Equivalence of Models of Computation for Nearly Linear Time

Vincent St-Amour and Paul Stansifer

April 11, 2011

Theorem 1 (Main Theorem). *Every non-deterministic RAM computation in time t can be simulated by a non-deterministic oblivious two-tape TM in time t polylog(t).*

We prove this theorem in two steps. First, we show how to simulate non-deterministic RAM machines using non-deterministic multi-tape TMs with polylogarithmic overhead. Then, we show how to simulate multi-tape TMs using oblivious two-tape TMs.

1 RAM to Multi-tape

Lemma 1. *Every non-deterministic RAM computation in time t can be simulated by a non-deterministic multi-tape TM in time t polylog(t).*

Definition 1 (Nearly linear time). *A function is in nearly linear time if and only if it is in time$(n(\log n)^{O(1)})$.*

Definition 2 (NQL). *NQL is the class of languages accepted by non-deterministic multi-tape TMs in nearly linear time.*

Definition 3 (NNLT). *NNLT is the class of languages accepted by non-deterministic random access machines in nearly linear time.*

We assume that random-access machines (RAM machines, random-access TMs, etc.) are equivalent for nearly linear time. The proofs are straightforward, but beyond the scope of this proof.

Lemma 2. *Multi-tape TMs can sort in nearly linear time.*

This lemma is due to Schnorr [4]. We will take it for granted. The idea is that since merging two lists can be done in linear time on a 3-tape TM, sorting a list using merge sort can be done in nearly linear time.

Theorem 2 (Theorem 2 of Gurevich and Shelah [1]). *NQL = NNLT*

Proof of theorem 2.

- NQL $\subseteq$ NNLT
  Obvious. Random-access TMs can efficiently simulate multi-tape TMs.
- NQL $\supseteq$ NNLT
  Let M be a non-deterministic random-access TM that accepts language L in nearly linear time $T(n)$.

Definition 4 (Trace). *The trace $\mathcal{T}$ of a random-access TM computation is the sequence of tuples*

$$(t, q_t, a_t, I_t, b_t, J_t, c_t) : t \leq T(n)$$

such that at time t, the TM is in state q_t, the address tape is a_t with the head at I_t, the auxiliary tape is b_t with the head at J_t and the character under the head of the main tape is c_t.

Let M' be a non-deterministic multi-tape TM. The input x is on one of its tapes, which we will call the *input tape*. M' guesses traces for the computation of M on input x, guessing tuples in order from $t = 0$ to $t = T(n)$. These traces are written on another tape, which we will call the *trace tape*.
Guessing traces takes nearly linear time. Since M runs in nearly linear time, there is a nearly linear number of tuples to guess. Each tuple requires guessing a $O(\text{polylog } n)$ amount of information. t is a number between 0 and $T(n)$, and requires $\log T(n) = O(\text{polylog } n)$ bits. q_t has constant size, and depends only on M. a_t and b_t are the address tape and the auxiliary tape of M, respectively, and are logarithmic size, by definition of random-access TMs. I_t and J_t are numbers representing head locations on these tapes, and are $O(\text{polylog } n)$ as well. Finally, c_t has constant size, depending only on the size of the alphabet handled by M.

Definition 5 (Consistency). *A tuple $(t', q', a', I', b', J', c')$ is consistent with the tuple (t, q, a, I, b, J, c) if, when in state q with address tape a with head at position I, auxiliary tape b with head at position J and observing character c on the main tape, M transitions to state q' updates the address tape to a' and moves its head to position I', updates the auxiliary tape to b' and moves its head to position J'.*
Note: consistency of the characters observed on the main tape is not *enforced, since it may require looking back arbitrarily far into the computation trace.*

If the entire trace is consistent (every tuple $t + 1$ is consistent with the tuple t) and state $q_{T(n)}$ is an accepting state, we keep going. Otherwise we reject.
Checking consistency can be done in nearly linear time.

Next, we need to check for consistency of characters on the main tape of M. To do so, M' sorts the tuples of the trace according to their a_t components. Then, M' sorts the tuples again (using a stable sort, to preserve the effect of the first sorting), this time according to their t components. The result is a sequence of tuples grouped by address. Each group of tuples is ordered in chronological order.
Using lemma 2, we know that this sorting process can be done in nearly linear time.

To check consistency with regards to the main tape of M, we consider each block in order. We start with the first block, which refers to the first cell of the main tape. We position the head of M' input tape on the first cell. The first tuple of the first group is of the form:

$$(0, q_{start}, 0, I, b, J, c)$$

since tuples are sorted accoring to their third and first components. The c component of this tuple should be the same as the character M' observes under its input head; indeed, at time 0, the contents of the first cell of M's main tape is the first character of the input. If this is not the case, reject.

