Project Proposal: Parallel Convex Hull

COMS 4995 Parallel Functional Programming

Andrei Coman

November 27, 2022

1 Introduction

In this project, I will work on the Convex Hull Problem: given a large dataset D of points in the xy-plane, find the smallest convex polygon which covers all the points in D. While there are many efficient solutions to this problem (most of which have a linear time complexity in the size of the dataset), the constants in their runtimes are large due to expensive arithmetic operations and branching. For this reason, I will parallelize one of these algorithms and analyze the performance improvement.

2 Background

The algorithm that I will parallelize is known as Graham's algorithm [2, pg.1029] and performs the following operations at a high level:

- Identify a point on the convex-hull (usually the leftmost/bottommost point)
- Take that point to be the origin and sort all the remaining data points in clockwise order
- Iterate through the dataset while also maintaining a stack; at every step, insert a new point and pop the stack as long as necessary to maintain its convexity

This is my complete C++ implementation of the idea described above:

C++ Convex Hull Sequential Implementation

```
int n;
struct Point{double x, y;} v[1 + MAXN];
bool cmpXY(Point A, Point B){
    return (A.x < B.x ||
            (A.x == B.x && A.y < B.y));
}
bool cmpTan(Point A, Point B){
    if(A.x == v[1].x && B.x == v[1].x)
        return A.y < B.y;
    if(A.x == v[1].x) return 1;
    if(B.x == v[1].x) return 0;
    return (A.x - v[1].x) * (B.y - v[1].y) <
           (B.x - v[1].x) * (A.y - v[1].y);
}
bool convex(int a, int b, int c){
    float S = (v[a].x * v[b].y + v[b].x * v[c].y
                              + v[c].x * v[a].y)
            - (v[a].x * v[c].y + v[b].x * v[a].y
                              + v[c].x * v[b].y);
    return S < 0;
}

int Stack[1 + MAXN], StackSize;
void convexHull(){
    std::sort(v + 1, v + n + 1, cmpXY);
    std::sort(v + 2, v + n + 1, cmpTan);

    Stack[++StackSize] = 1;
    Stack[++StackSize] = 2;
    for(int i = 3; i <= n; i++){
        while(!convex(Stack[StackSize - 1],
                      Stack[StackSize],
                      i))
            StackSize--;
        Stack[++StackSize] = i;
    }
}
```

3 Parallelism

Before parallelizing the algorithm, I make the following slight modification: Instead of computing the entire convex hull in one pass through the dataset, I separate the computation into two stages - computing the upper hull and the lower hull.

Using this version of the algorithm, I first fragment the xy-plane into vertical strips. Then, each thread-/spark is assigned the computation of the convex hull for one particular strip using the modified Graham's algorithm. In the end, the main thread collects all the "partial" convex hulls and performs a merging algorithm on them.

To merge two linearly separated convex hulls (which is true in our case since we aligned the points according to the vertical strips), it is enough to find their common upper-tangent and lower-tangent. This is made easier by the previous modification [3]:

- Knowing the upper/lower hull for all fragments, we know the leftmost/rightmost points for all hulls.
- At a merge, we start with the line between the rightmost point of the left hull and the leftmost point at the right hull. We move along the hulls iteratively until we find the common tangent.

If time permits, I will also consider parallelizing the merge operations in a divide-and-conquer approach - merge the first half of the hulls and merge the second half of the hulls in parallel, then merge the result.

4 Expected Results

I expect this idea to allow for a considerable parallel speedup for large datasets when combined with Haskell Vectors/Arrays. This is because the more costly arithmetic operations can be parallelized almost completely, while merging should be relatively inexpensive. I have also found a similar parallel implementation in Python [1] which gives good results.

Due to the nature of this algorithm, the distribution of the points in the xy-plane is not very significant, so large datasets can be generated randomly.

References

[1] Miller Russ Chen, Weiyang. Parallel Implementation of the Convex Hull Problem. https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Weiyang-Chen-Spring-2020.pdf.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[3] Petr Felkel. Convex Hulls. https://cw.fel.cvut.cz/b181/_media/courses/cg/lectures/04-convexhull.pdf.