Algorithms for Memory Hierarchies
Lecture 13

Lecturer: Nodari Sitchinava
Scribe: Emanuel Schrade

1 Parallel Distribution Sweeping Framework

1.1 General Framework

Let $d = min\left\{\sqrt{\frac{N}{P}}, \frac{M}{B}, P\right\}$

1. Partition space into d vertical slabs and P horizontal slabs with equal number of objects in each slabs.
2. Preprocess objects (specific for a problem) in each vertical slab using all processors.
3. Sweep each horizontal slab using one processor,
   process all horizontal objects that span at least one vertical slab,
   distribute objects into appropriate slab lists
4. Recursively solve problem on each vertical slab.

Base case: One processor per slab, run sequential I/O-efficient solution.
Figure 1 shows an example of the general framework. At the beginning one processor divides the space through $d = 4$ vertical slabs. In the next recursive call each of these four partitions is again divided into d parts. The recursion ends, when the base case, as defined above, is reached.

1.2 Orthogonal Line Intersection - First Attempt

As an example, we can compute intersections of orthognonal line segments using this framework:
Let $d = min\left\{\sqrt{\frac{N}{P}}, \frac{M}{B}, P\right\}$

1. Partition so that the number of endpoints is equal in each slab
2. Count for each portion of horizontal segments h_i, that spans σ_i, how many vertical segments h_i intersects in σ_i.

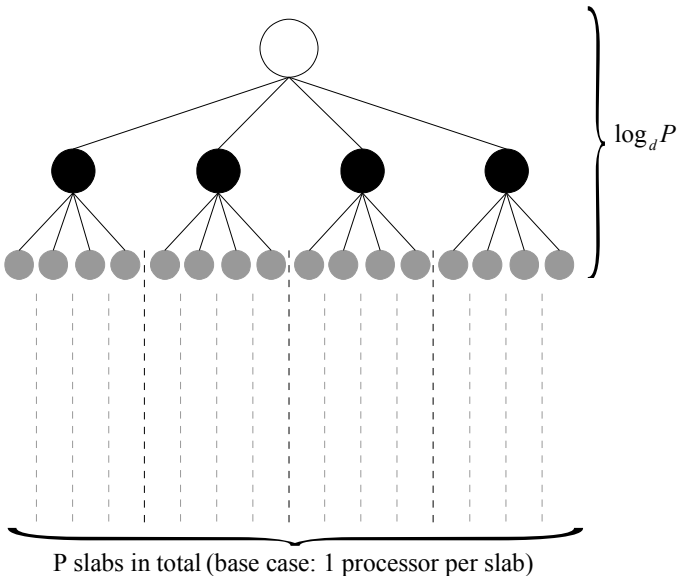

Figure 1: Recursion Tree of the algorithm

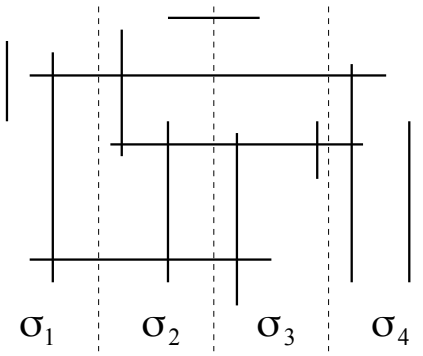

Figure 2: Orthogonal line intersection

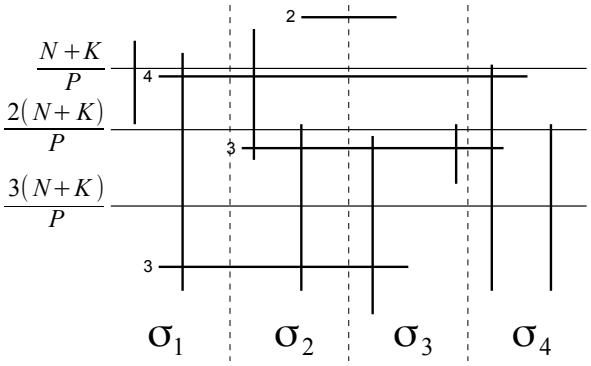

Figure 3: Division of the work using prefix sums.

