

## Lecture 7: Parallel MIS

### 1 Randomized Complexity Classes

#### 1.1 P - Polynomial-Time Computable

**Definition 1.1.1** *A language  $L \in \mathbf{P}$  iff there exists a deterministic Turing Machine  $M$  that decides  $L$  in polynomial time.*

#### 1.2 NP - Nondeterministic Polynomial-Time

**Definition 1.2.1** *A language  $L \in \mathbf{NP}$  iff there exists a Nondeterministic Turing Machine  $M$  that decides  $L$  in polynomial time. That is to say:*

- $x \in L$  iff there exists a computation path in  $M$  that accepts  $x$ .
- $x \notin L$  iff there doesn't exist any computation path in  $M$  that accepts  $x$ .

#### 1.3 BPP - Bounded-Error Probabilistic Polynomial-Time

**Definition 1.3.1** *A language  $L \in \mathbf{BPP}$  iff there exists a Nondeterministic Turing Machine  $M$  that decides  $L$  in polynomial time such that:*

- $x \in L$  iff  $\geq \frac{2}{3}$  of  $M$ 's computation paths accept  $x$ .
- $x \notin L$  iff  $\leq \frac{1}{3}$  of  $M$ 's computation paths accept  $x$ .

Languages in  $BPP$  are subject to **2-sided error**. This means that by picking a computation path at random, one could pick an accepting path for an  $x \notin L$ , and one could also pick a non-accepting path for an  $x \in L$ .

It is believed that  $BPP$  describes every efficient algorithm.  $BPP$  is closed under compliment.

Relative to other classes, we know that  $P \subseteq BPP$ , and  $BPP \subseteq NP^{NP}$  ( $NP^{NP}$  is the set of languages decided in polynomial-time by nondeterministic access to nondeterministic machines.  $NP^{NP}$  is more powerful than  $NP$ , and it is necessary because you must not only determine whether an accepting path exists for a given  $x$ , but you must also count the accepting paths). It is possible that  $BPP = EXP$ , but it is believed that  $P = BPP$ .

## 1.4 RP - Randomized Polynomial-Time

**Definition 1.4.1** *A language  $L \in \mathbf{RP}$  iff there exists a Nondeterministic Turing Machine  $M$  that decides  $L$  in polynomial time such that:*

- $x \in L$  iff  $\geq \frac{2}{3}$  of  $M$ 's computation paths accept  $x$ .
- $x \notin L$  iff all of  $M$ 's computation paths reject  $x$ .

$RP \subseteq BPP$ .  $RP$  algorithms are likely to be less efficient than  $BPP$  algorithms, but most natural problems are in  $RP$ .

$RP$  is not closed under compliment.  $RP$  is subject to **1-sided error** only, since one could pick a computation path that does not accept for  $x \in L$  but it is not possible to pick a computation path that accepts for  $x \notin L$ .

$RP$  corresponds to the **Monte Carlo** class of randomized algorithms.

## 1.5 coRP - The Compliment of RP

**Definition 1.5.1** *A language  $L \in \mathbf{coRP}$  iff there exists a Nondeterministic Turing Machine  $M$  that decides  $L$  in polynomial time such that:*

- $x \notin L$  iff  $\geq \frac{2}{3}$  of  $M$ 's computation paths don't accept it.
- $x \in L$  iff all of  $M$ 's computation paths accept it.

## 1.6 ZPP - Zero-Error Probabilistic Polynomial-Time

**Definition 1.6.1** A language  $L \in \mathbf{ZPP}$  iff  $L \in RP \cap coRP$ .

$ZPP$  corresponds to the Las Vegas class of randomized algorithms.  $ZPP$  is closed under compliment. It is trivially in  $NP$ . There exists an exponentially small but real possibility that  $ZPP$  algorithms can take exponentially long.

## 1.7 PP - Probabilistic Polynomial Time

**Definition 1.7.1** A language  $L \in \mathbf{PP}$  iff there exists a Nondeterministic Turing Machine  $M$  that decides  $L$  in polynomial time such that:

- $x \in L$  iff  $\geq \frac{1}{2}$  of  $M$ 's computation paths accept it.
- $x \notin L$  iff  $\leq \frac{1}{2}$  of  $M$ 's computation paths accept it.

Problems in  $PP$  are thought to be very hard.

## 1.8 MA - Merlin-Arthur

**Definition 1.8.1** A language  $L \in \mathbf{MA}$  iff it is decideable by a **Merlin-Arthur protocol**: Merlin, with unlimited resources, sends Arthur a solution to a problem with answer "Yes", which Arthur must verify in  $BPP$ .

Graph Isomorphism, a problem that we have not been able to show is polynomial-time solvable or NP-hard, is in MA.

