CS 467/567: The Parallel Computation Thesis

Stefan D. Bruda

Winter 2023
A Turing machine $M$ is $s(n)$-space bounded, $s : \mathbb{N} \rightarrow \mathbb{N}$ if
A Turing machine $M$ is $s(n)$-space bounded, $s : \mathbb{N} \to \mathbb{N}$ if

- $M$ is a Turing machine with a read-only input tape, a write-only output tape, and a (read-write) work tape.
- The output tape is initially empty and each time the machine writes on that tape it writes a symbol into the square immediately adjacent to the right of the last overwritten tape square.
- A configuration of $M$ is a tuple $\{(q, w, u\overline{a}v, \alpha)\}$ where $q$ is the current state, $w$ is the (read only) input, $u\overline{a}v$ is the content of the work tape, and $\alpha$ is the output produced so far.
- There is no configuration $(q, w, u\overline{a}v, \alpha)$ such that $(s, w, \varepsilon, \varepsilon) \xrightarrow{\star} M (q, w, u\overline{a}v, \alpha)$ and $|uav| > s(|w|)$.

**DSPACE($s(n)$) / NSPACE($s(n)$)** → the class of all the decision problems solved by $s(n)$-space bounded, deterministic/nondeterministic Turing machines.
A Turing machine \( M \) is \( s(n) \)-space bounded, \( s : \mathbb{N} \rightarrow \mathbb{N} \) if

- \( M \) is a Turing machine with a read-only input tape, a write-only output tape, and a (read-write) work tape
- The output tape is initially empty and each time the machine writes on that tape it writes a symbol into the square immediately adjacent to the right of the last overwritten tape square
- A configuration of \( M \) is a tuple \( \{(q, w, uαv, α)\} \) where \( q \) is the current state, \( w \) is the (read only) input, \( uαv \) is the content of the work tape, and \( α \) is the output produced so far.
- There is no configuration \( (q, w, uαv, α) \) such that \( (s, w, ε, ε) \xrightarrow{*} (q, w, uαv, α) \) and \( |uαv| > s(|w|) \).

\( \text{DSPACE}(s(n)) / \text{NSPACE}(s(n)) \rightarrow \) the class of all the decision problems solved by \( s(n) \)-space bounded, deterministic/nondeterministic Turing machines

- Shorthand: \( L = \text{DSPACE}(\log n) \), \( NL = \text{NSPACE}(\log n) \), \( \text{POLYLOGSPACE} = \bigcup_{k \geq 1} \text{DSPACE}(\log^k n) = \text{DSPACE}(\log^{O(1)} n) \)
- Note in passing: \( \text{DSPACE}(s(n)) = \text{DSPACE}(s(n)/c) \) for all \( c \in \mathbb{N} \)
SPACE-BOUNDDED COMPUTATIONS

- A Turing machine $M$ is $s(n)$-space bounded, $s : \mathbb{N} \to \mathbb{N}$ if
  - $M$ is a Turing machine with a read-only input tape, a write-only output tape, and a (read-write) work tape
  - The output tape is initially empty and each time the machine writes on that tape it writes a symbol into the square immediately adjacent to the right of the last overwritten tape square
  - A configuration of $M$ is a tuple $\{(q, w, u\alpha v, \alpha)\}$ where $q$ is the current state, $w$ is the (read only) input, $u\alpha v$ is the content of the work tape, and $\alpha$ is the output produced so far.
  - There is no configuration $(q, w, u\alpha v, \alpha)$ such that $(s, w, \varepsilon, \varepsilon) \xrightarrow{\ast} (q, w, u\alpha v, \alpha)$ and $|u\alpha v| > s(|w|)$.

