COL380 Lab-1

Question1

Used binary tree based algorithm for calculating prefix sum. Implemented binary tree using the input vector itself. Algorithm just contains 2 loops, 1 for going bottom-up in the binary tree and other for going top down in the binary tree.

First loop of the algorithm:

```
for( d=0 ; d<log2(n) ; d++ ){

   #pragma omp parallel for schedule(static)
   for( i=0 ; i<n ; i += 2^(d+1) ){
      input[i + 2^(d+1) – 1]+=input[i + 2^(d) -1];
   }

}
```

This loop goes bottom-up in the binary tree and adds the value of left children to each node.

Second loop of the algorithm:

```
for( d=log2(n)-2 ; d>=0 ; d-- ){

   #pragma omp parallel for schedule(static)
   for( i=2^(d) ; i<n ; i += 2^(d+1) ){
      input[i + 2^(d+1) -1]+=input[i + 2^(d) -1];
   }

}
```

This loop goes top-down in the binary tree and adds the values of parent to each node.
Although the update statements look same in the 2 loops but the starting values of i and d are different in the 2 loops and also the direction of change in d.

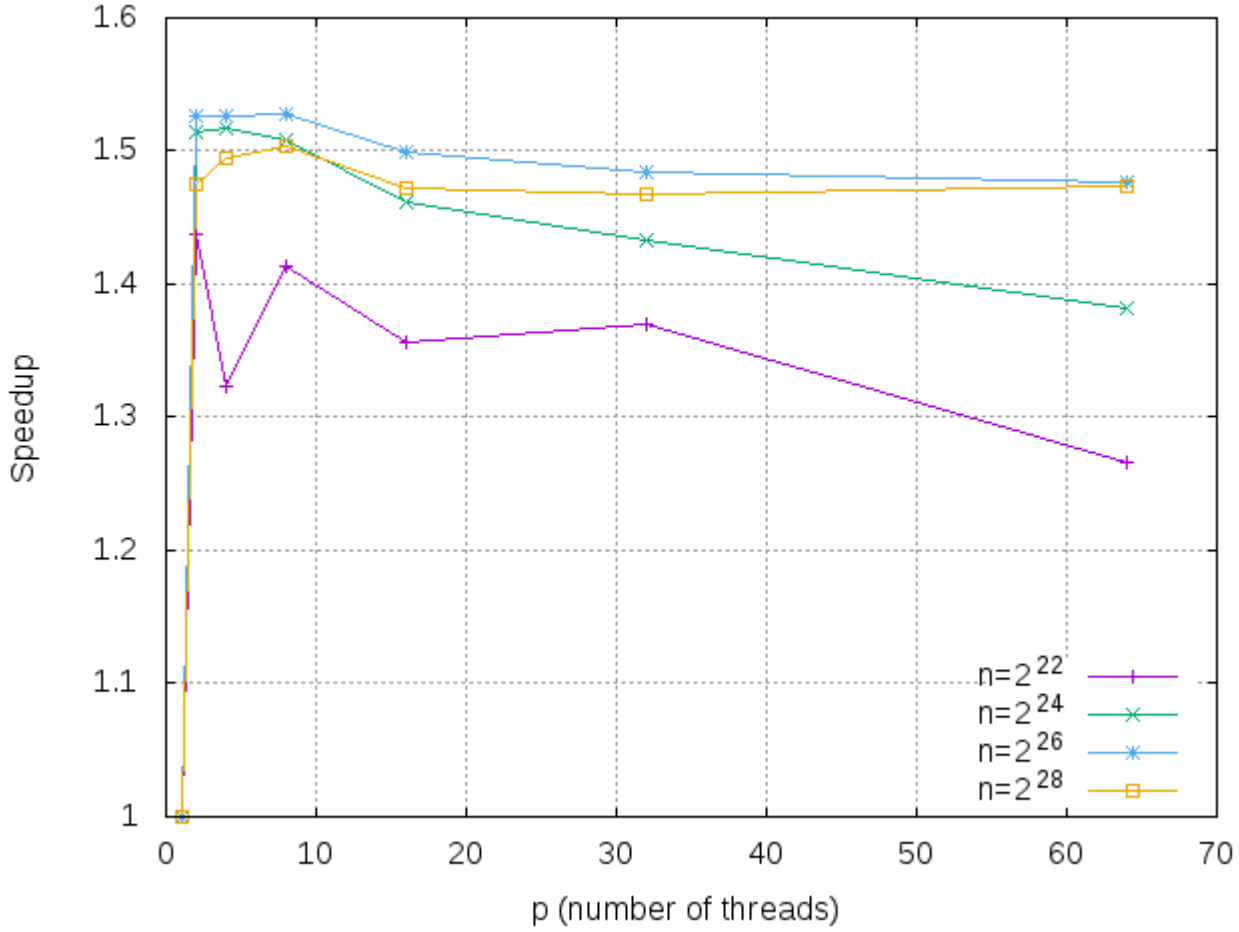
The total amount of work done by the algorithm is $\mathcal{O}(n)$ in $\mathcal{O}(logn)$ phases.

