Lecture 11: The PRAM Model

1 Basic Notations

The PRAM stands for Parallel Random Access Machine, where Parallel means we have multiple processors that can each compute on the input, where we assume that they each have access to the same shared memory.

The following figure depicts the machines each accessing the same shared memory space.

Figure 1: A simple example of parallel.

In the Figure 1, assume we have $M$ processors and a shared memory which has size $N$. They also follow the same clock. This allows us to describe computation in synchronous rounds.

Then we can say the $M$ processors work in parallel. The shared memory and the same clock guarantee the cooperation of these processors.
There are many slightly different models of shared-memory access, such as CRCW “Concurrent Read/Concurrent Write” and EREW “Exclusive read/exclusive write” indicating whether multiple processors are allowed to write to the same location in memory at the same time or not. For the purposes of this lecture, we will not worry about these low level differences: you can assume our PRAMS are all CRCW.

**Definition 1.0.1 complexity class NC.** A language is in the complexity class $\text{NC}$ if it can be solved using a deterministic algorithm on a PRAM with polynomial number of processors using poly-logarithmic time. i.e. the time is $O(\log^c n)$ for some constant $c$.

If the algorithm is randomized, it is $\text{RNC}$, where $\text{RNC}$ means Random $\text{NC}$.

**Definition 1.0.2 MIS.** A Maximal Independent Set (MIS) of a graph $G = (V,E)$ is a set of vertices $I \subseteq V$.

**s.t.**

- $x \in I$, $\forall y$ s.t. $(x,y) \in E$, $y \notin I$
- $x \notin I$, $\exists y$ s.t. $(x,y) \in E$, $y \in I$

![Figure 2: A comparison of M(aximal)IS and M(aximum)IS.](image-url)
The Figure 2 shows that a maximal independent set is different from a maximum cardinality independent set. Maximal independent sets need only be independent sets that can’t be extended; they can be of different sizes. We can find a MIS in $O(E)$ time using a greedy algorithm, whereas maximum independent set is an NP-hard problem.

2 An RNC algorithm to create an MIS in parallel

We consider a model where there are $N$ processors, one for each vertex in the graph. Computation will proceed in synchronous rounds.

2.1 Algorithm

One solution is to run processors in rounds. For example, we do something in round $i$ and then round $i + 1$ with same clock. The processors work one round by one round.

For each vertex, we assign a random variable $x_i$ associated to vertex $i$ as follows, and the variable $x_i$ affects the decision on whether a vertex join or not.

$$x_i = \begin{cases} 
1 & \text{with some probability } p \\
0 & \text{with some probability } 1 - p 
\end{cases} \quad (1)$$

If $x_i = 1$ in round $j$ and vertex $i$ doesn’t yet have a neighbor in the MIS, then vertex $x_i$ will compete to try to enter the MIS in round $j$. Every round, every vertex checks if it has a neighbor in MIS. If it has no neighbor in the MIS, it will try to enter with probability $p$.

If $x_i = 1$ and $x_k = 0$ for all neighbors $k$ of $i$ then $i$ joins the MIS in round $j$. Then we can go to the next round.

Let $D$ be the maximal degree in the graph $G$, then we set $p = \frac{1}{4D}$ in Equation (1).
Then, call a vertex “satisfied” if either it enters the MIS or one of its neighbors does.

**Definition 2.1.1** An indicator \( y_i \), which says whether or not vertex \( i \) succeeds in entering the MIS in round \( i \),

\[
y_i = x_i \times \prod_{(i,j) \in E} (1 - x_j) \tag{2}
\]

Let \( z_i \) tells us that either it, or at least one of its neighbors enters the MIS in round \( i \), where

\[
z_i = \begin{cases} 
1 & \text{iff } (y_i + \sum_{(i,j) \in E} y_j) \geq 1 \\
0 & \text{otherwise}
\end{cases} \tag{3}
\]

Then we sort the degrees of \( G \) into \( \log_2 n \) buckets.

\[
1 - 2, \ 3 - 4, \ 5 - 8, \ 9 - 16, \ldots \frac{D}{2} - D
\]

\( \frac{D}{2} - D \) calls anyone with vertices from \( 2^{[\log_2(D-1)]} \leq \deg(1) \leq 2 \). We also call these the big degree nodes. There are \( \log_2 n \) buckets of degrees in total.

**Claim 2.1.2** In round \( i \) we satisfy a constant function of the “big degree” nodes. There are \( O(\log_2 n) \) rounds in round \( i \). So for \( i = 1, \ldots, \log_2 n \), there are \( O(\log_2^2 n) \) rounds in total.

