Lecture 16: Parallel Computing

**Simultaneous Resource Classes**

**Def:** ATIME-ALT$[t(n), a(n)]$ is the set of problems solved by alternating Turing machines in time $O(t(n))$, while making at most $a(n)$ alternations between existential and universal states, starting with existential.

**Example:**

\[
\text{ATIME-ALT}[t(n), 0] = \text{NTIME}[t(n)]
\]

Similarly, define the simultaneous classes:

\[
\text{ASPACE-TIME}[s(n), t(n)], \quad \text{ASPACE-ALT}[s(n), a(n)]
\]
LH and PH

Define the LOGTIME hierarchy (LH) and the PTIME hierarchy (PH) as follows:

\[
\begin{align*}
LH &= \text{ATIME-ALT}[\log n, O(1)] \\
PH &= \text{ATIME-ALT}[n^{O(1)}, O(1)]
\end{align*}
\]

**Thm:** \( PH = SO \)

**Proof:** [idea] Follows from Fagin’s Thm: \( \text{NP} = \text{SO}\exists \).

**Fact:** \( LH = FO \)
\[ \text{CRAM}[t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}] \]
synchronous, concurrent, \( n^{O(1)} \) processors and memory

priority write: lowest number processor wins conflict

common write: no conflicts allowed
**Thm:** For all \( t(n) \), \( \text{CRAM}[t(n)] = \text{AC}[t(n)] \).

**Proof:** (Sketch)

\( \text{CRAM}[1] \subseteq \text{AC}[1] = \text{AC}^0 = \text{FO} \):

Only the global memory is tricky: all else is sequential.

\( B(i, w, b) \) means that bit \( i \) of global memory word \( w \) is \( b \).

\[
B'(i, w, b) \equiv B(i, w, b) \land \forall p \text{ ("p didn’t just write into word w") } \lor \exists p \text{ ("p just wrote into word w " } \land R(p, i, b))
\]
CRAM[1] ⊇ FO:

∀x(α(x))

1. \( w := 1 \)
2. Each processor \( P_i \) in parallel, do {
3. \( \text{if } (\neg \alpha(i)) \text{ then Write}(0,w) \)
4. }

∃x(α(x))

1. \( w := 0 \)
2. Each processor \( P_i \) in parallel, do {
3. \( \text{if } (\alpha(i)) \text{ then Write}(1,w) \)
4. }

□
Def: $sAC^1$ (semi-unbounded $sAC^1$) is the subset of $AC^1$ where the and-gates are binary and only the or-gates are unbounded.

Fact: [Ruzzo] $sAC^1 = \log(CFL) = FO(CFL) = \{ S \mid \exists CFL C (S \leq C) \}$

Thm: $NC^1 \subseteq L \subseteq NL \subseteq sAC^1 \subseteq AC^1$

Proof:
Alternation as Parallelism

Fact: [Ruzzo and Tompa] For $t(n) \geq \log n$,
\[
\text{ASPACE-TIME}[\log n, t(n)] = \text{NC}[t(n)]
\]
\[
\text{ASPACE-ALT}[\log n, t(n)] = \text{AC}[t(n)]
\]

Cor: $\text{ATIME}[\log n] = \text{NC}^1$
\[
\text{ASPACE-TIME}[\log n, t(n)] = \text{NC}[t(n)]
\]
\[
\text{ASPACE-ALT}[\log n, t(n)] = \text{AC}[t(n)]
\]

**Proof:** (sketches)

The computation graph of an ASPACE-TIME\([\log n, t(n)]\) machine on an input of size \(n\) is an NC\([t(n)]\) circuit.

The computation graph of an ASPACE-ALT\([\log n, t(n)]\) machine is a series of \(t(n)\) computation graphs of NL or co-NL machines.

We can modify the ASPACE-ALT\([\log n, t(n)]\) machine to make a series of alternating guesses: ID\(_1\), ID\(_2\), \ldots\, ID\(_{t(n)}\) of the endpoints of each of these NL or co-NL computations, and **check only once** that ID\(_{i+1}\) follows correctly from ID\(_i\).

Since NL \(\subseteq\) AC\(^1\), the whole thing is an AC\([t(n) + \log n]\) = AC\([t(n)]\) circuit. \(\square\)