On the Necessity of Formal Models for Real-Time Parallel Computations*

Stefan D. Bruda and Selim G. Akl
Department of Computing and Information Science
Queen's University
Kingston, Ontario, K7L 3N6 Canada

Abstract

We assume the multitape real-time Turing machine as a formal model for parallel real-time computation. Then, we show that, for any positive integer k, there is at least one language L_k which is accepted by a k-tape real-time Turing machine, but cannot be accepted by a $(k-1)$-tape real-time Turing machine. It follows therefore that the languages accepted by real-time Turing machines form an infinite hierarchy with respect to the number of tapes used. Although this result was previously obtained in [1], our proof is considerably shorter, and explicitly builds the languages L_k.

The ability of the real-time Turing machine to model practical real-time and/or parallel computations is open to debate. Nevertheless, our result shows how a complexity theory based on a formal model can draw interesting results that are of more general nature than those derived from examples. Thus, we hope to offer a motivation for looking into realistic parallel real-time models of computation.

Keywords: Real-time computation, parallel computation, multitape Turing machines, computational complexity.

1 Introduction

Some time ago, one of the authors received a challenge [18]:

> Can one find any problem that is solvable by an algorithm that uses k processors, $k > 1$, and is not solvable by a sequential algorithm, even if this sequential algorithm runs on a machine whose processor is k times faster than each of the k processors used by the parallel implementation?

In terms of the conventional theory of parallel algorithms, this question is, of course, meaningless. Indeed, it is standard to assume (in analyses of running time) that each processor on a parallel computer is as fast as the single processor on the sequential computer used for comparison. However, the question does make sense in practice. Besides, questions of this kind are crucial for the process of developing a parallel real-time complexity theory. Indeed, a meaningful such theory should be invariant to secondary issues like the speed of some particular machine. Thus, an answer to the above question is also important from a theoretical point of view (specifically, from the point of view of parallel real-time complexity theory).

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

There are some noteworthy results in the area of parallel algorithms for real-time computations [3, 4, 5, 6, 7, 10, 11, 9, 16], but none appears to address the above challenge. A positive answer to the question for $k = 2$ is provided by (a slightly modified version of) the *pursuit and evasion on a ring* example presented in [2]. In this version, an entity A is in pursuit of another entity B on the circumference of a circle, such that A and B move at the same speed; clearly, A *never* catches B. Now, if two entities C and D are in pursuit of entity B on the circumference of a circle, then C and D *always* catch B, even if C and D move at $1/x$ the speed of A (and B), $x > 1$. A computational analog of this example consists of two streams of input, both of them having to be monitored in order to solve a problem. Thanks to parallelism, two (slow) processors C and D operating simultaneously invariably succeed in completing the computation, whereas a single processor A which is twice as fast as either C or D always fails. This example can be easily extended to cases where $k > 2$.

One should note that most of the areas where superunitary behavior is manifested involve a real-time component. However, real-time computations received less attention from theoreticians. Even the term *real-time* is used by the complexity theorists in a somewhat different manner than in the real-time systems community: The systems researchers use the term to refer to those computations in which the notion of correctness is linked to the notion of time [21]. By contrast, theorists often use real-time as a synonym for *on-line.* As a consequence, there is a huge body of substantial theoretical work on on-line complexity [14], but real-time computations received considerably less attention. Indeed, there are few (sequential) formal models for such computations, and, to our knowledge, no parallel model at all. Therefore, a consistent parallel complexity theory of real-time systems has first to overcome this absence.

This paper does not attempt to introduce a new model. Instead, we shall consider an existing sequential model, and see whether variants of this model are suitable for modeling parallel computations. Specifically, we consider what appears to be the oldest model for real-time computations, namely the *real-time Turing machine* introduced in [22]. Machines belonging to this model, and the languages accepted by them, called *real-time definable languages*, were further studied in many papers, e.g., [1, 12, 17, 19, 20]. The model used is a deterministic one, but nondeterministic extensions were also studied, like the real-time Turing machines with restricted nondeterminism [13], and nondeterministic real-time Turing machines [8] (the languages accepted by the latter model being called *quasi-real-time languages*).

