPTMs are fully deterministic computing devices (not resorting to nondeterminism) using a natural acceptance mechanism: an agreement of all processes obtained via broadcasting. The latter is to be compared with a quite tricky acceptance mechanism of ATMs which, in fact, makes a "hidden use" of inter-processor communication when reporting the accpetance/rejection to the roots of the computational subtrees in the ATM computational tree (cf. [1]). Hencefore, the WPTMs can be seen as a parallel computational model which is more realistic than ATM and which retains the computational efficiency of the latter model. Last but not least, referring to the original motivation (cf. [6]), the WPTMs demonstrate that there is a huge computational potential hidden in the instances of the wireless mobile computing which are covered by our modelling.

The structure of the paper is as follows. In Section 2 we recall a formal definition of a WPTM and that of its computation. In Section 3 we show a novel simulation of a WPTM by an ATM which is simultaneously linear in time and space, and in Section 4 we prove a similar reverse simulation. In Section 5 we prove consequences of the previous results on the relation of WPTMs to uniform circuit families. Finally, in Conclusions we summarize our achievements.

The present paper has been written so as to be independent from the original paper [6] in which the model of a WPTM had been introduced. However, to learn more about the original background of our studies and the relation of our design to the related models studied within the complexity theory the reading of [6] is recommended.

2 Wireless Parallel Turing Machine

Now we give the definition of the basic model of the WPTM (with minor changes when compared to [6]).

Definition 1 *A k–tape wireless parallel Turing machine (WPTM) with a separate read–only input tape and a separate channel tape is an eleven–tuple* $M = (k, Q, R, \Sigma, \Gamma, \Delta, q_0, r_0, \varepsilon, q_{accept}, q_{reject})$

where

- $k \geq 1$ *is the number of work tapes;*
- $Q \times R$ *is the finite set of states;*
  - Q *is the set of working states with* initial state $q_0 \in Q$;
  - R *is the set of communication states also containing four distinguished states: initial communication state* r_0*, empty communication state* ε *and states* q_{accept}, q_{reject} *which are* accepting *and* rejecting *states, respectively;*
- Σ *is a finite input alphabet (* $\$ \notin \Sigma$ *is an endmarker);*
- Γ *is a finite work tape alphabet (*$\sharp \in \Gamma$ *is the blank symbol,* $\sharp \notin \Sigma$*);*
- $\Delta \subseteq Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1} \times Q \times R \times (\Gamma - \{\sharp\})^{k+1} \times \{-1, 0, 1\}^{k+2}$ *is the next move relation.*

The elements of Δ are called transitions. The machine has a read–only input tape with endmarkers, one work tape and one channel tape. The work tape and the channel tape, jointly be referred to as tapes, are initially blank. The tapes are unbounded to the right, with their cells numbered from 0.

Let $\delta = \langle q, r, x, a_1, \ldots, a_{k+1}, q', r', a'_1, \ldots, a'_{k+1}, d_1, \ldots, d_{k+2} \rangle \in \Delta$ be a transition of M. According to this transition in a single *step* M finding itself in working state q, in communication state r, reading symbol x from the input tape and a_i from the i-th tape, for $i = 1, 2, \ldots, k+1$, enters a new working state q', new communication state r', writes symbol a'_i on the i-th tape and moves each of the $k+2$ heads by one tape cell to the left (if $d_j = -1$), right (if $d_j = +1$), or performs no head move at all (if $d_j = 0$), for $j = 1, 2, \ldots, k+2$.

A *configuration* of a WPTM M is an element of $\mathcal{C} = Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2}$ representing the working and communication state of the finite control, the input, the non–blank contents of $k+1$ tapes, and $k+2$ head positions on all tapes.

A *head configuration* of M is an element of $Q \times R \times (\Sigma \cup \{\$\}) \times \Gamma^{k+1}$ representing the working and communication state of the finite control and the contents of cells scanned by each head.

We say that a transition with the new communication state $r' \neq \varepsilon$ broadcasts state $r' \in R$. Such a transition is called a *broadcasting transition.* For broadcasting transitions there is one syntactic restriction called "unanimous broadcast rule": *the broadcasting transitions pertinent to the same head configuration must all broadcast the same communication state.* They can differ in the remaining parts, i.e., they can prescribe entering different working states and different rewritings and movements. We say that a configuration is *tuned to channel* c if it has string c written to the left from the current channel tape head position. If c is a non–empty string, then it is also called a *channel number.* A configuration tuned to c to which a transition with the new communication state $r' \neq \varepsilon$ applies is said to broadcast r' on channel c. A configuration broadcasting ε is effectively considered as no–broadcasting (or *silent*) configuration (cf. the definition of a transition modified by a broadcasting below). The silent transitions are useful in situations when a process "does not want" to broadcast any information, e.g., when re–tuning its channel.

A configuration β is a δ–successor of a configuration α with respect to transition $\delta \in \Delta$ (written as $\alpha \vdash^{\delta} \beta$, or as $\beta = \delta(\alpha)$) if β follows from α in one step, according to transition δ. The move $\alpha \vdash^{\delta} \beta$ is called a *simple step* of M.

A configuration without successors is called a *terminal configuration.* Note that a configuration can have several different δ–successors.

In order to formally define a computation of M we need a couple of further preliminary definitions.

Function $Tuned : \mathcal{C} \to (\Gamma - \{\sharp\})^*$ assigns to each configuration its channel number.

