# HPC Lab Week 3 Report 2

| | |
|---|---|
| Name : | Prateek Agrawal |
| Roll Number : | CED18I040 |
| Programming Environment : | OpenMP |
| Problem Statement : | Vector Dot Product |
| Date : | 26th August 2021 |

## Systems Specifications :

| | |
|---|---|
| CPU Name : | Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz |
| CPU Type : | Intel Coffeelake processor |
| CPU Stepping : | 10 |
| Number of Sockets: : | 1 |
| Cores per Socket : | 6 |
| Threads per core : | 2 |
| L1 Cache size | 32 kB |
| L2 Cache size | 256 kB |
| L3 Cache size: | 9 MB |
| RAM | 32 GB |

## Serial Code:

```
/*
* @Author: prateek
* @Date:   2021-08-26 15:51:41
* @Last Modified by:   prateek
* @Last Modified time: 2021-08-26 16:10:52
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define VECTOR_LENGTH 10000
int main (int argc, char *argv[])
{

   double arr_1[VECTOR_LENGTH] ;
   double arr_2[VECTOR_LENGTH] ;
   double start, end;
   for (int i = 0; i < VECTOR_LENGTH; i++)
   {
      arr_1[i] = arr_2[i] = i * 1.0;
   }

   double sum = 0;

   start = omp_get_wtime();

   for (int k = 0; k < 1000000; k++)
   {
      for (int i = 0; i < VECTOR_LENGTH; i++)
      {
         sum += arr_1[i] * arr_2[i];
      }
   }

   end = omp_get_wtime();
   printf("%lf", end - start);

   return 0;

}
```

# Parallel Code : (Reduction)

```
/*
* @Author: prateek
* @Date:   2021-08-26 15:37:42
* @Last Modified by:   prateek
* @Last Modified time: 2021-08-26 16:13:58
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define VECTOR_LENGTH 10000
int main (int argc, char *argv[])
{

   double arr_1[VECTOR_LENGTH] ;
   double arr_2[VECTOR_LENGTH] ;
   double start, end;
   for (int i = 0; i < VECTOR_LENGTH; i++)
   {
      arr_1[i] = arr_2[i] = i * 1.0;
   }

   double sum = 0;

   start = omp_get_wtime();

   #pragma omp parallel for reduction (+:sum)
   for (int k = 0; k < 1000000; k++)
   {

         for (int i = 0; i < VECTOR_LENGTH; i++)
         {
            sum += arr_1[i] * arr_2[i];
         }
   }

   end = omp_get_wtime();
   printf("%lf", end - start);

   return 0;
}
```

# Parallel Code : (Reduction + Critical Section)

```
/*
* @Author: prateek
* @Date:   2021-08-26 16:20:56
* @Last Modified by:   prateek
* @Last Modified time: 2021-08-26 16:28:16
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#define VECTOR_LENGTH 10000
int main (int argc, char *argv[])
{

   double arr_1[VECTOR_LENGTH] ;
   double arr_2[VECTOR_LENGTH] ;
   double start, end;
   for (int i = 0; i < VECTOR_LENGTH; i++)
   {
      arr_1[i] = arr_2[i] = i * 1.0;
   }
   int i;
   double sum = 0;
   double psum = 0;

   start = omp_get_wtime();

   #pragma omp parallel shared(arr_1,arr_2,sum) private(i,psum)
   {
      psum = 0.0;
      #pragma omp for
      for (int k = 0; k < 1000000; k++)
      {
         for ( i = 0; i < VECTOR_LENGTH; i++)
         {
            psum += arr_1[i] * arr_2[i];
         }
         #pragma omp critical
         {
            sum+=psum;
```

```
        }
      }
    }
    end = omp_get_wtime();
    printf("%lf", end - start);

    return 0;

}
```

