CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2012, Article 07, pages 1–24
http://cjtcs.cs.uchicago.edu/

Computational Models with No Linear Speedup

Amir M. Ben-Amram   Niels H. Christensen   Jakob Grue Simonsen[*]

Received: October 11, 2011; revised: October 1st, 2012; published: October 16, 2012.

Abstract: The linear speedup theorem states, informally, that constants do not matter: It is essentially always possible to find a program solving any decision problem a factor of 2 faster. This result is a classical theorem in computing, but also one of the most debated. The main ingredient of the typical proof of the linear speedup theorem is tape compression, where a fast machine is constructed with tape alphabet or number of tapes far greater than that of the original machine. In this paper, we prove that limiting Turing machines to a fixed alphabet and a fixed number of tapes rules out linear speedup. Specifically, we describe a language that can be recognized in linear time (e. g., $1.51n$), and provide a proof, based on Kolmogorov complexity, that the computation cannot be sped up (e. g., below $1.49n$). Without the tape and alphabet limitation, the linear speedup theorem does hold and yields machines of time complexity of the form $(1+\varepsilon)n$ for arbitrarily small $\varepsilon > 0$.

Earlier results negating linear speedup in alternative models of computation have often been based on the existence of very efficient universal machines. In the vernacular of programming language theory: These models have very efficient self-interpreters. As the second contribution of this paper, we define a class, PICSTI, of computation models that exactly captures this property, and we disprove the Linear Speedup Theorem for every model in this class, thus generalizing all similar, model-specific proofs.

[*]Jakob Grue Simonsen is partially supported by the Danish Council for Independent Research *Sapere Aude* grant "Complexity through Logic and Algebra" (COLA).

Key words and phrases: complexity, hierarchy theorem, linear speedup, self-interpretation

DOI: 10.4086/cjtcs.2012.007

1 Introduction

The classical linear speedup theorem (a.k.a. the constant-factor speedup theorem) is due to Hartmanis and Stearns [11]. In stating the theorem, one has to be precise regarding the Turing Machine model assumed. We cite a version from [48, Thm. 18.12]:

Theorem 1.1. *Let $k \geq 1$ and consider the class of Turing machines with 1-way, read-only input tape and k one- or two-way work tapes. For every $t : \mathbb{N} \longrightarrow \mathbb{N}$, if there exists a real number $c > 1$ such that $t(n) \geq cn$ for all but finitely many n, then any problem decidable in deterministic time $O(t(n))$ is decidable in deterministic time $t(n)$.*

The linear speedup theorem remains valid for non-deterministic time and for machines where the input tape is two-way, or identified with one of the work tapes (when there is more than one of those).

All known proofs of the theorem are constructive and work by a technique that can aptly be called *tape compression*: *Either* increasing the size of the alphabet *or* allowing the number of tapes of M' to be much greater than that of M. As an artifact of the definition of time complexity for Turing machines, tape compression allows the reading or writing of a great number of bits simultaneously in a single computation step. Furthermore, the number of bits can be as large as you like, a property whose basis in realistic computation is quite dubious.

In this paper, we prove that tape compression is not only sufficient, but also necessary for the linear speedup theorem to hold: Limiting Turing machines to a fixed alphabet and a fixed number of tapes disproves the linear speedup theorem. Specifically, we describe a language that can be recognized in linear time (e. g.$1.51n$) and prove that the computation cannot be sped up (e. g., below $1.49n$); without the limitation on tape and alphabet, the speedup theorem does hold and yields solutions of time complexity of the form $(1+\varepsilon)n$ for arbitrarily small $\varepsilon > 0$.

Linear speedup seems to be particular to Turing machines. In more sophisticated models we often have hierarchy theorems that contradict speedup. These theorems are mostly proved by diagonalization; for the hierarchy to be tight, we need very efficient universal machines ("self-interpreters" in programming language vernacular). In Turing machines, an interpreter intuitively has to scan the simulated program in each step of the simulation, due to the lack of efficient memory access.

As the second contribution of this paper, we define a class, PICSTI, of computation models that captures the property of having a highly efficient self-interpreter (along with a few, more technical, properties, required for establishing a diagonalization argument). We disprove the Linear Speedup Theorem for every model in this class, thus generalizing all similar, model-specific proofs.

Organization of the paper Section 2 shows (by means of a specific counter-example) that the linear speedup theorem fails for Turing machines when alphabet- or tape compression is disallowed. Section 3 introduces the class of models that are PICSTI—have program-independent, constant-time self-interpretation overhead. Section 3.4 shows that the linear speedup theorem does not hold in any PICSTI model. Section 4 concludes with a list of open problems related to our results.

The reader is assumed to be familiar with introductory computability and complexity theory at the level of standard textbooks [17, 44, 18, 27].

1.1 Related work

The term "speedup" seems to have been employed for two related, but quite different phenomena: (i) *linear speedup* as in Theorem 1.1 above, (ii) the lack of an (asymptotically) optimal algorithm, sometimes called *Blum Speedup*. In its original form [4, 23], the Blum Speedup Theorem described a construction of a decision problem for which it could be shown that no asymptotically fastest algorithm exists. More precisely, given a (suitable) *speedup function*, a decision problem is generated so that for every algorithm solving the problem, an algorithm faster by the given "amount of speedup" exists. Thus there is an infinite sequence of increasingly faster algorithms. Schnorr and Stumpf later showed that one could control the complexity of the algorithms, so that, for example, they all lie in PTIME, or even quasi-linear time [38].

While the problems exhibiting speedup considered by Blum, Schnorr and Stumpf were all artificially constructed for the purpose, Coppersmith and Winograd showed that for a particular class of algorithms for matrix multiplication in polynomial time, any of these algorithms could always be replaced by another with slightly smaller exponent. It is conjectured that a similar result holds true for any algorithm for matrix multiplication [25, 26].

The present paper considers only *linear* speedup; we next review some previous work on this problem.

Speedup theorems à-la Turing machine, but for different models, are scarce. Regan [33] generalized the linear speedup of Turing machines to his class of *Block-Move* models, with a certain range of cost measures. A Block-Move model operates on a tape, like a Turing machine, but using a more complex programming model in which finite-state transducers are invoked to operate on data as they are copied from one part of the tape to another.

A linear speedup theorem is also known to hold for cellular automata, as proved by Mazoyer and Reimen [22]. The construction increases the number of possible states of *each* of the potentially infinite number of cells in a run of the automaton by an amount exponential in the speedup factor, hence the method effectively extends the amount of "memory" readable in a single computation step in the same fashion as the standard proof for Turing machines.

For counter machines with a one-way input tape, Fischer, Meyer and Rosenberg [8] proved a linear speedup theorem stating that for any real constant $c > 0$ and any k-counter machine ($k \geq 1$) operating in time $n + E(n)$, for some function E, there exists an equivalent counter machine operating in time $n + \lfloor c \cdot E(n) \rfloor$. Interestingly, this speedup does not require some analogue of tape compression or the use of additional counters; it only records a greater amount of information in the finite-state control.

For several restrictions or variants of the Turing machine, linear speedup does *not* hold.

For *single*-tape Turing machines, Šakuov claimed in [36, 37] the existence of problems, decidable in time between $n \log n$ and n^2, that do not admit linear speedup, but no proofs were given, nor have they ever appeared.

For *online* Turing machines, where the input is a sequence $x_1, \ldots, x_n$ of queries and the machine must answer query x_i before accessing query x_{i+1}, it is in general easier for a malicious adversary to force the machine to use more computational resources; thus lower-bound results (counteracting speedup) have sometimes been obtained in this model, whereas off-line counterparts have not been proved as yet. A line of research in which the on-line assumption has often been made is the comparison of the power of different abstract storage units (e.g., simulating a multi-tape storage device using fewer tapes); see [29, 28, 15] and references within. Several proofs of this kind can possibly be simplified, or new proofs enabled, by the use of Kolmogorov complexity theory, as done for example in [30].

Hühne [13] considered online Turing machines with an efficient tree-structured memory model instead of the usual tape, and showed, using Kolmogorov complexity, that constant (time) factor speedup does not hold for online machines of this kind. Regan [32] later explained the success of Hühne's proof (in contradiction with the fact that such a result cannot be proved for ordinary TMs, given the speedup theorem). He gave an informal argument according to which only machine models that have *polynomial vicinity* can have the speedup phenomenon. Polynomial vicinity means that $O(t^c)$ storage cells can be reached in t steps from any given configuration, where c is some constant. This class of models has been noted before (with a definition nearly identical to Regan's) by Cook and Anderaa [6], who proved, for such machines, a non-linear lower bound for online multiplication. However, none of the authors achieved a constant-factor tight lower bound (contradicting linear speedup) for the full range of such models. Regan *has* been able to refine Hühne's arguments and show that linear speedup does not hold for online machines in his class of *Block-Move* models, with a certain range of cost measures. These measures endow the machines with super-polynomial vicinity (and differ, of course, from those measures for which he proved linear speedup, as mentioned above).

