Turing Networks: Complexity Theory for Message-Passing Parallelism

Jack Romo

University of York

jr1161@york.ac.uk

April 2019

Overview

Complexity Theory

- The study of resource requirements to solve problems
  - eg. Time, memory, ...
- Alternatively, the study of **complexity classes**, ie. sets of problems with the same computational overhead
  - How are they related?
  - eg. $P \subseteq NP$
- Usually studied in context of Turing Machines
- Large amount of theory developed here over the last century
- Would be ideal to reuse it in other problems
- Parallel complexity theory?

Parallel Complexity Theory

- Complexity of parallel computations are **extremely nontrivial**
- Turing Machines to parallelism: *Parallel Turing Machines*
  - Same, but can create new identical read-write heads at any step

Parallel Complexity Theory

- Complexity of parallel computations are **extremely nontrivial**
- Turing Machines to parallelism: *Parallel Turing Machines*
  - Same, but can create new identical read-write heads at any step
- However, this models **shared memory** only, not message passing
- Other models can emulate message passing parallelism, but do not connect to classical complexity theory

Parallel Complexity Theory

- Complexity of parallel computations are **extremely nontrivial**
- Turing Machines to parallelism: *Parallel Turing Machines*
  - Same, but can create new identical read-write heads at any step
- However, this models **shared memory** only, not message passing
- Other models can emulate message passing parallelism, but do not connect to classical complexity theory
- This is a problem!

Modelling Message-Passing Parallelism

We would like a model that...

1. Intuitively emulates a network of communicating processors.
2. Allows for substantial complexity analysis.
3. Relates classical complexity classes to its own parallel ones.
4. Can simulate other models of parallelism with good complexity.

Network topologies

- Take a simple undirected graph $G = \langle V, E \rangle$ of constant degree, $1 \in V \subseteq \mathbb{N}$, $E \subseteq V \times V$ symmetric relation
- Choose a vertex, index its neighbors from 1 to n; do this for every vertex
  - Call this the graph's **orientation**, $\phi : V \times \mathbb{N} \nrightarrow V$
- Call a pair $\overline{G} = \langle G, \phi \rangle$ a **network topology**

Network topologies

- Take a simple undirected graph $G = \langle V, E \rangle$ of constant degree, $1 \in V \subseteq \mathbb{N}$, $E \subseteq V \times V$ symmetric relation
- Choose a vertex, index its neighbors from 1 to n; do this for every vertex
  - Call this the graph's **orientation**, $\phi : V \times \mathbb{N} \nrightarrow V$
- Call a pair $\overline{G} = \langle G, \phi \rangle$ a **network topology**
- eg. Linked List
  - $V = \mathbb{N}$
  - $E = \{(v, v+1) \mid v \in V\}$'s symmetric closure
  - $\phi(1, 1) = 1$
  - $\phi(n+1, 1) = n, \quad \phi(n+1, 2) = n+2, \quad n \in \mathbb{N}$

Communicative Turing Machines

- Each vertex is a Turing Machine
- However, need some capacity to communicate
- Add 'special transitions' to send/receive a character
- Index neighbors to commune with by orientation
- Two start states, a 'master' state for vertex 1 and 'slave' state for the rest
    - Slaves must start by waiting for a message

Communicative Turing Machines

Definition (Communicative Turing Machine)

A **Communicative Turing Machine**, or CTM, is a 10-tuple

$$T = \langle Q, \Sigma, \Gamma, q_m, q_s, h_a, h_r, \delta_t, \delta_s, \delta_r \rangle$$

where Q, Σ are nonempty and finite, $\Sigma \cup \{\Lambda\} \subset \Gamma$, $q_m, q_s, h_a, h_r \in Q$, and

$$\delta_t : Q \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R, S\}$$
$$\delta_s : Q \times \Gamma \nrightarrow \mathbb{N} \times \Gamma \times Q^2$$
$$\delta_r : Q \nrightarrow \mathbb{N} \times Q$$

are partial functions, where $\delta_t(q_s, x), \delta_s(q_s, x)$ are undefined $\forall\, x \in \Gamma$.

Turing Networks

Definition (Turing Network)

A Turing Network is a pair $\mathcal{T} = \langle \overline{G}, T \rangle$, such that $\overline{G}$ is a network topology and T is a Communicative Turing machine.