The Figure 3 is satisfied. In this figure, the node \( i \) is happy because \( j \)’s neighbor enters but all \( j \)’s neighbors denied (except \( i \)).

Now let \( BDN \) be the set of big degree nodes and let \( T \) be the number of nodes in \( BDN \) that get satisfied in round \( r \).

\[
T = \sum_{j \in BDN} Z \tag{4}
\]
We want to show that

$$E(T) \geq \frac{|BDN|}{constant} \quad (5)$$

We will define a new variable $R_{ij}$ to represent the event that vertex $i$ is satisfied because a unique neighbor $j$ tries to enter the MIS in round $r$.

$$R_{ij} = x_j \prod_{(j,k)\in E} (1-x_k) \prod_{(i,l)\in E, l\neq j} (1-x_l) \quad (6)$$

Notice that $R_{ij}$ are disjoint for every neighbor $j$ of $i$. And since every event $R_{ij}$ is a case where $i$ is satisfied, we have:

$$Z_i \geq \sum_{(i,j)\in E} R_{ij} \quad (7)$$

Therefore,
\[ T = \sum_{i \in BDN} Z_i \]
\[ \geq \sum_{i \in BDN} \sum_{(i,j) \in E} R_{ij} \]
\[ \geq \sum_{i \in BDN} \sum_{(i,j) \in E} p - \sum_{(i,j) \in E} p^2 - \sum_{(k,l) \in E, k \neq l} p^2 \]
\[ \geq |BDN|1/2(D_p - D_p^2 - D_p^2) \]
\[ \geq |BDN|/16 \]

where the last line follows because we set \( p = 1/4D \).

So we have presented an RNC PRAM algorithm for MIS that runs in \( O(\log^2 n) \) rounds with \( O(n) \) processors for an \( n \) vertex graph. Note that this algorithm does not really require the shared memory space: the only way individual nodes need any knowledge of the global structure of the network beyond their local neighbors is in order to set \( p \) (the probability of entering the MIS) appropriately. There is a modification where each vertex chooses to enter the MIS with probability 1/4d where \( d \) is its own local degree; but in this case if two neighboring nodes both try to enter the MIS at the same time, we break ties in favor of the node of higher degree. The analysis can be shown to be similar: see Luby’s paper.

### 2.2 Application

One application to the former algorithm is coloring a graph with \( \Delta + 1 \) colors, where \( \Delta \) is the maximal degree.

What we can do is to color everything in MIS with color 1 and remove(everything in MIS). Then all uncolored nodes have degree at most \( \Delta - 1 \) in the graph of uncolored nodes.
2.3 Derandomization

We gave an RNC algorithm for MIS. We now show that we could derandomize this algorithm and give an NC algorithm for MIS.

We have some ideas when we look at the Figure 3. We find that whether a node satisfies the $R_{ij}$ variable needed for the analysis depends only on pairwise independence, meaning a node’s variable only needs to be fully independent from its neighbors. If we can construct a polynomial size sample space (as in a previous lecture) with pairwise independent coins that are 1 and 0 with the appropriate probabilities, then we can try them all in parallel with only a polynomial blowup in the needed number of processors.

The trick listed before is that we want to come up with a set of coins.

![Figure 4: An example set of coin values.](image)

Processor is free, and we all have $\log^2 n$ rounds. So each processor needs $\log^2 n$ coins. Thus looking at all the processors, in total we flip $n \log^2 n$ coins. So if we exhaustively tried all the coin settings, that would be $2^{n \log^2 n}$ sample points we would have to search in order to find the good setting of the coins implied by the probabilistic analysis if the coins were completely independent.

If we try different coins, randomization is done. Instead, we will show that there exists a pairwise independent sample space that only uses $n^2$ coins in total.

Where pairwise independence is formally defined as the property:
\[ P_r[x_i, x_j] = P_r[x_i] P_r[x_j] \]  \hspace{1cm} (9)

We need to construct a pairwise independent sample space where the probabilities of flipping a 1 and a 0 are highly unequal, namely:

\[
\begin{align*}
P_r[\text{coin} = 1] &= \frac{1}{4\Delta} \\
P_r[\text{coin} = 0] &= 1 - \frac{1}{4\Delta}
\end{align*}
\]  \hspace{1cm} (10)

where \( \Delta = \text{max degree} \)

Note that in order to search this sample space the processors need to agree on which sample point they are using at each point in time, so they need to use the shared memory of the PRAM.

Here is the construction.