In a similar vein we define function $Broadcast : \mathcal{C} \to R$ returning to each configuration the unique state broadcasted by the transition applicable to that configuration (remember the "unanimous broadcast rule"). With a slight abuse of notation this function will naturally be extended to a set $L \neq \emptyset$ of configurations as follows:

$$Broadcast(L) = \begin{cases} b & \text{iff for all } \alpha, \beta \in L, Tuned(\alpha) = Tuned(\beta) \\ & \text{and } Broadcast(\alpha) = b \\ \perp & \text{otherwise} \end{cases}$$

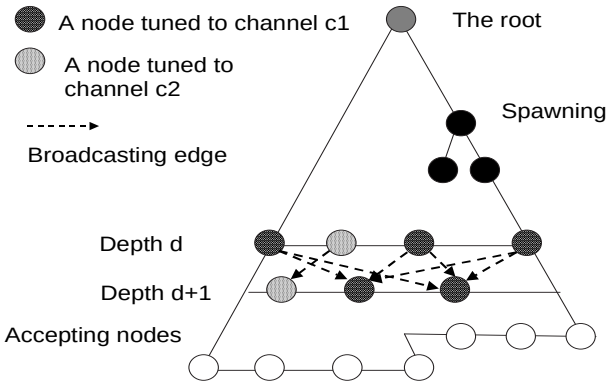

Figure 2.1: The computational graph of M

(symbol $\perp$ denotes an undefined value).

Finally, we define projection $Comm : \mathcal{C} \to R$ assigning to each configuration its communication state.

To formalize the action of broadcasting/receiving a message, let $Comm(\alpha) = u$ and let $\alpha|_{u:=v}$ denote the configuration α in which communication state u is changed to v. Let $\mathcal{C}_e \subseteq \mathcal{C}$ be the subset of all configurations from $\mathcal{C}$ tuned to e, let $\alpha, \beta \in \mathcal{C}_e$ and $\alpha \vdash^{\delta} \beta$ a simple step. Then configuration γ, so–called $\delta_{\mathcal{C}_e}$*–successor* of α w.r.t. transition δ *modified by broadcasting* from $\mathcal{C}_e$ (denoted as $\alpha \vdash^{\delta_{\mathcal{C}_e}} \gamma$), is defined as follows (configuration γ is $\Delta_{\mathcal{C}_e}$*–successor of* α if there exists $\delta \in \Delta : \alpha \vdash^{\delta_{\mathcal{C}_e}} \gamma$):

$$\gamma := \begin{cases} \beta & \text{iff } \forall \varphi \in \mathcal{C}_e : \ Broadcast(\varphi) = \varepsilon \\ \beta|_{Comm(\beta):=b} & \text{iff } \exists \varphi \in \mathcal{C}_e : Broadcast(\varphi) = b \\ & \quad \text{and } \forall \varphi \in \mathcal{C}_e : \ Broadcast(\varphi) \in \{b, \varepsilon\} \\ \perp & \text{otherwise} \end{cases}$$

Note that there can be several computational steps possible from a given α. This occurs when there are several transitions applicable to α. If this is the case we say that α *spawns* all configurations γ_i for which there exists $\delta_i \in \Delta$ such that $\alpha \vdash^{(\delta_i)_C} \gamma_i$, for some $r \geq 1$ and $1 \leq i \leq r$. Formally, spawning corresponds to the universal branching in ATMs.

For any input w to M the *computational graph* $G(w)$ of M on that input is a rooted, directed, possibly infinite acyclic multigraph whose nodes are configurations of M and edges correspond to transition and communication links (Fig. 2.1). The *depth* of a node in $G(w)$ is its distance from the root. This graph is defined inductively:

1. The *initial configuration* $c_{init} = (q_0, \varepsilon, w, \underbrace{\nu, \ldots, \nu}_{k+1}, \underbrace{0, \ldots, 0}_{k+2})$ with ν being the null string, is the root of $G(w)$ at depth $d = 0$.

2. Let C_d be the set of configurations in $G(w)$ at depth $d \geq 0$. Then for all non–terminal configurations $\alpha \in C_d$, set C_{d+1} contains all Δ_{C_d}–successors of α, i.e., all configurations spawned by α. If some of the Δ_{C_d}–successors of α is undefined, then the whole graph $G(w)$ is undefined.

3. In $G(w)$ there are two kinds of edges: *transition edges* leading from each $\alpha \in C_d$ to each of its Δ_{C_d}–successors $\gamma \in C_{d+1}$ and *broadcasting edges* leading from each broadcasting configuration $\alpha \in C_d$ to each configuration $\beta \in C_{d+1}$, with $Tuned(\alpha) = Tuned(\beta)$.

Configuration $c \in C_d$ is called *reachable at depth* d. The nodes of $G(w)$ without successors are called *leaves* of $G(w)$. Note that the computational graph is built in an entirely deterministic manner.

Definition 2 *A computational graph* $G(w)$ *of* M *is a* computational graph accepting input w *if it satisfies the following conditions:*

1. Finiteness: $G(w)$ *is a finite graph;*

2. Acceptance agreement: *all leaves of* $G(w)$ *are terminal configurations in communication state* q_{accept} *tuned to the same channel.*

¿From practical point of view the acceptance agreement means that all processes share the information that all of them have accepted the input.

We say that M *accepts* w if M's computational graph accepts w; we define $L(M)$ to be the set of strings accepted by M.

The space complexity of a configuration is the maximum of lengths of the non–blank contents of all tapes. The space of a computational graph G is the maximum space of any configuration in G; the channel space of G is defined similarly. The time of G is the maximum length of any path in G.

