COMS E6998-9: Algorithms for Massive Data (Spring'19) Apr 09, 2019

Lecture 21: Cache Oblivious I/O Models, Parallel Algorithms

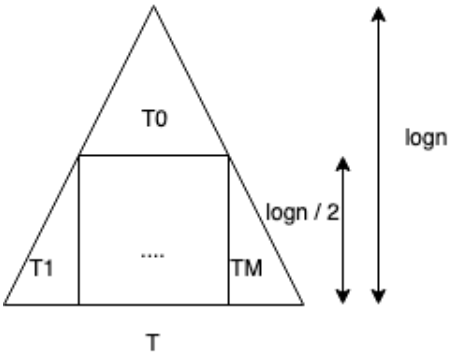
Instructor: *Alex Andoni* Scribes: *Prerna Kashyap*

1 Introduction

In the last lecture we saw that cache oblivious models are I/O models that are not dependent on M (size of the cache) or B (size of one block of consecutive addresses in the main memory). These models are also called external memory models. The **motivation** for such models was that we wanted algorithms that would work for any B and could deal with several layers of memory hierarchy. The **goal** is to minimize the number of times the main memory is accessed.

2 Van Emde Boas Layout

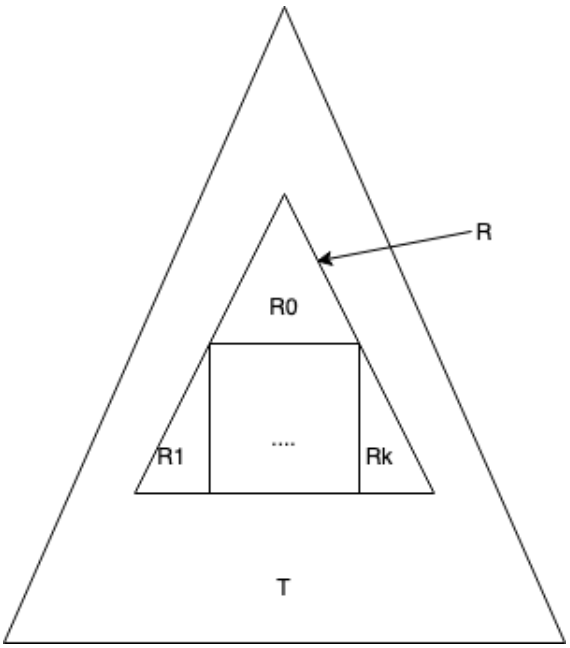
Let T be the tree such that $T_0, T_1, ..., T_M$ are its various subtrees as shown below. The height of the tree is *logn*, where n is the number of leaves. We know that $m \approx \sqrt{n}$. The idea is to store the subtrees recursively.



$vEB(T) = vEB(T_0) + vEB(T_1) + ... + vEB(T_M)$
$vEB(T_0) = vEB(T_{00}) + vEB(T_{01}) + ... + vEB(T_{0M})$... and so on.
The base case is until $|T| \leq$, then store explicitly.
The search procedure in vEB works like the generic search on T.

Claim 1. *The search procedure runs in $O(log_B n)$ time. Here time is the number of main memory accesses.*

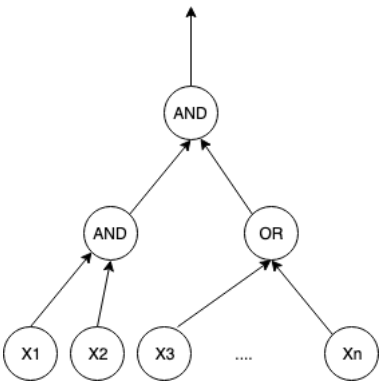
Proof. Consider the following vEB tree T and any subtree R.



