# L5: I/O Extensions

scribe: *Gita Sreekumar*

Modern machines have complicated memory hierarchy. Levels get larger and slower further away from CPU. Operating system moves the data between levels using large blocks as needed.

## 5.1 External Memory Model

This is a standard model which defines a computer as having 2 levels:

- The cache which is near the CPU, cheap to access but limited in space
- The disk which is distant from CPU, expensive to access but nearly limitless in space
- N = No: of items in the problem instance
- B = No: of items per disk block
- M = No: of items that fit in the main memory
- T = No: of items in the output.
- I/O : Move block between memory and disk.

We assume that $M > B^2$.

### 5.1.1 Fundamental Bounds

| | $Internal$ | $External$ |
|---|---|---|
| $Scanning$ | $N$ | O((N/B) + 1) |
| $Sorting$ | $N \log N$ | O((N/B) $\log_{(M/B)}(N/B)$) |
| $Searching$ | $\log_2 N$ | O($\log_B(N/B)$) |

## 5.2 Cache Oblivious Algorithms

The principle idea of Cache oblivious algorithms is to design external memory algorithm without knowing B and M. External memory algorithm bound I/Os in terms of N, M, B. But in cache oblivious algorithm, no knowledge of B and M is required.

### 5.2.1 Advantages

- Cache oblivious algorithms perform well on multi level memory hierarchy without knowing any parameters of the hierarchy, only knowing the existence of a hierarchy. Equivalently, a single cache oblivious algorithm is efficient on all memory hierarchies simultaneously.
- Self tuning: Typical cache efficient algorithms require tuning to several cache parameters which are not always available from the manufacturer and often difficult to extract automatically. Parameter tuning makes code portability difficult. Cache oblivious algorithms can work on all machines without tuning.

### 5.2.2 Model Assumptions

Cache oblivious model assumes an ideal cache which advocates a utopian viewpoint: page replacement is optimal, and the cache is fully associative.

- Optimal page replacement-This specifies that the page replacement strategy knows the future and always evict the page that will be accessed farthest in the future. In contrast, the real world caches do not know the future and employ more realistic page replacement strategies such as evicting the least recently used (LRU) block or the oldest block (FIFO). algorithm is efficient on all memory hierarchies simultaneously.
- Tall cache-It is common to assume that a cache is taller than it is wide, that is, the number of blocks, M/B, is larger than the size of block, B.

### 5.2.3 Algorithms

#### Scanning

- Assumption: The data is lay out in contiguous part of the disk, in any order, and implement the N-element traversal by scanning the elements one-by-one in the order in which they are stored. This layout and traversal algorithm do not require the knowledge of B or M.
- Analysis: The main issue here is alignment: where the block boundaries are relative to the beginning of the contiguous set of elements in the disk. In the worst case, first block has just one element and last block has just one element. In between, though, every block is fully occupied, so there are at most [N/B] of such blocks, for a total of at most $\lceil \frac{N}{B} \rceil + 2$ blocks.
- Worst case bound: $\frac{N}{B} + 1$

#### Scanning

Divide and conquer repeatedly refines the problem size. Eventually, the problem will fit in cache and later, the problem will fit in a single block. While the divide and conquer recursion completes all the way down to constant size, the analysis can consider moments at which the problem fits in a cache and fit in block and prove the memory transfer is small in these cases. For a divide and conquer algorithm dominated by leaf cost, i.e. in which the number of leaves in the recursion tree is polynomially larger than divide/ merge cost, such an algorithm will use within a constant factor of the optimal number of memory transfers. On the other hand, if the divide and merge can be done using few memory transfers, then the divide and conquer approach will be efficient even when the cost is not dominated by the leaves.

#### Sorting

The soring bound for cache oblivious algorithm is $O(\frac{N}{B} \log_2 \frac{N}{B})$.

#### Median

Analysis: Internal Memory: $T(N) = N + T(\frac{N}{5}) + T(\frac{7}{10}N) = O(N)$
Cache obilvous algorithm:

- $Step1 + Step2 = 2(\frac{N}{B} + 2)$ I/Os
- $Step3 = T(\frac{N}{5})$
- Step 4 can be done with three parallel scans, one reading the array, two others writing the partitioned arrays. Parallel scans use O(N/B) memory transfers.
- Step 5 is a recursive call of at most $\frac{7}{10}N$.

Thus we obtain the following recurrence on the number of memory transfers:
$T(N) = O(\frac{N}{B}) + T(\frac{N}{5}) + T(\frac{7}{10}N) = O(\frac{N}{B})$
The whole algorithm did not need block size to implement it.

**Algorithm 5.2.1** Median$(D, N/2)$

Split data into N/5 sets of size 5
Find median of each set and store it to Q, a queue to write to memory
Recursively compute median m on Q: Median$(Q, Q/2)$
Split data set D into $L = (d \in D)|d < m$
$R = (d \in D)|D \geq m$
**if** $(|L| \leq N/2)$ **then**
  Median$(L, N/2)$
**else**
  Median$(R, N/2 - |L|)$

## 5.3 Parallel Disk Model (PDM)

Parallel Disk Model assumes a single CPU, single RAM and multiple disks. The two key mechanisms for efficient algorithm design in PDM is locality of reference (which takes advantage of block design) and parallel disk access (which takes advantage of multiple disks). In a single I/O, each of the D disks can simultaneously transfer a block of B contiguous data items. So, point of interest is whether it is possible to speed up the I/Os by a factor of D.
PDM assumes that the data for the problem are initially striped across the D disks and we require the final data to be similarly striped. Striped format allows a file of N data items to be input or output in $O(\frac{N}{DB})$ I/Os, which is optimal.
Scanning bound for PDM is $O(\frac{N}{BD})$
Sorting bound for PDM is $O(\frac{N}{BD} \log_{(M/B)} \frac{N}{B})$

## 5.4 Parallel External Memory (PEM)

The PEM is a computational model with P processors and P internal memories sharing an external memory or disk. Each internal memory is of size M and is partitions into blocks of size B and is exclusive to each processor. To perform any operation on a data, the processor must have the data in its internal memory. The data is transferred between disk and internal memory in blocks of size B.

Scanning bound for PEM is $O(\frac{N}{BP} + \log P)$
Sorting bound for PEM is $O(\frac{N}{BP} \log_{(M/B)} \frac{N}{B})$