Linear Speed-Up, Information Vicinity, and Finite-State Machines

Kenneth W. Regan[a]

[a]Department of Computer Science, University at Buffalo
226 Bell Hall, Buffalo NY 14260-2000, USA

Connections are shown between two properties of a machine model: *linear speed-up* and *polynomial vicinity*. In the context of the author's *Block Move* (BM) model, these relate to: "How long does it take to simulate a finite transducer S on a given input z?" This question is related to the century-old problem of finding economical representations for finite groups. Under some cost measures for computing $S(z)$, the BM enjoys the linear speed-up property, but under more-realistic measures, and subject to a reasonable but unproved hypothesis, it has the antithetical property of a *constant-factor time hierarchy*.

Keyword Codes: F.1.1; F.2.2
Keywords: Models of Computation; Nonnumerical Algorithms and Problems.

1. Speed-Up and Vicinity

Hartmanis and Stearns [7] proved that the standard multitape Turing machine (TM) model enjoys the following property, for which we give a general statement:

Definition 1.1. A machine model $\mathcal{M}$ has the *linear speed-up property* if there exists $k_0 > 0$ such that for every $\epsilon > 0$ and $t(n)$ time-bounded $\mathcal{M}$-machine M, there exists a $\mathcal{M}$-machine M' that computes the same function as M and runs in time $\epsilon t(n) + k_0 n$.

The term $k_0 n$ accounts for an initial step in which the input $x \in \Sigma^n$ given to M is "compressed" to length ϵn by being rewritten over an alphabet Σ' of size more than $|\Sigma|^{1/\epsilon}$. Their proof for Turing machines has $k_0 = 1$, and in the case $t(n) = O(n)$, the machine M' runs in time $(1 + \epsilon)n$. It also makes M' simulate M *on-line*, meaning that (after the initial step) every $\lceil \epsilon t \rceil$ moves of M' simulate t moves of M. Hühne [10] gives a formal definition of on-line speedup.

The "compression" is really an illusion in the TM proof, since the alphabet size expands greatly, and the result can be criticized as an artifact of changeable units. However, it connects to a concrete and important general property of machine models. Feldman and Shapiro [5] informally define the *vicinity* of a machine to be the maximum number of data bits on which the evolution of the computation over the next t time units can possibly depend. Their definition applies not only to conventional models in which data is *stored* between steps, but also to machines they describe in which data is propagated in free space. We concentrate on the former case, and suppose that data is stored in cells of machine-dependent size C. Let a "semi-ID" H of a machine M specify the current locations of storage read heads, but not the actual content of memory or current program state. Say that a cell is *in reach of H within t steps* if there is a configuration I extending H such that the machine started in I scans the cell within t steps.

Definition 1.2. Given a machine model $\mathcal{M}$ and $t > 0$, let $u(t)$ be the maximum over all semi-IDs H of machines in $\mathcal{M}$ of the number of cells in reach of H within t steps. Then we say that $\mathcal{M}$ has *vicinity of order* $u(t)$, and that an individual machine M in $\mathcal{M}$, with capacity constant C, has *vicinity* $v(t) = Cu(t)$.

This definition does not try to take account of everything, but suffices for this paper. All of the popular RAM models, including the *log-cost* RAM of Cook and Reckhow [3] and the *polynomially compact* RAM of Grandjean and Robson [6], have exponential vicinity. So do the pointer models of Schönhage [14] and Jones [11], and the Turing machines with tree-structured tapes considered by Hühne [10]. Turing machines with k tapes have $u(t) = k(2t + 1)$, and those with d-dimensional tapes have $u(t) = \Theta(t^d)$. Feldman and Shapiro argue that to be realistic, a machine model should have polynomial vicinity, indeed at most cubic vicinity under physics as we know it today.