- $\text{DSPACE}(s(n)) / \text{NSPACE}(s(n)) \rightarrow$ the class of all the decision problems solved by $s(n)$-space bounded, deterministic/nondeterministic Turing machines

- Shorthand: $L = \text{DSPACE}(\log n)$, $NL = \text{NSPACE}(\log n)$,
  $\text{POLYLOGSPACE} = \bigcup_{k \geq 1} \text{DSPACE}(\log^k n) = \text{DSPACE}(\log^{O(1)} n)$
  - Note in passing: $\text{DSPACE}(s(n)) = \text{DSPACE}(s(n)/c)$ for all $c \in \mathbb{N}$

- $L \subseteq NL \subseteq \mathcal{P}$; widely believed (but not proven) that all the inclusions are strict
THE GRAPH ACCESSIBILITY PROBLEM (GAP)

- **GAP**: Given a directed graph $G = (V, E)$ and two vertices $u, v \in V$, determine whether there exists a path from $u$ to $v$

- **GAP $\in$ NL**:

  Algorithm $\text{N-GAP}(G = (V, E), u, v)$ returns $\top/\bot$:

  1. $x \leftarrow u$
  2. while $x \neq v$ do
     1. nondeterministically guess a value $y \in V$
     2. if $(x, y) \notin E$ then return $\bot$
     3. $x \leftarrow y$
  3. return $\top$

- **GAP $\in$ DSPACE$(\log^2 n)$**:

  Algorithm $\text{D-GAP}(G = (V, E), u, v)$ returns $\top/\bot$:

  return $\text{PATH}(G, u, v, |V|)$

  Algorithm $\text{PATH}(G = (V, E), i, j, k)$ returns $\top/\bot$:

  1. if $k = 0$ then return $i = j$ else if $k = 1$ then return $(i, j) \in E$
  2. else return $\exists l \in V : \text{PATH}(i, l, \lceil k/2 \rceil) \land \text{PATH}(l, j, \lceil k/2 \rceil)$

  $O(\log n)$ recursion depth and $O(\log n)$ storage per level = $O(\log^2 n)$ space

  - GAP can be solved in parallel in $O(\log^2 n)$ time (see hypercube algorithm)
Theorem (Savitch’s theorem)

\[ \text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2) \] for most useful functions \( s(n) = \Omega(\log n) \) including polynomials and poly-logarithms (space-constructible functions)

- Let \( M \) be an \( s(n) \)-space bounded Turing machine
- Size of configuration graph: \( 2^{O(s(n))} \) vertices
- Use GAP to determine whether the accepting configuration is accessible from the initial configuration \( \rightarrow (\log 2^{O(s(n))})^2 = O(s(n)^2) \) space

Corollary

- \( \text{NL} \subseteq \text{DSPACE}(O(\log^2 n)) \)
- \( \text{NSPACE}(\log^{O(1)} n) = \text{DSPACE}(\log^{O(1)} n) (= \text{POLYLOGSPACE}) \)
- \( \text{DSPACE}(n^{O(1)}) = \text{NSPACE}(n^{O(1)}) (= \text{PSPACE}) \)

- Known that \( \mathcal{P} \neq \text{POLYLOGSPACE} \); conjectured that \( \mathcal{P} \nsubseteq \text{POLYLOGSPACE} \) and \( \text{POLYLOGSPACE} \nsubseteq \mathcal{P} \)
A language $A$ is log-space reducible to language $B$ ($A \leq_{\text{log}} B$) iff there exists a function $\tau$ computable in logarithmic space such that $x \in A$ iff $\tau(x) \in B$

Let $C$ be a class of languages:
- $B$ is log-space hard for $C$ if $A \leq_{\text{log}} B$ for all $A \in C$
- $B$ is log-space complete for $C$ if $B$ is log-space hard for $C$ and $B \in C$
- $\mathcal{P}$-complete stands for “log-space complete for $\mathcal{P}$”

How can we conclude that if a problem is $\mathcal{P}$-complete and also in POLYLOGSPACE then $\mathcal{P} \subseteq \text{POLYLOGSPACE}$?