Definition 6 (Main Tape Consistency). *A tuple (t, q, a, I, b, J, c) is consistent with the tuple $(t', q', a, I', b', J', c')$ with regards to the main tape of M if either:*

1. $c = c'$ (the main tape is unchanged at a at this step)

2. given the configuration for M described by the first tuple, M would write c' at address a.

Note: the first tuple contains the necessary information to predict which character would be written.

We then look at the other tuples of the group in chronological order and check for main tape consistency. As we go along, M' updates its input tape (always at address a) to reflect the changes M makes to its main tape. Since the original contents of M's main tape at address a are known to be correct (consistent with input x) and we check for consistency of each modification, each write M' simulates is exactly the write M would do at that point in time.

We repeat the process for each group of tuples. If all of them are consistent with regards to the main tape, we accept.

We scan the sorted sequence of tuples in order, and advance M''s input head sequentially. Therefore, this consistency check can be done in linear time. The whole process thus takes nearly linear time.

□

Proof of lemma 1. Follows naturally from theorem 2. □

2 Multi-tape to 2-tape

For the simulation of multi-tape TMs by two-tape TMs, we present two strategies. First, we present a simulation strategy due to Hennie and Stearns [2][1] that achieves a logarithmic overhead, is simple and intuitive, but is not immediately doable in an oblivious fashion. Then, we present a slightly trickier strategy, due to Pippenger and Fischer [3], that achieves both the logarithmic overhead and the obliviousness required for theorem 4. Both these strategies work equally well in a deterministic and a non-deterministic context.

Theorem 3. *Every non-deterministic multi-tape TM computation in time t can be simulated by a non-deterministic two-tape TM in time $t\ log(t)$.*

Proof. Let M be a non-deterministic k-tape TM. Let M' be a non-deterministic 2-tape TM that simulates M.

We begin by encoding the tapes of M as tracks on one of the tapes of M', that we will call the *main tape*. The simulated head of each track will always be at the origin — the symbols on the tape will move under it. M''s main tape has an additional k tracks, paired with the encoded tracks. We call these new tracks *spill tracks*. To store all these tracks on a single tape, we extend the alphabet handled by M', such that a character of M' can encode $2k$ characters (one for each track) of M. We assume, without loss of generality, that M''s main tape is infinite in both directions. The second tape of M', henceforth known as the *auxiliary tape*, is used for copying, and to hold temporary information.

[1]A good presentation of this proof is appears in lecture notes from Paul Beame: `http://www.cs.duke.edu/~reif/courses/complectures/Beame/lect05.pdf`

To allow the virtual head associated with each track to move independently from the others, we will move tracks from under the heads instead of moving the heads themselves. A naïve solution to moving tracks would involve copying the entire non-blank contents of a track every time its head moves, one step to the left or to the right, depending on the movement of the head. This solution potentially involves $O(n)$ operations of M' to simulate a single step of M, which is unacceptable. A more reasonable solution requires copying only part of the tape at every step, and is doable with logarithmic overhead.

Definition 7 (Blocks). *We separate M''s main tape into blocks of cells.*
Block 0 contains the cell at the origin (center) of the tape.
Block $i : i > 0$ contains the 2^{i-1} cells to the right of block $i - 1$.
Block $i : i < 0$ contains the 2^{i+1} cells to the left of block $i + 1$.
Intuitively, block numbers increase (decrease, on the left side) as we get further from the origin, and blocks double in size each time.

To allow for moving tracks without copying their entire contents, we will allow tracks to "bunch up" and "stretch out". For example, assume we want to move a given track to the right (which corresponds to the head moving to the left). To do so, we move the contents of block -1 to block 0, leaving block -1 empty (or "stretched out"). We also need to move the original contents of block 0, which now belong on block 1. However, block 1 already has contents. We allow the track to "bunch up" by storing the original contents of block 1 on block 1 of the appropriate spill track. It is easy to preserve the ordering of the tape in the process.

To move the track further to the right, since block -1 is empty, we would need to move the contents of block -2 to block -1 (stretching block -2 out and bunching up block -1 in the process), then move one of these symbols to block 0. Then, on the right, since block 1 is already bunched up and cannot accomodate more symbols, we move its contents to block 2 (bunching it up in the process), then move the contents of block 0 to block 1. Since each block is twice as large as its predecessor, it can always hold enough symbols to store its own contents, plus those of its bunched up predecessor.