3. Read just horizontal slab boundaries such that each slab contains $\theta\left(\frac{N}{P}\right)$ copies of horizontal segments. (using prefix sums)
   During sweep of horizontal slabs, each processor creates a copy of each horizontal segment h_i, that spans σ_i and intersects at least one vertical segment in σ_i or has endpoint in σ_i, in σ_i's slab list and creates copies of vertical segments in σ_i in σ_i's slab list

4. Recurse on slab lists.

Base case: Run sequential I/O-efficient line segment intersection solution.

1.3 Analysis

Per recursive call the I/O-complexity for steps two and three is:

2. $Q(N, P)$
3. $scan\left(\frac{N}{P}\right) + scan\left(K_k\right) \leq \mathcal{O}\left(\frac{N}{PB} + \frac{K}{PB}\right)$, if $P < \frac{N}{P \log N}$

Where K_k is the number of copies we create in the recursive call.
Base case:

$$\mathcal{O}\left(\frac{N'}{B} \cdot log_{\frac{M}{B}} \frac{N'}{B} + \frac{K'}{B}\right) = \mathcal{O}\left(\frac{N+K}{PB} \log_{\frac{M}{B}} \frac{N+K}{PB} + \frac{K}{PB}\right) \text{ with } N' = \Theta\left(\frac{N+K}{P}\right)$$

To ensure, that each processor has the same amount of work to do, we divide it using prefix sums, also accounting for the number of copies we have to create of each horizontal line. (see Figure 3)
In total the number of I/Os sums up to:

$$\sum_{k=1}^{\log_d P} \mathcal{O}\left(Q(N,P) + \frac{N+K}{PB}\right) + \mathcal{O}\left(\frac{N+K}{PB} \log_{\frac{M}{B}} \frac{N+K}{PB} + \frac{K}{PB}\right) \quad (1)$$

The number of intersections in step 2 can be caluclated using the following algorithm: (see Figure 4)

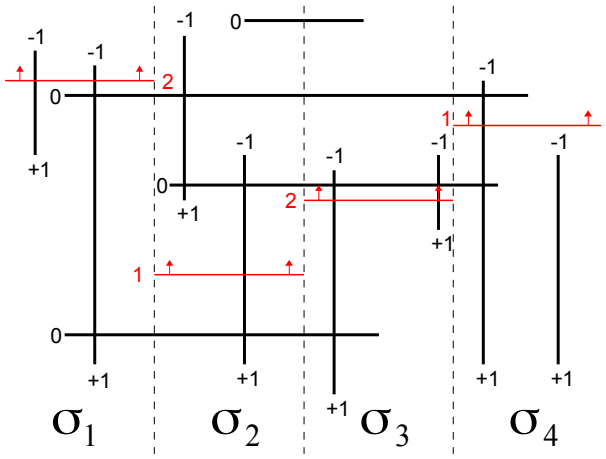

Figure 4: Prefix sums for counting the number of intersections

a) Set weights in each slab σ_i as follows:
   +1 on bottom endpoint
   -1 on top endpoint
   0 on horizontal segment spanning σ_i

b) Run prefix sums

The I/O-complexity of this algorithm is:

$$Q(N,P) = \mathcal{O}\left(\frac{N'}{PB} + \log P\right) \overbrace{=}^{P<\frac{N}{P\log N}} \mathcal{O}\left(\frac{N''}{PB}\right) \leq \mathcal{O}\left(\frac{N+K}{PB}\right) \qquad (2)$$

In this analysis we observe that N'' can be as large as N. However we did not create any copies in the current recursive step yet. There may only be up to K copies from previous recursive calls, so that in total $N'' \leq N + K$.
Inserting Equation 2 into Equation 1 leads to the following total I/O-complexity:

$$\begin{aligned} Total = \cdots &= \sum_{k=1}^{\log_d P} \mathcal{O}\left(\frac{N+K}{PB}\right) + \mathcal{O}\left(\frac{N+K}{PB}\log_{\frac{M}{B}}\frac{N+K}{PB}\right) \\ &= \mathcal{O}\left(\frac{N+K}{PB}\log_d P\right) + \mathcal{O}\left(\frac{N+K}{PB}\log_{\frac{M}{B}}\frac{N+K}{PB}\right) \\ &= \mathcal{O}\left(\frac{N+K}{PB}\log_{\frac{M}{B}}\frac{N+K}{PB}\right) \\ &= \mathcal{O}(sort_P(N+K)) \end{aligned}$$

