Introduction

This report outlines and analyses an implementation of a parallel algorithm for the following problem: Given a grid, where each square has a positive integer as its cost, find the shortest 8 connected path from the top left most square to the bottom right most square, where the distance of a path is defined as the sum of costs of all squares on the path.

The algorithm I've devised is a modified version of the delta stepping algorithm from Mayer and Sanders [1], described in the following pseudocode:

```
1: x_end ← x_size − 1
2: y_end ← y_size − 1
3: function FindDelta( )
4:     for x ← 0 to x_end do in parallel
5:         for y ← 0 to y_end do in parallel
6:             cost_max ← Max(cost_max, Cost(x, y))
7:         end for
8:     end for
9:     return cost_max/8
10: end function
11: distance[0][0] ← Cost(0, 0)
12: delta ← FindDelta()
13: function Relax(to-relax, relaxed, removed)   ▷ pass NULL as removed will cause the function to only Relax heavy edges. Pass a set as removed will cause the function to call Remove for each coordinate in to-relax and add them to removed.
14:     for each (x, y) ∈ to-relax do
15:         if x = -1 then                       ▷ Marked as removed already.
16:             continue
17:         end if
18:         if removed ≠ NULL then
19:             Remove(x, y)
20:             removed ← removed ∩ (x, y)
21:         end if
22:         for dx ← −1, 0, 1 do
23:             for dy ← −1, 0, 1 do
24:                 x_next ← x + dx
25:                 y_next ← y + dy
26:                 if (x_next, y_next) is valid coordinate ∧ (x_next, y_next) ≠ (x, y) then
27:                     if (Cost(x_next, y_next) ≤ delta) ⊕ (removed = NULL) then   ▷ If removed is null, only relax heavy edges, otherwise only relax light edges.
28:                         distance_new ← distance[x][y] + Cost(x_next, y_next)
29:                         if distance_new < distance[x_next][y_next] then
30:                             distance[x_next][y_next] ← distance_new
31:                             relaxed ← relaxed ∩ (x_next, y_next)
32:                         end if
33:                     end if
34:                 end if
35:             end for
36:         end for
37:     end for
38: end function
```

Note that $\textsc{Cost}(x, y)$ returns the cost of the cell on x_{th} row and y_{th}. Furthermore, while the cost takes a long time to calculate for the first time. This function caches the cost in a 2D array so subsequence call returns instantly. This has the additional effect of calculating the cost of each cell upfront in a parallel fashion during the find_ delta procedure. Not only does this allow delta to be calculated at the optimal value. It also allows the load of calculating the cost of each cell to be spread evenly among all processors, taking advantage of parallelization for the most expensive part of the algorithm. Furthermore, to ensure that relaxed and removed can be queried in constant time as well as being made emptied in constant

time, I implemented them as 2D int arrays where (x, y) is in the set if set[x][y] = true-value. Here true value is an int. By incrementing true-value by 1 we can empty the set very quickly.

Methodology

To confirm and measure the performance of my algorithm, I've

Experiments

Discussion

Conclusion

References

[1] U. Meyer and P. Sanders, ``Δ-stepping: A parallelizable shortest path algorithm,'' *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 1998, 1998 European Symposium on Algorithms, ISSN: 0196-6774. DOI: https://doi.org/10.1016/S0196-6774(03)00076-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196677403000762.