Professor Satish Rao
Lecturer: Satish Rao Last revised Scribe so far: Satish Rao cribbing from previous years lectures

Lecture 20/21 PRAM lower bounds.

1 Overview

Today, we will learn about parallel complexity. In particular, we introduce Nick's Class (NC) as the class of problems that can be parallelized in a sense. Then, we give several examples of problems that, were they in NC any polynomial time solvable problem would be in NC.

2 Definitions

Before we define NC, it is useful to recall the class of problems that we are most concerned with for sequential computing. This is the class of polynomial time solvable problems, which we refer to as P.

We define the class, (**NC**), or "Nick's class" as the set of problems to be solvable in time $O(\log^c n)$ on a PRAM with a polynomial number of processors, for some constant c. For example, we have seen that the problem of list-ranking, approximate list coloring, matrix multiplication are in **NC**.

Perhaps it is the case that all problems in P are in NC. We think not, at this point. This can be seen as analagous to the question of whether all problems in **NP**(that is, nondeterministic polynomial time) are in **P**. Thus, we have the analagous definition to **NP**-completeness, **P**-completeness.

Or, more specifically, we define **P**-hard as follows.

DEFINITION 1 *A language L is* **P***-hard, if any language L' reduces to L through some NC algorithm. That is, for any string x, there is a* **NC***-computable function $r(x)$ where $x \in L'$ if and only if $x \in L$.*

Or course, a problem is **P**-complete if it is **P**-hard and is also in **P**.

Some **P**-complete problems include the Monotone Circuit Value Problem (MCVP), most yes/no questions about depth-first search, Greedy Independent Set, and Max Flow/Linear Programming. We show NC-reductions in the following sections.

3 Monotone Circuit Value Problem

The monotone circuit value problem is composed of a set of gates $g_1, \ldots, g_n$ where each is an AND gate,

$$g_i = g_j \wedge g_k$$

, or an OR gate,

$$g_i = g_j \vee g_k$$

or a constant value, $g_i =$ **true** or $g_i =$ **false**.

We wish to compute the value of g_n. This is clearly easy to do in polynomial time. However, we prove the following theorem.

THEOREM 1
MCVP is P-complete.

PROOF:

To show P-hardness, consider any language L in P. We will design an NC algorithm that given x computes $r(x)$ such that $x \in L$ if and only if $r(x) \in MCVP$, i.e., the value of the gate g_n is true.

Recall the definition of a turing machine. It consists of a tape of binary memory and a read/write head for the tape. In each step, the head can move right or left or write a symbol at its head and then moves to the next program instruction (which may be a jump backwards). Thus, its state at any step is represented by a program instruction number p_i, a head location l_h and the contents of the tape.

Any language in **P** has a turing machine program that solves it. For example, given a string $x \in L$, it will halt with a 1 in position 1 on the tape ("outputs" 1), otherwise it "outputs" 0.

Given a Turing Machine program for L and a particular input x, we wish to create a layered circuit $r(x)$ which evaluates to **true** if and only $x \in L$ (i.e., the program outputs 1.)

For simplicity of computing, we represent our state as a tape where each spot consists of a bit and either an empty spot or the program counter. That is, $s_{t,j}$ is the value of the tape at cell j at time t.

With this representation, the value of $s_{t,j}$ can be completely determined by seeing only the value of $s_{t-1,i}$ for $i \in \{j-1, j, j+1\}$. For each $s_{t,j}$ there are $2(p+1)$ possible values, where p is the length of the program. We can then represent each of the possible values of this state by a variable, $g_{t,j,s}$ which can be true if and only if the cell has that state. The value of $g_{t,j,s}$ can be computed using a constant depth circuit. For example, $g_{t,j,s}$ for a state $(0, \text{``}noprogramcounter''\text{)}$ is true if none of j,$j-1$, or $j+1$ have the program counter at time $t-1$ and the value at time $t-1$ was 0. This can be computed using a boolean circuit.

The final gate of the circuit could be defined as $g_{(T(|x|),1,s)}$ where s is the state that corresponds to a 1 being written in the cell. The timestep $T(|x|)$ is determined by the running time of the polynomial bound on the running time of the program that determines L. Moreover, the number of tape cells in each step can also be upper bounded by this term. Thus, the total number of states is bounded by a polynomial in this construction.

Now, this may contain not gates. This can be remedied by pushing the nots into the expressions. For example, the expression $\neg(a \vee b) = (\neg a \wedge \neg b)$. This can be done until we reach the inputs. In this case, we can replace the $\neg$**true** with **false** and vice versa.

The construction can be done in NC by assigning one processor to each cell. There is a constant (proportional to the size of the program) number of gates for each cell, and the wiring can be done using only information about the program. The total number of cells is polynomial in $|x|$, thus the reduction is in NC.

□

4 Greedy Independent Set

Greedy Independent Set(GIS) is the output of the following greedy algorithm for finding a maximal independent set of nodes in a graph with some ordering of the nodes; choose the first node in the ordering, place it in the set, and remove its neighbors, and then continue with the next node that in the order that remains in the graph. Formally, the problem is given a graph $G = (V, E)$ with a bijective map $num : V \to \{1, \ldots, n\}$ find the unique set S such that $v \in S$, if and only if for all $num(v') < num(v)$, $v' \in S$ there is no edge between v' and v.