The sequential algorithm took $\mathcal{O}\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B} + \frac{K}{B}\right) = sort(N) + scan(K)$ I/Os. Ideally, in the PEM model, we would like to have an algorithm using $sort_p(N)+$

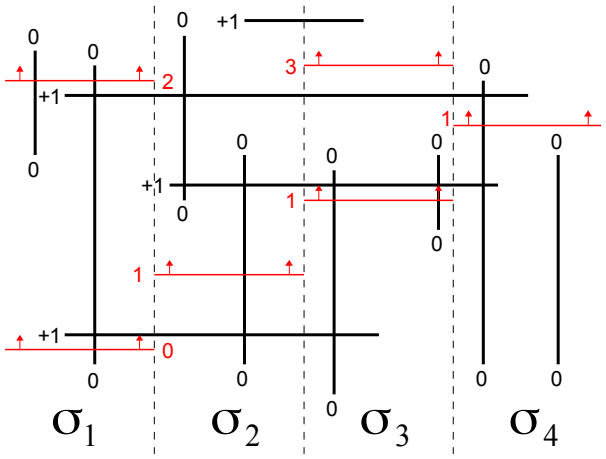

Figure 5: Prefix sums for counting the number of intersections

$scan_p(K) = \mathcal{O}\left(\frac{N}{PB}\log_{\frac{M}{B}}\frac{N}{B} + \frac{K}{PB}\right)$ I/Os. Hence, our first approach is not as efficient as it could be.

1.4 Orthogonal line intersection - Second Attempt

The following algorithm has an I/O-complexity of $sort_P(N)\log_d P + scan_P(K)$:
Let $d = min\left\{\sqrt{\frac{N}{P}}, \frac{M}{B}, P\right\}$

1. Partition input data so that the number of endpoints is equal in each slab.
2. Count the number of horizontal segments for each partition.
   Also count the number of intersections w_i of vertical segments with spanning horizontal segments.
3. Report intersections with spanning horizontal segments and create copies of segments for σ_i stablist if σ_i contains endpoints.
4. Recurse on slablist.

To ensure, that each processor reports the same number of intersections we can use prefix sums again. Figure 5 depicts an example. We compute the number of intersections by calculating the difference between the prefix sum at the endpoint and the prefix sum at the beginning point. I.e. in σ_3 the first vertical segment begins at a value of zero for the prefix sum, that is, no horizontal line was passed yet, and ends after passing two horizontal lines. Hence the vertical line intersects two horizontal lines. To prevent from big jumps in the array for computing the differences, which would be I/O costly, we sort the endpoints by their x-coordinates. Afterwards the differences can be caluclated by accessing consecutive elements in the array.

1.5 Analysis

The I/O-complexity of the algorithm is:

2. $Q(N, P) = \mathcal{O}(sort_P(N))$
3. $scan\left(\frac{N}{P}\right) + scan\left(\frac{K_k}{P}\right) = \mathcal{O}\left(\frac{N+K_k}{PB}\right)$

Base case:

$$\mathcal{O}\left(\frac{N'}{B}\log_{\frac{M}{B}}\frac{N}{B} + \frac{K'}{B}\right) = \mathcal{O}\left(\frac{N}{PB}\log_{\frac{M}{B}}\frac{N}{PB} + \frac{K'}{PB}\right)$$

The number of I/Os the algorithm performs in total is:

$$\begin{aligned}\sum_{k=1}^{\log_d P} \mathcal{O}\left(sort_P(N) + \frac{K_k}{PB}\right) + \mathcal{O}\left(\frac{N}{PB}\log_{\frac{M}{B}}\frac{N}{PB} + \frac{K'}{PB}\right) \\ = \mathcal{O}\left(sort_p(N)\cdot log_d P + \sum_{k=1}^{\log_d P}\frac{K_K}{PB} + \frac{K'}{PB}\right) \\ = \mathcal{O}\left(sort_P(N)\cdot\log_d P + \frac{K}{PB}\right)\end{aligned}$$

In the second step we use the fact, that $\sum_{k=1}^{\log_d P}\frac{K_K}{PB} + \frac{K'}{PB} = K$.