Žák [47] considered a somewhat artificial complexity model for Turing machines, in which the machine has to be executed by a single fixed interpreter, and the time of interpretation is measured. In this model he shows that a constant-factor separation holds.

Contrasting the result of Fischer, Meyer and Rosenberg [8], Petersen [31] proves that linear speedup does *not* hold for k-counter machine with a 2-way input tape, for any fixed k.

For *space complexity* of Turing machines, some hierarchy results that contradict speedup are known. An early example is [14] which establishes constant-factor tight hierarchies within polynomial-space complexity classes: specifically, they prove that for any real constants $b > a > 0$ and $r \geq 1$, when the alphabet size (≥ 2) is fixed, more sets can be decided in space an^r than in space bn^r, for machines with a fixed number of tapes. Increasing the number of tapes, or the size of the alphabet, is tantamount, roughly, to increasing the space bound (an observation originally made by Stearns [45]). Another "folklore theorem" is that for machines working within space S, an extra head is worth additional space proportional to $\log S$ (namely, the representation of the head position).

A further investigation of tradeoffs between alphabet size, work space, and tape heads was done by Seiferas [42, 41]. For example, given a sufficiently large $m \in \mathbb{N}$ and space-constructible $S(n)$, which does not grow too fast, there are problems decidable by an offline Turing machine M having alphabet size $m+1$ within space $S(n)$ that are not decidable by any Turing machine M' with alphabet size m within such space, no matter the number of tape heads available to M'.

Seiferas and Meyer [43] wrote an exposition of a fundamental theorem by Levin [20], characterizing the sets of functions that describe the space usage of Turing machines solving some problem. This sets the scene in general for both speedup and hierarchy results; a corollary closely related to our work is a very tight hierarchy theorem for the special case of space on a Turing machine of a fixed alphabet, showing that every constructible space bound is the space complexity of some problem, which cannot be "sped up" even by the tiniest constant factor (note that this is stronger than a hierarchy theorem as in [14]). Next, we mention some results that contradict Linear speedup (in time) for several models of computation quite distinct from the Turing machine.

For random access machines (RAMs), Sudborough and Zalcberg showed that the unit-cost RAM does not have linear speedup by proving a constant-factor hierarchy theorem [46]:

Theorem 1.2. *For every RAM time-constructible function $T(n)$ where $T(n) \geq n$, there is a constant $c > 1$ and a language $L \subseteq \{0,1\}^*$ such that*
(i) L is decided by a RAM in time at most $cT(n)$, and
(ii) L is not decided by any RAM in time at most $T(n)$.

The proof, like all other proofs mentioned below, uses the standard diagonalization argument, and the tightness of the result is due to the efficiency of the interpreter.

Blass and Gurevich [3] independently rediscovered the hierarchy theorem for random access machines and—significantly—extended the result to the Abstract State Machines of Gurevich.

For Jones' language I — a simple, Turing-complete imperative programming language where all data are binary trees — Jones [16] and Jones and Ben-Amram [2] showed that linear speedup does not hold. Rose [35] showed that speedup does not hold for the class of *Categorical Abstract Machines*, a model of higher-order functional programming languages.

All of the above results concern *deterministic* computation. Surprisingly, even when explicitly disallowing tape compression, *non-determinism* allows for linear speedup, as shown by the following remarkable result by Geffert [10]:

Theorem 1.3. *For every* nondeterministic *single-tape Turing machine M with alphabet $\{0,1\}$ such that the running time of M is bounded above by $T(n) \geq n^2$, and each $K \geq 1$, there exists an equivalent one-tape Turing machine M' with alphabet $\{0,1\}$ whose running time is bounded above by $T(n)/K$.*

Geffert also showed that for time bounds $T(n)$ that do not meet the requirement $T(n) \geq n^2$, the machine M' can be constructed by adding a only a single symbol to the original tape alphabet.

Geffert posed two open problems [10, p. 64]: Whether it is possible to extend the above theorem to (i) deterministic machines, and (ii) to two-tape non-deterministic machines. The results of the present paper provide a very partial answer to (i): The theorem cannot be extended to *multi*-tape deterministic machines (the case for single-tape machines is still unknown). Geffert's question (ii) is still open.

Many Turing-machine lower bounds are proved using the method of *crossing sequences*—even constant-factor tight bounds: for example, Kobayashi [19] proved a constant-factor tight hierarchy theorem for one-tape non-deterministic Turing machines, for a restricted class of time bounds. However, this theorem shows that increasing the time bound increases the class of decidable sets under the assumption that the *number of control states cannot be increased*. In general, the calculation of the number of crossing sequences involves the number of states, which makes it difficult to prove a constant-factor separation if a bound on the number of states is not imposed. Note that the lower bound we present in this paper does not assume a bound on the number of states. Kobayashi noted: "Crossing sequences are also difficult to use for the analysis of how the size of the working-tape alphabets affects efficiency of computations." An earlier, related paper by Hennie [12] also used crossing sequences for a separation that becomes constant-factor tight if the number of states is bounded. Hennie remarked that the method "does not seem to be useful for obtaining time bounds for (two-tape) machines". It therefore comes as a surprise that Petersen's result [31] for counter machines does use crossing sequence arguments for the lower bound (which is so formulated that the dependence on the number of states disappears when input size grows); it rests the single example of its kind known to us.

2 Linear speedup fails for multi-tape Turing machines without tape compression

This section establishes that for offline Turing machines with a *fixed* alphabet size and a *fixed* number of tapes, there is a decision problem that defies linear speedup (as expressed by Theorem 1.1).

Note that offline machines are special cases of online machines: The sequence of queries is trivial, the machine gets all of its input at once. Consequently, our results in this section are valid for online models as well.

Tight complexity results regarding Turing machines may be sensitive to the specification of the model. We consider Turing machines with input and work-tape alphabet containing '0' and '1', where 0 doubles as blank. On a given input, a machine either halts in the designated "accept" or "reject" states, or fails to halt.

Each Turing machine has a read-only, one-way input tape and two (two-way, read-write) work tapes. Each tape is assumed to have exactly one head. In the taxonomy of Wagner and Wechsung [48], our machines are 1-2T1-DMs (the first tape is one-way and read-only by convention, the next two tapes are 2-way, and the machine is deterministic).

We assume the work-tapes to be semi-infinite, with a beginning-of-tape mark on the left (as in [27]); the input tape also has and end marker. In the initial configuration, all heads scan the beginning-of-tape marks. The cells of the work tapes are numbered consecutively starting with 1.

Our goal is to prove the following theorem—note the contrast with Theorem 1.1.

Theorem 2.1. *There is a family of languages $\{L_k \mid k > 0\}$ such that for every $\varepsilon > 0$, there is a k such that L_k is decidable by a TM in our class in at most $(1+\varepsilon)\cdot 1.5n$ steps for all n, while any machine deciding L_k must use at least $(1-\varepsilon)\cdot 1.5n$ steps on infinitely many inputs.*

Proof. The proof comprises the definition of the languages (given next in this subsection), An upper bound (Lemma 2.5), and a lower bound (Corollary 2.14). □

As intermediate steps in both the proof of the upper bound and of the lower bound we will use a single problem L (independent of ε) and a setting which is easier to work with—specifically, we include an additional symbol $\sharp$ in the alphabet.

Definition 2.2. When $b_0 \dots b_{m-1} \in \{0,1\}^m$ and n is a natural number (in binary, least significant bit first), define

$$get(n, b_0 \dots b_{m-1}) = \begin{cases} 0 & \text{when } n \geq m \\ b_n & \text{otherwise} \end{cases}$$

We define L to be the language $\{w\sharp i \mid get(i \mod 2^{\lceil \log |w| \rceil}, w) = 1\}$

Example: The following four values are in L: 011♯1, 011♯01, 011♯101, 011♯011. None of the following four values are in L: 011♯0, 011♯11, 011♯001, 011♯111.

For this language, we will have an upper bound as in Theorem 2.1, but the lower bound will be weaker, and to enable our tight result, we replace L with a family L_k of languages over the binary alphabet.