A WPTM M *operates in space* $S(n)$ if for every string $w \in L(M)$ of length n there is a computational graph of M of space at most $S(n)$ that accepts w. Similarly, M *operates in time* $T(n)$ if for every string $w \in L(M)$ of length n there is a computational graph of M of time at most $T(n)$ that accepts w.

When the transition relation becomes a function it is easily seen that a WPTM turns into the classical deterministic Turing machine (in such a case, broadcasting does not make any sense). The above described basic model can be extended in several ways. First of all, in order to be able to study also computations with sub–linear times, we can extend our model by so–called *index tape* to which non-negative binary numbers can be written. If there is a number i written in binary on the index tape and the machine enters a special, so–called *index state* q_{index}, then the input head on the input tape relocates itself in a single move to the i-th cell of that tape, for any $1 \leq i \leq n$, and to the $(n+1)$-st cell otherwise. For a similar mechanism in the context of ATMs, cf. [1]. In general, one can also consider a model with several channel tapes and also nondeterministic versions of the previous models.

3 Linear time-space simulation of wireless communication by alternation

Now we will be interested in characterizing the computational efficiency of WPTMs. Since the underlying graph structure of a WPTM computation reminds a computational tree of an ATM computation extended by additional links among communicating nodes, an ATM appears to be a good candidate for investigating its relation to a WPTM.

In the sequel we will consider a model of a WPTM with $k \geq 1$ working tapes and one channel tape. In the proof of the following theorem we will need a couple of new notions.

Let G_W be the computational graph of a WPTM W on input w, let C_i be the set of configurations in G_W at depth $i \geq 0$ (we do not distinguish between configurations and the respective nodes in G_W). A configuration c is reachable in G_W at depth i if and only if $c \in C_i$. For any non–terminal configuration $c \in C_i$, $Succ(c) = \{d | (\exists \delta \in \Delta : c \vdash^{\delta} d$ and $(L_{Tuned(d)} = \emptyset$ or $Broadcast(L_{Tuned(d)} = \varepsilon)))$ or $(\exists \delta \in \Delta, \exists e \in C_i : c \vdash^{\delta_{\{e\}}} d)\} \subseteq C_{i+1}$ is the set of all successors of c.

Let $\delta \in \Delta$ be a simple transition. If $c, c' \in \mathcal{C}$ are configurations such that $c \vdash^{\delta} c'$, then δ^{-1} for which $c' \vdash^{\delta^{-1}} c$ is called the *inverse transition* to δ.

For any configuration c let $Head(c)$ denote its head configuration. Let d be a configuration obtained from c by "pruning" the contents of tapes represented in c from both ends, but not beyond the positions of tape heads in c. If this is the case d is called a *sub–configuration* of c (or c is a *super–configuration* of d) and we shall write $d \sqsubseteq c$ ($c \sqsupseteq d$). Note that for any such d, $Head(d) = Head(c)$ holds and that (thus) the heads both in d and c scan the same symbols. Obviously, any sub–configuration is a configuration and therefore transitions and inverse transitions can be applied to a sub–configuration just like to complete configurations.

Finally, we will introduce a special kind of sub–configurations called kernels. A *kernel* of a configuration $c \in Q \times R \times \Sigma^* \times ((\Gamma - \{\sharp\})^*)^{k+1} \times \mathbb{N}^{k+2}$, denoted as $Ker(c)$, is configuration $Ker(c) \in Q \times R \times \Sigma \times (\Gamma - \{\sharp\})^k \times (\Gamma - \{\sharp\})^* \times \{1\}^k \times \mathbb{N}$ which is a sub–configuration of c representing the working and the communication state of the finite control, the symbols scanned by the input and

working tape heads, the non–blank contents of the channel tape, the working head positions and the position of the channel–tape head which remains the same as in c. Note that in order to know what instructions can be applied to a configuration c and on what channel c will broadcast it is enough to know $Ker(c)$ (or any sub–configuration of c). We will use this fact to our advantage in the following simulation.

Theorem 1

(i) Let W be a single channel tape WPTM of time complexity $T(n) \geq n$ and of space complexity $S(n)$. Then W can be simulated by an alternating Turing machine A simultaneously in time $O(T(n))$ and in space $O(T(n))$.

(ii) Moreover, if $S(n)$ is fully space constructible then A is of simultaneous time complexity $O(T(n))$ and space complexity $O(S(n) + \log T(n) + n)$.

(iii) Finally, if both A and W are equipped by the index tape, $T(n) \geq \log n$ and $S(n)$ is fully space constructible, than A is of simultaneous time complexity $O(T(n))$ and space complexity $O(S(n)+ \log T(n))$.

Sketch of the proof: Assume first that $T(n) \geq n$ and W is not equipped by the index tape. Let G_W be the computational graph of W on input w, let C_i be the set of configurations in G_W at depth $i \geq 0$.

The simulating machine A will have the same number of working tapes as W plus one extra write-once working tape called the *false input tape.* The latter tape will be used for reconstructing the contents of the input tape. We also extend the working alphabet of A by a new distinguished symbol $\mathfrak{c}$.

At the heart of simulation of W by A there is procedure *Simulate* (Fig. 3.1). It is a recursive procedure which proceeds in rounds. In the i–th round, procedure *Simulate* computes $Succ(c) \subseteq C_{i+1}$ for any nonterminal $c \in C_i$. The procedure ends by reaching terminal configurations at depth at most $T(n)$ and returns *true* ($false$) if the configuration at hand is an accepting (rejecting) terminal configuration.

