COMS 4995-2: Advanced Algorithms (Spring'20) Apr 16, 2020

Lecture Lecture 22: Cache Oblivious Models, Parallel Algorithms

Instructor: *Alex Andoni* Scribes: *Kundan Guha*

1 Introduction

Topics Covered Today

- Cache Oblivious Model
- Parallel Algorithms

2 Cache Oblivious Model Continued

2.1 Problem: Search (in a search tree)

We have a set of numbers to search over, target $O(\log_B N + 1)$ cache line misses for searching for some number B.

2.2 Solution

BST (balanced), van Emde Boas layout
Layout: $\text{vEB}(T) = \text{vEB}(T_0), \text{vEB}(T_1), \ldots, \text{vEB}(T_{\sqrt{n}})$
Search: As usual

2.3 Analysis of Query

(1) Suppose subtree at recursion level e, is size $N^{(1/2)^e} \equiv k$. UB On traversing this subtree: $O\left(1 + \left\lceil \frac{k}{B} \right\rceil\right)$ is $k \in B \Rightarrow O(1)$ time (CLM).

Height: $\log k$

(2) # size-k subtrees: $O\left(\frac{\log N}{\log k}\right)$

3 Cache Line Transfers (in Cache-Oblivious Setting)

When cache is full, what to throw out? Rely on automatic *paging algorithm*:

- FIFO (First In First Out): Throw out the CL that was brought in earliest $\rightarrow$ Best when older memory is known to be constantly reused, so keep it in. Not great in other situations
- LRU (Least Recently Used): Throw out the pages that have been least recently used, still not optimal for all tasks.

Could be auto paging algorithms lead to performance *worse* than what the algorithm designer intends (thinking of the best CL to keep in cache)

Theorem 1. *For any algorithm A that has T cache line misses (time) on cache of size M, when optimally using the cache, if we run A with FIFO/LRU, on cache of size $2M$, the runtime is at most $2T$.*

Alternatively: A of auto paging on cache size M is "as good as" optimal paging on cache of size $\frac{M}{2}$ up to factor of 2.

This is an example of online algorithms,

- FIFO/LRU have to make decisions locally, without knowledge of future
- Compare them to algorithms making decisions, knowing the future

Measure of performance: *competitive ratio* $\rightarrow \frac{\text{Cost of Online Algorithm}}{\text{Cost of Future-Knowing Algorithms}}$

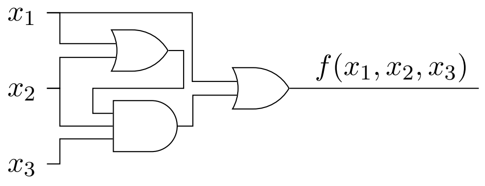
4 Parallel Algorithms

Solve a problem of size n by a number of computing units. Goal is to have faster (wall clock) runtime than just with one CPU. Any analysis of such algorithms inherently relies on the underlying computational model (as any analysis would), i.e. cores on a laptop vs cores on a super computer vs cores on a GPU, etc. In this class will mainly go over three such models.

4.1 First model: Circuits

Computes function f using gates.
Q1: Set of basis gates

- { OR, AND, NOT } which is enough to compute any f
- { NAND } which is equally enough to compute any f



Performance measures of circuits:

- Number of gates $\approx$ total time (in standard non-parallel model)
- Depth of circuit: Largest computational path from input (x) to output $\approx$ <u>parallel time</u> (wall-clock time)

Goal:

Reduce p time (depth) to ideally be much less than time in standard model, while keeping the number of gates bounded (not blow up in size)

Theorem 2. *For any function $f\colon \{0,1\}^n \mapsto \{0,1\}$, exist a circuit to compute f with depth of 4 and 2^{n+1} gates.*

Proof:

Can create a set of y inputs such that $f(y) = 1$ and use a line of AND gates to compare x inputs against all possible such y. Final result is depth of 4 using the total 2^{n+1} gate space to exhaustively output f.

There are two versions of circuit models, one allowing for unbounded fan-in (i.e. multi input gates are allowed) and the other with bounded fan-in. Bounded fan-in models may be mapped to unbounded fan-in versions through splitting / combining multi-input AND/OR gates to a set of connected AND/OR gates as necessary.

4.2 Complexity Classes

- AC: Problems solvable with circuits (unbounded fan-in) with $(\log N)^{O(1)}$ depth and $N^{O(1)}$ number of gates.
- NC: As above, but bounded fan-in
- AC$_k$: As AC but depth is $O\left((\log N)^k\right)$
- NC$_k$: As above but with bounded fan-in

Examples:

1. AND of n bits: $f(x_1, \dots, x_n) = \text{AND}(x_1, \dots, x_n)$
   - AC: Depth of 2
   - NC: Depth of $\log n$

     Proof: AND with bounded fan-in is a binary operation, chaining n bits requires $\log n$ total such chains.

2. XOR on n bits
   - AC: Depth possible is $O\left(\frac{\log N}{\log\log N}\right)$ for poly(N) # gates (if basis is {AND, OR, NOT})
   - NC: $O(\log N)$

5 Next Class

Go over the remaining two parallel models:

(2) PRAM: p processors, shared memory

(3) MPC (Map Reduce): p machines, local memory