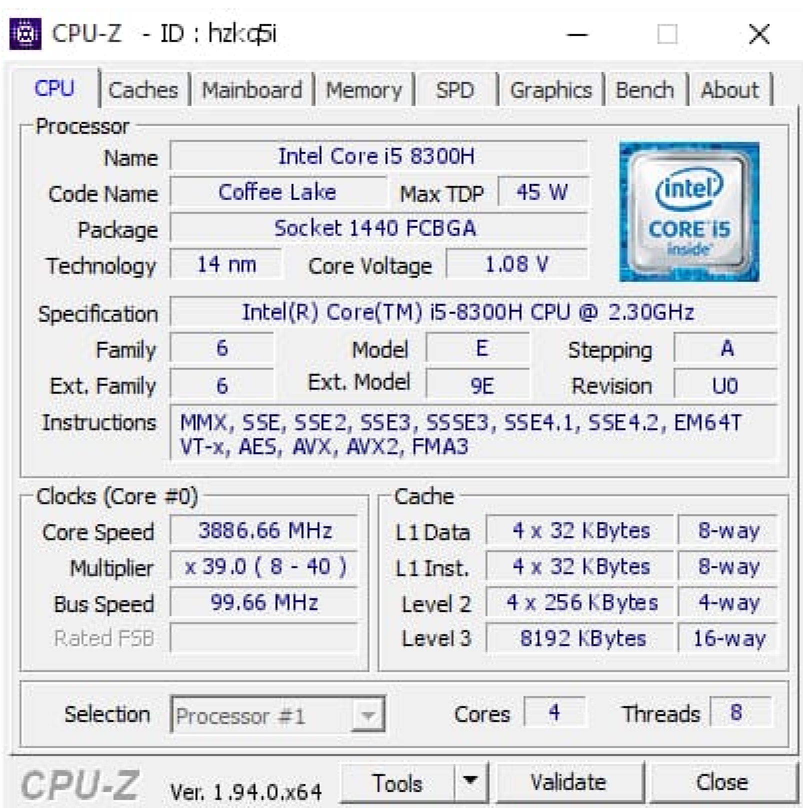
Paleti Krishnasai
CED18I039

HPC LAB - Vector Multiplication

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: OpenMP
Problem: Vector Multiplication
Date: 19th August 2021

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 Krishnasai
CED18I039

```
paleti@paleti-Lenovo-ideapad-330-15ICH:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           158
Model name:                      Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Stepping:                        10
CPU MHz:                         900.021
CPU max MHz:                     4000.0000
CPU min MHz:                     800.0000
BogoMIPS:                        4599.93
Virtualization:                  VT-x
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        8 MiB
NUMA node0 CPU(s):               0-7
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Mitigation; PTE Inversion; VMX conditional cach
                                 e flushes, SMT vulnerable
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled v
                                 ia prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user
                                  pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB condit
                                 ional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:             Mitigation; Microcode
Vulnerability Tsx async abort:   Not affected
```

Paleti Krishnasai
CED18I039

Serial Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define n 100000

int main()
{
   double a[n],b[n], c[n], random_a, random_b;
   float startTime, endTime,execTime;
   int i;
   int omp_rank;
   srand(time(0));

   startTime = omp_get_wtime();
   {
       for(i=0;i<n;i++)
       {
           random_a = rand() , random_b = rand();
           omp_rank = omp_get_thread_num();
           a[i] = i * random_a;
           b[i] = i * random_b;
           for(int j=1;j<n;j++)
           c[i] = a[i] * b[i];
       }

