Software Performance Engineering

Recitation 2.4

Sophia Sun
Tuesday, November 18, 2025

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

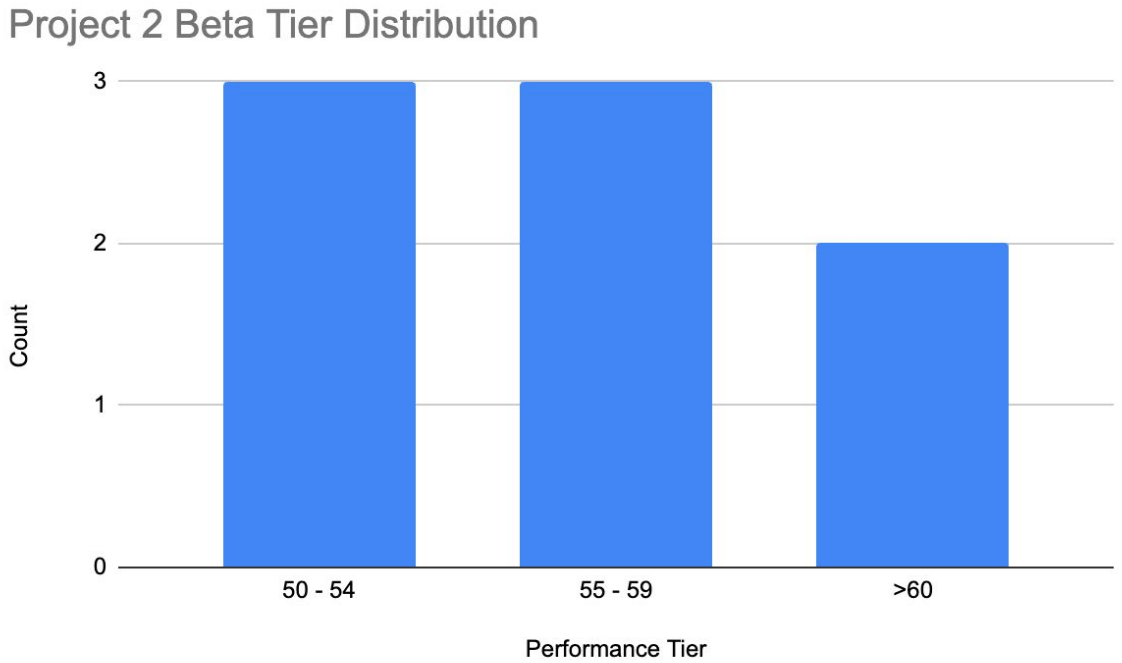
Some due dates

- Homework 6 is due Monday, December 1.
- Project 2 Final is due Thursday, December 4.
- Project 3A and 3B are due Monday, December 8.

- Don’t forget to do the weekly report.
  - You can talk about your project progress in your report.
- Review your team contract with your team member.
  - Revise it if necessary.
  - Make sure every member maintains good progress.

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Project 2 Beta



To get a B in final submission: reach tier 60

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Project 3A

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Project 3A

- For project 3A, the goal is to implement a GPU renderer
  - kinda similar to the project 2 renderer we're working on right now
- 3 parts:
  - Saxpy: correctly measuring GPU kernel runtime
  - Exclusive prefix sum -> Find_repeats
    - Input: [a0, a1, a2, a3… an]
    - Output: [0, a0, a0 + a1, a0 + a1 + a2… a0 + a1 + … + a(n-1)]
    - Can be used to find the location of duplicate values
  - GPU renderer
    - Implement a GPU renderer that renders predefined scenes

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Exclusive Prefix Sum

- Parallel exclusive prefix sum -> Find repeats
- How it works: input + marker + exclusive sum = dup indices
  - Index: [0, 1, 2, 3, 4, 5, 6]
  - Input: [1, 1, 2, 4, 4, 4, 5]
  - Marker: [0, 1, 0, 0, 1, 1, 0] -> mark duplicate as 1
  - Exclusive scan: [0, 0, 1, 1, 1, 2, 3] -> where to write the output
  - Output: [1, 4, 5] -> duplicates are at idx 1,4,5
- Based on the same approach, you can easily collect the index of any kind of marked values.

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Exclusive Prefix Sum

- Parallel exclusive prefix sum -> Find repeats
- How it works: input + marker + exclusive sum = dup indices
  - Index: [0, 1, 2, 3, 4, 5, 6]
  - Input: [1, 1, 2, 4, 4, 4, 5]
  - Marker: [0, 1, 0, 0, 1, 1, 0] -> mark duplicate as 1
  - Exclusive scan: [0, 0, 1, 1, 1, 2, 3] -> where to write the output
  - Output: [1, 4, 5] -> duplicates are at idx 1,4,5
- Based on the same approach, you can easily collect the index of any kind of marked values.

