So far we have assumed the following model:

Random Access Model (RAM)

A central model to describe:

Graph Algorithms.

Algs that ignore locality

Other Models:

1) Agents/Ants on the graph.
2) Pointer machines
RAM is unrealistic as n goes to infinity.

1) Speed of light (large size machines)
2) Quantum effects (small size machines)

Bottom Line: RAM

1) Many important algorithms were found using this model.
2) Most algorithms are coded in a RAM-like language.
   eg C
Parallel Models

Fixed connection machines
machines = finite state machine
2) RAM

A) Cellular Arrays 1D, 2D, 3D

1940's von Neumann
60's, 70's algorithms for CA
Alvy Ray Smith 1974
80's Wolfram
Klaus Sutner
Conway Game of Life
60's Edgar Codd
80's HT Kung
Highly connected models

1) Hypercube \( (V, E) \) 1980's

\[ V = \{ (a_1, \ldots, a_m) \mid a_i \in \{0, 1\} \} \]

\[ m = \log n \]

\[ (a_1, \ldots, a_m), (a_1, \ldots, \bar{a}_i, \ldots, a_m) \in E \]

2) Shuffle-exchange graph 1980's

\[ V = \{ (a_1, \ldots, a_m) \mid a_i \in \{0, 1\} \} \]

\[ (a_1, \ldots, a_m), (\bar{a}, a_2, \ldots, a_m) \in E \]

\[ (a_1, \ldots, a_m), (a_m a_1, \ldots, a_{m-1}) \in E \]

3) Randomly connected graphs possible models of the brain (Valiant)
Shared memory models

1) PRAM (Parallel Random Access Machine)

Processors

Memory

Unit time ops

$+, \times \div$

read/write

ER EW

CR CW

Wyllie 1979
**PRAM Issues:**

What should be the effect of a CW on a EW machine?

1) Machine crashes!
2) Garbage Reads!

How is synchronization handled?

1) Synch after each unit of time!
2) Bulk Synchronous Parallel (BSP) Valiant.
3) None!

We will mostly use 1).
Circuit Model

Inputs in either bits or words

DAG

Node

Nodes:
1. $\land, \lor, \neg$ gates
2. Arithmetic ops

1. Constant fan in
2. Arbitrary fan out

Size = # nodes = Work
Time = longest path from input to output
Span (15-210)
Depth

Neural Nets

Deep Learning!!
Naive Matrix Multiply in the Circuit Model

Input $A^{n \times n}$ & $B^{n \times n}$

Output $C^{n \times n}$ where $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

Circuit

Input $A_{11} \ldots A_{nn}$ $B_{11} \ldots B_{nn}$ $n^2$ input wires

for each $C_{ij}$

addition subcircuit
depth $= \log n$
size $= n-1$

$n^2(n-1)$ add nodes

$C_{11} \ldots C_{ij} \ldots C_{nn}$ $n^2$ output wires

Circuit Total: work: $O(n^3)$
Time: $O(\log n)$
Let's simulate Naive Matrix-Multi on PRAM

Let \( P \) = \# processors we request.

\( T \) = Parallel time they need.

First implementation

1) Request: 1 processor/node of circuit.
   Request a CREW PRAM.

   We get \( P = O(n^3) \) \( T = O(n \log n) \) CREW Alg.

2) If each processor reads its arguments we only need a CREW machine.

3) To get an EREW machine
   We use binary tree to make copies (these are new processors).

   E.g. We need \( n \) copies of \( A_{11} \)

\[
\begin{array}{c}
\uparrow \\
A_{11} \\
/ \quad \leftarrow A_{11} \\
A_{11} \quad A_{11} \\
\end{array}
\]

Increase in depth by additive \( \log n \)

Processor increase \( n^2(n-1) \) new write processors.
**PRAM Work**

**Definition:**
- \( P = \# \) of processors used for life of run
- \( T = \) total time.

**Matrix Mult**
- So far, \( O(n^3) \in P \) \& \( T = O(\log n) \) or
  \[ P \cdot T = O(n^3 \log n) \]

**Claim:** \( O(n^3/\log n) \) processor
- \( O(\log n) \) time for Naïve MM.

**Proof:**
- Let's start with \( n^3 \) mult:

\[
\begin{array}{c}
A_{11} \times B_{11} & \cdots & \cdots & A_{nn} \times B_{nn} \\
\end{array}
\]

- \( \log n \) size blocks

Each \( P_i \) computes \( n^3 \) mult in \( O(\log n) \) time.

**Mult:**
- \( P = O(n^3/\log n) \)
- \( T = O(\log n) \)
Additions: eg

\[ a_1, a_2, \ldots, a_{n-1}, a_n \]

Replace with:

\[ (a_1 + a_2 + \ldots + a_n), (a_{n+1} - a_2 - a_3 - \ldots - a_n), \ldots, (a_{n-1} - a_n) \]

Add \( n \)-numbers with \( O(n/\log n) \) processors in \( O(n \log n) \) time.

Finished Claim
The Slow Down Principle:

Given an Parallel alg using $T$ time & $P$ processors.

$\forall P \leq P' \exists$ parallel alg using $(\frac{P}{P'})T$ time and only $P'$ processors.

pf Each real processor simulates $\frac{P}{P'}$ virtual proc.

Let's implement Strassen's Alg on a PRAM

Let $SMM = \text{Strassen's Alg}$

Recall Recurrence

$SMM(n) = 7SMM(\frac{n}{2}) + Cn^2$

\[ \text{7 recursive calls} \quad \text{matrix additions} \]
Parallel Strassen Analysis

Note: Matrix addition is
\( \mathcal{O}(n^2) \) processors
\( \mathcal{O}(1) \) time

Timing Recurrence:
\[
T(n) = T(n^{1/3}) + \mathcal{O}(1)
\]
we do parallel calls

Processor Recurrence:
\[
P(n) = 7P(n^{1/3}) + cn^2
\]
we must hire processor for each call

thus \( \mathcal{O}(n^{2.8}) \)

\[
P \cdot T = Work = \mathcal{O}(n^{\log_2 7 \log_3 n})
\]