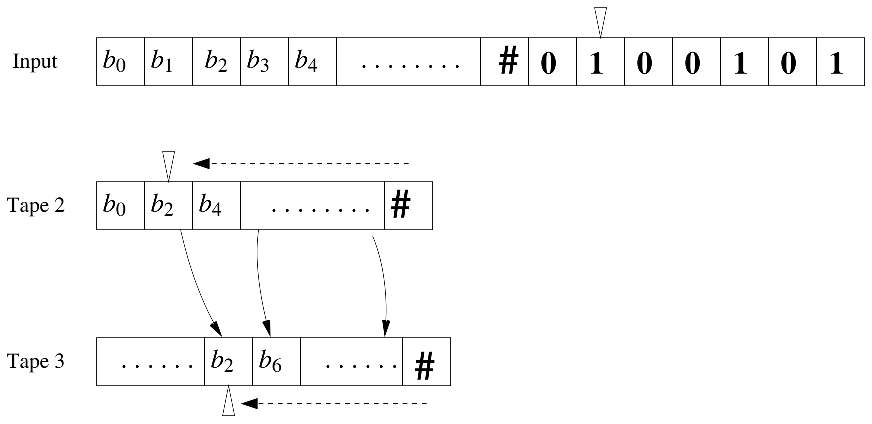
Input
b_0 b_1 b_2 b_3 b_4 .... # 0 1 0 0 1 0 1
Tape 2
b_0 b_2 b_4 .... #
Tape 3
...... b_2 b_6 ...... #

Figure 1: The contents of the tapes of machine M_1, as the first iteration of the main loop (3) is nearing its end. The dashed arrows indicate the trajectory of the work-tape heads during this iteration. Note that the symbols copied to Tape 3 have indices which match the two lower bits of i.

Definition 2.3. For any $k > 0$, define $L_k = \{c_k(w)0i \mid w\sharp i \in L\}$, where $c_k(w)$ is the string obtained from w by padding it with zeros, if necessary, to a multiple of k bits, and inserting a 1 before every block of k consecutive bits.

Observe that $|c_k(w)| = (k+1) \cdot \lceil (1/k) \cdot |w| \rceil$, and that, when reading a word in L_k, it is easy to recognize where the first part (representing w) ends and the representation of i begins.

2.1 Upper bound

Lemma 2.4. *For every $\varepsilon > 0$, there exists a 1-2T1-DM with alphabet $\{0,1,\sharp\}$ that decides L using at most $(1+\varepsilon) \cdot 1.5n$ steps for all n.*

Proof. For clarity, we describe the general idea by giving a machine M_1 that obtains the bound $(1+\varepsilon)2n$. The idea is then refined by a machine M_b that obtains the desired bound. b is a parameter to be specified later.

We assume that the input has the form $w\sharp i$ where $w = b_0 \dots b_{m-1} \in \{0,1\}^m$, $m \leq n$, and i represents a binary string which is interpreted as a number, as in Definition 2.2. It is trivial to ensure that the machine rejects strings which are not in this form. We use the identifiers w, i and b_j for the substrings they represent in this description (b_j is a single bit). Note that the essential task of the machine is to retrieve b_i.

Description of M_1.

1. Split the input word (up to the $\sharp$) among tapes 2 and 3. That is, copy $b_0b_2b_4\dots$ to Tape 2, and $b_1b_3b_5\dots$ to Tape 3, all in one pass over w. In case that the length of w is odd, an extra 0 is copied to Tape 3, so that the same number of cells has been written on both tapes. The machine keeps track of the parity of the number of cells written (in other words, the position in the tape is maintained modulo 2).

2. Skip the $\sharp$ on the input tape; write a $\sharp$ on each of the work tapes, thus marking the end of the part of w stored there.

3. Let the first digit of i be $x \in \{0,1\}$. If $x = 0$ then the value of i is an even number and the symbol which we want to retrieve, b_i, is known to have been copied to Tape 2. If $x = 1$, b_i has been copied to Tape 3. Hence, $t = 2 + x$ is the index of the tape of interest. The value of t is recorded in the finite control once x has been read. Denote by $\bar{t}$ the index of the other work tape $(2 + (1 - x))$. Now, enter the main loop:

   - Move the input head to the next digit of i. Let this digit be $x \in \{0,1\}$. Scanning Tape t in a direction inverse to the last scan, copy only the even (if $x = 0$) or only the odd (if $x = 1$) positions from tape t to tape $\bar{t}$, overwriting its previous contents. The copying stops when either the beginning-of-tape marker or the $\sharp$ is encountered[1]; in the case $x = 0$, if the stop condition occurs after an odd number of symbols was scanned, another 0 is "copied." As above, the number of symbols written to tape $\bar{t}$ is maintained modulo 2, allowing the machine to identify the odd or even positions when scanning backwards in the next iteration. Switch t with $\bar{t}$.

4. The main loop ends when a single bit is all that remains of w after the repeated splitting[2]. That remaining bit should be a 1 for the machine to accept.

The reader may observe that the "rounding up" to an even string length, each time we copy, is equivalent to "virtually" extending w with zeros up to the next power of two, namely to a length of $2^{\lceil \log |w| \rceil}$ (cf. Definition 2.2). At iteration k of the main loop, tape t contains the bits of w in positions that agree with i on their k least-significant bits. The correctness of the result should now be obvious.

Analysis of M_1: The sum of the lengths of the tape scans is

$$m + \left\lceil \frac{m}{2} \right\rceil + \left\lceil \frac{m}{4} \right\rceil + \cdots + 1 \leq 2m + \log m$$

and the extra work at the end of each scan (recognizing the $\sharp$ or beginning-of-tape marker and moving the input head) is one step more, so we have a bound of $2m + 2\log m$, which, for all $\varepsilon > 0$, is bounded by $(1+\varepsilon) \cdot 2m$ for m large enough. Recall that $m \leq n$ and that small n can always be handled by hard-wiring results into the machine, thus the bound can be met for all n.

Description of M_b: The improvement is obtained by putting more burden on the Turing machine's finite control. This machine handles not one bit of i at a time, but b (so that the case $b = 1$ gives M_1). For example, if $b = 3$, the machine will read 3 bits at a time from i. Suppose that they are 101, that is, 5 in binary; then the machine will copy to the other tape the positions whose index equals 5 modulo 8. Generally, M_b begins by splitting the w part of the input among tapes 2 and 3 just as in M_1. The first bit of i tells which tape will be, initially, t. A slight difference is that M_b keeps track of the position on the work-tapes modulo 2^b (that is, the b least significant bits of the position are maintained). Then the main loop is performed:

[1]The interested reader should be able to see that the beginning-of-tape marker will always be met on the tape that was initially t, and the $\sharp$ on the tape that was initially $\bar{t}$.

[2]If the input representing i ends earlier, the machine proceeds as if it were extended with zeros.

- Move the input head over the next b digits of i. Let $0 \le x < 2^b$ be the value represented by these bits. The position on Tape t is known modulo 2^b. Scanning the tape in a direction inverse to the last scan, copy only the positions that match x modulo 2^b to tape $\bar{t}$. The copying stops when either the beginning-of-tape marker or the $\sharp$ is encountered. Switch t with $\bar{t}$.

As above, the number of symbols written to tape $\bar{t}$ is maintained modulo 2^b, and, as long as it exceeds 2^b, is also rounded up to a multiple of 2^b by adding zeros, which achieves the effect of virtually extending the input w with zeros so that all divisions by 2^b are exact.

The main loop ends when no more than 2^b bits remain of w; these are read in one chunk. If r bits are read, the next $\lceil \log r \rceil$ bits of i are used to select the position according to which the machine accepts or rejects.

Analysis of M_b: The sum of the lengths of the tape scans is

$$m + \left\lceil \frac{m}{2} \right\rceil + \left\lceil \frac{m}{2^{1+b}} \right\rceil + \left\lceil \frac{m}{2^{1+2b}} \right\rceil + \cdots \le m + \frac{m}{2} \sum_{i=0}^{\infty} \frac{1}{2^{ib}} + \log m \le \left(1 + \frac{1}{2} \cdot \frac{1}{1-2^{-b}}\right) m + \log m.$$

The extra work at the end of each scan amounts to b additional steps, so the time complexity of M_b is bounded by $(1 + \frac{1}{2} \cdot \frac{1}{1-2^{-b}})m + (1+b)\log m$.

