Parallel algorithms and parallel computers (in4026)

Assignment A

Sverre Rabbelier
srabbelier@gmail.com
1308211

Introduction

Asked is to provide an algorithm that, given a list of unique numbers, produces the lowest number for the first/last i numbers, with i ranging from from 1 to n.
My implementation does this by first calculating for each pair of numbers which is the lowest. It repeats this log(n) time, resulting in a log(n)*n dimensional array. This step will hereafter be

described as the prefix step. Subsequently, it uses this array to figure out what the minimum for the entire row is by starting at the bottom of the array, and using the previously calculated minima as a lower bound. This second step will hereafter be described as the minima step. My algorithm requires that the input is a power of two.

Time-complexity analysis

The algorithm consists of two steps, the prefix and the minima step.

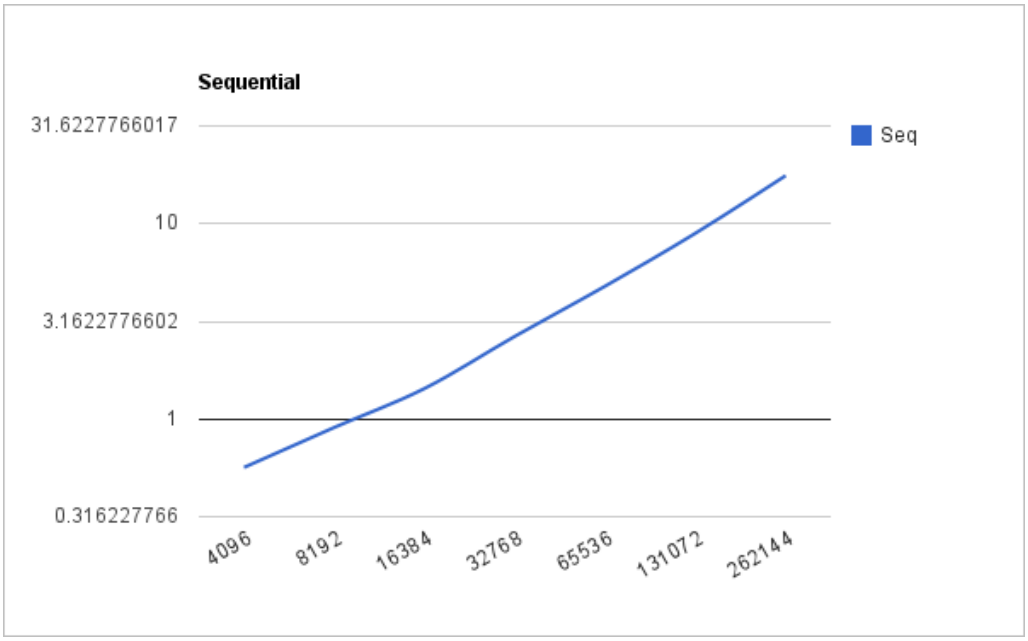
The prefix step consists of two loops. The first loop runs in O(1,n) work-time complexity. The second loop is executed log(n) times, and contains an O(1,n) inner loop resulting in a O(log(n), n*log(n)) work-time complexity for the second loop. The prefix step thus has a O(log(n), n*log(n)) work-time complexity.

The minima step consists of one loop which is executed log(n) times, containing an O(1,n) inner loop, resulting in a O(log(n), n*log(n)) work-time complexity for the minima step.

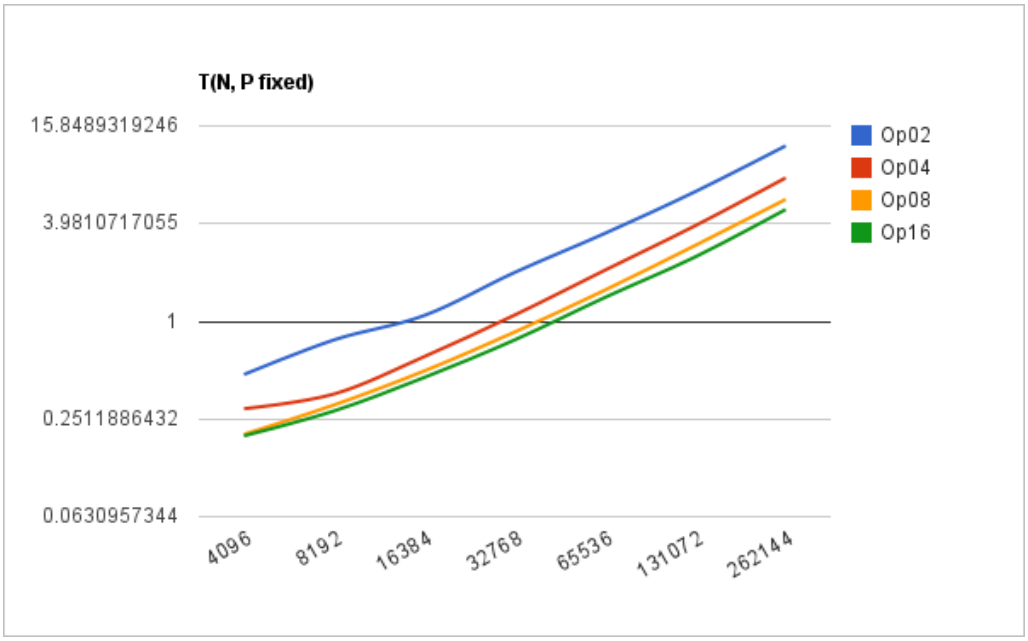
The algorithm thus has a O(log(n), n*log(n)) work-time complexity.