Defining Computations

- Define a configuration of one CTM and a transition
- Extend to a configuration/transition of a TN
- Derivation sequences
- Computations as terminating derivation sequences

Configurations

Definition (CTM Configuration)

A **CTM configuration** of a CTM T is a 4-tuple of the form $C \in \Gamma^* \times \Gamma \times Q \times \Gamma^*$. We name the set of all CTM configurations for the CTM T $\mathcal{C}(T)$.
We say, for CTM configurations $C_n = \langle r_n, s_n, q_n, t_n \rangle, n \in \mathbb{N}$, a network topology $G = \langle V, E \rangle$ and $v_1, v_2 \in V$,

$$C_1 \vdash C_2 \Leftrightarrow C_1 \text{ transitions to } C_2 \text{ as TM configs}$$

$$\langle C_1, C_2 \rangle \vdash_{v_1}^{v_2} \langle C_3, C_4 \rangle \Leftrightarrow v_1 \text{ in config } C_1 \text{ sends a char to } v_2 \text{ in config } C_2, \text{ transitioning to } C_3 \text{ and } C_4$$

$$C_1 \not\rightarrow_{v_1} C_2 \Leftrightarrow v_1 \text{ sends to a nonexistent neighbor}$$

Configurations

Definition (TN Configuration)

A **TN configuration** of a TN $\mathcal{T}$ is a function of the form $\Omega : V \to \mathcal{C}(T)$.

We say, for CTM configurations $\Omega_n, n \in \mathbb{N}$ of a Turing network $\mathcal{T}$ and $v_1, v_2 \in V$,

$$\begin{aligned}
\Omega_1 \vdash_{v_1} \Omega_2 \Leftrightarrow\ & \Omega_1 \mid_{V \setminus \{v_1\}} = \Omega_2 \mid_{V \setminus \{v_1\}} \\
& \wedge \left(\Omega_1(v_1) \vdash \Omega_2(v_1) \vee \Omega_1(v_1) \nrightarrow_{v_1} \Omega_2(v_1)\right) \\
\Omega_1 \vdash_{v_1}^{v_2} \Omega_2 \Leftrightarrow\ & \Omega_1 \mid_{V \setminus \{v_1\}} = \Omega_2 \mid_{V \setminus \{v_1\}} \\
& \wedge \langle \Omega_1(v_1), \Omega_1(v_2) \rangle \vdash_{v_1}^{v_2} \langle \Omega_2(v_1), \Omega_2(v_2) \rangle \\
\Omega_1 \vdash \Omega_2 \Leftrightarrow\ & (\exists\, v \in V \bullet \Omega_1 \vdash_v \Omega_2) \\
& \vee (\exists\, v_1, v_2 \in V \bullet \Omega_1 \vdash_{v_1}^{v_2} \Omega_2)
\end{aligned}$$

Initial and Final States

Definition (Initial State)

An **initial state** of a TN $\mathcal{T}$ is a configuration Ω_S for some $S \in \Sigma^*$ where

$$\Omega_S(1) = \langle \lambda, \Lambda, q_m, S \rangle$$
$$\Omega_S(n+1) = \langle \lambda, \Lambda, q_s, \lambda \rangle \qquad \forall\, n \in \mathbb{N}$$

Definition (Final State)

A **final state** of $\mathcal{T}$ is a configuration Ω_h where $\Omega_h(1) = \langle A, b, q, C \rangle$ where $q \in \{h_a, h_r\}$. The output string is AbC with all characters not in Σ deleted. We say Ω_h is **accepting** if $q = h_a$ and **rejecting** otherwise.

Computations

Definition (Derivation Sequence)

A **derivation sequence** $\Psi = \{\Omega_n\}_{n \in X}$ is a sequence of indexed configurations of $\mathcal{T}$ where $X \subseteq \mathbb{N}$ and for any $n, m \in X$, $\Omega_n \vdash \Omega_m$ if m is the least element of X greater than n.

Say that, for two derivation sequences of $\mathcal{T}$, Ψ_1, Ψ_2, $\Psi_1 < \Psi_2$ if the former is a prefix of the latter as a sequence.

Definition (Computation)

We say a derivation sequence Ψ is a **computation** if it starts with an initial state and ends with a final state.

Acceptance, Rejection and Computing Functions

Definition (Acceptance and Rejection)

We say $\mathcal{T}$ **accepts** a string $S \in \Sigma^*$ if every derivation sequence starting with Ω_S is less than an accepting computation and all rejecting computations are greater than an accepting computation. We say it **rejects** if there exists a rejecting computation not greater than some accepting computation.

Definition (Computing Functions)

Say T **computes** a function $f : \Sigma^* \nrightarrow \Sigma^*$ if, for every input string $s \in dom(f)$, every computation of T with input string s has a final state with output string $f(s)$.

