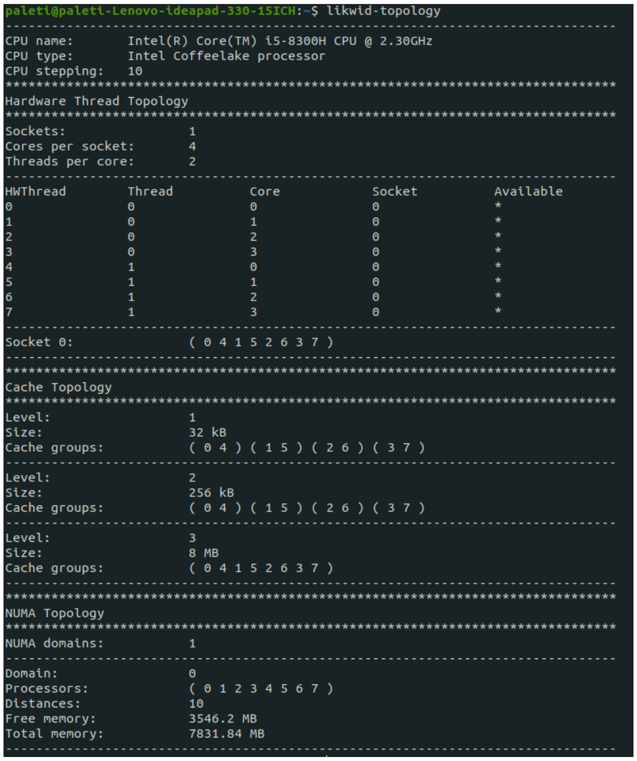
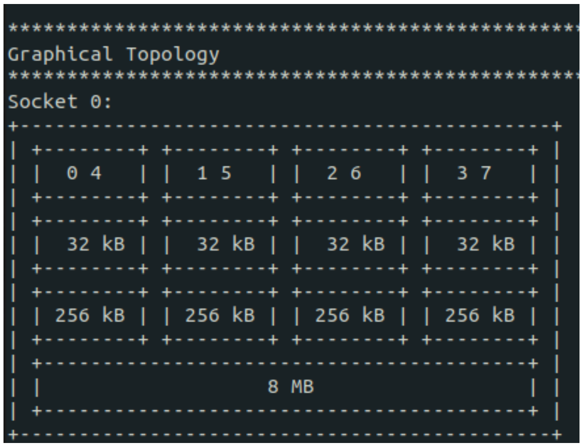
HPC LAB - Vector Multiplication CUDA

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: CUDA (collab)
Problem: Vector Multiplication

Hardware Configuration:
PU NAME: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Number of Sockets: 1
Cores per Socket: 4
Threads per core: 2
L1d cache: 128 KiB
L1i cache: 128 KiB
L2 cache: 1 MiB
L3 cache: 8 MiB



```
paleti@paleti-Lenovo-ideapad-330-15ICH:~$ likwid-topology
--------------------------------------------------------------------------------
CPU name:       Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
CPU type:       Intel Coffeelake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                1
Cores per socket:       4
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread          Core            Socket          Available
0               0               0               0               *
1               0               1               0               *
2               0               2               0               *
3               0               3               0               *
4               1               0               0               *
5               1               1               0               *
6               1               2               0               *
7               1               3               0               *
--------------------------------------------------------------------------------
Socket 0:               ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:                  1
Size:                   32 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  2
Size:                   256 kB
Cache groups:           ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:                  3
Size:                   8 MB
Cache groups:           ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:           1
--------------------------------------------------------------------------------
Domain:                 0
Processors:             ( 0 1 2 3 4 5 6 7 )
Distances:              10
Free memory:            3546.2 MB
Total memory:           7831.84 MB
--------------------------------------------------------------------------------
```



```
*************************************************
Graphical Topology
*************************************************
Socket 0:
+---------------------------------------------+
| +--------+ +--------+ +--------+ +--------+ |
| |  0 4   | |  1 5   | |  2 6   | |  3 7   | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| |  32 kB | |  32 kB | |  32 kB | |  32 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +--------+ +--------+ +--------+ +--------+ |
| | 256 kB | | 256 kB | | 256 kB | | 256 kB | |
| +--------+ +--------+ +--------+ +--------+ |
| +-----------------------------------------+ |
| |                  8 MB                   | |
| +-----------------------------------------+ |
+---------------------------------------------+
```

