Algorithms for Memory Hierarchies
Lecture 12

Lecturer: Nodari Sitchinava
Scribe: Fabian Klute, Romain Gratia

1 List ranking in $O(sort_p(N))$

Last time we discussed List ranking in $O(sort_p(N)log^*(N))$ I/O's, today we will get rid of the $log^*(N)$ factor. As a general reminder the algorithm we saw last time did the following:

1. Find independent set of size $\Theta(N)$
2. Bridge out the nodes, which are in the independent set
3. Solve the problem recursively on the remaining items
4. Bridge the nodes back in

We showed how we can do this in the PEM model using a technique called deterministic coin tossing. But still we remain with a non constant factor of $\log^*(N)$ for the I/O's.
To understand the first step of our new algorithm we look back at the list ranking algorithm from last time. We can see that deterministic coin tossing is nothing else, than a colouring of the nodes and the number of repetitions of the colouring determine the number of colors. In fact if we repeat the process t times we get an $O(\log^t(N))$ colouring. We will call this a k colouring Now we will use the induced colors and sort the nodes by color. We get groups and want the first group G_0 to contain the most nodes. Now for each group G_i we add the group to the independent set and remove all their neighbours in other groups. Trivially we can do this at start with G_0, then with G_1 since all neighbours G_0 and G_1 have are already deleted and so on. Thus we get a valid independent set. The I/O complexity here is in $O(sort_p(N) + ksort_p(N)) = O((k + 1)sort_p(N)) = O(sort_p(N)\log^t(N))$. Now it looks like we did not win anything, still there is a logarithmic factor in the I/O complexity.

1.1 Delayed Pointer processing

To get pure sorting complexity we introduce a new technique called "Delayed pointer processing". We start as above.

1. Run the deterministic coin toss algorithm t times. $O(tsort_p(N))$
2. Store with each item their colors and ID's of neighbours $O(sort_p(N))$
3. Group the nodes by colors, such that G_0 contains the most nodes $O(sort_p(N))$

So far nothing special happened. As above we add G_0 to the independent set and, if a node in G_0 has a neighbour in another group, add to that group a duplicate. Now for each group do the following:

1. Sort the group and remove the duplicated nodes $O(sort_{p(N_i))})$
2. Add the remaining nodes to the independent set $O(scan_p(N_i))$
3. Add duplicates to the appropriate groups $O(scan_p(N_i)) + \log^t(N)$

The I/O complexity can be written as:

$$\begin{aligned}
&O(sort_p)(N) + \sum_{i=1}^{k-1} sort_p(N_i) + \log^t(N) \\
&= sort_p(N) + \sum_{i=1}^{k} \frac{N}{PB} \log_{\frac{M}{B}} \frac{N_i}{B} \\
&\leq sort_p(N) + \frac{1}{PB} \log_{\frac{M}{B}} \frac{N}{B} \sum_{i=1}^{k} N_i + (\log^t N)^2) \\
&= O(sort_p)(N)) + (\log^t N)^2)
\end{aligned}$$

Now we still have a logarithmic factor in our equation, but if we choose the number of processers appropriatly we can eliminate it. The question is for which values of p is the logarithmic factor dominating. Obviously we have to solve this inequation:

$$\frac{N}{PB} \log_{\frac{M}{B}} \frac{N}{B} < (\log^t N)^2$$

Shifting it to p yields:

$$p < \frac{N}{B(\log^t(N))^2} = O(\frac{N}{B \log N}$$

2 Parallel Distribution Sweeping

The aim here is to adapt the distribution sweeping technique to multicores architectures. The I/O complexity reached on this kind of architecture with p cores is at best:

$$O(\tfrac{N}{B} log_{\frac{M}{B}}(\tfrac{N}{B}) + \tfrac{k}{pB})$$

But in this lesson is going to limit to a solution whose I/O complexity is:

$$O(\tfrac{N+k}{B} log_{\frac{M}{B}}(\tfrac{N+k}{B}) = O(sort_p(N+k))$$

Let

$$d = min\{\sqrt{\frac{N}{p}}; \frac{M}{B}; p\}$$

2.1 General Algorithm

Algorithm:

1. Partition the space into d vertical slabs & p horizontal slabs with equal number of objects in each slab
2. Preprocess objects in each vertical slab using all processors (specific for a problem)
3. Sweep each horizontal slab using one processor. Process all horizontal objects that span ¿ 1 vertical slabs, distribute objects into appropriate slab lists
4. Recursively solve the problem on each vertical slab (allocate proportional number of processors)

Base case: one processor per slab runs sequential I/O efficient solution.