Although Turing machines are essentially sequential models, because of their finite control, the model offers the possibility of multiple tapes. A multitape machine is allowed to perform elementary actions on all of its tapes in parallel. Therefore, one can arguably call the real-time Turing machine a model for real-time parallel computation.

In this setting, an interesting problem related to superunitary behavior is whether an addition to the number of tapes of a real-time Turing machine increases its computational power. A partial answer was given in [17], where it is shown that a 2-tape real-time Turing machine is strictly more powerful than a 1-tape one. The general problem is addressed in [1], where it is shown that, for any positive integer k, a k-tape real-time Turing machine is strictly more powerful than a $(k-1)$-tape one. The proof of this result is based on the notion of overlap, and it is quite long. In addition, the languages L_k recognizable in real time by a k-tape Turing machine but not by a $(k-1)$-tape one are constructed implicitly, as languages accepted by specific Turing machines.

We offer a new proof of this result. By contrast with [1], our proof is considerably shorter and intuitive, and the languages L_k are explicitly constructed. Moreover, we identify some open questions in the area of real–time parallel computation that are suggested by the result of this paper.

The scope of this paper is therefore twofold: First, we give an improved result in the theory of real-time Turing machines. Second, we intend to open a discussion on formal models for real-time parallel computations. Granted, one can justifiably question our claim that Turing machines offer a realistic model for such computations. Moreover, as noted above, arguments can be made that the real-time Turing machine is not even a good enough model for practical real-time computations (that is, real-time computations as seen by the systems community) in general, no matter whether those computations are sequential or parallel. Therefore, we don't intend to suggest that one should stick to the real-time Truing machine. Instead, we emphasize the need for a realistic parallel model of real-time computations, by showing in this paper that the use of a formal model leads to more general results than those drawn from examples. To our knowledge, no work in the direction of a realistic model for parallel real-time computations exists to date. We hope that this paper offers a motivation for such work.

We present in the next section a concise introduction to real-time Turing machines and real-time definable languages. Then, in section 3, we show that real-time Turing machines form an infinite hierarchy with respect to the number of tapes used. We conclude in section 4.

2 Real-Time Turing Machines

This section is provided only for completeness. The reader familiar with real-time Turing machines may skip directly to section 3.

Given some alphabet A, the set A^k is defined recursively by $A^1 = A$, and $A^i = A \times A^{i-1}$ for $i > 1$.

We assume that the reader is familiar with the concept of a Turing machine; therefore we do not define the terms that are usually covered in a textbook on such a subject (e.g., [15]). A Turing machine M is said to accept some language L if, for any input string w, M stops (that is, M reaches the halt state h) iff $w \in L$ [15]. The empty word is denoted by λ.

We shall use the definition of real-time Turing machines presented in [19]:

Definition 2.1 [19]

1. For some constant k, $k \geq 1$, an *on-line Turing machine* is a deterministic $(k+1)$-tape Turing machine (with k working tapes and one input tape) $M = (K_p, K_a, \Sigma, W, \delta, s_0)$, where $K_p \cup K_a$ is the set of states, not containing the halt state h, s_0 is the initial state, Σ is the input alphabet, W is the alphabet of working symbols, containing the blank symbol #, and δ is the state transition function, $\delta : (K_p \times \Sigma \times W^k) \cup (K_a \times W^k) \longrightarrow (K_p \cup K_a \cup \{h\}) \times (\{R, L, N\}^k \times W^k)$. The head on the input tape is allowed to move only to the right.

   A *configuration* of an on-line k-tape Turing machine is a $(k+2)$-tuple $C = (q, t, x_1\underline{a_1}y_1, \ldots, x_k\underline{a_k}y_k)$, where q is a state, $t \in \Sigma^*$ is the (not yet considered) content of the input tape, for any i, $1 \leq i \leq k$, $x_i a_i y_i$ is the content of the i-th working tape, and a_i is the symbol that is currently scanned by the head of tape i. If a configuration C_1 yields another configuration C_2, we write $C_1 \vdash_M C_2$. As usual, $\vdash_M^*$ denotes the transitive and reflexive closure of $\vdash_M$.