We contend that *every machine model $\mathcal{M}$ with the linear speedup property must have polynomial vicinity*. The general reasoning is as follows: If linear speed-up holds, then for all machines M in $\mathcal{M}$ and $D > 0$, there is a machine M' in $\mathcal{M}$ that requires only t/D time units to simulate M for t time units. For this to be possible, M' needs access within t/D time units to as much information as M has within t time units; i.e., $v'(t/D) \geq v(t)$. Then $C'u(t/D) \geq Cu(t)$, so u has the property $(\forall D > 1)(\exists E > 1)(\forall^\infty t)\,[u(t/D) \geq u(t)/E]$. Every such nondecreasing function u must be polynomially bounded.

This is short of a *proof*, first because it only talks about *on-line* linear speed-up, and second because the word "needs" and the assumption that M and M' have the same "hardware," hence the same $u(t)$, must be justified in specific cases. This paper addresses these concerns in the case of the *Block Move* (BM) model of [13,12], which has useful generality because it comes with a parameter $\mu : \mathbf{N} \to \mathbf{N}$ called a *memory access cost function* that calibrates its vicinity.

2. The BM Model

A *straight-line BM program* is a sequence of *block move* instructions of the form $S\ [a_1 \ldots b_1]$ *into* $[a_2 \ldots b_2]$. Formally S is a deterministic generalized sequential machine (DGSM), as defined in [9]. If z is the string formed by the characters in cells $a_1 \ldots b_1$, then $S(z)$ is written into the block $a_2 \ldots b_2$ beginning at a_2. Overflows or underflows of the target block may be permitted, and $b_1 < a_1$ and/or $b_2 < a_2$ is allowed. The *blank* B is a writable character in block moves, with the proviso that every B appearing in the output stream $S(z)$ leaves the previous content of its target cell unchanged. Given the parameter μ, the *μ-time* for the block move has the general form

$$\mu(a) + \mathit{time}(S, z) \qquad \text{where} \qquad a = \max\{\, a_1, b_1, a_2, b_2 \,\}. \tag{1}$$

An individual BM *machine* M has an input alphabet Σ and a work alphabet Γ, a single tape that initially holds its input x, and a finite control that contains finitely-many DGSMs and transition rules that, at the end of one block move, determine the addresses a_1, b_1, a_2, b_2 and DGSM S used for the next block move (or cause M to halt). Specific machine forms may be found in [13], along with results that for the functions $\mu_d(a) = a^{1/d}$, the forms all simulate each other up to linear μ_d-time. For other μ functions, and for most of this paper, it suffices to consider the straight-line program given by the computation of M on a specific argument x. To talk about on-line simulations of straight-line programs, we consider simulators P, P' that can take a sequence of instructions (whether block moves or any kind) on an auxiliary read-only tape.

The main issue here is how to define $time(S, z)$. We suppose that it is linear in the length of z, on the grounds that nonlinearities due to the length of z are already accounted for by the term $\mu(a)$ above. Thus we may write:

$$time(S, z) = norm(S) \cdot |z| + lag(S), \tag{2}$$

where $norm(S)$ represents the propagation rate or "clock cycle" of S, and $lag(S)$ is the delay between first input and first output as data is piped through S. These are analogous to clock-cycle and lag for systolic automata as described by Even and Litman [4].

3. Results

Under the parameters $\mu_d(a) = a^{1/d}$, the BM has vicinity $O(t^d)$, simply because any access outside the first t^d tape cells incurs a charge $\mu(a) > t$. For smaller μ functions, we show by refining the argument in [10] that not only does the BM under μ lack polynomial vicinity, but also linear speed-up, regardless of how $time(S, z)$ is defined.

Theorem 3.1 *Suppose that for all $\delta > 0$, $\mu(n) = o(n^\delta)$. Then the BM under μ does not have the* on-line *form of the linear speed-up property.*

Proof. First we note that if μ has the property $(\forall E > 1)(\exists D, K > 1)(\forall^\infty n)\ \mu(n/D) \leq \mu(n)/E + K$, then there must exist $\epsilon > 0$ such that $\mu = \Omega(n^\epsilon)$. So by hypothesis of μ, there exists $E > 1$ such that for all $D, K > 1$ there exist infinitely-many n such that $\mu(n/D) > (1/E)\mu(n) + K$. We design a P along the lines of the "Storage-Retrieval Problem" in [8,10] such that P cannot be sped up on-line by a factor of E under μ.