2.2 The Batched Stabbing Query Problem

Algorithm:

1. Partition the space into d vertical slabs & p horizontal slabs with equal number of objects in each slab
2. Count for each portion of horizontal segment h_i that spans σ_j how many vertical segments h_i intersects σ_j. Readjust horizontal slab boundaries such that each slab cointains $\theta(\frac{N}{p})$ vertical segment & $\theta(\frac{k}{p})$ copies of horizontal segment (using prefix sum).
3. Create a copy of each h_i that spans σ_j & intersect $\geqslant 1$ vertical segment or has endpoint in σ_j in σ_j's slab list. Create copies of vertical segments in σ_j in σ_j's slab list
4. Recursively solve the problem on each vertical slab (allocate proportional number of processors)

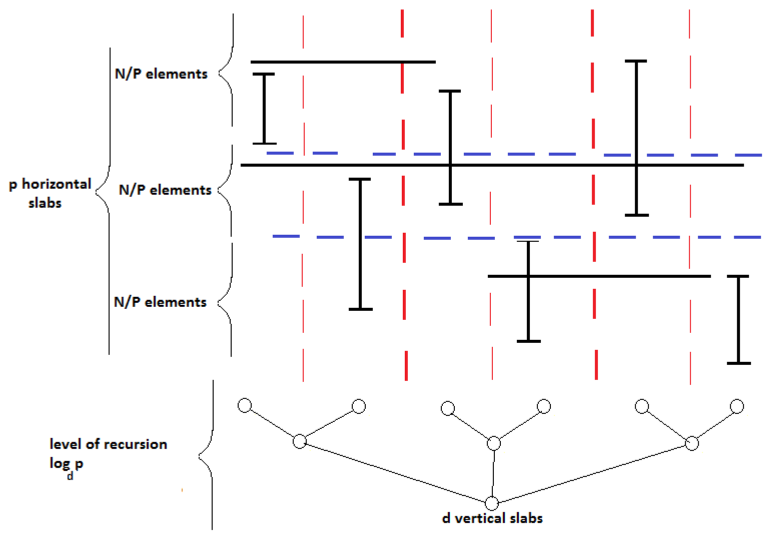

Figure 1: Divisions in slabs

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

Now we need to detail a little more the step 2:

a Set weights in each slab σ_j as follows: +1 on bottoms end point and -1 on top end point

b Run prefix sums: this allows to know how many an horizontal line intersects vertical lines as only sums which differs from 0 are those which are intersected:

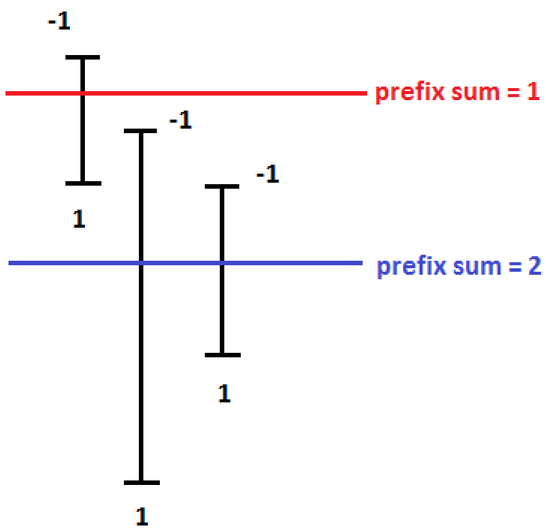

Figure 2: Prefix sum uses to count intresections

2.3 I/O complexity per recursive call in the case of The Batched Stabbing Query Problem

- Second step: Q(N,P)
- Third step:$scan(\frac{N}{p}) + scan(\frac{k_k}{p})) \leqslant O(\frac{N}{pB} + \frac{k}{pB})$

I/O complexity of base case: $O(\frac{N'}{B} log_{\frac{M}{B}}(\frac{N'}{B}) + \frac{k}{B})$

With ih this case: $N' = \theta(\frac{N+k}{p})$

Total I/O complexity:

$$Total = \sum_{k=1}^{log_d p} O(Q(N;p) + \frac{N+k}{pB}) + O(\frac{N+k}{pB} log_{\frac{M}{B}}(\frac{N+k}{pB}) + \frac{k}{B})$$

$Q(N;p) = O(\frac{N''}{pB} + logp) = O(\frac{N''}{pB}) \leqslant O(\frac{N+k}{pB})$

If we consider that the copies have been done.

$$Total = \sum_{k=1}^{log_d p} O(\frac{N+k}{pB}) + O(\frac{N+k}{pB} log_{\frac{M}{B}}(\frac{N+k}{pB}) + \frac{k}{pB})$$

$$Total = \sum_{k=1}^{log_d p} O(\frac{N+k}{pB}) + O(\frac{N+k}{pB} log_{\frac{M}{B}}(\frac{N+k}{pB}))$$

$$Total = O(\frac{N+k}{pB} log_d p) + O(\frac{N+k}{pB} log_{\frac{M}{B}}(\frac{N+k}{pB}))$$

$$Total = O(\frac{N+k}{pB} log_{\frac{M}{B}}(\frac{N+k}{pB}))$$

$$Total = O(sort_p(N+k))$$

In sequential I/O solution:

$$O(\frac{N}{B} log_{\frac{M}{B}} \frac{N}{B} + \frac{k}{B}) = sort(N) + scan(k)$$