   The set of states is divided into two subsets: the set of *polling* states K_p and the set of *autonomous* states K_a. All the states that lead to h in one step are polling states, and the initial state is a polling state. In addition, the relation $\vdash_M$ has the following property: if $q \in K_p$, $q'' \in K_a$, and $q' \in K_p \cup K_a$, then $(q, abv, x_1, \ldots, x_k) \vdash_M (q', bv, x'_1, \ldots, x'_k)$, $(q'', abv, x_1, \ldots, x_k) \vdash_M (q', abv, x'_1, \ldots, x'_k)$, and $(q, \lambda, x_1, \ldots, x_k) \vdash_M (h, \lambda, x'_1, \ldots, x'_k)$.

M accepts the input w iff $(s_0, w, x_1, \ldots, x_k) \vdash_M^\tau (h, \lambda, x_1, \ldots, x_k)$, where $\tau > 0$ is called the *running time* of M on w.

2. A *real-time Turing machine* is an on-line Turing machine for which $K_a = \emptyset$. A language accepted by such a machine is called a *real-time definable language.*

□

In other words, an on-line Turing machine has a unidirectional input tape. Therefore, it has no knowledge about further input data. Between reading two input symbols, such a machine is allowed to go into a number of autonomous states, where it performs some work without considering any input. In addition to these requirements, a real-time Turing machine has no autonomous state, it being forced to consume an input datum at every step.

Definition 2.2 A nondeterministic real-time Turing machine is a machine that is identical to the one defined in definition 2.1, except that $\delta \subseteq ((K_p \times \Sigma \times W^k) \cup (K_a \times W^k)) \times ((K_p \cup K_a \cup \{h\}) \times (\{R, L, N\}^k \times W^k))$. The languages accepted by nondeterministic real-time Turing machines are called *quasi-real-time* languages [8]. □

3 k Tapes are More Powerful than $k-1$ Tapes

In the following, given a word u, u^r denotes the reversal of u. For a fixed positive integer k and given some alphabet Σ, and two symbols \$ and @, such that \$, @ $\notin \Sigma$, let us consider the language

$$\begin{aligned} L_k &= \{\$w_1\$w_2\$w_3\$...\$w_k@u^r \mid w_1, \ldots, w_k, u \in \Sigma^*, \\ &\quad \text{there exists some } i,\ 1 \le i \le k, \text{ such that } u = w_i\}. \end{aligned} \tag{1}$$

Let w be a word in L_k, $w = \$w_1\$w_2\$w_3\$...\$w_k@u^r$. We denote by w_{ij} the j-th symbol of the subword w_i, $1 \le i \le k$, and by u_j^r and u_j the j-th symbol of u^r and u, respectively. The length of some word x is denoted by $|x|$.

Lemma 3.1 *There is a k-tape (not counting the input tape) deterministic real-time Turing machine, with the working alphabet $\Sigma \cup \{\$, @\}$ that accepts L_k.*

Proof. Such a machine M works as follows. While reading the input w, it writes each subword $\$w_i$, $1 \le i \le k$, on tape i. That is, it starts by writing the initial \$ symbol on the first tape. Then, when the head on the input tape reads some symbol w_{ij}, M writes w_{ij} on its i-th tape, and advances that tape's head to the right. When the \$ symbol that terminates the subword w_i is read, M moves the head of tape i one cell to the left, and, at the same time, writes \$ on tape $i+1$ (provided that $i \le k-1$). Clearly, writing the word w_i requires precisely $|w_i|$ time.

After the @ symbol is read, M starts comparing simultaneously the currently read symbol u_j with the subwords stored on each of its k tapes. For any tape i, if $u_j = w_{ij}$, then the machine advances the head of tape i one cell to the left; otherwise, it writes @ on tape i, and never moves the head of that tape afterwards. This step also takes $|u|$ time.