Naïve approach: given input $x$ for some problem $A \in \mathcal{P}$, use the log-space machine $M_{\tau}$ that computes the log-space reduction from $A$ to a $\mathcal{P}$-complete problem $B$, then run the machine $M_B$ (that accepts $B$) on $M_{\tau}(x)$
A language $A$ is log-space reducible to language $B$ ($A \leq_{\log} B$) iff there exists a function $\tau$ computable in logarithmic space such that $x \in A$ iff $\tau(x) \in B$

Let $C$ be a class of languages
- $B$ is log-space hard for $C$ if $A \leq_{\log} B$ for all $A \in C$
- $B$ is log-space complete for $C$ if $B$ is log-space hard for $C$ and $B \in C$
- $\mathcal{P}$-complete stands for “log-space complete for $\mathcal{P}$”

How can we conclude that if a problem is $\mathcal{P}$-complete and also in POLYLOGSPACE then $\mathcal{P} \subseteq$ POLYLOGSPACE?
- Naïve approach: given input $x$ for some problem $A \in \mathcal{P}$, use the log-space machine $M_\tau$ that computes the log-space reduction from $A$ to a $\mathcal{P}$-complete problem $B$, then run the machine $M_B$ (that accepts $B$) on $M_\tau(x)$
- This approach fails (not enough space to store $M_\tau(x)$)
- However, we can modify the Turing machine $M_\tau$ to obtain $M'_\tau$ such that $M'_\tau(x, i) = \text{the } i\text{-th bit of } M_\tau(x)$
- Every transitions of $M_B$ depends on a single input bit
- So instead of computing all the input $M_\tau(x)$ in advance, we use $M'_\tau$ on demand to obtain the particular bit needed by the current transition of $M_B$
The parallel computation thesis

**Theorem (The parallel computation thesis)**

*Time on any reasonable parallel model is polynomially equivalent to the space used by a sequential machine*

- Technically a conjecture rather than theorem because of the presence of “reasonable”
  - A “reasonable” parallel machine usually features restrictions on word size, instruction set, and parallelism
- Powerful theoretical tool

**Corollary**

*All P-complete problems are inherently sequential unless* \( P \subseteq \text{POLYLOGSPACE} \)

- It is likely that no \( P \)-complete problem is in POLYLOGSPACE
- Therefore according to the parallel computation thesis they cannot be solved in parallel in \( O(\log^{O(1)} n) \) time
- The only possibility remaining is that they can be solved in parallel in polynomial time → no better than solving them sequentially
Theorem

An $s(n)$ space-bounded deterministic Turing machine can be simulated by a parallel machine with the minimal instruction set, of word size $O(s(n))$, and in time $O(s(n) \log s(n))$.

Theorem

A $t(n)$ time bounded parallel machine with word size $w(n)$ can be simulated by a deterministic Turing machine using space $t(n)(w(n) + \log t(n)) + s(n)$, where $s(n)$ is the space requires for the Turing machine to simulate a single instruction of a processor of the parallel machine.
Restrictions on the instruction set:

- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ space by a deterministic Turing machine, where $t(n)$ is the running time of the parallel machine.
- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ time by a deterministic Turing machine (stronger than the above).
Restrictions on the instruction set:

- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ space by a deterministic Turing machine, where $t(n)$ is the running time of the parallel machine.
- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ time by a deterministic Turing machine (stronger than the above).

Restrictions on the number of processors:

- Most people regard a parallel machine as feasible if the number of processors is $n^{O(1)}$ (small machine) and the running time is $\log^{O(1)} n$ (fast machine).
- However, the parallel computation thesis holds even if the number of processors is $2^{O(t(n))}$ or even $2^{O(t(n))^{O(1)}}$. 
Restricted on the instruction set:
- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ space by a deterministic Turing machine, where $t(n)$ is the running time of the parallel machine.
- One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ time by a deterministic Turing machine (stronger than the above).

Restricted on the number of processors:
- Most people regard a parallel machine as feasible if the number of processors is $n^{O(1)}$ (small machine) and the running time is $\log^{O(1)} n$ (fast machine).
- However, the parallel computation thesis holds even if the number of processors is $2^{O(t(n))}$ or even $2^{O(t(n))^{O(1)}}$.

Restricted on the word size:
- Normally the word size is $t(n)^{O(1)}$ though in practice the tighter restriction of $O(\log n)$ size is used for simplicity.