Our Time Function

- Insufficient to analyze length of computations; things happen in parallel!
- Define **parallel sequences**
- Analyze number of parallel sequences a computation is itself a sequence of
- $\tau : \Sigma^* \to \mathbb{N}$ gives longest parallel time of any computation starting with an input string

Our Time Function

Definition (Parallel Derivation Sequence)

A **parallel derivation sequence** of a Turing network $\mathcal{T}$ is a derivation sequence $\Psi = \{\Omega_n\}_{n \in X}$ where for all $i, j \in X, i \neq j$ and $v_n \in V$,

$$\begin{aligned}
\Omega_i \vdash_{v_1} \Omega_x \quad &\wedge \quad \Omega_j \vdash_{v_2} \Omega_y \quad \Rightarrow \quad v_1 \neq v_2 \\
\Omega_i \vdash_{v_1}^{v_2} \Omega_x \quad &\wedge \quad \Omega_j \vdash_{v_3} \Omega_y \quad \Rightarrow \quad v_1 \neq v_2, v_3 \\
\Omega_i \vdash_{v_1}^{v_2} \Omega_x \quad &\wedge \quad \Omega_j \vdash_{v_3}^{v_4} \Omega_y \quad \Rightarrow \quad v_1, v_2 \neq v_3, v_4
\end{aligned}$$

where $x, y \in X$ are the successors of i, j in X, if they exist.

Our Time Function

Definition (Parallel Time Function)

The **parallel time** of a derivation is the smallest number of parallel derivation sequences it is a concatenated sequence of.

The **time function** $\tau : \Sigma^* \to \mathbb{N}$ of a Turing network $\mathcal{T}$ is a mapping from input strings to the longest parallel time of any accepting computation starting with said string as input the Turing network is capable of.

Complexity Theory

We now have a time function; time to begin analyzing complexities!

Definition (Parallel Complexity)

For any given function $f : \mathbb{N}_0 \to \mathbb{N}_0$ and network topology $\overline{G}$,

$$PARTIME_{\overline{G}}(f(n)) = \{g \mid \ g : \Sigma^* \nrightarrow \Sigma^* \text{ computable by some TN } \mathcal{T} \text{ with topology } \overline{G} \text{ and } \tau_{\mathcal{T}}(n) = O(f(n))\}$$

Definition (Polynomial Parallel Time)

For a given network topology $\overline{G}$,

$$P_{par}\langle\overline{G}\rangle = \bigcup_{d\in\mathbb{N}} PARTIME_{\overline{G}}(n^d)$$

Simple Results

Theorem

For any given function $f : \mathbb{N}_0 \to \mathbb{N}_0$ *and network topologies* $\overline{G}, \overline{H}$,

$$DTIME(f(n)) \subseteq PARTIME_{\overline{G}}(f(n)) \tag{1}$$

$$\overline{H} \simeq \overline{G} \;\Rightarrow\; PARTIME_{\overline{H}}(f(n)) = PARTIME_{\overline{G}}(f(n)) \tag{2}$$

The Parallel Computation Thesis

- The time taken to compute a function in parallel is polynomially related to the space taken sequentially.
- Not all models satisfy this! (eg. Parallel Turing Machines)
- Inspiration for a complexity class relation?
- Investigate when $PSPACE \subseteq P_{par}\langle \overline{G} \rangle$
- Prove by deciding QBF in polynomial parallel time
  - a PSPACE-complete problem

Definition

For the alphabet $\Sigma = \{\forall, \exists, \bullet, \wedge, \vee, \neg, (,), \top, \bot, a\}$,

$$QBF = \{\alpha \in \Sigma^* : \alpha \text{ is a satisfiable Boolean formula}\}$$

Deciding QBF in Binary Trees

Assume a single CTM is capable of computing the following in polynomial time:

- Check whether a string is a valid Boolean formula.
- Converting a formula to Prenex Normal Form.
- Replacing all occurrences of a variable with 'true'/'false'.
- Checking whether any variables are left in a formula.
- Evaluating a formula with no variables.

Theorem (Parallel Computation Thesis, Binary Trees)

$PSPACE \subseteq P_{par}\langle\overline{B}\rangle$

Deciding QBF Generally

Definition (Neighborhood and Growth)

For a given network topology $\overline{G}$, the **neighborhood function** $N_{\overline{G}} : \mathbb{N} \to \mathcal{P}(V)$ is defined as

$$N_{\overline{G}}(n) = \{v \in V : \exists \text{ path in G from 1 to v of length} \leq n\}$$

and the **growth function** $\gamma_{\overline{G}} : \mathbb{N} \to \mathbb{N}$ is defined as

$$\gamma_{\overline{G}}(n) = |N_{\overline{G}}(n)|$$

Theorem (Parallel Computation Thesis)