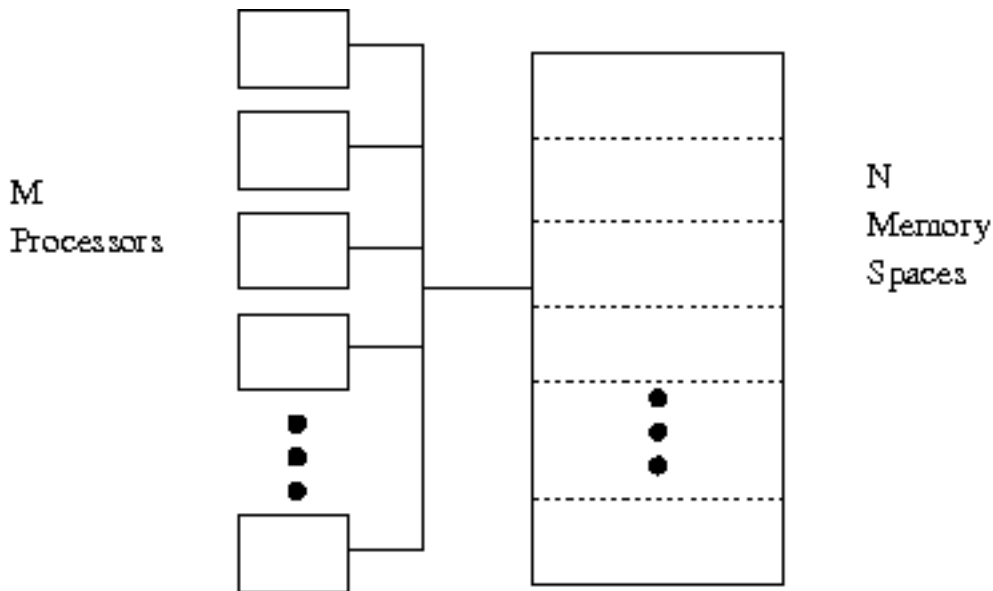


Figure 1: The PRAM

## 2 A Randomized Solution to the PRAM problem

### 2.1 The PRAM Model

A **Parallel Random Access Machine (PRAM)** is a machine with  $M$  processors and  $N$  memory spaces. For this system we assume a shared memory model, such that any processor can use any memory space. We also assume that the system is a **Concurrent Read Concurrent Write** system, which means that any number of processors can be concurrently writing or reading to or from the same memory space. However, for simplicity's sake, we also assume that any 2 processors writing to the same memory space at the same time must be writing the same value.

**Definition 2.1.1** *A language is in the complexity class  $NC$  if it can be solved using a deterministic algorithm on a PRAM with a polynomial number of processors in polylogarithmic time. If the algorithm is instead randomized, we say it is in  $RNC$ .*

We now present an *RNC* followed by an *NC* algorithm for the Maximal Independent set (MIS) problem.

**Definition 2.1.2** *A Maximal Independent Set (MIS) of a graph  $G = (V, E)$  is a set of vertices  $I \subseteq V$  such that*

- $x \in I \rightarrow \forall y$  such that  $(x, y) \in E, y \notin I$
- $x \notin I \rightarrow \exists y$  such that  $(x, y) \in E$  and  $y \in I$

To put this simply, this means that it is a subset of the vertices in a graph such that no 2 vertices in that subset are neighbors, and no other vertices can be added without disrupting that condition. The problem with attempting to solve this problem on a parallel machine is that 2 processors could pick neighbors in the graph simultaneously.

## 2.2 A Randomized Solution

To solve the MIS problem, we approach a solution in rounds. We begin with an empty set of vertices  $I$  and a graph  $G = (V, E)$ . In a given round, the vertices in  $\{V - I\}$  "compete" to be in  $I$ , which is done with randomness.

We assign value to random variable  $X_i$  as follows:

$$X_i = \begin{cases} 1 & \text{with some probability } p \\ 0 & \text{with some probability } 1 - p \end{cases}$$

If  $X_i = 1$ , then vertex  $i$  competes to be in  $I$ . If  $X_i = 0$ , then it does not. If vertex  $i$  decides to compete, and all of  $i$ 's neighbors do not, then  $i$  "wins" and is added to  $I$ . The probability that vertex  $i$  wins can be expressed as follows:

$$Y_i = X_i \times \prod_{(i,j) \in E} (X_j = 0)$$

A vertex is **satisfied** if it enters  $I$ , or if it doesn't compete and a neighbor enters  $I$ . The probability of  $i$  being satisfied can be expressed as a variable  $Z_i$ :

$$Z_i = 1 \text{ if } (Y_i + \sum_{(i,j) \in E} Y_j) \geq 1$$

Let  $D$  be the largest degree of an unsatisfied vertex in  $\{V - I\}$ .  $D$  is, in general,  $O(n)$ . Construct  $\log(D)$  buckets and distribute unsatisfied vertices into some bucket defined as follows:

$Bucket_i$  contains all unsatisfied vertices of degree  $\in [2^{i-1}, 2^i]$ . In each round, we'd like to reduce the number of unsatisfied vertices.