Here, $|R| > B$
Also, $|R_0|, |R_1|, |R_2|, ..., |R_k| < B$, i.e. $|R_0|, |R_1|, |R_2|, ..., |R_k| \geq \sqrt{B}$. R will have height of atleast $logB$. Entire R_0 is stored consecutively in memory. Once the search process touches R_0, it passes through R_0 using ≤ 2 I/O transfers.
For any tree of size $>$ B, there are at most 4 I/O transfers.
So, the number of trees of the same type as R that we pass through is at most $\frac{logn}{logB}$.
So, the nunber of I/O transfers $\leq 4\frac{logn}{logB} = O(log_B n)$.
NOTE: vEB layout has numerous applications. One of them is predecessor search. □

3 Parallel Algorithms

Parallel algorithms can be executed a piece at a time on many different processing devices, and then combined together again at the end to get the correct result.
Consider the following 'circuit' which can be represented as a DAG (Directed Acyclic Graph, where all computation flows in one direction). It consists of input wires $(X_1, X_2, ..., X_n)$, internal wires, gates($\vee, \wedge$, etc) and output wire. Think of the inputs as binary inputs or bits.



Let p-time be the parallel time i.e. the depth of the circuit (longest path from input to output). In the figure above, the depth is 2. Let work done be equal to the number of gates. It can be thought of as the amount of hardware required.

The Base of Gates can be: $\{AND, OR, NOT\}$ or $\{NAND\}$. A gate is called universal if it can construct any computational function.

Fan-in = # of wires that go into a gate.

Depending on these parameters a few complexity classes can be defined:

- AC_i =problems solvable with p-time $O((logn)^i)$, unbounded fan-in and $n^{O(1)}$ work.
- NC_i =problems solvable with p-time $((logn)^i)$, $O(1)$ fan-in and $n^{O(1)}$ work.
- $AC = \cup_i AC_i$
- $NC = \cup_i NC_i$

Example 1. *$AND(X_1, X_2, ..., X_n) \in AC_0$ circuit because the p-time is 1 and the work done is $O(n)$. It $\notin NC_0$ but $\in NC_1$.*

Theorem 1. *For computing $XOR(X_1, X_2, ..., X_n)$ we need $\Omega(\frac{logn}{loglogn})$ depth of AC circuits, # of gates $\leq n^{O(1)}$ and unbounded fan-in.*

Claim 2. *$\forall f : \{0,1\}^n -> \{0,1\} \exists$ circuit with fan-in $\leq n$, 2^n gates and depth $O(1)$.*

4 PRAM: Parallel Random Access Machine

Let us assume that there are P processors, the size of memory is M and the computation is synchronous (i.e. there is a shared clock amongst processors). Let p-time be the number of clock ticks (equivalent to the synchronous steps) required to solve the problem.
When 2 CPUs attempt to read/write at the same memory location, this can either be disallowed by the hardware or a decision is made regarding which processor wins (for example: the one with the smallest id might win). Depending on this there are variations of this model:

- **Concurrent Read Concurrent Write (CRCW) PRAM**: Both simultaneous reads and both simultaneous writes of the same memory cell are allowed.
- **Exclusive Read Exclusive Write (EREW) PRAM**: No two processors are allowed to read or write the same shared memory cell simultaneously.
- **Concurrent Read Exclusive Write (CREW) PRAM**: Simultaneous reads of the same memory cell are allowed, but only one processor may attempt to write to an individual cell.

If we think of work as the energy being consumed, then it is equal to (# of processors) $*$ (p-time).

Theorem 2. *Even on CRCW, XOR requires* $\Omega(\frac{logn}{loglogn})$ *p-time.*

5 MPC: Massively Parallel Computation

A large number of processors is used to perform a set of coordinated computations in parallel. Each processor works on a chunk of its input and once in a while talks to other processors.
Let n be the input size, m be the number of machines/processors and let each machine have s space.

Remark 1. $m * s \geq O(n)$

Here we define p-time to be the number of rounds to solve the problem. We define each round to have two steps: performing any computation on local information and then shuffling data and distributing to other machines. The second step i.e. shuffling is an expensive operation and that is why we want to minimise the number of rounds.

Remark 2. *output traffic size* $\leq s$
input size $\leq s$
If output size $> s$*, also store in distributed fashion.*
If $s = n^{\epsilon}$, $m = O(n^{1-\epsilon})$
If every machine stores a routing table, $s \geq m$ *i.e.* $s \geq \sqrt{n}$

Example 2. *The p-time for XOR is 2 rounds if* $s = \sqrt{n}$*. In general* $O(log_s n)$ *rounds for an s-ary tree.*