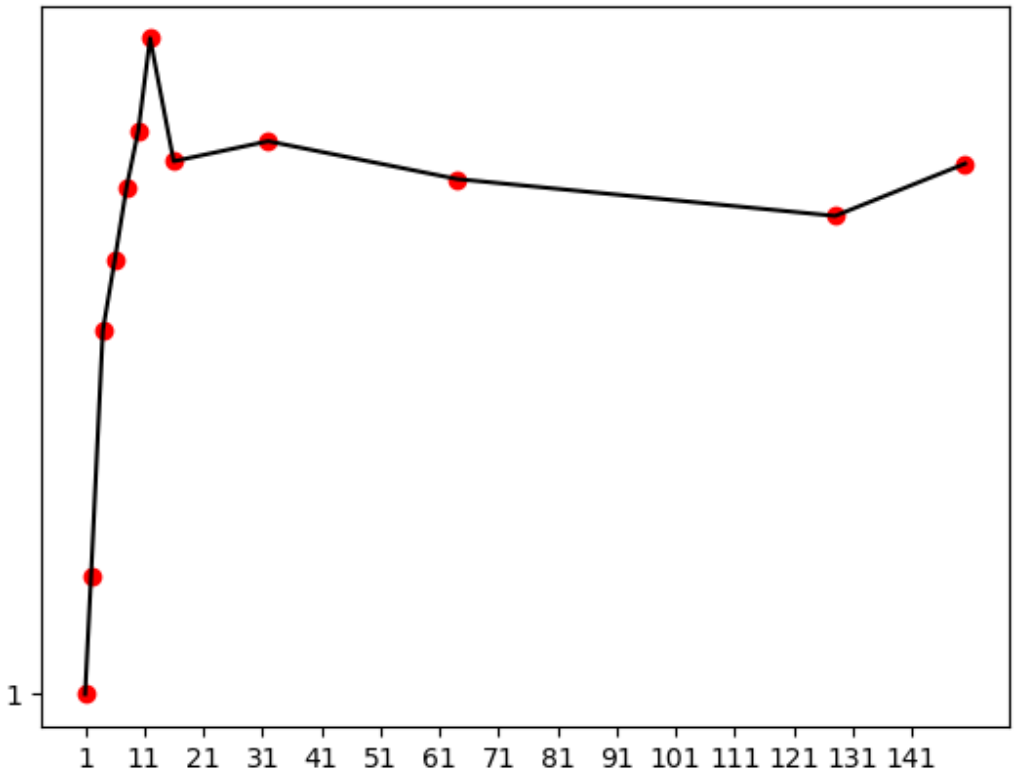
## Compilation and Execution:

For enabling OpenMP environment use -fopenmp flag while compiling using gcc.

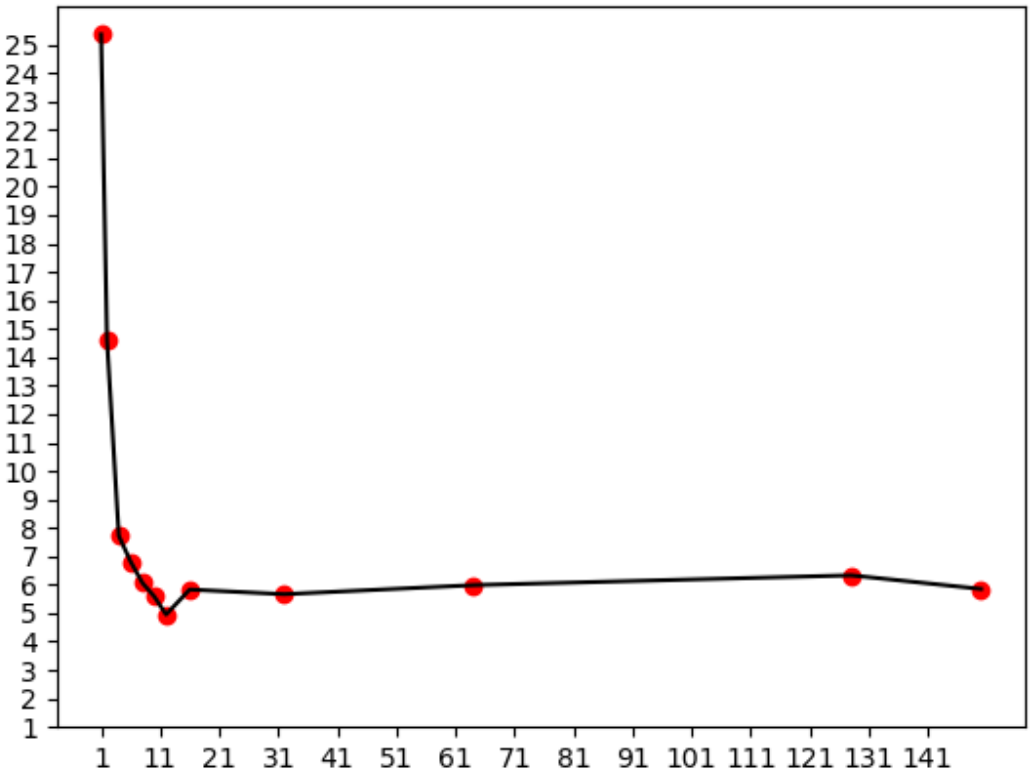
```
g++ -fopenmp vector_dot_product_parallel_reduction.cpp
```