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Exclusive Prefix Sum

- Parallel exclusive prefix sum -> Find repeats
- How it works: input + marker + exclusive sum = dup indices
  - Index: [0, 1, 2, 3, 4, 5, 6]
  - Input: [1, 1, 2, 4, 4, 4, 5]
  - Marker: [0, 1, 0, 0, 1, 1, 0] -> mark duplicate as 1
  - Exclusive scan: [0, 0, 1, 1, 1, 2, 3] -> where to write the output
  - Output: [1, 4, 5] -> duplicates are at idx 1,4,5
- Based on the same approach, you can easily collect the index of any kind of marked values.

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

GPU Renderer

- Given a list of circles (yes, 2d circles), render them and output an image
  - similar idea as project 2, but on GPU
  - which gives a lot more chance for parallelism
- 2 steps:
  - fix existing bugs to have a working solution
  - optimize it to make it run faster

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

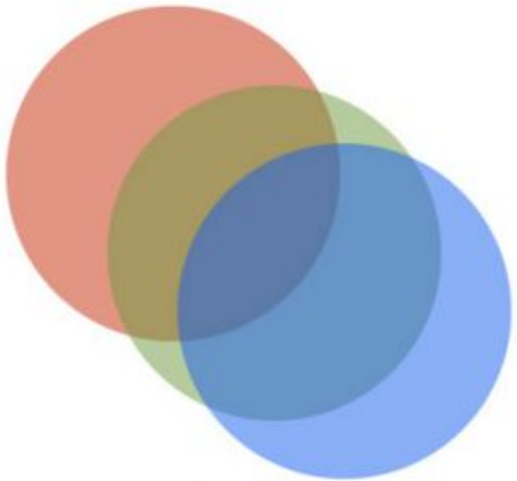
GPU Renderer

- Atomic coloring:
  - only 1 thread should be coloring 1 pixel at the same time
  - potential race condition - how to resolve?
- Color dependency:
  - for a pixel, it should be colored in the circle index order
  - which limits how you can parallelize it - why?