Testing

First I tested plotted the linear set to get a baseline. I chose a log scale because that works nicely with the exponentially increasing input size.

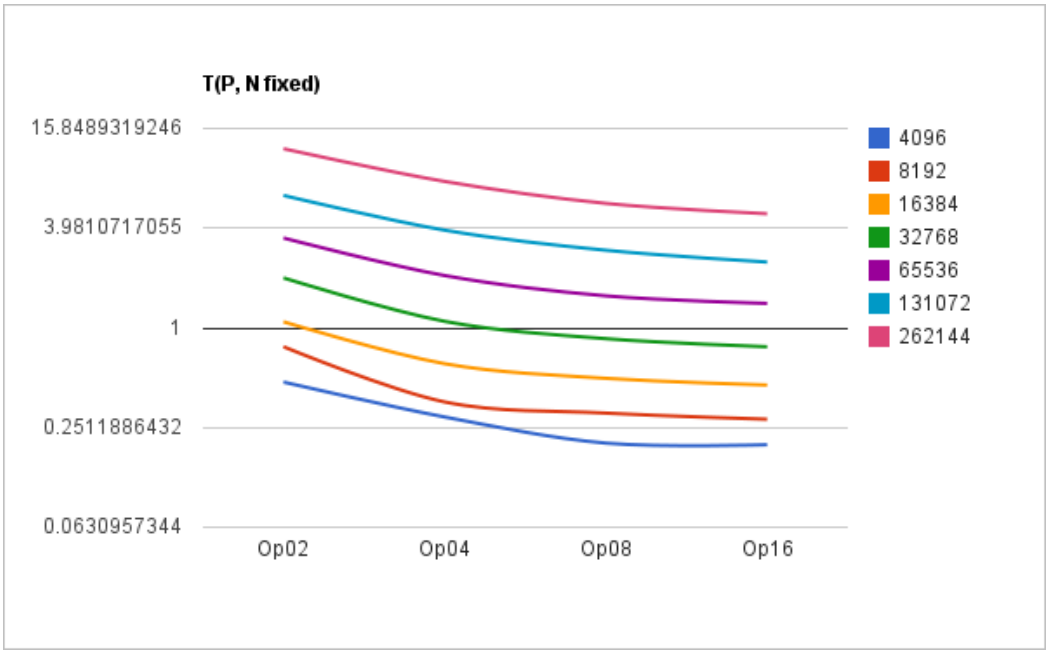


Now I'd expect the OMP to show something similar but of course with lower values for a higher thread count. Indeed, we find that OMP performs as expected:

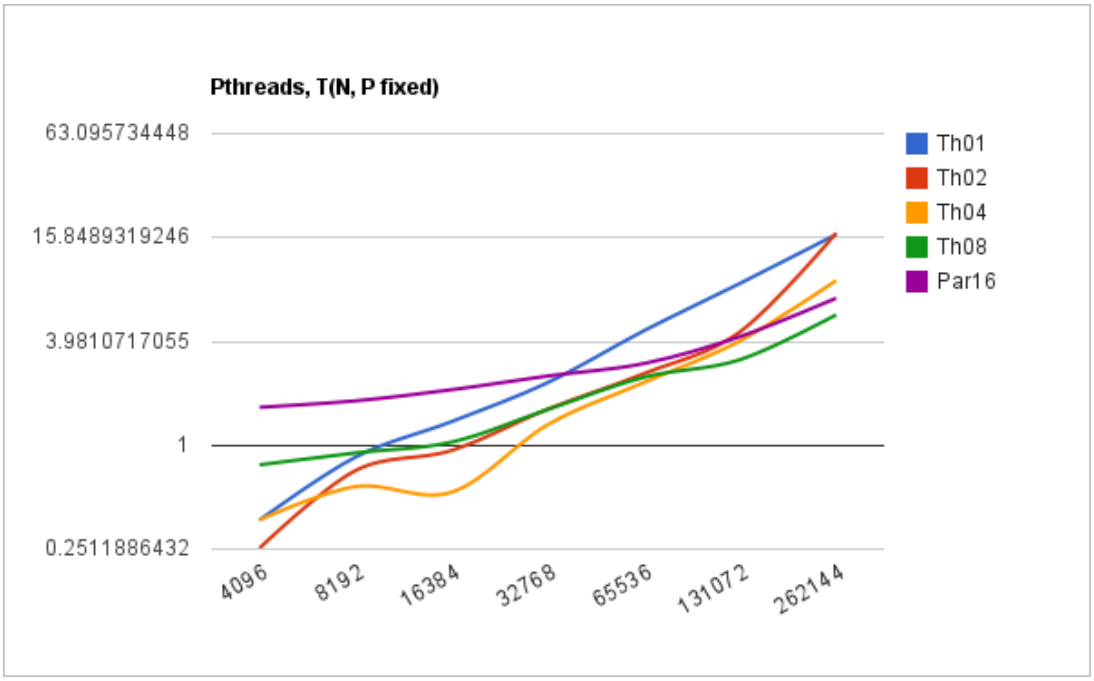


If we instead plot the threads against N we find a similar pattern, although there seems to be a tapering off when approaching 16 threads (which makes sense since the machine I tested on

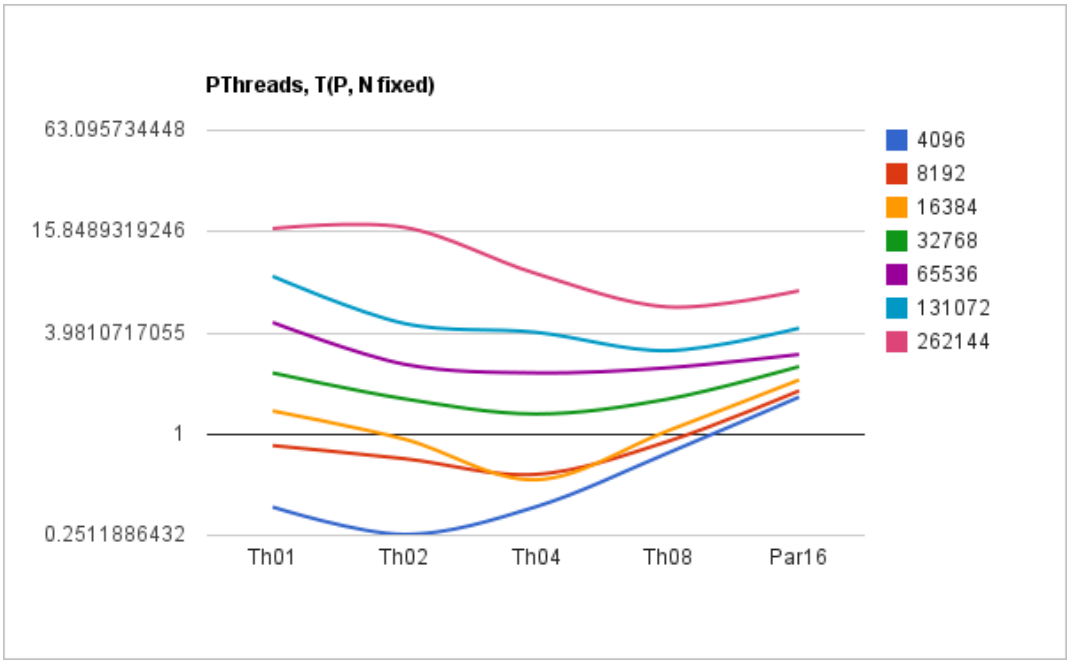
has 8 cores).



For pthreads I would hope for a similar result, but alas the result is all over the charts. There seems to be a general upward trent (as expected), but the results are far from consistent:



Keeping n fixed but varying the number of threads again gives a similarly distorted image. There is an observable upward trend for small n (whereas it should be downward).



Conclusions

As I expected, hand-crafting an optimized program with pthreads is _a lot_ harder than just using OpenMP. I suspect that in particular that OpenMP dealt better with the input size varying so much per iteration of the minima step than my hand-written pthreads solution. Using OpenMP was _a lot_ easier than pthreads here in particular, because in the minima step I had to recalculate the start/end step each time through the loop.