At the end of the input, on a tape i for which $u = w_i$, it is clear that the head advances each time an input symbol is read, and when the input is exhausted, the head points to the initiating \$ sign. On the other hand, assume that $|u| > |w_i|$. Then, after exhausting the word w_i, M eventually compares \$ with some symbol in Σ, which are obviously different. Then, it writes @ on tape i and never moves that head. Analogously, if $|u| < |w_i|$, then the head points to some symbol from w_i when the end of the input is reached, which is not \$. Finally, on those tapes i where there is a j such that $u_j \neq w_{ij}$, M writes @ and subsequently never moves the head (as mentioned in the previous paragraph). Therefore, at the end of the input, M accepts the input iff the head of at least one tape points to a \$ sign. As shown above, this happens iff there is some subword w_i such that $u = w_i$.

Moreover, it is clear that M reads one input symbol in each state, therefore M is a real-time Turing machine. □

Lemma 3.2 *There is no $(k-1)$-tape (not counting the input tape) deterministic real-time Turing machine that accepts L_k, even if the working alphabet of M is $(\Sigma \cup \{\$, @\})^{k'}$ for some k' that depends on k.*

Proof. Assume that there exists such a machine, and denote it by M. M has $k-1$ tapes but there are k subwords w_i. However, all the subwords have to be stored somewhere, since u can match any of them. Let the length of the input word be n.

We ignore for the moment the way in which the k subwords are written on the tapes of M, but focus instead on the computation that M has to perform when u^r is read. When reading a symbol u^r_j, $1 \leq j \leq |u|$, M has to compare this symbol with all the symbols $w_{ij'}$, where $1 \leq i \leq k$, with $|w_i| \geq |u|$, and $j' = |u| - j$, in exactly one step. The only way of achieving this is that the cells immediately accessible on the $k-1$ tapes of M must contain all the symbols $w_{ij'}$. That is, for any j, $1 \leq j \leq |u|$, at the moment when the symbol @ is read, there is some tape cell that contains at least the symbols $w_{lj'}$ and $w_{mj'}$ for some m and l, where $1 \leq l < m \leq k$ and $j' = |u| - j$. When a symbol $w_{mj'}$ is written in the same tape cell as some symbol $w_{lj'}$, we say that M *superimposes* $w_{mj'}$ [over $w_{lj'}$].

Since there are only $k-1$ tapes, and the maximum capacity of a tape cell is k' symbols, where both k and k' are constant with respect to n, the above result implies that there is at least one word (let it be w_m) in which p symbols are superimposed, where $p = O(n)$. Let these symbols be $w_{mj_1}, w_{mj_2}, \ldots, w_{mj_p}$, and note that, before superimposing any symbol, M has to know the length of w_m. Indeed, a symbol w_{mj_q} is superimposed over some symbol w_{lo}, where $|w_m| - j_q = |w_l| - o$. Therefore, w_{mj_q}, $1 \leq q \leq p$, can be superimposed only after the end of w_m is reached. However, it is immediate that M can superimpose in one step at most $k'(k-1)$ symbols. That is, after reaching the end of w_m, M needs at least $p/(k'(k-1))$ steps in order to complete the processing of w_m. But consider now those input strings for which $|w_{m+1}| + \cdots + |w_k| < p/(k'(k-1))$. Clearly, since M is a real-time Turing machine, it has to consume an input symbol at each step. However, there are more steps than input symbols. That is, M cannot complete the processing of w_m before @ is read, and therefore it cannot compare u with all the subwords $w_1, \ldots, w_k$ in $|u|$ steps, which is in contradiction to our initial assumption that M is real-time.

We have yet to justify some facts in order to complete the proof. First and most important, we assumed that at most k' symbols can be written on a tape cell, with k' constant with respect to the length of the current input. This is clearly true, even if the symbols are written using some (however clever) encoding, since the working alphabet of M cannot depend on the input. Indeed, it is immediate that (a) all the subwords w_i of w have to be stored somewhere on the working tapes, and (b) in the general case, once a symbol u^r_j of u^r, $1 \leq j \leq |u|$, has been read, it must be compared in one step with all the corresponding symbols in w_i, $1 \leq i \leq k$ (a real-time Turing machine is also on-line and, in addition, there are no autonomous states). Since only $k-1$ cells of the working tapes can be read in one step, it follows that, no matter how the input is encoded on the working tapes, two symbols w_{mj} and w_{lj}, $1 \leq m < l \leq k$, must coexist (whichever the meaning of "coexist" is) in the same tape cell. This is achievable in real-time only if *all* the symbols between w_{mj} and w_{lj} are stored without moving the head of the tape storing w_{mj} and w_{lj} more than one cell. As shown above, this is impossible, and this impossibility is invariant with the encoding/compression used—in fact, the requirement to move all the heads by more than one cell is given solely by the fact that the number of symbols between w_{mj} and w_{lj} on the input tape is not bounded by any constant and can be determined only after M reads w_{lj}.