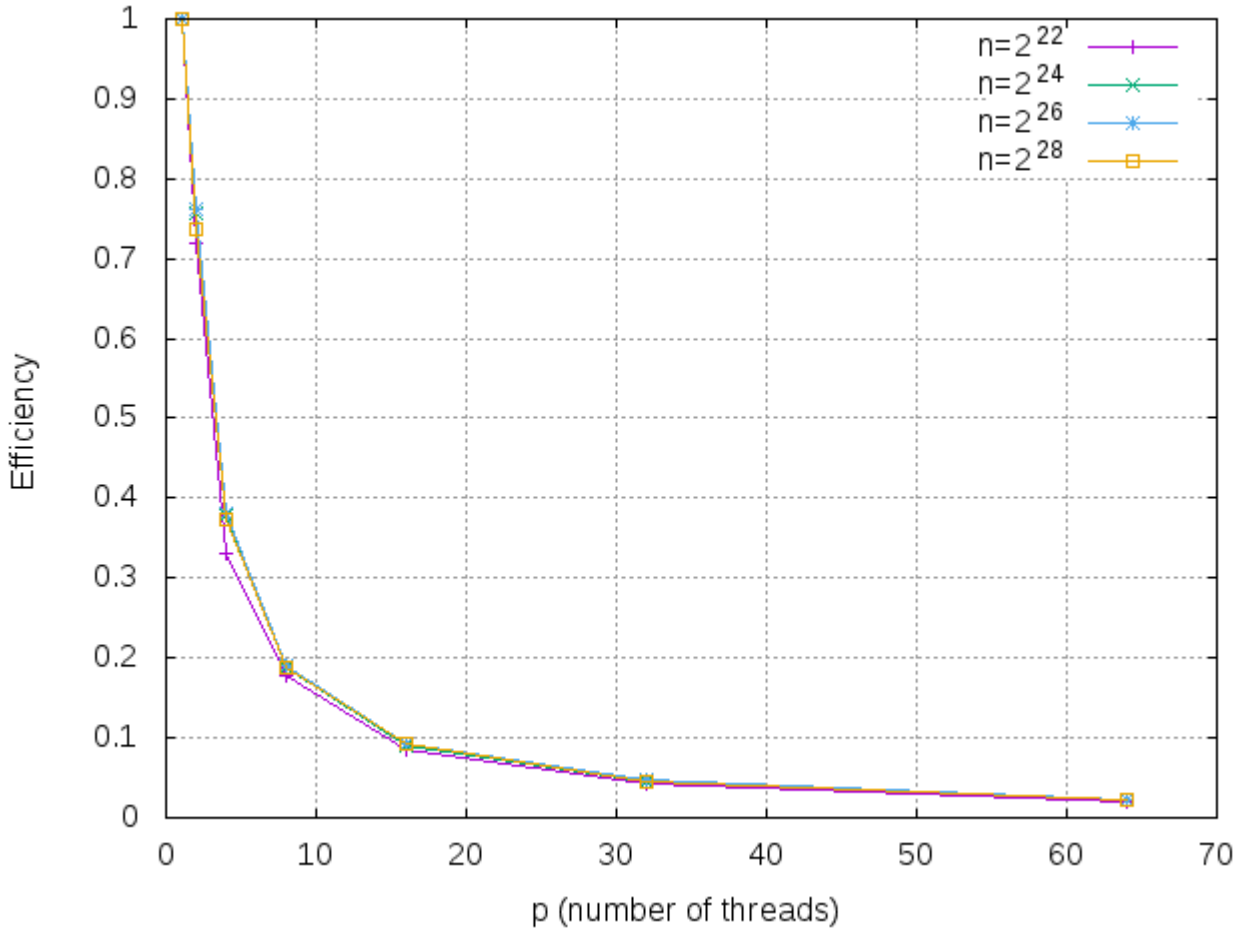
Used schedule(static) because it divides the loop between threads in continuous chunks, this prevents false sharing. Used first private for common variables accessed by threads so that threads don't access common variables.