The simulation is initiated by calling $Simulate(0, c_{init})$ where c_{init} is the initial configuration. In general, for any configuration c at depth i $Simulate(i, c)$ works as follows. It starts by testing whether c is a terminal configuration. Due to the definition of a WPTM computation, such a configuration must be achieved along any computational path in G_W. If c is a terminal accepting (rejecting) configuration then $Simulate(i, c)$ returns *true* ($false$). If c is neither accepting nor rejecting terminal configuration it returns $\perp$ (undefined). Otherwise, there is a constant number of transitions applicable to c. In the universal mode, all these transitions are applied to c. Let δ be one of them, with $c \vdash^{\delta} d$. For any such d *Simulate* proceeds as follows. It existentially splits into two branches, (a) and (b). Branch (a) corresponds to the case when there was no broadcasting at channel $Tuned(d)$ from any configuration in C_i, i.e., δ was a simple transition. Therefore no modification of d's communication state is necessary. We merely split universally into two branches calling $Verify(i+1, d)$ on one branch and $Simulate(i + 1, d)$ on the other branch. The task of the former procedure is to check whether $d \in C_{i+1}$, indeed, since we have only *assumed* that there was no broadcasting to d, but the call to $Verify$ will check this fact.

Branch (b) corresponds to the case when there was a broadcasting at channel $Tuned(d)$ from a configuration e, say, with $e \in C_i$. That is, δ was a transition modified by communication state $r \in R$, say, $r \neq \varepsilon$. Thus, $c \vdash^{\delta_{\{e\}}} d|_{Comm(d):=r}$. In this case we proceed as follows: we existentially split for each $r \in R, r \neq \varepsilon$. At each branch, we universally split into two branches and call $Verify(i+1, d|_{Comm(d):=r})$ and $Simulate(i+1, d|_{Comm(d):=r})$, respectively, on each of them. Note again that we have only *assumed* here state r was broadcasted to d and it is the task of $Verify$ to confirm this assumption. Thus, if the call to $Verify$ returns *true* we could again conclude that $d|_{Comm(d):=r} \in C_{i+1}$.

Hencefore, no matter whether case (a) or (b) will occur, for any $c \in C_i$ procedure *Simulate* computes set $Succ(c) \subseteq C_{i+1}$ because the procedure it verifies all members of $Succ(c)$ by trying to apply every $\delta \in \Delta$ to c. Subsequently, for all configurations in $Succ(c)$ the next recursive calls of *Simulate* can be launched.

alternating procedure $Simulate(i, c)$;
{returns *true* if from $c \in C_i$ an accepting configuration in G_W is reachable}

if $c = terminal\ configuration$ **then**
  if c is accepting **then return**$(true)$ **fi;**
  if c is rejecting **then return**$(false)$ **fi;**
  if c is neither accepting nor rejecting **then return**$(\perp)$ **fi**
else
  universally branch on each $\delta \in \Delta$ **applicable to** c**:**
    compute $d = \delta(c)$;
    existentially branch on (a) **or** (b) :
    (a) {there is no broadcasting to d – assume and verify}
        universally branching on two processes:
        return$((Simulate(i+1, d)) \ \wedge \ (Verify(i+1, d)))$;
    (b) {there is a broadcast to d – assume and verify}
        existentially branch on each $r \in R$, $r \neq \varepsilon$:
          compute $d' := d|_{Comm(d):=r}$
          universally branching on two processes :
          return$(Simulate(i+1, d') \ \wedge \ Verify(i+1, d'))$;
fi

Figure 3.1: Sketch of the procedure Simulate. $Simulate(i, c)$ returns true iff from $c \in C_i$ an accepting configuration in G_W is reachable. Procedure $Verify(i+1, d')$ returns true iff there exists $g \in C_{i+1}$ such that $g \sqsupseteq d'$

Now we describe the verification procedure.

As mentioned above, the goal of procedure $Verify(j, f)$ (Fig. 3.2) is to check for any $j \in \mathbb{N}$ and any syntactically correctly constructed configuration $f \in \mathcal{C}$ of W (i.e., f need not necessarily be a configuration reachable in G_W on a given input), whether there exists $g \in C_j$ such that $g \sqsupseteq f$.

Note that in order to learn that for a given configuration f there exists some super–configuration $g \in C_j$ with $g \sqsupseteq f$ it is enough to find a subgraph F of G_W rooted at c_{init} such that F contains all paths of length j leading from c_{init} to a configuration whose sub–configuration is f. Then g could be constructed (i.e., computed) if we followed the respective paths from c_{init} in F. However, for us a sub–configuration will be sufficient since we are interested merely in the action of the machine when reaching the corresponding full configuration — what symbol is broadcasted (if any) and on what channel. As remarked in the beginning of this section this action is entirely determined by any sub–configuration (and in particular, by the kernel) of the corresponding full configuration. Thus, the idea of procedure $Verify$ is to check that a subgraph F exists (note that for given f there can be several such subgraphs in G_W, each corresponding to a different full configuration $g \sqsupseteq f$), by starting at f and by recursively climbing against the directed edges of G_W towards its root level by level, by guessing and applying the inverse transitions to intermediate configurations until eventually c_{init} is reached. The recursive reverse traversal of paths G_W is possible thanks to the uniqueness of these paths.

Procedure $Verify$ starts by checking, whether $j = 0$ and f equals to a prefix of c_{init}. By a prefix of the initial configuration we mean a configuration describing any prefix of the input on the false input tape, with all other tapes empty and with the heads on all tapes in their leftmost positions. In the before mentioned test we do not consider the entire initial configuration because a computational path in G_W can terminate without having read the entire input, and therefore the reverse verification process (see in the sequel) can reconstruct only the respective prefix of the input. Clearly, if $j = 0$ and f equals to a prefix of c_{init}, then $f \sqsubseteq c_{init}$. Because $c_{init} \in C_0$, $Verify$ accepts. If $j = 0$ and f is not equal to a prefix of c_{init} $Verify$ rejects. Otherwise, for any $j > 0$ we guess whether g has emerged by applying a simple or a modified (by broadcasting) transition to g's predecessor.