For any network topology $\overline{G}$ *where* $\gamma_{\overline{G}}(n) = \Omega(2^n)$,

$$PSPACE \subseteq P_{par}\langle \overline{G} \rangle.$$

Lower Bounds

- We have a substantial lower bound, ie. a set of problems that can definitely be made tractable by parallelism
- What about an upper bound, ie. problems which can definitely NOT be made tractable?
- Solution: simulate a TN on a sequential Turing machine!
  - See whiteboard for explanation cuntz

Analyzing Our Simulation

Before we go further, we need to capture name size:

Definition (Name Complexity)

For any network topology $\overline{G}$, the **name growth function** $\kappa_{\overline{G}} : \mathbb{N} \to \mathbb{N}$ is defined as

$$\kappa_{\overline{G}}(n) = max(N_{\overline{G}}(n))$$

First tape, with input string of length x after n simulated transitions, has size $O(x + \gamma_{\overline{G}}(n)(\kappa_{\overline{G}}(n) + n))$

Second tape similarly has size $O(\gamma_{\overline{G}}(n)(\kappa_{\overline{G}}(n) + \Delta\kappa_{\overline{G}}(n+1)))$, where topology has maximum degree Δ.

Analyzing Our Simulation

After some magic, we have that transition n takes time

$$t(n) = O\Big((h(n) + k(n))^2 + \max_{v \in N_{\overline{G}}(n+1), m \in \mathbb{N}} t_\phi(v, m)\Big)$$

where $t_\phi : V \times \mathbb{N} \nrightarrow \mathbb{N}$ is the time taken to compute ϕ.
Adding together all the transitions, we have (by more magic!)

$$t_{\mathcal{T}}(x) = O\Big(\gamma_{\overline{G}}^2(\tau_{\mathcal{T}}(x))\tau_{\mathcal{T}}^2(x)\big((x + \gamma_{\overline{G}}(\tau_{\mathcal{T}}(x))\kappa_{\overline{G}}(\tau_{\mathcal{T}}(x) + 1)\big)^2 + \gamma_{\overline{G}}^2(\tau_{\mathcal{T}}(x))\Phi(\tau_{\mathcal{T}}(x))\big)\Big)$$

where $\Phi(n) = \max_{v \in N_{\overline{G}}(n+1), m \in \mathbb{N}} t_\phi(v, m)$.

What About $P_{par}\langle\overline{G}\rangle$?

- What upper bound can we get on this with our $t_{\mathcal{T}}$?
- Now $\tau_{\mathcal{T}}(n) = O(n^d)$
- Note also $\gamma_{\overline{G}}(n) = O(2^n)$, as network topologies have constant degree
- So, $t_{\mathcal{T}}(n) = O(2^{O(n^d)}(\kappa_{\overline{G}}(O(n^d)) + \Phi(O(n^d)))^2)$
- Contributing factors left are $\kappa_{\overline{G}}$ and t_ϕ

What About $P_{par}\langle\overline{G}\rangle$?

- Best we can hope for is an EXPTIME upper bound, since $2^{O(n^d)}$ in expression
- Need $\kappa_{\overline{G}}(n) = O(2^{n^k})$; is possible!
- Note $t_\phi(n, m) = O(f(n))\,\forall\, m$ means $\Phi(n) = O(f(\kappa_{\overline{G}}(n)))$
- Hence, need $f(O(2^{n^d})) = O(2^{n^h})$
- Nontrivial to find biggest possible f; however, quasi-polynomials, or $f(n) = O(2^{\log^c n})$, work!
- $f(n) = O(2^n)$ is in fact too big, would get a 2-EXPTIME upper bound instead; upper bound between these two...

Our Upper Bound

Theorem

For any network topology $\overline{G}$ *such that* $\kappa_{\overline{G}}(n) = O(2^{n^d})$ *and* $t_\phi(v, n) = O(2^{\log^c v})$,

$$P_{par}\langle \overline{G} \rangle \subseteq EXPTIME$$

Implications

- This means a problem outside of EXPTIME can only be made tractable by encoding hard computations in the network topology
- eg. Presburger arithmetic cannot be made tractable by any polynomial-time computable network topology

Simulation

We have covered the first three goals of our model; the last was "simulate other models of parallelism with good complexity."

Will try and simulate BSP and Boolean Circuits.

- BSP is a practical model of parallelism
- Boolean Circuits connect to parallelism in a theoretically important way

Conclusions

- Overall, have created a model of message-passing parallelism which meets all of our requirements
- Have shown how to connect Turing machines to this form of parallelism
- Our model is limited by a static network of constant degree; beyond the scope of this project