We complete the proof of the lemma by noting that for all $\varepsilon > 0$, a value of b can be chosen sufficiently large to make the running time less than $(1+\varepsilon) \cdot 1.5n$ for n large enough and consequently for all n. □

Recall the language $L_k = \{c_k(w)0i \mid w\sharp i \in L\}$, where $c_k(w)$ is the string obtained from w by padding it with zeros, if necessary, to a multiple of k bits, and inserting a 1 before every block of k consecutive bits.

Lemma 2.5. *For every $k > 0$, and every $\varepsilon > 0$, there exists a 1-2T1-DM with alphabet $\{0,1\}$ that decides L_k using at most $(1+\varepsilon) \cdot 1.5n$ steps for all n.*

Proof. We use a machine M'_b which is similar to M_b above, however it uses the first out of every $k+1$ consecutive tape cells for a mark, in accordance with the definition of L_k. Thus, the "sharp sign" is represented by a mark of 0. Maintaining the work-tapes in this representation means, for example, that after the first splitting, we have $\frac{k+1}{k} \cdot \frac{m}{2}$ cells written, rather than $m/2$; in general we have a factor of $(k+1)/k$ in the cost of each iteration, but this is also the ratio of $|c_k(w)|$ to $|w|$, so the running time in terms of the input length remains $(1+\varepsilon) \cdot 1.5n$ as before. □

2.2 Lower bound

As for the upper bound, we develop the lower bound proof first for the language L. We obtain a lower bound that matches our upper bound up to an arbitrary small factor (more precisely, arbitrarily close to 1), but under the assumption that the work-tape alphabet is binary, which is a mismatch with the upper bound (and also with the input alphabet). Therefore, we later extend the result to the languages L_k, where the input alphabet as well as the working-tape alphabet used in the upper-bound proof are binary.

The proof uses a sequence of lemmas that characterize the computation of a Turing machine that decides L, culminating in Corollary 2.14, which gives the lower bound. The main tool for the proof is an incompressibility argument, that is, an argument based on the Kolmogorov Complexity (also known as description complexity) of strings. This tool has been successfully used in many lower-bound proofs for all kinds of Turing machines and other computational models; for introductory material, as well as a survey of many applications of Kolmogorov Complexity, the reader is referred to the excellent textbook of Li and Vitányi [21]. In particular, we use conditional Kolmogorov complexity $C(x \mid y)$ [21, Section 3.2.6].

Roughly, the Kolmogorov complexity of a string x, denoted $C(x)$, is the length of a minimal string p such that a universal Turing machine produces x on input p; the conditional Kolmogorov complexity of string x given string y, $C(x \mid y)$, is the length of a minimimal string p such that a suitable kind of universal Turing machine produces x on input $\bar{y}p$, where $\bar{y}$ is a self-delimiting encoding of y.

We shall also need the notion of a Kolmogorov-random infinite binary sequence [21, Section 2.5]; the following two facts are all one needs to know about the latter subject, for our purpose.

1. Random infinite binary sequences exist.

2. Let x_ω be a random infinite binary sequence. Let $x_{1:n}$ be the n-bit prefix of x_ω. There is a constant n_0 such that for all $n \geq n_0$, $C(x_{1:n} \mid n) \geq n - 2\log n$, where $C(x \mid y)$ is the conditional Kolmogorov complexity of x given y.

For the rest of this analysis, fix a Turing Machine M, more specifically a 1-2T1-DM with work-tape alphabet $\{0,1\}$, that decides L, and a random infinite binary string x_ω.

While the problem of deciding membership in L is an offline problem (and those are typically more difficult to prove lower bounds for), the nature of our problem is such that it is "almost online," as it consists of data followed by a query on these data; moreover, in order to achieve a processing time below $2n$, some non-trivial processing of the data has to occur when the query is not yet known. Just copying the input to a work-tape and deferring all decisions to when the query is received will not work for going below $2n$, as the reader can surely verify.

We will analyze the computation of M on prefixes x_ω of arbitrary length, as well as computations which receive such a prefix of length m followed by a query, that is, a $\sharp$ followed by the representation of the query position i.

In the sequel, the meaning of *writing to a blank cell* means writing to a work-tape cell that has never been written to before (it may be convenient to imagine that such cells contain a special blank symbol, although to the machine it is indistinguishable from a 0). The *end* of a work-tape is the first such blank cell. Recall that our tapes are semi-infinite and the non-blank portion can only grow by writing at their current end.

For all $m > 0$, we refer by *Step m* to the transition where the machine first enters cell m on the input tape (thus there are two "clocks" that we are using—number of transitions, and number of steps). Define:

$P(m)$ the time used until Step m.

$Q(m)$ the worst-case cost of a query at Step m. This is defined as the worst-case number of transitions from this point until the end of the computation, if a $\sharp$ appears in cell m.

$H_i(m)$ the distance of head i ($\in \{2,3\}$) from the end of its tape at Step m (the value $H_i(m)$ is 0 if the head moves to a blank cell together with the input head moving into cell m).

$S_i(m)$ the *material space* on tape i ($i = 2,3$) at Step m. This is defined as the number of distinct tape cells c on tape i such that: (a) M has visited c before Step m, and (b) there exists at least one query at Step m such that c will be visited by M while processing this query.

$S(m)$ is $S_2(m) + S_3(m)$.

Note that, ex definitione, $P(m) \geq m$ and it is also easy to verify the following inequalities:

$$Q(m) \geq (1/2)S_i(m) \qquad \text{for } i \in \{2,3\} \tag{2.1}$$

(argument: $(1/2)S_i$ is the radius of head movements necessary to cover the material space on tape i.)

$$Q(m) \geq S_i(m) - H_i(m) \qquad \text{for } i \in \{2,3\} \tag{2.2}$$

(argument: Head i has H_i non-blank positions to its right, so even if they are all material, there will be $S_i - H_i$ material positions to its left, which some query must visit)

$$Q(m) \geq (1/2)S(m) - \max(H_2(m), H_3(m)). \tag{2.3}$$

(follows from (2.2) and the observation that one of the tapes must contain at least half of the material space.)

Lemma 2.6 (Sufficiency Lemma). *For a certain constant c_1, it holds for almost all m that $S(m) > m - c_1(\log m)$.*

Proof. Intuitively, we claim that $S(m)$ should suffice for storing approximately m bits. More precisely, we know, by randomness, that the Kolmogorov Complexity of $x_{1:m}$ is at least $m - 2\log m$, even given m. Suppose that for some m, $S(m) < m$.

If we have the complete configuration of M at Step m, except for the input tape, we can reconstruct the portion of the input read so far. This can be achieved by simulating M on all inputs on the form $x_{1:m}\sharp i$ with $i = 0, 1, \ldots, m-1$. In fact, the following information suffices for performing this task:

1. A description of M and its current state—a constant number of bits,
2. The values $S_i(m)$ (satisfying $S_2(m) + S_3(m) < m$),
3. For each tape, the contents of the $S_i(m)$ cells that constitute the material space,
4. The position of each head of M relative to the material section of its tape—this is $\log m$ bits.

This proves $C(x_{1:m} \mid m) \leq S(m) + c\log m$ for some constant c. By randomness, $C(x_{1:m} \mid m) \geq m - 2\log m$, so we have $S(m) > m - c_1 \log m$ for an appropriate constant c_1. □

In the remainder of the section, we shall repeatedly refer to the constant c_1 guaranteed to exist by the above lemma.

We will sometimes use $f(m) >_{a.e.} g(m)$ as a shorthand for "for almost all m, $f(m) > g(m)$."