Paleti Krishnasai
CED18I039

CUDA code:

```
//VEctor Mul
%%cu
#include <stdio.h>
#include <stdlib.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<unistd.h>
//#define Threads 2
#define n_size 100000
__global__ void mul(float *a, float *b, float *c) {
int index=threadIdx.x+blockIdx.x*blockDim.x;
c[index]=a[index]*b[index];
}
void random_init(float a[],int ch)
{
  srand(time(NULL));
  if(ch==0)
  {
      for(int i=0;i<n_size;i++)
      {
          a[i]=((float)rand()/(float)(RAND_MAX)) * 5.0;
      }
  }
  else
  {
      for(int i=0;i<n_size;i++)
      {
            a[i]=(i+1);
      }
  }
 }
int main() {
 float a[n_size], b[n_size],c[n_size];
cudaEvent_t start, end;
// host copies of variables a, b & c
float *d_a, *d_b, *d_c;
// device copies of variables a, b & c
```

```
int size = n_size*sizeof(float);
// Allocate space for device copies of a, b, c
cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size);
cudaMalloc((void **)&d_c, size);
// Create Event for time
cudaEventCreate(&start);
cudaEventCreate(&end);
// Setup input values
random_init(a,0);
random_init(b,0);
// Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, &b, size, cudaMemcpyHostToDevice);
int Thread[]={1,2,4,6,8,10,12,16,20,32,64,128,150};
int thread_arr_size=13;
for(int i=0;i<thread_arr_size;i++)
{       sleep(1);
      int Threads=Thread[i];
      cudaEventRecord(start);
      // Launch add() kernel on GPU
      mul<<<n_size/Threads,Threads>>>(d_a, d_b, d_c);
      cudaEventRecord(end);
      cudaEventSynchronize(end);
      float time = 0;
      cudaEventElapsedTime(&time, start, end);
      // Copy result back to host
      cudaError err = cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
      if(err!=cudaSuccess) {
          printf("CUDA error copying to Host: %s\n",
cudaGetErrorString(err));
      }
          int flag=0;
      for(int i=0;i<n_size;i++)
      {   //printf("Result[%d]=%f\n",i+1,c[i]);
          if(c[i]!=(a[i]*b[i]))
          {
              flag=1;
          break;
          }
      }
      if(flag==0)
      {
          printf("Program Executed as Expected\n");
          printf("Time Taken by the program for %d 
Threads=%f\n",Threads,time);
          //printf("%f\n",time);
      }
      else
      {
          printf("Vector Addition hasnt been done properly,Mismatch in 
Values!!!\n");
      }

}
// Cleanup
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
return 0;
}
```