We first construct an \( N^2 \) size sample space (with \( N^2 \) sample points). These pairwise independent sample space approximately satisfies:

\[
\begin{align*}
P_r[\text{coin} = 1] &= \frac{1}{4\Delta} \\
P_r[\text{coin} = 0] &= 1 - \frac{1}{4\Delta}
\end{align*}
\]

Set \( Q = \text{max}(8\Delta, N) \). And select the pairwise independent random variables \( B_1, B_2, ..., B_N \) from set \([0, 1, ..., Q - 1]\). Then we will convert the \( B_i \)’s to \( x_i \)’s.

What we also need to do is to determine the \( B_i \)’s.

We choose \( \alpha \) and \( \beta \) completely independently from the set \([0, 1, ..., Q - 1]\).

Set \( B_i = \alpha i + \beta (\text{mod } Q) \). Where we show below that the \( B_i \) are pairwise independent.

The deterministic algorithm simply exhaustively tries all \( n^2 \) choices for \( \alpha \) and \( \beta \). Thus we need a multiplicative factor of \( n^2 \) more processors to try all sample points in parallel to complete a round of the derandomized algorithm.
Claim 2.3.1 The $B_i$ are pairwise independent.

Proof. If $i \neq j$,
\[
B_r[B_i = a, B_j = b] = P_r[\alpha + \beta = a \land \alpha_j + \beta = b] = P_r[\alpha(i - j) = a - b, \beta = b - \alpha_j] = P_r[\alpha = \frac{a - b}{i - j}, \beta = b - \frac{a - b}{i - j}] = \frac{1}{Q^2}
\]

Now convert the $B_i$’s to $x_i$’s as follows:

Select an integer $h$,

\[
s.t. \quad \frac{1}{8\Delta} \leq \frac{h}{Q} \leq \frac{1}{4\Delta}
\]

if we set $p = \frac{h}{Q}$,

\[
\frac{1}{8\Delta} \leq p \leq \frac{1}{4\Delta}
\]

Set
\[
x_i = \begin{cases} 
1 & \text{if } B_i \in \{0, 1, ..., h - 1\} \\
0 & \text{otherwise}
\end{cases}
\]

3 Low-Diameter graph decomposition

3.1 Notation

Definition 3.1.1 $A(\chi, d)$ Decomposition $A(\chi, d)$ decomposition of $G$ is a $\chi$ coloring of the vertices.

s.t.
1. Each color class is partitioned into an arbitrary number of disjoint clusters.

2. The distance between any pair of nodes in a cluster is at most $d$.

3. Clusters of the same color are at least distance 2 apart.

Figure 5: The two corner cases of $A(\chi, d)$ Decomposition

Figure 5 shows two simple corner cases. But in general, each graph has a $(\chi, d)$ decomposition, with $\chi = \log n$ and $d = 2 \log n$, like Figure 6.

### 3.2 Simple sequential greedy algorithm

#### 3.2.1 Step 1

Construct all the yellow clusters.

Start from any uncolored vertex. Look at the number of vertices within distance $r$ (the interior) and compare to number of vertices at distance $r + 1$ (the border).

If number of uncolored vertices in interior is no more than number in the border, then grow again (in other words, increase $r$ to $r + 1$), otherwise stop. The Figure 7 show the clusters when this step ends.
Figure 6: $A(\chi, d)$ Decomposition in general

Figure 7: $A(\chi, d)$ Decomposition in general
Claim 3.2.1 \( \text{Diameter(cluster)} \leq 2 \log n. \)

\[ \text{Radius} = \text{number of times I grow a cluster} \leq \log_2 n \]

Because each time I grow, the number of nodes in the cluster doubles.

Claim 3.2.2 \textit{At the end when every node is either yellow or in the border, more than } \( \frac{1}{2} \) \textit{number of the total nodes of } G \textit{ are yellow.}

3.2.2 Step 2

Construct all the pink clusters in graph from step1 if there are vertices uncolored in graph. All operations are same as step 1.

3.2.3 Summary

Follow the instructions in step1 and step2 to color the vertices in graph until all vertices are colored. As we can see, each step colors half of all uncolored vertices in graph. So we need \( O(\log n) \) steps in total to color all vertices in the graph.

3.3 Linial and Saks \textit{RNC algorithm}

Linial and Saks also gave an \textit{RNC} algorithm that constructs the decomposition in parallel.

Each uncolored vertex \( v \) picks a radius \( r_v \) where,

\[
P_r[r_v = j] = \begin{cases} 
\left(\frac{1}{2}\right)^{j+1} & 0 \leq j \leq \log n \\
\frac{1}{n} & \text{for } j = \log n
\end{cases}
\] (13)