In the former case, a transition δ is guessed, the predecessor $\delta^{-1}(f)$ of f is computed and the procedure is applied to this predecessor. In parallel it is checked whether there was no broadcasting

alternating procedure $Verify(j, f)$;
{returns *true* iff there exists $g \in C_j$ such that $g \sqsupseteq f$}

if $j = 0$ **then return(**$f = Prefix(c_{init})$**)**
else existentially branch on (a) **or** (b)**:**
(a) {f has not been modified by broadcasting}
universally branch on $(a1)$ **and** $(a2)$**:**
(a1) **existentially branch on each** $\delta \in \Delta$**:**
if $\delta^{-1}(f) = f'$ *is defined* **then return(**$Verify(j-1, f')$**)**
else return($false$**) fi;**
(a2) **universally branch on each kernel** h **tuned to** $Tuned(f)$:
if $Broadcast(h) \neq \varepsilon$ **then return(**$\neg Verify(j-1, h)$**)**
else return($true$**) fi;**
(b) {f has been modified by broadcasting, r is its original communication state}
existentially branch on each (r, δ) such that
$r \in \{\varepsilon, Comm(f)\}, f_1 = \delta^{-1}(f|_{Comm(f):=r})$ defined:
universally branch on $(b1)$ **and** $(b2)$ **and** $(b3)$**:**
(b1) **return(**$Verify(j-1, f_1)$**);**
(b2) **existentially branch on each kernel** h **tuned to** $Tuned(f)$**:**
if $Broadcast(h) = Comm(f)$ **then return(**$Verify(j-1, h)$ **fi;**
(b3) **universally branch** on each kernel h tuned to $Tuned(f)$:
if $Broadcast(k) \notin \{Comm(f), \varepsilon\}$ **then return(**$\neg Verify(j-1, k)$**) fi**
fi;

Figure 3.2: Sketch of the procedure Verify. Procedure $Verify(j, f)$ returns true iff there exists $g \in C_j$ such that $g \sqsupseteq f$.

to f from any configuration in C_{j-1}, indeed. In order to do so, the kernels of all configurations tuned to $Tuned(f)$ are guessed (note that there is but a constant number of such kernels) and in case that a kernel h broadcasts a state different from ε it is verified that h is not reachable in G_W. This is determined by checking whether $Verify(j-1, h)$ returns $false$ in such a case.

In the latter case, the original communication state of f before its modification by broadcasting is guessed. Let r be this original state. Thus, we can construct a configuration $f|_{Comm(f):=r} = f'$. For this configuration there must exist a simple transition δ and two configurations $f_1, f_2 \in C_{j-1}$ such that $f_1 \vdash^{\delta} f'$ and $f_1 \vdash^{\delta\{f_2\}} f$. Note that $Tuned(f) = Tuned(f_2) = Tuned(Ker(f_2))$ and $Broadcast(f_2) = Broadcast(Ker(f_2)) = Comm(f)$. Similarly as before, knowing δ, $f_1 = \delta^{-1}(f')$ is constructed and $Ker(f_2)$ tuned to $Tuned(f)$ and broadcasting $Comm(f)$ is guessed. Then, it is verifies in parallel that both f_1 and $Ker(f_2)$ are reachable in G_W (by calling $Verify(j-1, f_1)$ and $Verify(j-1, Ker(f_2))$, respectively) and that there is no broadcasting conflict on $Tuned(f)$. The latter condition is confirmed by checking whether $Verify(j-1, h)$ returns $false$ for each kernel h broadcasting at $Tuned(f)$ a state different from $Comm(f)$ and ε. Note again that there is but a constant number of such kernels.

In order to efficiently implement the previous algorithm we indicate how the kernels, or in general, the sub–configurations, will be represented on the tapes of A in order to enable their efficient generation and operations over them. Each kernel h will be generated from a configuration f in which A finds itself in that time. Note that at all occasions when the kernels are constructed we always have the situation that $Tuned(h) = Tuned(f)$. Therefore, on the channel tape we mark the position of its head (since the channel number is determined by the contents of the channel tape left to the head position) and otherwise the channel tape in h remains as it was in f. Should we generate a kernel $h = Ker(f)$, then, clearly, the heads already point to the right symbols. However, in order to distinguish the head configuration of h from the remaining symbols in f we embrace the symbol scanned at each working tape by a pair of ¢ symbols. During the subsequent computation, the "valid" part of (sub–)configurations which keeps developing from the given kernel will always be embraced between the pair of ¢ symbols. Each time when a head of a simulated machine is about to enter a cell containing the

left (right) ¢ symbol this symbol is moved left (right) rewriting whatever symbol was there before and leaving the blank symbol on the original position of the ¢ symbol. The input symbols need a special treatment since we cannot put marks on the input tape. Instead of the input tape we therefore make use of the false input tape on which the input symbol scanned at time of a kernel creation is copied at the first position. Then, as the configuration keeps evolving the additional symbols are placed to this tape. Obviously, correct guesses must not rewrite the symbols once written to that tape, and therefore the false input tape is a write–once tape. If at some occasion the head on any tape finds itself at the leftmost tape position and is about to move left, then the standard solution — using the second track on that tape — is used. A similar procedure is also used when generating a kernel that is not the kernel of the current configuration c. The only difference is that now the symbols under the input tape head and the working tape heads must be guessed.