Definition 8 (Smoothing). *A block is smooth if it is neither stretched out nor bunched up, that is, each of its cells contains a single symbol. It is possible to smooth a bunched up block by "pushing" the excess symbols further on the tape. Similarly, it is possible to smooth a stretched out block by "pulling" symbols from further on the tape. Smoothing can happen naturally when the head changes direction, or it can be done explicitly.*

Starting from a state where blocks $-j$ to j are smooth, it is possible to simulate M for $O(2^j)$ steps without having to access beyond either block j or $-j$. After each step i of the simulation, we explicitly smooth blocks such that after step $i = k \cdot 2^{j-1}$, blocks $-j$ to j are smooth. Therefore, we smooth blocks 1 and -1 every step, blocks 2 and -2 every two steps, and so on, adding a block on each side and doing it half as often each time. Thus, by the time we would need to access beyond either block j or $-j$, a smoothing of blocks $-j$ to j would be scheduled to happen, and the simulation always has a reasonable amount of smooth blocks to play with.

The cost of smoothing a block is linear in its size. Since we need to smooth block j, which has size 2^{j-1}, $\frac{1}{2^j}$ of the steps, smoothing a given block has add a constant amortized cost to each step. Over the course of simulating M for $T(n)$ steps, the furthest block we can reach is $O(\log T(n))$. Since we see a logarithmic number of blocks, and that each adds a constant amortized cost to each step, the simulation overhead is logarithmic. □

We now present the second simulation strategy, which guarantees both a logarithmic overhead and obliviousness.

Definition 9 (Oblivious TM). *An oblivious TM is a TM whose head movements depend only on the input length and not on the input itself. Intuitively, the head movements are "pre-programmed" and cannot change depending on the input.*

Theorem 4. *Every non-deterministic multi-tape TM computation in time t can be simulated by a non-deterministic oblivious two-tape TM in time t log(t).*

Proof. Let M be a non-deterministic TM, and M' be an oblivious TM simulating M. As in the previous proof, M's tapes will be stored as tracks on M''s main tape, but, over time, the order of symbols on each track will be permuted in a systemic fashion. The positions of the virtual heads are marked on the tape using a special symbol. The simulation process outlined below operates on each track in turn. The auxiliary tape of M' is used for copying and as a stack.

Definition 10 (COMPRESS(j)). *Let COMPRESS(j) be a subprocedure that takes the symbols in a $2^{j+1}-1$ radius around the origin and rotates them by a distance of 2^{j-1} in the direction that moves the virtual head towards the origin.*
The rotation is done obliviously; M''s head moves as if it was rotating in both directions one after the other, but only modifies the tape when doing the appropriate rotation.
COMPRESS also pushes a note on the stack (auxiliary tape) indicating in which direction the rotation was performed.

Definition 11 (EXPAND(j)). *Let EXPAND(j) be the inverse of COMPRESS(j); it rotates symbols by the same amount as EXPAND(j), but in the opposite direction as the previous call to EXPAND(j), popping the direction information from the stack on the auxiliary tape.*

Definition 12 (PHASE(j)). *The PHASE(j) subprocedure is defined as follows:*
If $j = 0$, simulate a step of M's execution obliviously by moving the virtual head markers.

```
Otherwise do {
    COMPRESS(j);
    PHASE(j − 1);
    COMPRESS(j);
    PHASE(j − 1);
    EXPAND(j);
    EXPAND(j);
}
```

At the end, the tape has been restored to its original position, and 2^j steps of M have been simulated.

To simulate M, we invoke PHASE with successively larger arguments. Each of these invocations will temporarily mangle the tape, gradually rotating successively smaller portions of the tape by successively smaller amounts until the head is positionned at the origin.

The correctness of this simulation hinges on two things. First, the region being rotated is large enough that all steps of M simulated while it is out of the original order don't need to reach outside it, reaching a discontinuity in the tape introduced by the rotation. The exponential sizing of these rotations ensures this.

Second, the rotation process will always move the virtual head to the origin, where the actual steps of M always take place. At the start of an invocation of PHASE(j), the virtual head marker is at most $2^{j+1} - 1$ spaces from the origin. Each COMPRESS(j) brings it closer by 2^j. By the expressibility of natural numbers in binary notation, the correct portion of the tape is in position by the time a step is executed. The second invocation of COMPRESS(j) ensures that the tape is repositioned to compensate for the virtual head's motion.

The overall overhead is logarithmic. Observe that each of COMPRESS(j) and EXPAND(j) takes $O(j)$ time. There are also j steps "underneath" those invocations, so the cost of these rotation operations is proportional to the number of layers of PHASE(j) invocations. This is logarithmic in total simulated running time. $\square$

Proof of Main Theorem. Follows naturally from lemma 1 and theorem 4. $\square$

References

[1] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at Botik '89*, pages 108–118, London, UK, 1989. Springer-Verlag.

[2] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13:533–546, October 1966.

[3] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26:361–381, April 1979.

[4] C. P. Schnorr. Satisfiability is quasilinear complete in NQL. *J. ACM*, 25:136–145, January 1978.