Lemma 2.7 (Growing Tapes Lemma)**.** *Suppose that $Q(m) < (1/2)m - c_1 \log m$ infinitely often. Then both functions $S_i(m)$ are unbounded.*

Proof. Assume to the contrary that the material space on one of the tapes, say Tape 2, does not grow beyond some constant K. By the Sufficiency Lemma, we get (for almost all m) $S_3(m) > m - c_1(\log m) - K$, and hence, by (2.1),

$$Q(m) >_{a.e.} (1/2)(m - c_1(\log m) - K) = (1/2)m - (1/2)c_1(\log m) - (1/2)K >_{a.e.} (1/2)m - c_1(\log m)$$

A contradiction to the assumption. □

Lemma 2.8 (Growing Distance Lemma)**.** *Suppose that $Q(m) <_{a.e.} (1/2)m - c_1 \log m$. Then there is no constant K such that for infinitely many m, $H_2(m) + H_3(m) \le K$.*

Proof. Suppose that such a K exists. Then for infinitely many m we have, by (2.3) and the Sufficiency Lemma, $Q(m) > (1/2)S(m) - K > (1/2)(m - c_1(\log m)) - K$, which contradicts the lemma's assumption. □

Lemma 2.9 (Crossing Lemma)**.** *Suppose that $Q(m) < (1/2)m - c_1 \log m$ infinitely often. Then there are infinitely many transitions where the heads are located at the same (or almost the same) distance from the ends of their tapes (where* almost *refers to a difference bounded by 1).*

Proof. By the Growing Tapes lemma, both tapes' contents grow infinitely. Thus, there are infinitely many transitions $t_1^{(2)} < t_2^{(2)} < t_3^{(2)} < \cdots$, where Head 2 is at the end of its tape, and similarly for Head 3. Fix some i, and let j be the smallest index so that $t_j^{(3)} > t_i^{(2)}$. Thus from transition $t_i^{(2)}$ to transition $t_j^{(3)}$, Head 3 moved from its position at transition $t_i^{(2)}$ to the end of its tape, while Head 2 moved, possibly, away from the end of its tape. Clearly, at some point between transition $t_i^{(2)}$ to transition $t_j^{(3)}$, their distances from the respective ends must have been equal, or at most 1 apart (if we draw graphs of the distance of each head from its tape's end, the graphs will cross). □

Definition 2.10. We denote by m_t for the number of the first *step* following transition t, that is,

$$m_t = \min\{m \in \mathbb{N} | P(m) > t\} .$$

Lemma 2.11 (Writing Lemma)**.** *Suppose that $P(m) + Q(m) <_{a.e.} (1.5)m - c_1(\log m)$. Then there are only finitely many transitions where both heads write to a blank cell.*

Proof. Assume to the contrary that there are infinitely many transitions where both heads write to a blank cell. Let t be the index of such a transition. Consider the effect of a query in Step m_t. For t large enough, the Sufficiency Lemma gives $S(m_t) > m_t - c_1(\log m_t)$, and hence, by (2.3),

$$Q(m_t) > (1/2)(m_t - c_1(\log m_t)) - \max(H_2(m_t), H_3(m_t)).$$

Recall that $P(m_t)$ is the number of transitions until Step m_t; these can be divided into three sections: (1) The transitions until Step $m_t - 1$ (surely at least $m_t - 1$ of these); (2) The transitions from Step $m_t - 1$ to Transition t (maybe 0 of these); (3) The transitions from Transition t until Step m_t. Note that at transition t, both heads wrote a blank cell, while when Step m_t is reached, they are positioned $H_i(m_t)$ cells away from the respective end of tape. Hence, we deduce that

$$P(m_t) \geq m_t - 1 + \max(H_2(m_t), H_3(m_t))$$

Summing the last two equations,

$$P(m_t) + Q(m_t) > (1/2)(m_t - c_1(\log m_t)) + m_t - 1 \geq (1.5)m_t - c_1(\log m_t), \tag{2.4}$$

and there will be infinitely many such m_t, in contradiction to the lemma's assumption. □

A transition in which one or both of the heads write into a blank tape cell will be called *a neograph*.

Corollary 2.12. *Suppose that $P(m) + Q(m) <_{a.e.} (1.5)m - c_1(\log m)$. Then there is a constant K such that for all t, the number of neographs during the first t transitions is at least $m_t - c_1(\log m_t) - K$.*

Proof. This is an easy corollary of the Writing Lemma and the Sufficiency Lemma; K accounts for the finite number of transitions that write two blank cells at once, as well as for the finite number of transitions in which $S(m)$ falls below the lower bound $m - c_1(\log m)$. □

Lemma 2.13. *For infinitely many values of m, $P(m) + Q(m) \geq 1.5m - 2c_1 \log m$.*

Proof. Assume, in contradiction, that $P(m) + Q(m) < (1.5)m - 2c_1 \log m$ for almost all m. Then Lemmas 2.6–2.11 and Corollary 2.12 apply. Let t be any of the transition numbers established by the Crossing Lemma. For convenience, assume that $H_2(m_t) \leq H_3(m_t)$. Let D be the distance of Head 3 to the end of its tape at transition t (the other one is at distance $D-1$, D or $D+1$).

By Corollary 2.12, there is a constant K such that the number of neographs up to transition t is at least $m_t - c_1(\log m_t) - K$. After the last neograph before t, the head that did the writing had to move away about D positions from the end; hence we have

$$t \geq m_t - c_1(\log m_t) - K + D.$$

By definition, at Step m_t, Head 3 is $H_3(m_t)$ positions away from the end of its tape; hence between transition t and the reading of the next (m_t's) input, it moves a distance of $|H_3(m_t) - D|$. Thus

$$P(m_t) \geq t + |H_3(m_t) - D| \geq m_t - c_1(\log m_t) - K + H_3(m_t)\,.$$

Since

$$\begin{aligned} Q(m_t) &\geq (1/2)S(m_t) - \max(H_2(m_t), H_3(m_t)) \\ &> (1/2)(m_t - c_1(\log m_t)) - H_3(m_t), \end{aligned}$$

we get

$$P(m_t) + Q(m_t) > 1.5(m_t - c_1(\log m_t)) - K$$

which contradicts the initial assumption. □

Corollary 2.14. *For every $\varepsilon > 0$, any 1-2T1-DM with work-tape alphabet $\{0,1\}$ deciding L must use at least $(1-\varepsilon)\cdot 1.5n$ steps for infinitely many values of n.*

Proof. Choose k sufficiently large that $(2c_1 \log k)/1.5k \le \varepsilon$. Then, for all $n \ge k$, we have $(1-\varepsilon)\cdot 1.5n \le 1.5n - 2c_1 \log n$, so by Lemma 2.13, the running time of the machine M is bounded below by $(1-\varepsilon)\cdot 1.5n$ for infinitely many values of n. □

We have found it somewhat unnatural to restrict the work-tape alphabet to an alphabet smaller than the input alphabet which is $\{0,1,\sharp\}$. Therefore, we also prove a result for the language L_k, whose definition disposes of the $\sharp$ symbol. Recall that $L_k = \{c_k(w)0i \mid w\sharp i \in L\}$, where $c_k(w)$ is the string obtained from w by padding it with zeros, if necessary, to a multiple of k bits, and inserting a 1 before every block of k consecutive bits.

Lemma 2.15. *For every $\varepsilon > 0$, there is a k such that any 1-2T1-DM with alphabet $\{0,1\}$ deciding L_k must use at least $(1-\varepsilon)\cdot 1.5n$ steps for infinitely many values of n.*

Proof. We use the same argument, considering input taken out of x_ω and formatted into blocks of k bits delimited by marker bits. Let m denote the length of the prefix of x_ω. It is convenient to concentrate on prefixes of length divisible by k, so they constitute a whole number of blocks and their encoding is of length $(1+1/k)m$ exactly. This has no effect on the claims made up to Lemma 2.11. Here, the argument leading to Eq. (2.4) has to be amended: the "previous step" is not Step $m-1$, but Step $m-k$, so the number of input-head movements up to that step is $(1+1/k)(m-k)$. The claim $P(m) \ge m-1+\max(H_2(m),H_3(m))$ changes into $P(m) \ge (1+1/k)(m-k)+\max(H_2(m),H_3(m))$ and Eq. (2.4) changes to

$$P(m)+Q(m) > (1/2)(m-c_1(\log m)) + (1+1/k)(m-k) \ge_{a.e.} (1.5+1/k)m - c_1(\log m). \tag{2.5}$$

Therefore, the statement of Lemma 2.11 is now the following:

Lemma 2.16 (Writing Lemma, amended)**.** *Suppose that $P(m)+Q(m) < (1.5+1/k)m - c_1(\log m)$ for almost all m. Then it is also true in almost all transitions, that when one of the heads writes to a blank cell, the other does not.*

This change propagates to the sequel, so that Lemma 2.13 changes to:

$$P(m)+Q(m) \ge_{a.e.} (1.5+1/k)m - 2c_1 \log m\,.$$

Expressed in terms of the input length length $n = (1+1/k)m$, our lower bound becomes (after some calculation) $(1.5 - 1/(2k+2))n - 2c_1(\log n)$. We deduce Lemma 2.15 by choosing $k \ge 1/(2\varepsilon)$. □

2.3 Changing the work-tape alphabet

We can generalize our result to an alphabet Γ of more than 2 letters in a simple way: the input w becomes now any word over Γ. The language L^Γ consists of words $w\sharp si$ where s is a single symbol, i is a binary number as before, and $\sharp si$ is in L^Γ if and only if the ith symbol of w (modulo a power of two, as before) is an s. The language L_k^Γ is obtained using the same encoding as for L_k. It is quite obvious that the algorithm that furnishes our upper bound still works, and the lower bound proofs do too, by considering x_ω to be a random infinite string over Γ.

