Parallel Algorithms and Programming
Week 9

Kishore Kothapalli

October 27, 2007

Chapter 10

Parallel Complexity Theory and Lower Bounds

In the final topic of this lecture, we look at the relative power of the various PRAM models and also some lower bound results. We also define parallel complexity classes and their relation to classical complexity classes.

10.1 Simulations of PRAM models

10.1.1 CRCW Priority by EREW

Let us simulate a CRCW Priority machine by the EREW PRAM. The strategy would be to simulate every read and write of the CRCW PRAM by exclusive reads and writes. The following theorem can be shown.

Theorem 10.1.1 *A concurrent read instruction of a p–processor CRCW Priority PRAM can be simulated by a p–processor EREW PRAM in $O(\log p)$ steps.*

Proof. The proof gives a step-by-step translation of each concurrent read instruction as follows.

Let $C_1, C_2, \cdots, C_p$ be the p processors of the CRCW Priority PRAM and $E_1, E_2, \cdots, E_p$ be the p processors of the EREW PRAM. Let us look at a concurrent read instruction.

Let processor C_i wish to read memory cell M_i for $i = 1$ to p. A temporary storage of p cells $T_1, T_2, \cdots, T_p$ will be used by our approach. Processor C_i writes a pair $\langle M_i, i\rangle$ in slot i of T. Since each processor has a different cell, these writes are exclusive. Notice however that because of a concurrent read it may happen that $M_i = M_j$ for $i \neq j$. Sort the entries in T lexicographically so that $\langle a, b\rangle < \langle c, d\rangle$ if $a < b$ or $c < d$ if $a = b$. This sorting is done in the EREW model using the optimal sorting algorithm that sorts in $O(\log p)$ time using $O(p)$ processors.

Now there are blocks of processors that are grouped according to the memory cell they read. In each such block, find the processor with the least index. This can be done in $O(1)$ time per block in parallel. Call this processor as the representative of the block.

For each block b the representative reads the memory cell M_b and broadcasts this value of all other processors in that block. This can be done in $O(\log p)$ time using the binary tree approach.

Simulating a concurrent write instruction will be part of homework. □

10.1.2 CRCW Priority by CRCW Common

Now we shall consider the relative power of the various concurrent write models of PRAM. It is easy to see that a COMMON can be simulated by ARBITRARY as well as a PRIORITY without any change.

Similarly, an algorithm designed on the ARBITRARY will run without change on a PRIORITY. However, it is not obvious how a COMMON can simulate a PRIORITY. We will show below that a COMMON can indeed simulate a PRIORITY provided with few extra resources with no slowdown.

To help us in the simulation and as a generalized problem that has wider scope, we first define the following problem.

Definition 10.1.2 (Leftmost Alive Processor Problem) *Consider p processors of which some are "alive" and some are "dead". An alive processor is characterized by a value 1 and and dead processor by 0. The problem is to find the lowest indexed alive processor with the condition that the dead processors cannot participate in the compuatation.*

The above problem is very similar to the leftmost 1 problem with the only difference being the additional rule that dead processors cannot participate in the computation. The reason for the restriction is that typically the alive processors correspond to processors in a particular write conflict. So the processors dead are not in conflict at this cell but could be in conflict elsewhere and hence would be part of a separate instance of the leftmost alive problem.

We first show how to solve this problem on the CRCW Common PRAM provided that there are $\log p$ additional processors per each live processor.

Theorem 10.1.3 *The leftmost alive processor problem can be solved in $O(1)$ time on the CRCW COMMON PRAM using an additional $\log p$ processors per live processor.*

Proof. The proof of the theorem follows using the $O(1)$ time minima algorithm for restricted domains. Hence the proof is skipped in this version. □

We get the following results as corrollaries.

Corollary 10.1.4 *A CRCW COMMON PRAM with $O(p \log p)$ processors can simulate a CRCW PRIORITY PRAM of p processors with no slowdown.*

One can ask what would be the situation if no extra resources are made available for the COMMON PRAM to simulate the PRIORITY PRAM. In this case, as the following results shows, a slowdown of $O(\frac{\log p}{\log \log p})$ suffices.

Theorem 10.1.5

10.2 Lower Bound Results

We next turn to prove lower bounds. First we concentrate on the CREW PRAM. To show the applicability of our lower bounds, we strengthen the PRAM model slightly to define an "ideal" PRAM. If we show a lower bound on this (stronger) model then certainly it should hold on our regular PRAM model.

10.2.1 The Ideal PRAM

We do not go overboard in strengthening our PRAM model. We only assume that the shared memory has *unbounded size*. Each processor has *unlimited* local memory. Each computation step will be viewed as 3 steps: a read step where values are read from the shared memory, a compute step for local computation, and a write step to write values into shared memory. We shall allow arbitrary amounts of local computation in each step. The restrictions concerning concurrent reads and writes as similar to that our standard definition of the PRAM.

10.2.2 Lower Bound for Boolean-OR on a CREW PRAM

Let f be an n–input Boolean function. We denote the input to f by $I = x_1x_2\cdots x_n$. If the ith bit of I is flipped we denote such input by $I(i)$. An input I is said to be *critical* for f if and only if $f(I) \neq f(I(i))$ for all $1 \leq i \leq n$. For example the input $000\cdots0$ is critical for the Boolean OR function. If an input I is critical, then the idea is that if any bit in I, say x_i, is flipped then the output changes. Hence the computation of f should run for enough time steps to let x_i affect the output.

Let us assume that for computing the function f, the inputs are given in memory cells $M_1, M_2, \cdots, M_n$ of the shared memory. We shall also assume that the cell M_1 contains the output at the end of the computation.