Table 1 - Parallel Code with Reduction

| NUM Threads | Execution Time | Speed-Up | Parallelization Fraction |
|---|---|---|---|
| 1 | 25.349 | 1 | |
| 2 | 14.586 | 1.73789935554641 | 84.9185372204032 |
| 4 | 7.718 | 3.2844001036538 | 92.7373860901811 |
| 6 | 6.81 | 3.7223201174743 | 87.7620418951438 |
| 8 | 6.062 | 4.18162322665787 | 86.9552476006379 |
| 10 | 5.586 | 4.53795202291443 | 86.6262530627989 |
| 12 | 4.951 | 5.11997576247223 | 87.7839900444343 |
| 16 | 5.831 | 4.3472817698508 | 82.130261548779 |
| 32 | 5.667 | 4.47308981824599 | 80.1487365411119 |
| 64 | 5.985 | 4.23542188805347 | 77.6021345195672 |
| 128 | 6.331 | 4.00394882325067 | 75.6154011262616 |
| 150 | 5.851 | 4.33242180823791 | 77.4344513014426 |




NUMBER OF THREADS vs SPEED-UP (Parallel with Reduction)



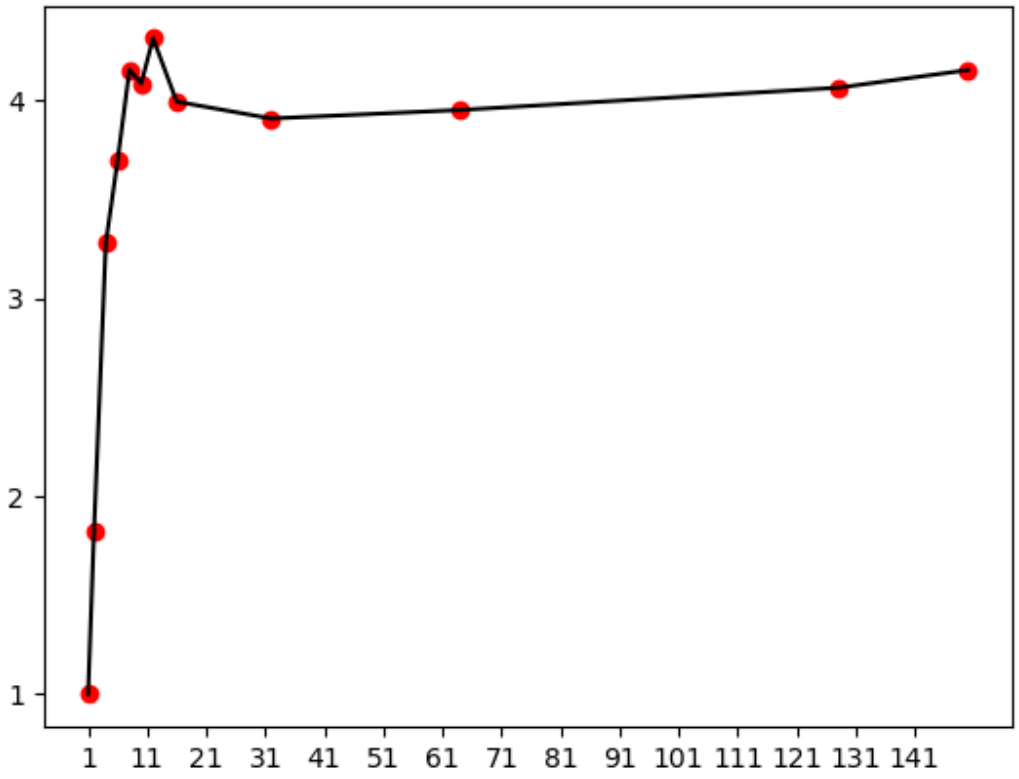

NUMBER OF THREADS vs Execution Time (Parallel with Reduction)