3 Linear speedup fails for models with efficient self-interpreters

We now introduce a general class of models of computation characterized by having highly efficient self-interpretation and demonstrate that this too defies general linear speedup.

3.1 Preliminaries

We base our definitions on the standard Blum complexity measures [4, 40], with one subtlety: we want to consider machine models that may employ a variety of data representations (integers, strings, trees…). Defining suitable notions of computability (let alone complexity) that will be consistent across a variety of data representations is a subtle endeavour; see [17, 5, 7]; our choice in this paper is to explicitly state just the assumptions that are important to our purpose, and avoid any further discussion of their ramifications, and also to simplify notation and definitions as much as we can.

Definition 3.1. A *data representation* is an infinite, countable domain $\mathbb{D}$, endowed with an embedding of $\mathbb{N} = \{0,1,2,\ldots\}$ in $\mathbb{D}$. The elements of $\mathbb{D}$ that represent, in this way, elements of $\mathbb{N}$ are called *numerals*. By $\mathrm{int}(x)$ we denote the number corresponding to numeral x. To simplify notation, we may use this embedding tacitly, identifying a number with its numeral; so we use, e.g., 0 for the numeral representing zero.

We further assume:

- A *size function* $|\cdot| : \mathbb{D} \longrightarrow \mathbb{N}$.
- A *tupling function* $\tau : \mathbb{D}^* \longrightarrow \mathbb{D}$. The value $\tau(x_1,\ldots,x_n)$ may also be written simply as $(x_1,\ldots,x_n)$. We assume that $|(x_1,\ldots,x_n)| = \mathcal{O}(|x_1| + \cdots + |x_n|)$.

Definition 3.2. A *programming system* consists of a data representation $\mathbb{D}$ and an indexed set $\{\phi_i\}_{i\in\mathbb{N}}$ of partial functions $\phi_i : \mathbb{D} \longrightarrow \mathbb{D}$; the indices i are called programs and the functions $\{\phi_i\}$, computable (in this system). We introduce the *typing notation* $\phi_i : \mathbb{D} \to \mathbb{N}$ to assert that the value of $\phi_i(x)$ (for any x such that it is defined) is a numeral.

Definition 3.3. A *model of computation with a complexity measure* is a pair $(\{\phi_i\},\{\Phi_i\})$ where $\{\phi_i\}$ is a programming system and for all $i \in \mathbb{N}$, $\Phi_i : \mathbb{D} \longrightarrow \mathbb{N}$ satisfies $dom(\Phi_i) = dom(\phi_i)$ (i. e., the functions are defined for the same inputs).

According to Blum [4], an (abstract) complexity measure has to satisfy the additional assumption that the predicate $\lambda i,x,b : \Phi_i(x) = b$ is decidable. We do not directly use this assumption, but our Assumption 1 below implies it (in reasonable computational models, including admissible enumerations of the recursive functions as defined in [34]).

Definition 3.4. A function $T : \mathbb{N} \longrightarrow \mathbb{N}$ is said to be $(\{\phi_i\},\{\Phi_i\})$-constructible if there is some $j \in \mathbb{N}$ such that $\phi_j : \mathbb{D} \longrightarrow \mathbb{N}$, $\forall x : \mathrm{int}(\phi_j(x)) = T(|x|)$ and $\Phi_j(x) \in \mathcal{O}(\lambda x.T(|x|))$.

When the programming model $\{\phi_i\}$ and the complexity measure $\{\Phi_i\}$ are given in the context, we will simply refer to $(\{\phi_i\},\{\Phi_i\})$-constructible functions as *constructible*.

3.2 Models with program-independent interpretation overhead

We now move on to a new definition. We are interested in models of computation with *efficient step-by-step interpretation*. By "efficient" we mean that the interpretation overhead should be a program-*independent* constant. Step-by-step interpretation is the kind of program interpretation that runs in lockstep with the interpreted program (the normal way to write an interpreter). In our context, the important consequence of step-by-step interpretation is that the interpreter can be equipped with a *clock* that can shut computation down after resource usage has reached a certain threshold.

We say that such a model of computation is "PICSTI" because it has Program-Independent Constant-overhead STep-by-step Interpretation. More precisely, we introduce four assumptions, of which the first one is the essential, while the rest are of a more technical nature.

Assumption 1 (Efficient interpretation)**.** There exists a program U such that

$$\begin{aligned} \forall t \in \mathbb{N} \forall x \in \mathbb{N} \forall y \in \mathbb{D} : \phi_U(t,x,y) &= \begin{cases} 0 & \text{when } \Phi_x(y) > t \\ \phi_x(y) & \text{otherwise} \end{cases}, \\ \exists a,b > 0 : \Phi_U &\leq a + b(t + |x| + |y|) \end{aligned}$$

Intuitively, the program U is a self-interpreter (also known as a *U*niversal program) that runs program x on y, shutting down the computation when the resource usage reaches t, in which case 0 is returned. If the simulation of running x on y halts with a result before the threshold is reached, U returns that result. The complexity of U is then assumed to be linear in the amount of resources, t, and the size of the interpreted program and its input. Intuitively, the constant b bounds the interpretation overhead of U (roughtly the time to simulate one step of program x).

Assumption 2 (Diagonalization)**.** For every $i \in \mathbb{N}$ there is a program $not(i) \in \mathbb{N}$ such that

$$\begin{aligned} \forall x \in dom_{\phi_i} : \phi_{not(i)}(x) &\neq \phi_i(x) \\ \forall x \in dom_{\phi_i} : \phi_{not(i)}(x) &\in \{0,1\} \\ \Phi_{not(i)} &\in \mathcal{O}(\Phi_i + 1) \end{aligned}$$

Assumption 3 (Efficient tupling)**.** For every $i_1,\ldots,i_k \in \mathbb{N}$ there is a program $tuple(i_1,\ldots,i_k) \in \mathbb{N}$ that applies all of $i_1,\ldots,i_k \in \mathbb{N}$ to its input (sequentially or in parallel) and creates a tuple of the outputs without using significant overhead. More precisely:

$$\begin{aligned} \forall x : \phi_{tuple(i_1,\ldots,i_k)}(x) &= (\phi_{i_1}(x),\ldots,\phi_{i_k}(x)) \\ \Phi_{tuple(i_1,\ldots,i_k)} &\in \mathcal{O}(\Phi_{i_1} + \cdots + \Phi_{i_k} + \lambda x.|x|) \end{aligned}$$

As an example of the Assumption 3, consider two programs i and i'. A typical implementation of $tuple(i,i')$ applied to input x would simply

- Apply program i to a copy of input x, obtaining the value $\phi_i(x)$, which is then stored.
- Apply program i' to a copy of input x, obtaining the value $\phi_{i'}(x)$, which is also stored.
- Construct and output the tuple $(\phi_i(x), \phi_{i'}(x))$.

Assumption 4 (Composition [3])**.** For every $i, j \in \mathbb{N}$ there is a $h(i, j) \in \mathbb{N}$ such that

$$\begin{array}{rcl} \phi_{h(i,j)} & = & \phi_i \circ \phi_j \\ \Phi_{h(i,j)} & \in & \mathcal{O}(\Phi_i \circ \phi_j + \Phi_j) \end{array}$$

3.3 Remarks on concrete computation models

3.3.1 Models which are PICSTI under the unit-cost time measure

There are several different computational models in the literature satisfying the above assumptions. For example, Unit-cost RAMs and Jones' programming languages I and F [17] are all PICSTI with the associated time measures. Mogensen proved that pure, untyped lambda calculus under several different cost models is PICSTI if β-reduction is restricted to reduction to weak head normal form under call-by-name, call-by-value, or call-by-need strategies [24]; Mogensen also showed that the existence of efficient self-interpreters is highly sensitive to the cost model used. As remarked in [2], the Storage Modification Machine introduced by Schönhage [39], when its alphabet is fixed, has an efficient self-interpreter under the time measure—in our terminology, it is PICSTI.

The input and output of I and F programs in [17] are binary trees, and in lambda calculus the input and output are similarly binary trees in the form of lambda terms. The other models are defined with string input and output.