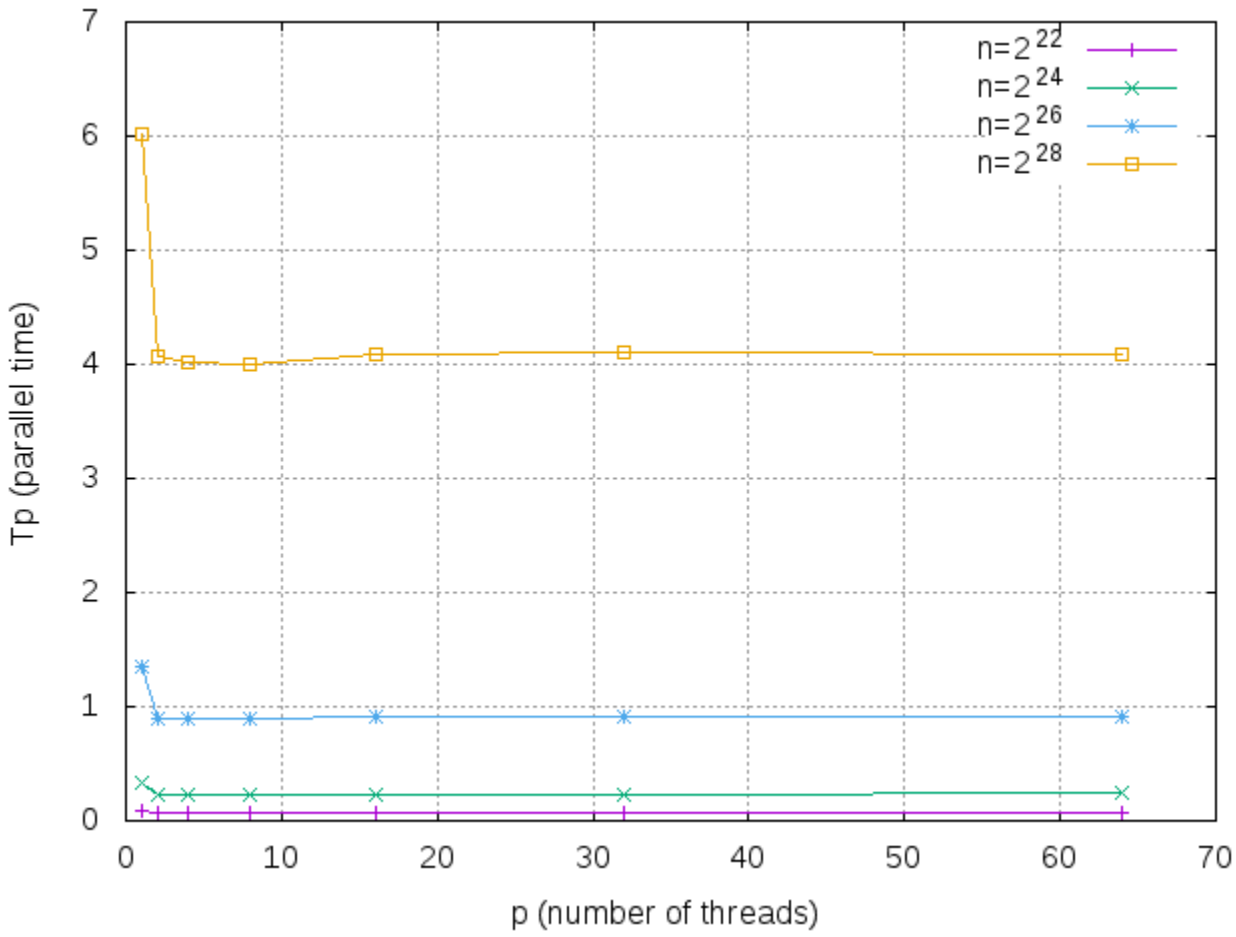
When n was not a power of 2 , did padding but did not actually pad 0 at end of input vector. Just skipped the statement when index>=n was being accessed. This works because the extension of vector would have contained zeros initially and we don't care what it contains after the computation is done.

The code was tested on input sizes $n = 2^{22}, 2^{24}, 2^{26}, 2^{28}$ and the input arrays were randomly generated. The code was run a machine with 2 physical cores and 7.7GB of RAM. The time was calculated after averaging over multiple runs.

The graphs for speedup, efficiency and time taken are:


Speedup
p (number of threads)
n=2^22
n=2^24
n=2^26
n=2^28
1.6
1.5
1.4
1.3
1.2
1.1
1
0
10
20
30
40
50
60
70


Efficiency
p (number of threads)
n=2^22
n=2^24
n=2^26
n=2^28
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
60
70


Tp (parallel time)
p (number of threads)
n=2^22
n=2^24
n=2^26
n=2^28
0
1
2
3
4
5
6
7
0
10
20
30
40
50
60
70