Then, note that M is deterministic. Moreover, while considering the subword w_m, M cannot determine the length of the subsequent words, since it is on-line. That is, the choice of m is independent of the not yet considered part of the input, and hence the above choice of the words $w_{m+1}, \ldots, w_k$ (more precisely, the choice of their length) is justified.

Finally, we also implicitly used a fact related to the one in the previous paragraph. Indeed, one can say that, when the choice of m turns out to be wrong (in the sense that there is no input to consume while M superimposes the symbols w_{mj_q}), M can return to a previous state and try another choice. However, the wrong choice manifests itself when M reads the symbol @, and then it has no time to reverse the computation, since it needs the remaining $|u|$ steps to read and compare the subword $|u|$.

Note that, in fact, the job of M would have to be even harder. Specifically, M would have to remember the indices of the symbols whose superimposing is postponed until the end of the current subword is reached, and this implies extra space and extra state transitions. However, we showed some facts that are enough to prove the impossibility of L_k's acceptance by M, and we hence completed the proof. □

Lemma 3.3 *There is a one-tape (not counting the input tape) nondeterministic real-time Turing machine that accepts L_k.*

Proof. The machine nondeterministically guesses that subword w_j that matches u, writes it on its working tape, and then compares u with the stored subword. □

The main result of our paper follows immediately from lemmas 3.1, 3.2, and 3.3:

Theorem 3.4 *For any positive integer k, there is at least one (quasi-real-time) language which is accepted by a k-tape real-time Turing machine, but cannot be accepted by a $(k-1)$-tape real-time Turing machine. Therefore, the languages accepted by k-tape real-time Turing machines form an infinite hierarchy with respect to k.* □

4 Conclusions

We showed in this paper that, for any positive integer k, a k-tape real-time Turing machine is strictly more powerful than a $(k-1)$-tape one.

Of course, an answer to the challenge articulated at the start of section 1 depends on one's definition of the word "solvable". However, if one replaces "algorithm" by "Turing machine" and "processor" by "tape" (since this is the only thing in a Turing machine that can grow as desired), then theorem 3.4 also represents a positive answer to the above challenge. Indeed, even if we restrict the working alphabet of a k-tape real-time Turing machine to the input alphabet Σ, but allow the one-tape Turing machine to use $\Sigma^{k'}$ as working alphabet, $k' \geq k$, hence giving to the unique working tape a k'-fold improvement in performance, the one-tape Turing machine is still not able to handle the acceptance of the language described in equation (1).

In this context, it is also evident that the very question of whether a k-tape Turing machine is equivalent in some sense to a k-processor computer is itself debatable. However, it is hoped that a discussion of this subject can lead to the development of a more realistic model for parallel real-time computations. Thus, we conclude this paper with a new challenge: Either prove or disprove the mentioned equivalence (between a real-time multi-processor machine and a real-time multi-tape Turing machine). We believe in fact that there is no such equivalence—not because of differences between a tape and a processor, but because the differences among the definitions of "real-time" across the computing community instead, as mentioned in the introductory section. However, even if the two models of computation are found not to be equivalent (as we believe to be the case), they still share some properties, since both of them model parallelism (if not real-time) in some sense. Then, an alternate question to the one above is: Do these models share the property of forming infinite hierarchies? That is, is there a (nontrivial) infinite hierarchy, this time with respect to the number of processors on a multi-processor abstract machine, which is similar in spirit to the hierarchy found in this paper for the real-time Turing machines? Of course, before being able to answer such a question at all, one must find *that* realistic parallel real-time model itself. We hope that this paper offers a motivation for such a pursuit.