To justify our claims, it is easy to see that each model satisfies Assumptions 2, 3 and 4. To see that Assumption 1 is also satisfied, note that all of these have a universal machine ("interpreter") with constant-time overhead that is independent of the program being interpreted, and, moreover, the interpreters all simulate one step of the interpreted program at a time, so that on its completion, the interpreter has the interpreted program's output stored in memory. Using such an interpreter, we can construct a program that, on input t, x, y:

1. Stores the threshold t in a counter.
2. Using the interpreter, simulates one step of x applied to y at the time, decrementing the counter for each step simulated.
3. At each step, if the counter has reached 0, the program outputs 0 and halts.
4. If no steps remain to be simulated, the result $\phi_x(y)$ would be in memory and the program is able to output that.

[3]Definition 12, [1], weakened.

In each of the models mentioned above these required operations are contant-time: storing the value of a numeral in a counter, decrementing a counter, checking for equality to 0, and concatenating 0 to a string (to completely justify it, the form of numerals has to be specified; for our purposes, several common choices, such as unary and binary representations, are all suitable).

3.3.2 Turing-machine space

For a Turing machine with $k \geq 1$ work tapes, write $S_i(n)$ for the maximum number of distinct space cells scanned on tape i, $1 \leq i \leq k$ in computation with input of size n. There are two standard definitions of the space complexity $S(n)$ of such a Turing machine, namely $S_\Sigma(n) = \sum_{i=1}^{k} S_i(n)$ and $S_{\max}(n) = \max_{i=1}^{k} S_i(n)$. It is immediate that $S_{\max}(n) \leq S_\Sigma(n) \leq k \cdot S_{\max}(n)$, whence S_Σ and $S_{\max}$ are asymptotically equivalent for the class of multi-tape Turing machines.

The following folklore result is easily proved.

Proposition 3.5. *The class of Turing-machines with an unlimited number of work tapes and alphabet* $\{0,1\}$*, with the space complexity measure* S_Σ*, is PICSTI.*

As above, the main issue is the efficiency of a universal machine, which has to be a machine with a fixed number k of work tapes that can simulate a machine with $k' > k$ work tapes. Here the measurement of space usage by S_Σ is crucial, as under $S_{\max}$ we do have a linear speedup phenomenon (so, according to our proof below, it cannot be PICSTI). In fact, for any given machine M with $S_{\max}(n) > 2n$, we can reduce $S_{\max}$ by half by doubling the number of work tapes. The symbols of each original work tape are distributed among two tapes of the new machine in an even-odd manner, so that it is easy to keep track of where the current symbol is.

3.4 Linear speedup fails for PICSTI models

We now proceed to show that, for any PICSTI model, there are constant-factor hierarchies, i. e., linear speedup *fails* in a strong sense. The proof generalizes similar results for concrete models [16, 2, 3] recast in an abstract setting.

Theorem 3.6. *Consider a model* $(\{\phi_i\},\{\Phi_i\})$ *satisfying Assumptions 1, 2, 3, and 4 and an* $\mathrm{id} \in \mathbb{N}$ *for which* $\phi_{id} = \lambda x.x$*. For any constructible function* $f : \mathbb{N} \longrightarrow \mathbb{N}$ *with* $\forall x : f(|x|) > |x| \wedge f(|x|) > \Phi_{id}(x)$*, there is a 0-1-valued function which is computable using resources* $\mathcal{O}(f(|\cdot|))$*, but is not computable using resources at most* $f(|\cdot|)$*.*

Proof. As f is constructible, there is an $F \in \mathbb{N}$ with $\mathrm{int}(\phi_F(x)) = f(|x|)$ and $\Phi_F \in \mathcal{O}(\lambda x.f(|x|))$. Recall that U is our three-parameter universal program, the first parameter being the resource limit after which interpretation is shut down. Recall also that h implements functional composition. Define $d \in \mathbb{N}$ by

$$d = not(h(U, tuple(F, id, id))))$$

Intuitively the program d creates a tuple with 3 copies of its input, applies F to the first copy, then applies U to the updated 3-tuple, and finally "negates" the result from U.

The 0-1-valued function in the Theorem is ϕ_d.

We know that this function can be computed (by d) using resources

$$\begin{array}{lll}
\Phi_d & \in & \mathcal{O}(\Phi_{h(U,tuple(F,id,id)))}) & \text{By definition of } d \text{ and Assumption 2} \\
& \subseteq & \mathcal{O}(\Phi_U \circ \phi_{tuple(F,id,id)} + \Phi_{tuple(F,id,id)}) & \text{By Assumption 4} \\
& \subset & \mathcal{O}(\mathrm{int} \circ \phi_F + \lambda x.|x| + \Phi_{tuple(F,id,id)}) & \text{By Assumption 1 and semantics of } tuple \\
& = & \mathcal{O}(\mathrm{int} \circ \phi_F + \Phi_{tuple(F,id,id)}) & \text{By the assumptions on } F \text{ and } id \\
& \subseteq & \mathcal{O}(\mathrm{int} \circ \phi_F + \Phi_F + \Phi_{id} + \Phi_{id} + \lambda x.|x|) & \text{By Assumption 3} \\
& = & \mathcal{O}(\mathrm{int} \circ \phi_F + \Phi_F) & \text{By the assumptions on } id \text{ and } f \\
& = & \mathcal{O}(\lambda x.f(|x|)) & \text{By definition of } F
\end{array}$$

It remains to prove that ϕ_d cannot be computed with resources less than f. Assume then, for the sake of contradiction, the existence of a program l with $\phi_l = \phi_d$ but for which $\Phi_l(x) \le f(|x|)$ for all x. It follows that $\Phi_l(l) \le f(|l|)$ which means

$$\begin{array}{lll}
\phi_l(l) & = & \phi_U(f(|l|), l, l) & \text{By Assumption 1} \\
& \neq & \phi_d(l) & \text{By the definition of } d \\
& = & \phi_l(l) & \text{By the assumption on } l
\end{array}$$

which is a contradiction. □

Thus, the linear speedup theorem fails for reasonably growing, constructible functions. The contrapositive of Theorem 3.6 is itself worth noting: If a model of computation *does* have linear speedup, it is *not* PICSTI. Thus, for the particular case of Turing machines and time complexity, we have:

Corollary 3.7. *Let $\{\phi_i\}$ be any of the machine classes below, and let $\{\Phi_i\}$ be its usual time resource measure. Then $(\{\phi_i\}, \{\Phi_i\})$ is not PICSTI (hence, the class does not have an efficient interpreter satisfying Assumption 1).*

- *The class of Turing machines with fixed number of tapes, but unlimited alphabet size.*
- *The class of Turing machines with unlimited number of tapes, but fixed alphabet size.*
- *The class of Turing machines with unlimited number of tapes and alphabet size.*

Proof. Combine Theorem 3.6 with the speedup theorem for these classes (e. g., [27, Thm. 2.2]); note that Assumptions 2–4 are satisfied by these models. □

Another contrast is with Fürer's classic result [9], which shows that we can decrease the asymptotic program-dependent overhead of self-interpretation in multitape Turing machines to a very slow-growing function. Corollary 3.7 shows that the overhead *cannot* be bounded by a (program-*in*dependent) constant. This can be interpreted as follows: If the given machine class does have a linear speed up, it cannot have an efficient universal machine.

4 Conclusion and open problems

We have proved that the linear speedup theorem does not hold for Turing machine time without increasing alphabet size or number of tapes. In addition, we have defined a class, PICSTI, of models of computation in which linear speedup does not hold and demonstrated how several existing models are elements of this class.

We leave the reader with the following open problems (on some of which we dare to pose conjectures).

1. Our result for Turing machines (Theorem 2.1) can perhaps be strengthened. For example, the lower bound assumed a one-way input tape. We conjecture that the lower bound also holds for machines where the input tape is two-way.

2. A more important weakness of Theorem 2.1 is that it only gives one particular counter-example to speedup, with a rather low time complexity. Theoretically, it might still be possible that linear speedup holds for machines whose time complexity is higher. We conjecture that this is not the case, and that there are problems with arbitrarily high time complexity that cannot be sped up.

3. We find it interesting that Theorem 2.1 does leave a gap between the upper and lower bounds, even though the gap can be made arbitrarily small. Moreover, by careful inspection of our upper bound, the reader would be able to see that a machine implementing our algorithm can always be improved by a very small constant factor (using additional states).

   We pose the following (somewhat vague) conjecture: *every* decision problem A whose complexity on a multi-tape Turing machine is within certain limits (for example, higher than $1.5n$?) has this form of weak linear speedup, that is, for every k-tape machine deciding it in worst-case time (space) $f(n)$ there is an $\varepsilon > 0$ and a k-tape machine M', of the same alphabet, that decides A in time (space) bounded above by $f(n)/(1+\varepsilon)$.