When calling $Verify(0, f)$, in order to realize the check $f = Prefix(c_{init})$ the contents of the false input and the original input tape are compared. If and only if the former is a prefix of the latter, and the other data in f and c_{init} coincide, the test is passed.

Now we sketch how to extend our simulation to the case when there is the index tape enabling to achieve sublinear running times of form $T(n) \geq \log n$. Under the assumption of the constructibility of $S(n)$ we will treat the index tape in a similar way as the channel tape. I.e., while proceeding down in graph G_W in procedure $Simulate$ the contents of the index tape is copied as well. In the verification procedure, while climbing back to the root along the spawning edges, we can verify in on-line manner the input symbols in accordance with the continuously retrospectively unfolded current contents of the index tape. However, when ascending along the broadcasting edges, we do not know the contents of the index tape in the broadcasting configuration since in general this contents is not related to that in the receiving configuration. Therefore, being at level $i > 1$ we extend procedure $Verify$ to start guessing also the contents of the index tape at the "referential" depth $i_{ref} = \min\{1, i - S(n)\}$. The contents of the index tape at the referential depth is guessed bit by bit, at each level. Along with it, on a special tape the actual movements (corresponding to the current depth j) of the head on the index tape are guessed and stored. Once the referential depth is reached (and for determining this depth, we will make use of the constructibility of $S(n)$), A can switch from the index and input tape guessing mode to on-line checking mode. That is, from now on the contents of the input tape as dictated by the index tape can be verified. Simultaneously, at the referential depth (for $j = i_{ref}$), A will in parallel (in universal mode) issue a special verification process whose task is to check whether the symbols read by the previously guessed input head movements are the same as those on the input tape. This, clearly, requires time of order $O(S(n))$ which, however, runs in parallel with the rest of computation.

As far as the time complexity of the simulation is concerned, it is proportional to the number od rounds multiplied by the complexity of actions needed to prepare the calls of $Simulate$ and $Verify$. There are $T(n)$ rounds all together. Note that when using the kernel representation proposed above, in order to prepare the parameters for each call only local changes in the vicinity of the current tape heads positions are needed. Therefore, preparing the parameters for each recursive call of $Simulate$ or $Verify$ takes a constant time. In a total, the entire simulation time of A is linear in terms of the original running time of W.

As far as the space complexity is concerned, note that in the case that a kernel of a configuration which is not reachable from the initial configuration is verified the decision about the termination of $Verify$ can occur as late as in depth $T(n)$. In that time, the size of the computed predecessor of that kernel can be of order $O(T(n))$ and this gives an upper bound on the space complexity of our simulation as in case (i) in the statement of the theorem. However, in case (ii), if $S(n)$ is known to be fully space constructible then A can keep track of the size of intermediate configurations and abort (and reject) the verification of the respective branch once a larger configuration was generated. In this case, the depth counter of size $O(\log T(n)$ must be taken into account since its size need not be absorbed by the $O(S(n))$ space bound. The size of the false input tape adds an other term of order $O(n)$. Finally, in case (iii) at most $O(S(n))$ space has been used on the false input tape since after reaching the reference point, the input tape head had been relocated to its current position and from now on, there is no need to record the symbol read from the input since these inputs can be verified in an on–line manner. Thus, the space complexity remains bounded by $O(S(n) + \log T(n))$.

□

4 Linear time-space simulation of alternation by wireless communication

Now we turn our attention to the reverse simulation, i.e., to the simulation of an ATM on a WPTM. We assume that both machines are equipped by the index tapes.

Theorem 2 *Let $S(n) \geq \log n$ be a space constructible function. Let A be an ATM of time complexity $T(n) \geq \log n$ and space complexity $S(n)$. The A can be simulated by a WPTM W simultaneously in time $O(T(n))$ and in space $O(S(n))$.*

Sketch of the proof: Assume that A is a machine with $k \geq 1$ working tapes, with a separate input tape and one index tape. For technical reasons we will assume that all branching instructions in A are binary and that the transitions applicable to configurations with the same head configuration are ordered. While simulating A on W we will represent tape contents of A straightforwardly on the corresponding tapes of W. Moreover, W will have one additional working tape, called a *configuration tape,* on which the current configuration, with the exception of the input tape contents (which is the same in all configurations and written on the input tape in all processes), of A will be represented. We will make use of two different forms of configuration representations: an explicit, and an implicit one.

An *explicit representation* of A's configuration will be an element of $\mathcal{C}$ (see the definition of a configuration in Section 2). Note that in a configuration, the position of the input head determines at the same time the contents of A's index tape.

We say that configuration c_2 is $s > 0$ steps apart from a given configuration c_1 if and only if c_2 can be reached from c_1 by applying s transitions $\delta_1, \delta_2, \ldots \delta_s \in \Delta$ (in that order) to c_1. If c_1, c_2 are as above, and c_1 is given in its explicit form, then c_2 in its *implicit form* w.r.t. c_1 is given by $c_1 \# \delta_1 \# \ldots \# \delta_s$.

Now we are ready to describe the simulation of A by W. Let T be the computational tree of A on a given input. It is a binary tree of depth $T(n)$ whose branches are ordered (due to our assumption on instruction ordering) containing in its nodes configurations of A. First assume that $T(n)$ is computable in space $S(n)$. The simulation will consist of two phases.

