Introduction to Parallel & Distributed Programming
COL380 -Programming Assignment 1

Ankesh Gupta
2015CS10435

Parallel Prefix Sum

Data Structures and Design Decisions

1. The algorithm was adapted from *Work Efficient Prefix Sum* algorithm on *Wikipedia*.
2. It starts by computing sums of consecutive items(first item has even position).
3. Recursively, prefix sum of new $n/2$ elements are calculated and combined with original to get sum of n elements.
4. For problem size of n and p resources, partial prefix sum is calculated in $O(n/p)$ time and $O(n)$ work.
5. The p partial sums are then combined using the *wiki* algorithm mentioned in point 1 in $O(\log p)$ time and $O(p)$.
6. Again, those p partial sums are reflected back on original array in $O(n/p)$ time and $O(n)$ work.
7. For all parts, arrays and vectors were used which support fast random access.
8. The main logic to perform point 4 was to exploit *cache locality*.
9. There was no need to take care about ***false sharing*** as because in calculating partial sums, question of false sharing is irrelevant as threads work on different arrays chunks.
10. In combining partial prefix sum's, array accesses quite random(i and 2^i) and so preventing false sharing is of little use.

Analyzing Figures

1. The experiments were performed on a 6 core Processor.
2. Amongst the experiments performed, graph for $speedup$ showed similar trends. The best speedup was from graph appears around $(4-5)$.
3. Speedup declines because of increased overhead when threads are greater than cores.
4. Speedup increases with increase in problem size, as serial component remains same(***Gustafson's Law***). Also, speedup appears to peak around 1.5 for a given problem size(***Amdahl's Law***).
5. Efficiency graph shows a consistent trend. It decreases with increasing processor.
6. The above phenomenon was intuitive because single thread programs are most efficient.
7. Passing vector by reference showed significant improvement(Speedup ≈ 2.5).

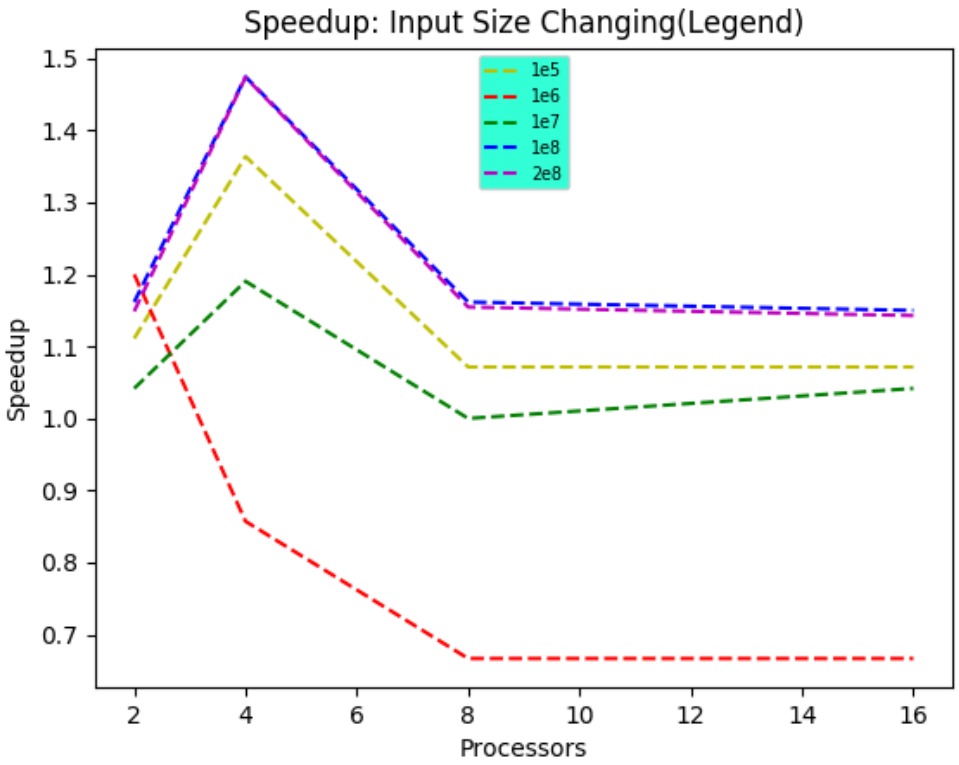

Figure 1: Speedup vs Processors for Varying input

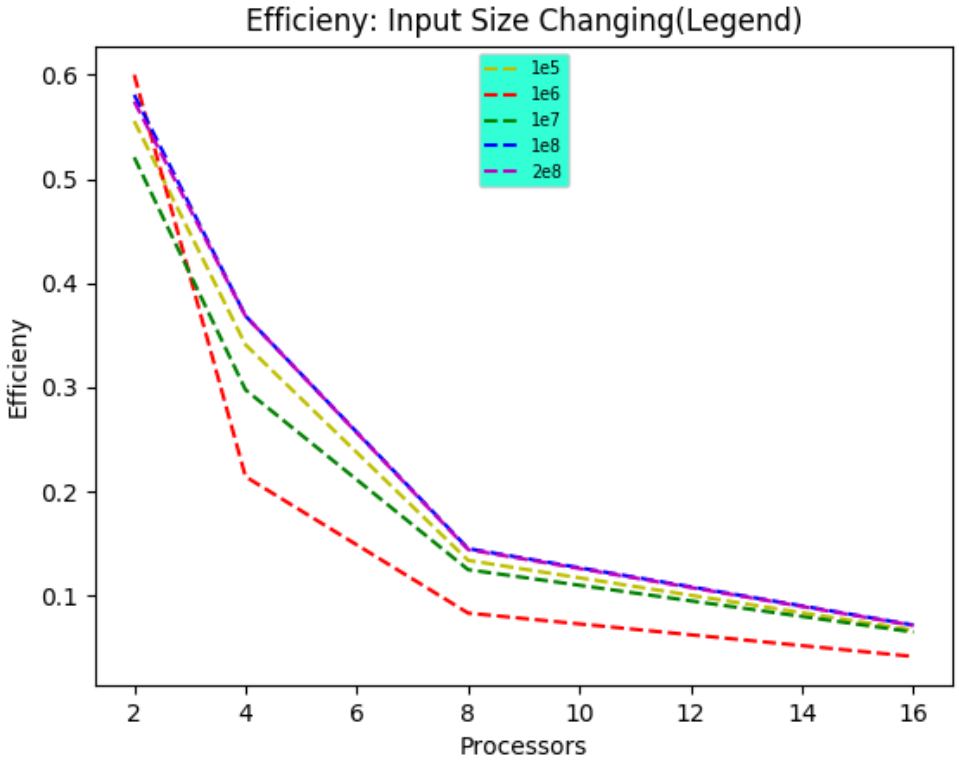

Figure 2: Efficiency vs Processors for Varying input