   Observe that this conjecture is at odds with neither the results of the present paper, nor the previous open problems: ε is dependent on the machine M, hence may become smaller the faster M is. Thus, linear speedup may still fail to hold even if the conjecture holds.

4. As noted in Section 3.3.1, the Storage Modification Machine of Schönhage [39], with a fixed alphabet, is PICSTI under the time measure, hence has no linear speedup. Given the result of Hühne [13] it seems reasonable that even if the alphabet were not fixed, there would still be no speedup. But this problem is, to our best knowledge, still open.

5. Finally, we pose a question regarding a version of the linear speedup problem for random access machines. Some of the algorithms literature considers unit-cost RAMs of a finite word length w that is related to the input size n, for example $w = \Theta(\log n)$ is a common choice. In order to avoid difficulties with, e.g., composing algorithms, a more specific expression for w, like $w = 2\log n$, is not used. Question: Is it true that for every program p recognizing a language (say of binary strings) in time $t(n)$ using word length $w = \Theta(f(n))$ there is a program q using word length $\Theta(f(n))$ (but possibly larger than used by p) and running in $cn + (1/2)t(n)$ time (for some constant c)? Intuitively, this question has to do with whether every program can be sped up using hardware with increased bit-parallelism—an issue quite related to the practice of computer systems.

Acknowledgments

The authors wish to thank Neil D. Jones, Andreas Milton Maniotis, Hunter Monroe, Holger Petersen and Simon Yoffe for useful comments that have helped us improve this paper. We also wish to thank Janos Simon and the anonymous referees for their insightful and thorough comments. Part of this work was done while Amir Ben-Amram was visiting The Department of Computer Science, University of Copenhagen (DIKU).

References

[1] A. Asperti. The intensional content of Rice's theorem. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '08)*, pages 113–119. The ACM Press, 2008. 17

[2] A. Ben-Amram and N. Jones. Computational complexity via programming languages: Constant factors do matter. *Acta Informatica*, 37(2):83–120, 2000. 5, 17, 18

[3] A. Blass and Y. Gurevich. The linear time hierarchy for abstract state machines and RAMs. *Journal of Universal Computer Science*, 3(4):247–278, 1997. 5, 18

[4] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the Association for Computing Machinery*, 14(2):332–336, 1967. 3, 15

[5] U. Boker and N. Dershowitz. The influence of domain interpretations on computational models. *Applied Mathematics and Computation*, 215(4):1323 – 1339, 2009. 15

[6] S. A. Cook and S. O. Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969. 4

[7] N. Dershowitz and E. Falkovich. Honest universality. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1971):3340–3348, 2012. 15

[8] P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Theory of Computing Systems*, 2:265–283, 1968. 3, 4

[9] M. Fürer. The tight deterministic time hierarchy. In *Proceedings of the 14th ACM Symposium on the Theory of Computing (STOC '82)*, pages 8–16. The ACM Press, 1982. 19

[10] V. Geffert. A speed-up theorem without tape compression. *Theoretical Computer Science*, 118(1):49–65, 1993. 5

[11] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. 2

[12] F. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, 8(5):553–578, 1965. 5

[13] M. Hühne. Linear speed-up does not hold for Turing machines with tree storages. *Information Processing Letters*, 47:313–318, 1993. 4, 20

[14] O. H. Ibarra and S. K. Sahni. Hierarchies of turing machines with restricted tape alphabet size. *Journal of Computer and System Sciences*, 11(1):56–67, Aug. 1975. 4

[15] T. Jiang, J. I. Seiferas, and P. M. B. Vitányi. Two heads are better than two tapes. *Journal of the ACM*, 44(2):237–256, Mar. 1997. note erratum in JACM 44:4. 3

[16] N. Jones. Constant time factors do matter. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 602–611, 1993. 5, 18

[17] N. Jones. *Computability and Complexity from a Programming Perspective*. The MIT Press, 1997. 2, 15, 17

[18] K.-I. Ko and D.-Z. Du. *Theory of Computational Complexity*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc., New York, 2000. 2

[19] K. Kobayashi. On the structure of one-tape nondeterministic Turing machine time hierarchy. *Theoretical Computer Science*, 40(2-3):175–193, 1985. 5

[20] L. A. Levin. Computational complexity of functions. *Theoretical Computer Science*, 157(2):267–271, May 1996. 4

[21] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer-Verlag, 1997. 10

[22] J. Mazoyer and N. Reimen. A linear speed-up theorem for cellular automata. *Theoretical Computer Science*, 101:59–98, 1992. 3

[23] A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *Journal of Symbolic Logic*, 37(1):55–68, 1972. 3

[24] T. Æ. Mogensen. Linear-time self-interpretation of the pure lambda calculus. *Higher-Order and Symbolic Computation*, 13(3):217–237, 2000. 17

[25] H. Monroe. Are there natural problems with speedup? *Bulletin of the European Association for Theoretical Computer Science*, 94:212–220, 2008. 3

[26] H. Monroe. Speedup for natural problems and noncomputability. *Theoretical Computer Science*, 412(4-5):478–481, 2011. 3

[27] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. 2, 6, 19

[28] R. Paturi, J. I. Seiferas, J. Simon, and R. E. Newman-Wolfe. Milking the Aanderaa argument. *Information and Computation*, 88(1):88–104, Sept. 1990. 3

[29] W. J. Paul. On heads versus tapes. *Theoretical Computer Science*, 28(1–2):1–12, Jan. 1984. 3

[30] W. J. Paul, J. I. Seiferas, and J. Simon. An information-theoretic approach to time bounds for on-line computation. *Journal of Computer and System Sciences*, 23(2):108 – 126, 1981. 3

[31] H. Petersen. Bounded counter languages. In *DCFS 2012, Descriptional Complexity of Formal Systems, Braga, Portugal*, Lecture Notes in Computer Science. Springer, 2012. 4, 5

[32] K. Regan. Linear speed-up, information vicinity, and finite-state machines. In *Proceedings of the IFIP 13th World Computer Congress*, volume 1, pages 609–614, 1994. 4

[33] K. W. Regan. Linear time and memory-efficient computation. *SIAM Journal on Computing*, 25(1):133–168, Feb. 1996. 3

[34] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. The MIT Press, paperback edition, 1987. 15

[35] E. Rose. Linear-time hierarchies for a functional language machine model. *Science of Computer Programming*, 32(1–3):109–143, 1998. 5

[36] S. Šakuov. Linear acceleration of the operating time of single-tape Turing machines. *Soviet math. Dokl.*, 17(5):1407–1409, 1976. 3

[37] S. Šakuov. Fast Turing computations and their linear speedup. *Soviet Math. Dokl.*, 18(5), 1977. 3

[38] C. Schnorr and G. Stumpf. A characterization of complexity sequences. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 21:47–56, 1975. 3

[39] A. Schönhage. Storage modification machines. *SIAM J. Comput.*, 9:492–508, 1980. 17, 20

[40] J. Seiferas. Machine-independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 163–186. Elsevier, 1990. 15

[41] J. I. Seiferas. Relating refined space complexity classes. *Journal of Computer and Systems Sciences*, 14:100–129, 1977. 4

[42] J. I. Seiferas. Techniques for separating space complexity classes. *Journal of Computer and Systems Sciences*, 14:73–99, 1977. 4

[43] J. I. Seiferas and A. R. Meyer. Characterization of realizable space complexities. *Annals of Pure and Applied Logic*, 73(2):171–190, 1995. 4

[44] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2006. 2

[45] R. Stearns, J. Hartmanis, and P. Lewis. Hierarchies of memory limited computations. In *IEEE Conference Record on Switching Circuit Theory and Logical Design*, pages 179–190, 1965. 4

[46] I. Sudborough and A. Zalcberg. On families of languages defined by time-bounded random access machines. *SIAM Journal of Computing*, 5(2):217–230, 1976. 4

[47] S. Žák. A Turing machine time hierarchy (note). *Theoretical Computer Science*, 26(3):327–333, 1983. 4

[48] K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its Applications. D. Reidel Publishing Company, 1986. 2, 6

AUTHORS

Niels Christensen, Issuu (`www.issuu.com`)
mrnc@diku.dk

Jakob Grue Simonsen, Department of Computer Science, University of Copenhagen (DIKU)
simonsen@diku.dk
`http://www.diku.dk/~simonsen`

Amir M. Ben-Amram, School of Computer Science, Academic College of Tel-Aviv Yafo
amirben@mta.ac.il
`http://www2.mta.ac.il/~amirben`