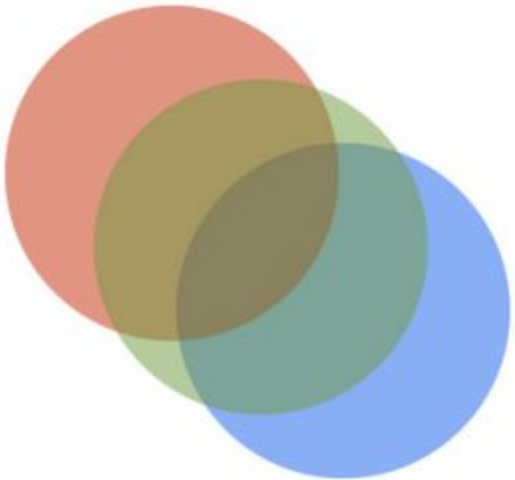
1. Compare and Swap
2. Exclusive Scan

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

GPU Renderer



Correct order:
blue over green over red



Incorrect order

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

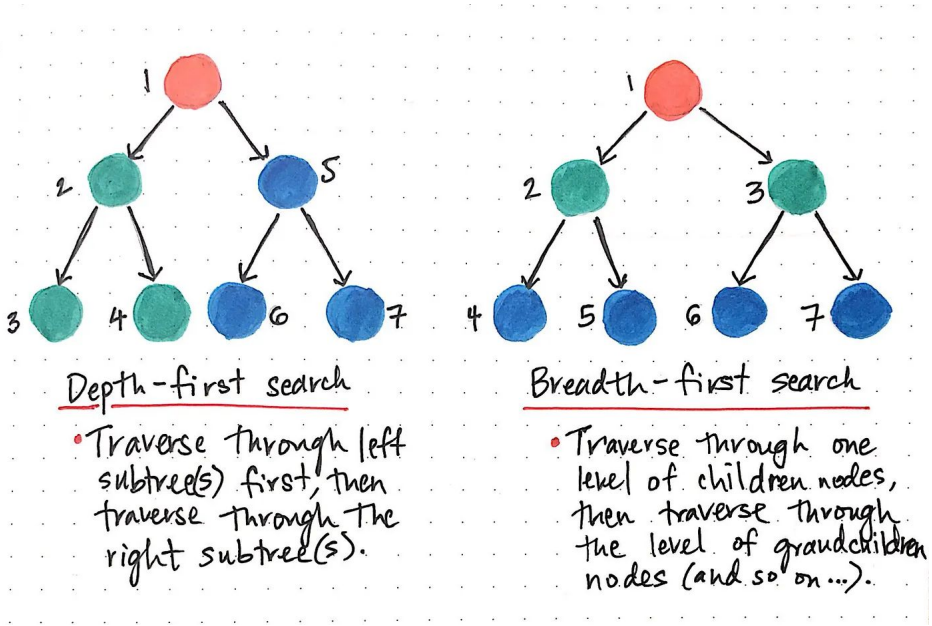
Project 3B

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Big Graph Processing

- For project 3B, implement 3 types of breadth-first search
  - Top-down
  - Bottom-up
  - Hybrid

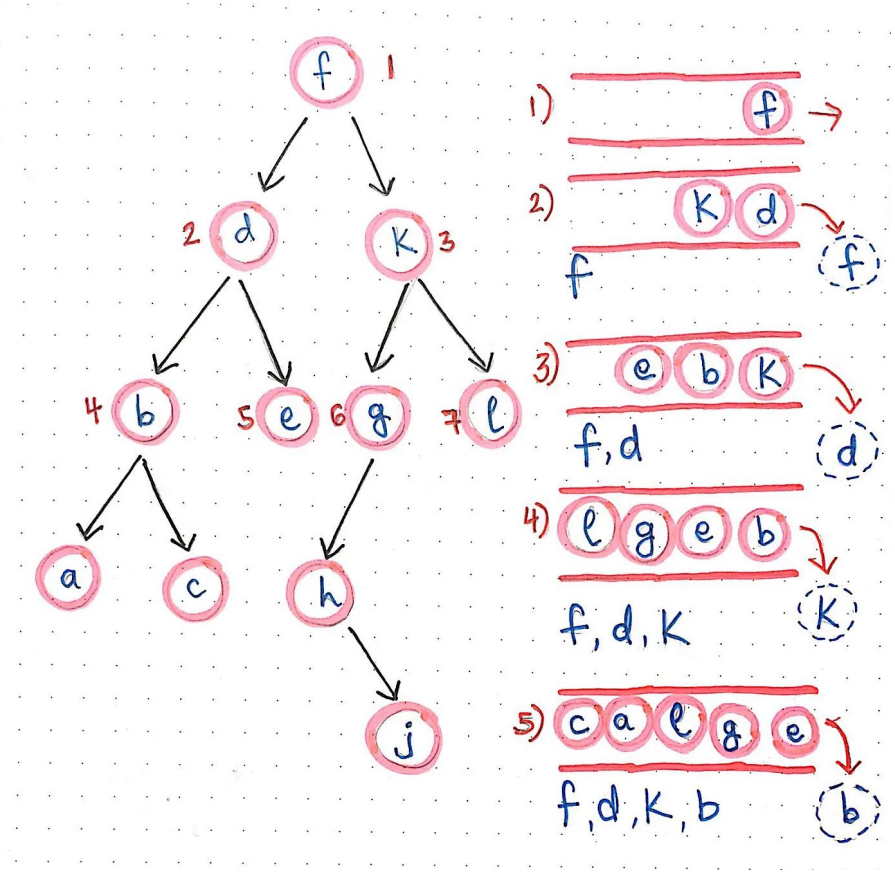
You'll work with million-scale dataset, so **be fast!**



© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Top-down BFS

1. Start from root
2. Visit and mark all the neighbors
3. Done with root
4. Move on to each of the neighbors
5. Keep going until all nodes are marked as visited



© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Bottom-up BFS

```
for each vertex v in graph:
    if v has not been visited AND
    v shares an incoming edge with a vertex u on the frontier:
        add vertex v to frontier;
```

- Instead of checking neighbors of all nodes on the frontier
- Check all unvisited nodes, if they have any neighbor on the frontier
- If yes, mark this node as visited, put to next frontier

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

Hybrid BFS

- Time complexity of the previous 2?
- Top down BFS: go through each node and each edge
  - O (V + E)
- Bottom up BFS: go through some nodes and some edges
  - Worst case: O(V + E)

- In some cases, Top down BFS is faster
- In other cases, Bottom up BFS is faster
- Tune your program to find the best balance!

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers

CODING TIME!

© 2008–2024 by the MIT 6.172 and 6.106 Lecturers