We say that an input index i *affects* a memory location M at time t on some input I if the contents of M at time t and input I differs from the contents of M at time t on input $I(i)$. To capture this definition, we define the set $G(M, t, I)$ as follows:

$$G(M, t, I) = \{i \mid i \text{ affects } M \text{ at time } t \text{ on input } I\}$$

Along the same lines, we can also look at how the state of a processor is affected by an input index. By the state of the processor we can take all that describes the current state including the contents of the reigsters and the local memory of the processor. The state of a processor may change by some read operation. Let us define the set $C(P, t, I)$ as follows:

$$C(P, t, I) = \{i \mid i \text{ affects } P \text{ at time } t \text{ on input } I\}$$

To come up with a lower bound for Boolean OR compuataion, we have to study how the sets C and G evolve. We first present two lemmata.

Lemma 10.2.1 *If $i \in C(P, t, I)$ then:*

- *either $i \in C(P, t-1, I)$, or*
- *P reads a gloabl memory location M on input I at time t and $i \in G(P, t-1, I)$.*

Proof. We are given that the state of P at time t on input I at time t is different from the state of P at time t on input $I(i)$. Suppose that $i \notin C(P, t-1, I)$. Then, it means that the state of P at time $t-1$ does not change regardless of the input being I or $I(i)$. So it must be the case that P read some global memory location that is affected by index i in time step $t-1$. Hence, $i \in G(P, t-1, I)$. The other part of the proof is similar. □

Lemma 10.2.2 *Let $i \in G(M, t, I)$. Then,*

- *either $i \in C(P, t, I)$ and processor P writes into M at time t on input I, or*
- *No processor writes into M at time $t-1$ on input I and either $i \in G(M, t-1, I)$ or a processor P writes into M at time t on input $I(i)$.*

The proof of this lemma is left as a homework problem.

We use the above lemmata to show the following key theorem.

Theorem 10.2.3 *Let P be a processor and M be a memory cell in a CREW PRAM. Then after t steps of computation,*

$$|G(M, t, I)| \leq b^t \text{ and } |C(P, t, I)| \leq b^t$$

where $b = \frac{5+\sqrt{21}}{2}$.

Proof. To simplify notation let $|C(P,t,I)| = c_t$ and $G(M,t,I) = g_t$. Based on the above lemmata, $C(P,t,I) = C(P,t-1,I) \cup G(M,t-1,I)$ so it holds that $c(t+1) = c(t) + g(t)$.

For $G(P,t+1,I)$, notice that in the first case where a processor writes into cell M at time t on input I, i.e., $i \in C(P,t,I)$, it holds that

$$|G(P,t+1,I)| = |C(P,t,I)| \leq c(t) + g(t)$$

There is another case for i to be in $G(P,t+1,I)$. It is that no processors writes into cell M at time $t+1$ on input I but either i affects M at time t or a processor P writes into M at time $t+1$ on input $I(i)$. The latter set is actually $G(M,t+1,I(i))$. However, it is not difficult to estimate the size of this set using the following observation.

Observation 10.2.4 *Let index u_i cause processor P_{w_j} to write into cell M at time $t+1$ on input $I(u_i)$. For all pairs u_i, u_j such that $P_{w_i} \neq P_{w_j}$ either $u_i \in C(P_{w_j}, t, I(u_j))$ or $u_j \in C(P_{w_i}, t, I(u_i))$.*

The observation follows from the fact that concurrent writes are not allowed.

Finally, consider a bipartite graph with the parition being $U = \{u_1, u_2, \cdots, u_r\}$ and $V = I(u_i) \times \{P_{w_1}, \cdots, P_{w_r}\}$. An edge exists between u_i and $((u_j), P_{w_j})$ if and only if u_i affects P_{w_j} at time $t+1$ on input $I(u_j)$. In the above, $r = |G(M,t+1,I(i))|$. To get a lower bound on r, we notice that the number of edges in the above graph is at most $r \cdot ck(t+1)$. This follows because for every vertex $v = (I(u_j), P_{w_j})$ we have at most $|C(P_{w_j}, t+1, I(u_j))| \leq c(t+1)$ neighbours in U. It also holds that, excluding concurrent writes, the number of pairs (u_i, u_j) such that $P_{w_i} \neq P_{w_j}$ are at least $r(r - c(t+1)$. This is because for u_i we have r choices and once we choose a u_i for u_j the number of choices is at least $r - c(t+1)$. (There are only at most $|C(P_{w_j}, t+1, I)| \leq c(t+1)$ that have to be excluded). Thus, the number of edges in the graph is at least $r(r - c(t+1))/2$ and at most $rc(t+1)$. Putting both inequalities together and solving for r gives that $r \leq 3c(t+1) \leq 3c(t) + 3g(t)$.

Combining both parts, we get that $g(t+1) \leq g(t) + r \leq 3c(t) + 4g(t)$.

To finish the proof, consider the pair of recurrence relations $c()$ and $g()$ with suitable boundary conditions and solve them using standard techniques. □

Now, the lower bound for Boolean-OR follows from the following theorem.

Theorem 10.2.5 *Let $f : \{0,1\}^n \rightarrow \{0,1\}$ have a critical input. Then, $\Omega(\log n)$ steps are required to compute f on a CREW PRAM.*

Proof. Notice that n indices affect the output of f. Also, we require that the output be stored in M_1. Thus, we require that n indices affect M_1. In our notation from above, we have that $G(M_1, T, I) \leq n$ if T steps are required for computing f. Using the above bounds on $G()$, we get that $T = \Omega(\log n)$. □

Since Boolean OR has a critical input, the lower bound applies to computing Boolean OR.