Phase 1 – descending the tree: W simulates A straightforwardly, by descending T and splitting at each branching instruction of A, irrespectively whether this instruction was an existential or universal one. Each reached configuration of A is represented in W by a process which corresponds to that configuration of A by having the same tape contents on its working tapes, and on the index tape. Consider arbitrary depth d and write it as $d = iS(n) + j$, for $i = 1, 2 \ldots$ and $0 \leq j < S(n)$. For $j = 0$ the respective ds are called a *reorganization times*. Let $c_0, c_1, \ldots$ be the configurations of A along a computational path in T at reorganization times $t_0, t_1, \ldots$ in chronological order. For any $i \geq 0$ we will want the following invariant to hold in W:

- for any depth d such that $t_i < d < t_{i+1}$, both at the configuration tape and the channel tape of W, a configuration c of A is represented in its implicit form w.r.t. c_i.
- for $d = t_{i+1}$, (at the reorganization time), c_{i+1} is represented explicitly at the channel tape and implicitly, w.r.t c_i, on the configuration tape of W.

Moreover, from technical reasons we will require that information on each of the tapes mentioned previously is prefixed by the value of t_i. From now on, when speaking about a configuration at depth d, with $t_i \leq d < t_{i+1}$, we will always assume that this configuration is prefixed by t_i.

For $i = 0$ we start at level 0 with c_0 (initial configuration) written on both channel and configuration tape of W. While descending the respective branch in T we represent the respective configurations implicitly w.r.t c_0 simply by adding the code of the instruction realized in that step behind the representation of the configuration from the previous step. . Simultaneously, we count up to $S(n)$ and when reaching the next reorganization point, we rewrite the channel tape by the current configuration

c_1 of A. This is simply achieved by sequentially copying, in time $O(S(n))$, the contents of the remaining tapes to the channel tape. Now the invariant at t_1 is restored.

We proceed further down the tree. Assume that the invariant has been restored at some depth d_i, with c_i represented explicitly at the channel tape and implicitly, w.r.t c_{i-1}, on the configuration tape of W. In the next step, we copy the contents of the channel tape (i.e., c_i) to the configuration tape rewriting whatever was there. Then on both tapes we represent the current configuration of A implicitly w.r.t. c_i simply by adding the code of the instruction realized in that step behind the rewritten part on both tapes. In this way we proceed until the next reorganization point is achieved. Here we restore the invariant in the same way as before.

The tree descending process terminates in depth $T(n)$ (in order to determine this moment, we must keep a special counter counting up to $T(n)$ in space $O(S(n))$).

Note that copying operations at reorganization points, and right after it, take time $O(S(n))$. However, since these actions are done once per $S(n)$ steps, this additional work gets amortized.

Phase 2 – ascending the tree: Configurations of A at depth $T(n)$ represented in processes in W are of three kinds: accepting, rejecting, and non-terminating. These kinds are called the *quality* of a configuration. The respective processes start to send the respective qualities "upwards" in the tree, to their parent configurations, which can compute their own quality depending on their type (which could be either existential or universal one). The parent of a configuration is achievable at a channel whose number is computed from the implicit configuration stored at the channel tape: if $t_i\#c_i\#\delta_1\#\ldots\#\delta_s$ for some $s > 0$ and $i > 0$ is the contents of the channel tape at that moment, then the father of that configuration is tuned to $t_i\#c_i\#\delta_1\#\ldots\#\delta_{s-1}$. Thus, the necessary re–tuning can be done in constant time simply by moving the head left over one transition. However, if $s = 0$, then on the channel tape there is an explicit representation of configuration c_i. We change it into an equivalent implicit form by copying the contents of the configuration tape to the channel tape. This takes time $O(S(n))$, but again, this time gets amortized. Then, in order to determine the parent of a configuration, we can proceed as in the former case.

In general, a parent can have two sons. In order to prevent the broadcasting collisions, the sons sent their quality to their parent one after the other, in the order implied by the ordering of the instructions by which sons have been reached from their parent.

After sending their quality, the processes re–tune to channel 0.

In this way, the quality of tree nodes is evaluated, until eventually, after $O(T(n))$ steps, the root of the tree gets its quality. Then also the root re–tunes to channel 0 and sends the accepting/nonaccepting signal to all processes which can now terminate.

This ends the description of the simulation algorithm for the case when $T(n)$ is computable in space $S(n)$. If this is not the case, we modify the above–described algorithm as follows. Now, after sending their signal to their parents the processes do not re–tune to channel 0. Instead, they re–tune back to their original frequency. The idea of the modified algorithm is to try, after reaching each reorganization point while descending the tree, to evaluate the tree of A's computation computed till that time. Thus, the signals proceed towards the root level by level, in a pipelined manner. Once an acceptance signal had reached the root, the respective root process starts sending a signal at channel 0. In order to hear this signal, when reaching a reorganization point, while re–tuning from the explicit to the implicit representation each process goes through the situation when it is tuned to channel 0. At this occasion the process has the chance to learn that in fact the simulation has to terminate. If this is the the case, the process stops spawning further processes. Instead of expecting quality of nodes to arrive from the lower levels and sending it upwards, the process starts sending upwards a signal informing the nodes above it to terminate and to re–tune to channel 1. When this signal reaches the root, the root process sends on channel 1 the accepting/nonaccepting signal to all processes which can now terminate.