Besides its argument x on its main tape, P is given a list of integers $i_0, \ldots, i_l$ on an auxiliary tape. Each integer i is an instruction for P to retrieve the bit of x in cell i. For sake of definiteness, let b be maximum such that $2^b \leq n - 1$, where $n = |x|$, and let w be the portion of x in the interval $B = [2^{b-1} \ldots 2^b - 1]$; so $|w| \geq n/4$. Then we restrict attention to inputs in which each instruction i_j belongs to B and calls for P to execute the one-cell block move *copy* $[i_j \ldots i_j]$ *into* $[j \ldots j]$. The simple DGSM for *copy* is the only one P uses, so the $norm(S)$ and $lag(S)$ terms are constant, while all DGSMs are available to P'. We can choose x so that w has *Kolmogorov complexity* at least $n/4$.

To simulate P on-line, P' is first allowed some number of steps to "compress" the input x over its alphabet Γ' (if it so desires), and then must interpret the instructions given to P one-at-a-time. Consider any step at which P' has finished simulating one instruction to P and is beginning the next. Let I be the configuration of the tape of P' at that point, and let $C = \log_2 \|\Gamma'\|$. Now let a be the highest cell that P' would access if called upon in configuration I to interpret any instruction $i \in B$. Then this is a description of w:

- The contents I' of cells $[0, \ldots, a]$, and the number b.
- Descriptions of P' and the routine "For $i = 2^{b-1}$ to $2^b - 1$, print the bit output by P' starting from configuration I' given instruction i."

In bits, this has length $C(a + 1) + \log n + O(1)$. By the choice of w, this must be at least $n/4$, so $a = n/4C - O(\log n) \geq n/(4C + 1)$ for sufficiently large n.

Hence for every $l \geq 1$ there is a sequence of l instructions such that P' takes time at least $l\mu(a) \geq l\mu(n/(4C+1))$ to process them, just from the memory-access charges alone. For the same instructions, P takes time at most $l\mu(n) + lK$, where the constant K covers whatever norm and lag are assigned to the DGSM for *copy*. Thus for a factor-of-E speedup we must have $\mu(n/D) \leq \mu(n)/E + K/E$, where $D = 4C + 1$ and C depends only on E, but this contradicts the hypothesis on μ.

In machine forms of the BM, one must also take into account the time spent by a given M to interpret the addresses i and move its heads to the required locations. The above estimates still hold when μ is not polylog-bounded. □

Now we consider the functions μ_d, $d \geq 1$, and turn to the issue of defining $norm(S)$ and $lag(S)$. We write $S = (Q, \Gamma, \delta, \rho, s, F)$ and let $N = \|Q\|$ stand for the number of states, and $e = \|\delta\|$ for the number of transitions. Also C stands for $\lceil \log_2 \|\Gamma\| \rceil$, and l stands for the maximum number of characters over Γ that S can output in any one transition. It is no real loss of generality to suppose $l \leq 2$, less so for $l \leq \log N$.

Theorem 3.2 *If $norm(S)$ depends only on the number of states in S and $lag(S)$ is constant, then for any $d \geq 1$, the BM under μ_d has the linear speed-up property.*

Proof Sketch. The idea is that "speeding up" a DGSM to read k characters at a time does not increase the number of states. The proof of Theorem 5.1 in [13] for the case $time(S, z) = |z|$ extends to do the rest. □

We consider two more-realistic candidates for the correct measure of $time(S, z)$. The first argues that the cycle time $norm(S)$ should be proportional to the information-theoretic size of S, which we define by the upper bound $|S|_I = |\Gamma| N(\log_2 N + l)$. The second holds that $time(S, z)$ should be the time for the best *off-line* simulation of $S(z)$, by a single machine that is given *some* encoding of S on an auxiliary tape, as well as z.

