CS 467/567: The Parallel Computation Thesis

Stefan D. Bruda

Winter 2023

Space-bounded 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 $\{(q, w, u\underline{a}v, \alpha)\}$ where q is the current state, w is the (read only) input, $u\underline{a}v$ is the content of the work tape, and α is the output produced so far.
  - There is no configuration $(q, w, u\underline{a}v, \alpha)$ such that $(s, w, \varepsilon, \varepsilon) \vdash_M^* (q, w, u\underline{a}v, \alpha)$ 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
- Shorthand: L = DSPACE$(\log n)$, NL = NSPACE$(\log n)$, POLYLOGSPACE $= \bigcup_{k \geqslant 1}$ DSPACE$(\log^k n)$ = DSPACE$(\log^{O(1)} n)$
  - Note in passing: DSPACE$(s(n))$ = 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 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 D-GAP$(G = (V, E), u, v)$ **returns** $\top/\bot$:
  return PATH$(G, u, v, |V|)$

  Algorithm 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** $\exists l \in V$: PATH$(i, l, \lceil k/2 \rceil) \wedge$ 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)

Deterministic vs nondeterministic space

Theorem (Savitch's theorem)

NSPACE$(s(n)) \subseteq$ 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 → $(\log 2^{O(s(n))})^2 = O(s(n)^2)$ space

Corollary

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

- Known that $\mathcal{P} \neq$ POLYLOGSPACE; conjectured that $\mathcal{P} \nsubseteq$ POLYLOGSPACE and POLYLOGSPACE $\nsubseteq \mathcal{P}$

Log-space completeness

- A language A is log-space reducible to language B ($A \leqslant_{\log} B$) iff there exists a function τ 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 $A \leqslant_{\log} B$ 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 $A \in P$, use the log-space machine M_τ 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_τ to obtain M'_τ such that $M'_\tau(x, i) =$ the i-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'_τ 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 $\mathcal{P} \subseteq$ POLYLOGSPACE

- It is likely that no $\mathcal{P}$-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

The parallel computation thesis (cont'd)

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

"Reasonable" parallel models

- 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 $2^{O(t(n))^{O(1)}}$
- Restrictions 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