For the purposes of reductions, we use the decision version of GIS in which we decide whether the last node is in S or not.

Theorem 2
Greedy Independent Set is **P***-complete.*

Proof:
GIS is in P since the greedy algorithm makes it so. Now, we reduce MCVP to the decision version of GIS as follows. Given an instance of MCVP whose gates are $G = \{g_1, \ldots, g_n\}$. We introduce two nodes T_{g_i} and F_{g_i} which will be in the GIS dependending on the trueness or falseness of gate g_i. Clearly, we connect them with an edge so only one is in any independent set. Furthermore,

1. For input gates, we assign $num(F_{g_i}) = 2i - 1$ and $num(T_{g_i}) = 2i$, if g_i is **false**, and flip them otherwise. This ensures that the correct value is chosen for these.

2. For an OR gate g_i, we set $num(F_{g_i}) = 2i - 1$ and $num(T_{g_i}) = 2i$ and add edges from (F_{g_i}, T_{g_j}) for each input g_j. This will ensure that F_{g_i} is chosen if and only if all the false nodes for its inputs are chosen.

3. For an AND gate g_i, we set $num(F_{g_i}) = 2i - 1$ and $num(T_{g_i}) = 2i$ and add edges from (T_{g_i}, F_{g_j}) for each input g_j. This will ensure that T_{g_i} is chosen if and only if all the true nodes for its inputs are chosen.

We add a higher numbered node v_{out} which is connected to the node F_{g_n}.

By induction, one can prove that one of the two nodes will be chosen for each gate and that moreover, that they will correspond to the correct one for its purpose. That is, the true node will be in an independent set when the value of the gate is true.

Moreover, the edges and nodes for a gate can be constructed using a single processor. Thus, the reduction is an **NC**-reduction.

□

5 Max Flow

Another problem that is important is the Max Flow problem. Given a directed graph $G = (V, E)$, a capacity function $c : E \to Z$ on the edges, and two nodes $s, t \in V$. A *flow assignment* associates non-negative flows, $f(e)$ along each edge $e \in E$ such that the edge's capacity is not exceeded ($f(e) \leq c(e)$), and at each node other than s and t, the total flow along incoming edges is equal to the total flow along outgoing edges. We wish to compute the flow which maximizes the total flow out of s (or into t.)

We consider a particular decision version of the max-flow problem where we wish to decide if the flow is odd.

Theorem 3
Max Flow is **P***-complete.*

Proof:

First note that Max Flow is known to be in **P**. To show **P**-hardness, we reduce from MCVP. We have a circuit C whose gates are $\{g_1, \ldots, g_n\}$. We assume we have outdegree at most 2, which can be enforced in **NC**with some work.

We will construct (1) a graph representing C, and construct a corresponding flow which sends an odd flow if C outputs **true** and (3) prove that this flow is maximum.

We build a graph with a node for each gate g_i and additional nodes s and t. We assume the gates a topologically sorted (this can be enforced either by noting that our original MVCP reduction could do this, or by topologically sorting in **NC**.)

We add an edge from each source vertex to each constant gate, of capacity 0 if the gate is false and of capacity equal to its output capacity if the gate is true. We add an edge from g_i to g_j if g_i is an input of g_j. The capacity of the edge is 2^{n-i}. We add a single edge of capacity 1 from g_n to t.

We add *balance* edges to gate nodes as follows. For AND gates we add an edge to t, and for OR gates we add an edge back to s (this is unneccessary since it just creates a flow cycle, but it is easier to analyze.) The balance edges for a nonoutput node i with outdegree d and incoming edges of capacity 2^j and 2^k have cacpacity $2^j + 2^k - d2^i$. Notice that d is 1 or 2, thus if one input is saturated, all of its flow can go be directed through balance edges.

Given a truth assignment, we construct a corresponding flow. If the truth assignment evaluates to true the flow is odd. Furthermore, this flow is maximum, which shows the correspondence between the problems.

We begin by saturating all the edges from the source to the true constant nodes. We then proceed as follows.

1. If g_i is a **true** OR gate. Then one input is true and we can saturate its output edges which are destined for gate nodes. (We return the excess flow to the source.)

2. If g_i is a **false** OR node, it has no inflow and no outflow.

3. If g_i is a **true** AND gate. Then at least one input (indeed both) are true and we can saturate its output edges which are destined for gate nodes.

4. If g_i is a **false** AND gate. We send the flow directly to the sink, avoiding sending any flow to other gates. Since only one edge is true, this can be done, since the balance edge has enough capacity to do so.

Under this flow, the edge g_n to t is saturated when g_n is true and empty otherwise. Since this is the only odd edge and all edges to t are either empty or saturated. This flow is odd if and only if the circuit evaluates to true.

Question 1: To argue that the flow is maximum, one needs to show that there is no augmenting path in this flow. Give such an argument.

Question 2: Compete the **P**-completeness proof. We only showed that *this* flow is odd if the circuit evaluates to true. How do you complete the proof? (One line is sufficient.)

□