**Claim:** At the end of each round, the algorithm satisfies (expected) a constant fraction  $\frac{1}{\alpha}$  of "big degree" vertices. The "big degree" refers to the  $\log(D)^{th}$  bucket. If this claim is true, then all "big degree" vertices will be satisfied in  $O(\log n)$  rounds. After this time, the maximum degree of any vertex in the graph will be  $\frac{D}{2}$ . Therefore, in  $O(\log^2 n)$  rounds, all vertices will be satisfied.

**Proof:** Define  $T$  to be the number of big-degree vertices satisfied in a given round. Let  $B$  be the number of big-degree vertices at the start of a round.  $B = |BigBucket|$ . We can also say that  $T = \sum_{i \in B} Z_i$ . We'd like to be able to say that that  $E(T) \geq \frac{|B|}{\alpha}$ , for some constant  $\alpha$ .

As we know,  $E[Y_j] = E[X_j \prod_{(j,k) \in E} (1 - X_k)]$  where  $X_j$  is the chance that vertex  $j$  got picked, and  $\prod_{(j,k) \in E} (1 - X_k)$  is the chance that none of  $j$ 's neighbors got picked. From this we can see that we would need our variables to be  $D$ -wise independent. This is a problem we must circumvent. To do so, we will construct a new estimate,  $T'$ , with the following conditions:

- Condition 1:  $T'$  is an underestimate of  $T$ .
- Condition 2:  $T'$  requires pairwise independence only, among  $X_i$ 's.
- Condition 3:  $E[T']$  is at least as big as a constant fraction of the big-degree vertices.

$$E[T] \geq E[T'] \geq \frac{|B|}{\alpha}.$$

We define a new variable,  $R_{ij}$ , to replace the  $Y_i$ 's which we'd been using to represent the chance that a given vertex is entered into  $I$ . We let:  $R_{ij} = X_j \prod_{(j,k) \in E} (1 - X_k) \prod_{(i,l) \in E, l \neq j} (1 - X_l)$ . This represents the probability that vertex  $i$  is satisfied by exactly 1 neighbor,  $j$ , which means that  $j$  got chosen,

and no other neighbor of  $i$  was chosen, nor was  $i$  itself. Since we can see that the chance of  $i$  being satisfied by one neighbor only is a subset of the ways that vertex  $i$  can be satisfied at all, we can see that  $E[Z_i] \geq \sum_{(i,j) \in E} R_{ij}$ .

We define  $T' = \sum_{i \in B} \sum_{(i,j) \in E} R_{ij}$ .

We know that  $T' \leq \sum_{i \in B} Z_i = T$  so condition 1 is true.

We can also see that:

$$T' = \sum_{i \in B} \sum_{(i,j) \in E} R_{ij} = \sum_{i \in B} \sum_{(i,j) \in E} X_i \times (1 - \sum_{(j,k) \in E} X_k - \sum_{(i,l) \in E} X_l).$$

But the probability that each vertex competes to enter the independent set is simply  $p$ , a constant. So:

$$T' = p(1 - \sum_{(j,k) \in E} p - \sum_{(i,l) \in E} p) = (p - \sum_{(j,k) \in E} p^2 - \sum_{(i,l) \in E} p^2)$$

Since  $D$  is the greatest degree of any vertex in the graph, we can substitute it in for the summations to get:

$$T' \geq |B| \frac{D}{2} \times (p - Dp^2 - Dp^2)$$

If we set  $p = \frac{1}{4D}$  and  $\alpha = \frac{1}{16}$  then it works, and we have satisfied condition 3 as well. Since we can see that the algorithm always captures a minimum of a constant fraction ( $\frac{1}{16}$ , by the conservative estimate  $T'$ ) of the largest-degree vertices, we can see that it varies only by said constant from running in  $O(\log^2 n)$  time, and thus runs in  $O(\log^2 n)$  itself.

Thus we have demonstrated an RNC algorithm for MIS which runs in  $O(\log^2 n)$  rounds using  $M$  processors. In fact, the estimate  $T'$  using variables  $R_{ij}$  only depends on *pairwise* independence to compute, and thus there is, in fact, a trick similar to what we used for max cut to derandomize this algorithm using a smaller pairwise independent sample space. Though we don't have time to show this in class, the result is a deterministic  $NC$  algorithm for MIS which also runs in  $O(\log^2 n)$  rounds but uses  $O(n^3)$  processors (each point of an  $O(n^2)$  sized sample space is run in parallel).