One final remark related to (and in support of) the above challenge concerns the languages L_k developed in section 3. One can note that each of these languages models a search for some word (u) in a set of subwords ($\{w_1, \ldots, w_k\}$). Such a processing is commonplace in many algorithms of practical importance. Therefore, finding an equivalent hierarchy with respect to a more realistic parallel model would further strengthen the importance of parallelism in the real-time area.

Acknowledgments We thank the anonymous referees for their insightful comments.

References

[1] S. O. Aanderaa, *On k-tape versus $(k-1)$-tape real time computation*, in Complexity of Computation, R. Karp, ed., SIAM-AMS Proceedings, volume 7, 1974, pp. 75–96.

[2] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[3] ——, *Secure file transfer: A computational analog to the furniture moving paradigm*, in Proceedings of the Conference on Parallel and Distributed Computing Systems, Cambridge, MA, November 1999, pp. 227–233.

[4] S. G. Akl and S. D. Bruda, *Parallel real-time optimization: Beyond speedup*, Parallel Processing Letters, 9 (1999), pp. 499–509. For a preliminary version see http:// www.cs.queensu.ca/ ~akl/ techreports/ beyond.ps.

[5] ——, *Parallel real-time cryptography: Beyond speedup II*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, June 2000, pp. 1283–1290. For a preliminary version see http:// www.cs.queensu.ca/ ~akl/ techreports/ realcrypto.ps.

[6] ——, *Parallel real-time numerical computation: Beyond speedup III*, International Journal of Computers and their Applications, 7 (2000), pp. 31–38. For a preliminary version see http:// www.cs.queensu.ca/ ~akl/ techreports/ realnum.ps.

[7] S. G. Akl. and L. Fava Lindon, *Paradigms admitting superunitary behaviour in parallel computation*, Parallel Algorithms and Applications, 11 (1997), pp. 129–153.

[8] R. V. Book and S. A. Greibach, *Quasy–realtime languages*, Mathematical Systems Theory, 4 (1970), pp. 97–111.

[9] S. D. Bruda and S. G. Akl, *A case study in real-time parallel computation: Correcting algorithms*, To appear in Journal of Parallel and Distributed Computing. For a preliminary version see http:// www.cs.queensu.ca/ ~bruda/ www/ c-algorithms.

[10] ——, *On the data-accumulating paradigm*, in Proceedings of the Fourth International Conference on Computer Science and Informatics, Research Triangle Park, NC, October 1998, pp. 150–153. For a preliminary version see http:// www.cs.queensu.ca/ ~bruda/ www/ data_accum.

[11] ——, *The characterization of data-accumulating algorithms*, Theory of Computing Systems, 33 (2000), pp. 85–96. For a preliminary version see http:// www.cs.queensu.ca/ ~bruda/ www/ data_accum2.

[12] P. C. Fischer, *Turing machines with a schedule to keep*, Information and control, 11 (1967), pp. 138–146.

[13] P. C. Fischer and C. M. R. Kintala, *Real-time computations with restricted nondeterminism*, Mathematical Systems Theory, 12 (1979), pp. 219–231.

[14] S. Irani and A. R. Karlin, *Online computation*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., International Thomson Publishing, 1997, pp. 521–564.

[15] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[16] F. Luccio and L. Pagli, *Computing with time–varying data: Sequential complexity and parallel speed–up*, Theory of Computing Systems, 31 (1998), pp. 5–26.

[17] M. O. Rabin, *Real time computations*, Israel Journal of Mathematics, 1 (1963), pp. 203–211.

[18] D. Rappaport, *Private communication.*

[19] A. L. Rosenberg, *Real-time definable languages*, Journal of the ACM, 14 (1967), pp. 645–662.

[20] ——, *On the independence of real-time definability and certain structural properties of context-free languages*, Journal of the ACM, 15 (1968), pp. 672–679.

[21] Usenet, *Comp.realtime: Frequently asked questions*, Version 3.4 (May 1998). http:// www.faqs.org/ faqs/ realtime-computing/ faq/.

[22] H. Yamada, *Real-time computation and recursive functions not real-time computable*, IRE Transactions on Electronic Computers, EC-11 (1962), pp. 753–760.