The best off-line simulation we know first computes the *trajectory* $s, q_1, q_2, \ldots, q_n$ of states entered by S on input z, and then pulls off $S(z)$. It uses the following standard way of encoding the *monoid of transformations* of S over $\{0, 1\}$. Identify Q with a subset of $\{0, 1\}^r$, where $r = \lceil \log_2 N \rceil$, and Γ with a subset of $\{0, 1\}^C$. For each $c \in \Gamma$ let $g_c : Q \to Q$ denote the mapping on states induced by c. Each g_c is encoded as a list of N binary strings of length r, and by adding "dummy states," we may suppose $N = 2^r$. The complete encoding of δ then has length $E = 2^{r+C} r = N \log N \cdot \|\Gamma\|$. while the encoding of ρ has length at most $el = 2^{r+C} l = Nl\|\Gamma\|$.

Lemma 3.3 *There is a single BM that computes $S(z)$ with this encoding in time proportional to $T(S, z) = |S|_I \log^2 N \cdot |z|$ under μ_1, and the same under any μ_d.*

Proof Sketch. The basic idea is standard "parallel prefix sum" on the monoid of S. Let z have length n over Γ, and regard the binary encoding of z as a list of n elements of size C. We first compute the sequence $g_{z_1} g_{z_2} \ldots g_{z_n}$ of mappings to be composed (in left-to-right order) as follows: Make a list δ^n of n copies of δ. Use the first bits of the binary encodings of the characters in z to mark half of the entries in each δ for erasure. Then continue with the second bits of the encodings. By the successive halving, the running time for this (under any μ_d) is $O(|\delta| n + nC^2)$. Since always $2^C \geq 2C^2$, this is $O(|\delta| n) = O(2^{r+C} rn)$.

The sequence $g_{z_1}g_{z_2}\ldots g_{z_n}$ has bit-length $2^r rn$. Now let $u(r2^r)$ stand for the μ_1-time taken to compose two functions from $\{0,1\}^r$ into itself, and suppose that the method "vectorizes" so as to compose successive pairs of $g_{z_1}g_{z_2}\ldots g_{z_n}$ in μ_1-time $u(r2^r)n$. Then the standard recursion for computing all $g_i = g_{z_i} \circ \ldots \circ g_{z_1}$ runs in μ_1-time $O(u(r2^r)n)$. The first r bits of each g_i (for start state 0^r) give the trajectory T, and one sweep in time $2^r rn$ can pull these off. T is a list of n elements, each of length r.

Finally, $S(z)$ is obtained by creating ρ^n, which has length $eln = N2^C ln$. Then the first bit of each element of T is used to mark half of the entries in each ρ for erasure. This is the same trick as above, but applied to states rather than characters, and the time under μ_1 is $O(|\rho|n + nr^2) = O(2^{r+C}ln)$. The final running time is proportional to

$$(2^{r+C}(r+l) + u(r2^r))\cdot n = |S|_I n + u(N \log N)n. \tag{3}$$

The problem of composing two mappings reduces to sorting. Standard sequential sorting methods appear to be inefficient under any μ_d on lists with $O(\log N)$-sized elements. However, simulating the Batcher odd-even sorting network with $O(N\log^2 N)$ comparators runs in μ_1-time $O(N\log^3 N)$ on such lists, and since the input length is $N \log N$, this accounts for the extra $\log^2 N$ term in the statement. $\square$

We suspect that the AKS sorting network [1] can be simulated efficiently on the BM in μ_1-time $O(N\log^2 N)$, which would cut the extra term to $O(\log N)$, but have not yet checked this. If the time for composing two mappings were linear in $N \log N$, or even proportional to $|\Gamma|N\log N$, then the time for the off-line simulation would be $O(|S|_I \cdot n)$. Via the Krohn-Rhodes decomposition of DGSMs into "permutation" and "reset" machines (see [2]), we need only worry about the case where the mappings g_c generate a group acting on a set of size N. So we ask, *Is there a representation for such a permutation group that allows elements to be multiplied in linear time?* We note that the matrix methods proposed in [2] actually take quadratic time in worst case.

Our intended use for a good bound $T(S,z)$ on the time for the off-line simulation also depends at this moment on the following:

Weak Efficiency Hypothesis. For any μ_d and reasonable time bound $t(n) \geq n$, every function f computable in μ_d-time $t(n)$ is computable in μ_d-time $t(n)$ by a BM whose number $R(n)$ of block moves satisfies $R(n) = o(t(n))$.