As far as the correctness of the algorithm is concerned, note that the quality evaluation proceeds for any $d > 0$ from level d to level $d - 1$ because $d = t_i + s$ is implicitly represented in the contents $t_i\#c_i\#\delta_1\#\ldots\#\delta_s$ stored at that time on the channel tape. The broadcasting conflicts have been eliminated since processes belonging to a parent in a different configuration broadcast on different frequencies. In the case when a process has two sons no conflict can arise thanks to the instruction ordering. Note that in the process of tree ascending, the nodes of T corresponding to the same

configurations of A get the qualities from their sons, irrespectively which son belonged to which parent. Fortunately, this does not make any harm since the sons must have obtained the same quality, because the history of computations "below" these sons in T must have been the same once the configuration in sons were the same.

As far as the time complexity of the above-described algorithm is concerned, it clearly is $O(T(n)$ since either a (parallel) step performed in simulation is of constant time complexity, or is of complexity $O(S(n))$, but such steps occur once in every $S(n)$ steps. Clearly, the space complexity of simulation is $O(S(n))$.

□

Let $\mathsf{W}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n}))$ denote the class of WPTM computations (with the index tape) of a simultaneous time complexity $T(n)$ and space complexity $S(n)$, and $\mathsf{A}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n}))$ denote an analogous class for ATMs. The following corollary characterizes the equivalence between WPTMs and ATMs:

Corollary 4.1 *Let $S(n) \geq \log n$ be a space constructible function, let $T(n) \geq \log n$. Then* $\mathsf{W}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n})) = \mathsf{A}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n}))$.

5 WPTMs and the uniform circuits

Let $\mathsf{U}-\mathsf{SIZE}-\mathsf{DEPTH}(\mathsf{S}(\mathsf{n}),\mathsf{T}(\mathsf{n}))$ be the uniform family of bounded fan-in circuits of size $S(n)$ and depth $T(n)$ (where $\mathsf{U} \in \{\mathsf{U_E},\mathsf{U_E^*}\}$, using the notation from [4]). It is known that $\mathsf{A}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n})) = \mathsf{U}-\mathsf{SIZE}-\mathsf{DEPTH}(2^{\mathsf{O}(\mathsf{S}(\mathsf{n}))},\mathsf{T}(\mathsf{n}))$, for $S(n), T(n) \geq \log n$ deterministically computable in space $S(n))$ [4].

Let $\mathsf{NC^i} =_{\mathsf{def}} \mathsf{U}-\mathsf{SIZE}-\mathsf{DEPTH}(\mathsf{n}^{\mathsf{O}(1)},(\log \mathsf{n})^{\mathsf{i}})$ be the class of all sets $A \subseteq \{0,1\}^*$ for which there is a uniform circuit family of polynomial size and of depth $(\log n)^i$, let $\mathsf{NC} =_{\mathsf{def}} \cup_{\mathsf{i}\geq 0} \mathsf{NC^i}$.

By these relations we get further consequences of the previous result:

Corollary 5.1

(i) for $S(n), T(n) \geq \log n$ which are deterministically computable in space $S(n)$ we have $\mathsf{W}-\mathsf{TISP}(\mathsf{T}(\mathsf{n}),\mathsf{S}(\mathsf{n})) = \mathsf{U}-\mathsf{SIZE}-\mathsf{DEPTH}(2^{\mathsf{O}(\mathsf{S}(\mathsf{n}))},\mathsf{T}(\mathsf{n}))$;

(ii) $\mathsf{W}-\mathsf{TISP}((\log \mathsf{n})^{\mathsf{i}},\log \mathsf{n}) = \mathsf{NC^i}$

(iii) $\mathsf{W}-\mathsf{TISP}((\log \mathsf{n})^{\mathsf{O}(1)},\log \mathsf{n}) = \mathsf{NC}$

Thus, simultaneously bounded WPTM computations characterize exactly uniformly bounded circuit families.

6 Conclusion

We have prolonged the studies of a recently introduced new model of wireless parallel Turing machines whose design has been inspired by recent trends in wireless mobile networking. We have substantially improved the originally known complexity relations between WPTMs and ATMs given in [6]. Namely, we have shown that the time-space bounded computations on the WPTMs correspond to similarly bounded computations of ATMs. Via the latter mentioned machines these results extend further to uniform bounded fan-in circuit families, especially to classes $\mathsf{NC^i}$ and NC. Our results offer an alternative model to alternating Turing machines; they provide an exact characterization of a computational power of alternation in terms of parallel deterministic computations with broadcasting. This ranks the WPTMs among the fundamental machine models. A study of WPTMs computations with a bounded number of broadcasting steps, or with a bounded number of channels, seems to represent a challenging area for future research. The results of this type can throw a further light on the nature of alternation and basic relations among fundamental complexity classes.

Bibliography

[1] Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. Journal of the ACM, Vol. 28, pp. 114–133, 1981.

[2] Geffert, V.: A Communication Hierarchy of Parallel Computations. Theor. Comput. Sci., Vol 198, No. 1-2, pp. 99–130, 1998

[3] Hromkovič, J., Karhumäki, J., Rovan, B., Slobodová, A.: On the power of synchronization in parallel computations. Discrete Appl. Math. 32 (1991), 155-182

[4] Vollmer, H.: Introduction to Circuit Complexity. Springer-Verlag Berlin Heidelberg, 1999, 270 p.

[5] Wiedermann, J.: On the Power of Synchronization. Elektronische Informationsverarbeitung und Kybernetik (EIK), Vol. 25, No. 10, pp. 499–506, 1989

[6] Wiedermann, J., Pardubská, D.: On the Power of Broadcasting in Mobile Computing. In: B. Cooper, B. Löwe, A. Sorbi (eds.), New Computational Paradigms. To appear in Springer, 2006. See also Technical Report V-944, Institute of Computer Science, October 2005, Prague, accessible via http://www.cs.cas.cz