## Inference: (Note: Execution time, graph and inference will be based on hardware configuration)

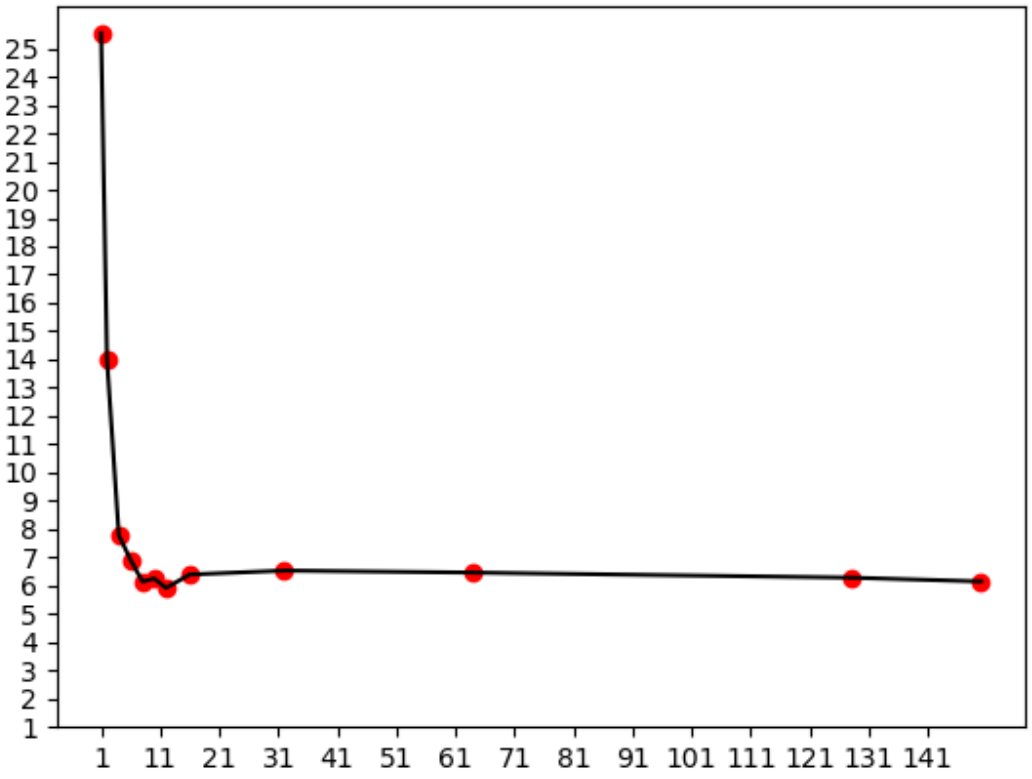
• At thread count 12 maximum speedup is observed.
• If thread count is more than 12 then the execution time increases/decreases slightly and tapers out.

Table 2 - Parallel Code with Reduction and Critical Section

| Threads | Execution Time | Speed-Up | Parallelization_Fraction |
|---|---|---|---|
| 1 | 25.538 | 1 | |
| 2 | 13.977 | 1.82714459469128 | 90.5395880648446 |
| 4 | 7.782 | 3.2816756617836 | 92.7036833999008 |
| 6 | 6.901 | 3.70062309810172 | 87.5730284282246 |
| 8 | 6.143 | 4.15725215692658 | 86.7950281373416 |
| 10 | 6.247 | 4.08804226028494 | 83.9315703831328 |
| 12 | 5.916 | 4.31676808654496 | 83.819477569967 |
| 16 | 6.391 | 3.99593177906431 | 79.9728509149764 |
| 32 | 6.53 | 3.91087289433384 | 76.8312369422922 |
| 64 | 6.459 | 3.95386282706301 | 75.8941235407678 |
| 128 | 6.28 | 4.06656050955414 | 76.0029673242838 |
| 150 | 6.145 | 4.15589910496339 | 76.4474679396042 |




NUMBER OF THREADS vs SPEED-UP (Parallel with Reduction and Critical Section)




NUMBER OF THREADS vs Execution Time (Parallel with Reduction and Critical Section)

## Inference: (Note: Execution time, graph and inference will be based on hardware configuration)

• At thread count 12 maximum speedup is observed.
• If thread count is more than 12 then the execution time increases/decreases slightly and tapers out.