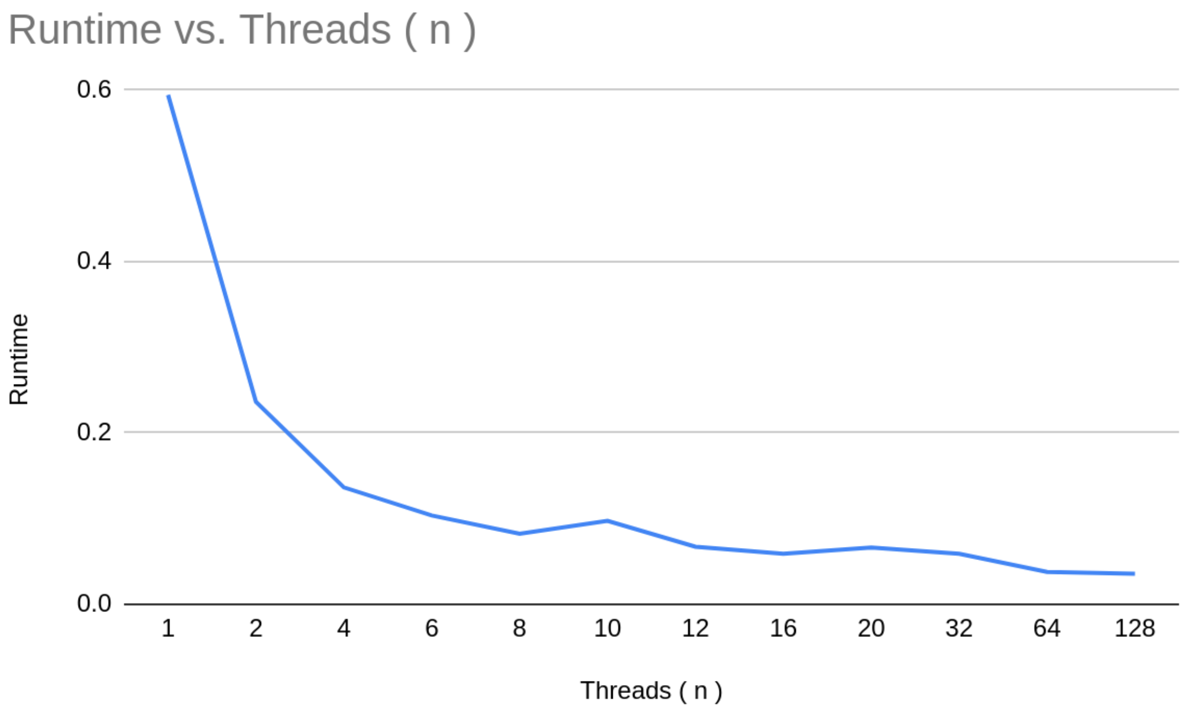
Paleti Krishnasai
CED18I039

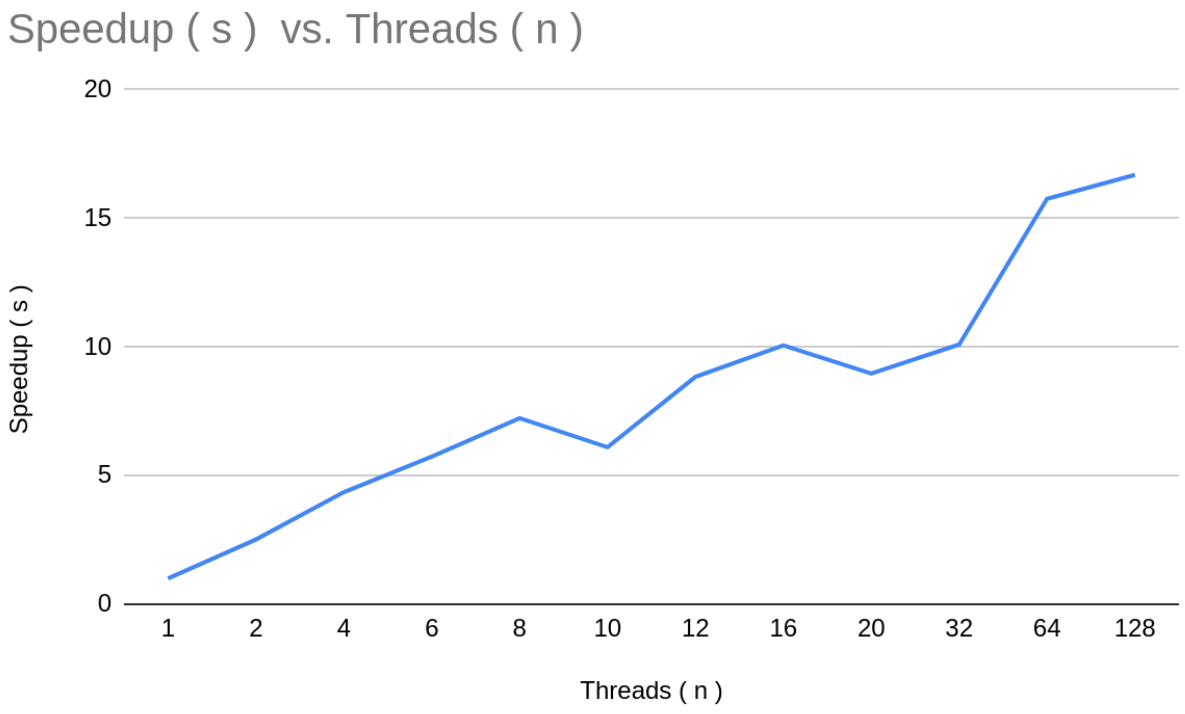
Observations:

The function add() is executed for an array of size ‘N’ on GPU for “N/T” times for ‘T’ Threads.

Blocks	Threads (n)	Runtime	Speedup (s)	Parallelization Fraction
N	1	0.594528	1	
N/2	2	0.236224	2.516797616	1.205339362
N/4	4	0.136448	4.35717636	1.027324757
N/6	6	0.103584	5.739573679	0.9909252382
N/8	8	0.08224	7.229182879	0.9847677485
N/10	10	0.09744	6.101477833	0.9290058668
N/12	12	0.067264	8.838725024	0.967485284
N/16	16	0.059072	10.06446371	0.9606832086
N/20	20	0.066304	8.966698842	0.9352381438
N/32	32	0.05888	10.09728261	0.9300267906
N/64	64	0.037728	15.75826972	0.9514069905
N/128	128	0.035616	16.69272237	0.9474959664

Graphical Inference:





Inference:

From the Graph, it is observed that the speedup increase in an up and down manner after 8 threads, and the highest Speedup is for 128 threads.

Runtime Decreases drastically from 1 to 4 threads and steadily decreases from then on till 128, with a minor rise at threads = 10.