- PRAM
- Vector RAM (VRAM)
- Circuit Models (NC, P)
- MP-RAM: Multiprocessor RAM

Goals for a parallel model:
- Simple
- Guide you (alg. designers) toward efficient algorithms
- Understand how perf/alg. scales as we vary the input size
- Robust across a variety of machines
- Useful for understanding alg. design techniques

It can naturally express algs. in pseudocode and by extension real code.

What about the PRAM?
- Perhaps not robust/implementable
- But alg. design techniques can be illustrated in the PRAM
- PRAM assumes fixed (p) processors; making it rather unwieldy for coding.

\[
\text{Work-Depth} / \text{Work-Span}
\]

- shared random access memory
- dynamic task creation / closure.

\[
\text{work} = \# \text{operations}
\]

\[
\text{depth/span} = \text{longest set of sequential dependencies}
\]

\[
\text{MP-RAM : Multiprocessor-RAM}
\]

- set of dynamically evolving processes
- unbounded memory

```
begin
  \text{RAM program}
end
```

\[
p = \text{pivot}
\]

\[
\text{done in parallel}
\]
- augment with \texttt{FORK}(k)
- k children processes created
- child i gets "id" i

- nested forks OK!
Binary forking vs k-ary forking

Work-stealing scheduler (P. Blume-Leinerson)

\[
\text{work} = \text{total number of basic instructions}
\]

\[
\text{depth} = \text{longest chain of sequential dependencies}
\]

\[
W = 10
\]

\[
D = 5
\]
Work = time using 1 processor
Depth = time using \( \infty \) processors.

\[ T \geq \frac{w}{P} \implies T \geq \max \left( \frac{w}{P}, D \right) \]

\( T \geq D \)

\textbf{Brent’s Thm:} Given a DAG with work \( w \), depth \( D \), it can be scheduled in \( O \left( \frac{w}{P} + D \right) \) steps.

\[ \max \left( \frac{w}{P}, D \right) \leq T \leq O \left( \frac{w}{P} + D \right) \]

work-stealing

\[ \frac{w}{P} + D \quad \text{time} \]

\[ P = \frac{w}{D} \quad \text{parallelism} \]
- Parallelism: \[ \frac{3}{D} \]

- \( w = O(n \log n) \)
- \( D = O(\log n) \)

\[ \frac{w}{D} = O(n) \]

\[ \text{parallel for loop: } \{0, 1, 2, \ldots, |A|-1\} \]

\[ \text{parfor } i \text{ in } [0:|A|] \]

\[ B[i] := f(A[i]) \]

\[ \text{fork } (i|A|) \]

\[ \text{par do, parallel do, fork: } i \]

\[ \text{Sum}(A) = \begin{cases} + \sum \text{A}[0:|A|/2] \end{cases} \]

- if \(|A| = 1\) return \text{A}[0]
- \( l \leftarrow \text{sum}(\text{A}[0, |A|/2]) \)
- \( r \leftarrow \text{sum}(\text{A}[|A|/2, |A|]) \)
- return \( l + r \)
\[ C = A \cup B \]
\[ w(c) = w(A) + w(B) \]
\[ D(c) = \max(D(A), D(B)) + 1 \]

\[
\begin{align*}
\text{Sum:} \\
\quad w(n) &= 2w(\sqrt{n}) + o(1) = O(n) \\
\quad D(n) &= \max(D(\sqrt{n}), D(\sqrt{n})) + o(1) = O(\log n)
\end{align*}
\]

Filter \((A, f)\): \[ [1, 2, 3, \ldots, 10] \]
\[
\begin{align*}
\quad w &= O(n) \\
\quad D &= O(\log n)
\end{align*}
\]

\[ D = \text{alloc} \left( |L| + |R| \right) \]
Filter \((A, f)\):

\[
\text{if } |A| = 1:
\]
\[
\text{if } f(A[0]) \text{ return } A
\]
\[
\text{else return } []
\]
\[
\text{endif}
\]
\[
(L, R) = \text{filter}(A_{1}) \cup \text{filter}(A_{2})
\]
\[
O = \text{alloc}(|L| + |R|)
\]
\[
\{ \text{copy } L, R \text{ into } O \}
\]
\[
\text{return } O
\]
\[
\text{endfunc}
\]

\[
D(n) = D(n/2) + O(\log n)
\]
\[
\in O(\log^2 n)
\]
Nich's class:
- poly work
- poly-logarithmic depth

SSSP:
\[ n^3 \text{ with } \log^2 n \text{ depth} \]
\[ \downarrow \]
\[ n \log(n) \text{ work/depth} \]

Work-efficiency: parallel alg uses the same amount of work (asymptotically) as that of the best seq. algorithm.

\[ \frac{W}{P} + O(D) \]