This is reasonable, because if $R(n) \geq t(n)/k$ where k is constant, this means that the great majority of the computation is done entirely in cells $[1\ldots k]$, and we would not expect good programs for f to spend so much time there (for infinitely many n). An analogous result for straight-line programs is proved in [12], but we have not proved the hypothesis for *machines*, which seems needed for the following:

Theorem 3.4 *Let time(S,z) in block moves be defined as $T(S,z)$ from Lemma 3.3, and suppose the "weak efficiency hypothesis" holds. Then there is a single BM M_U and a constant C such that for any μ_d and BM M that runs in μ_d-time $t(n)$, M_U given the code of M on an auxiliary tape simulates M in μ_d-time $Ct(n)$.*

Furthermore, there is a constant $C' > C$ and a language accepted in μ_d-time $C't(n)$ but not in μ_d-time $t(n)$. This would imply that linear speed-up does not hold, even off-line, for the BM under any μ_d.

Proof Sketch. M_U has one tape for that of M and two tapes for the scratchwork in Lemma 3.3. For each block move by M, M_U pulls off the addressed string z and the code of the DGSM S used. The memory access charge for this equals the $\mu(a)$ charged to M, *plus* a constant $c(M)$ for fetching S. The only reason for the weak efficiency hypothesis is to make the term $c(M)R(n)$ less than $t(n)$ as $n \to \infty$, because the choice of $T(S, z)$ makes every other constant in the simulation independent of M. (The lag term is similarly absorbed.) The existence of C' follows by analyzing the tape-reduction theorem of [13] for μ_d and the choice of $T(S, z)$, and the consequence of such a "constant-factor time hierarchy" was observed by Jones [11]. $\square$

In conclusion, we note the large gap between $norm(S) = N$ for which linear speedup holds, and $norm(S) = N \log^3 N \cdot |\Gamma|$ (with l constant) for which it appears not to hold. Can it be closed? What happens with $\log |\Gamma|$ in place of $|\Gamma|$? In the larger sense, we are asking: *Is there a more-efficient way to simulate a finite transduction off-line than by composing state mappings?* We inquire whether the systolic approach of [4] or the BDD-approach of [15] can yield more-efficient simulations, at least on the BM. As noted above, a positive answer would have a derivative effect on computational group theory.

REFERENCES

1. M. Ajtai, J. Komlos, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proc. 15th STOC*, pages 1–9, 1983.
2. M. Conner. Sequential machines realized by group representations. *Info. Comp.*, 85:183–201, 1990.
3. S. Cook and R. Reckhow. Time bounded random access machines. *J. Comp. Sys. Sci.*, 7:354–375, 1973.
4. S. Even and A. Litman. On the capabilities of systolic systems. *Math. Sys. Thy.*, 27:3–28, 1994.
5. Y. Feldman and E. Shapiro. Spatial machines: A more-realistic approach to parallel computation. *Comm. ACM*, 35:60–73, October 1992.
6. E. Grandjean and J. Robson. RAM with compact memory: a robust and realistic model of computation. In *Proc. CSL '90, LNCS 533*, pages 195–233, 1991.
7. J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.
8. F. Hennie. On-line Turing machine computation. *IEEE Trans. Electr. Comp.*, EC-15:35–44, 1966.
9. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.
10. M. Hühne. Linear speed-up does not hold for Turing machines with tree storages. *Inf. Proc. Lett.*, 47:313–318, 1993.
11. N. Jones. Constant factors do matter. In *Proc. 25th STOC*, pages 602–611, 1993.
12. K. Regan. Linear-time algorithms in memory hierarchies, 1994. This proceedings.
13. K. Regan. Linear time and memory efficient computation, 1994. Revision of UB-CS-TR 92-28, accepted to *SIAM J. Comput.*
14. A. Schönhage. Storage modification machines. *SIAM J. Comput.*, 9:490–508, 1980.
15. D. Sieling and I. Wegener. Reduction of OBDDs in linear time. *Inf. Proc. Lett.*, 48:139–144, 1993.