   }
   endTime = omp_get_wtime();

   execTime = endTime - startTime;
   printf("%f \n",execTime);
   return(0);
}
```

Paleti Krishnasai
CED18I039

Parallel Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define n 100000

int main()
{
   double a[n],b[n], c[n], random_a, random_b;
   float startTime, endTime,execTime;
   int i;
   int omp_rank;
   srand(time(0));

   startTime = omp_get_wtime();
   #pragma omp parallel private (i) shared (a,b,c)
   {
       #pragma omp for
       for(i=0;i<n;i++)
       {
           random_a = rand() , random_b = rand();
           omp_rank = omp_get_thread_num();
           a[i] = i * random_a;
           b[i] = i * random_b;
           for(int j=1;j<n;j++)
           c[i] = a[i] * b[i];
           //printf("The value of a[%d] = %lf and b[%d] = %lf and result
c[%d] = %lf  Thread rank = %d\n", i, a[i], i, b[i], i, c[i], omp_rank);
       }

   }
   endTime = omp_get_wtime();

   execTime = endTime - startTime;
   printf("%f \n",execTime);
   return(0);
}
```

Paleti Krishnasai
CED18I039

Compilation and Execution:

To enable OpenMP environment, use -fopenmp flag while compiling using gcc.

gcc -O0 -fopenmp vector_mul_mp.c

Then,

export OMP_NUM_THREADS= no of threads for parallel execution.

./a.out

Observations :

Threads (n)	Runtime	Speedup (s)	Parallelization Fraction
1	32.630859	1	
2	16.083984	2.028779623	1.014185682
4	10.212891	3.195065824	0.9160232037
8	8.671875	3.762837795	0.8391350041
16	9.207031	3.544123942	0.7656990131
32	9.503906	3.433415587	0.7316075786
64	8.613281	3.788435441	0.7477219462
128	8.714844	3.744284924	0.7386973809

Speed up can be found using the following formula,
S(n)=T(1)/T(n)
where, **S(n)** = Speedup for thread count ‘n’
T(1) = Execution Time for Thread count ‘1’ (serial code)
T(n) = Execution Time for Thread count ‘n’ (serial code)

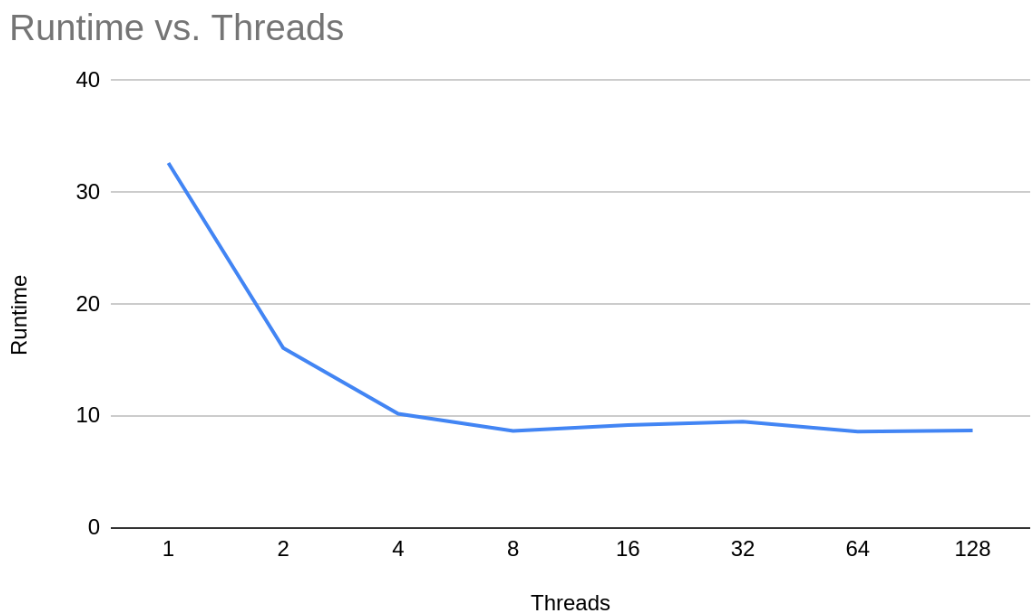
Parallelization Fraction can be found using the following formula,
S(n)=1/((1 - p) + p/n)
where, **S(n)** = Speedup for thread count ‘n’
n = Number of threads
p = Parallelization fraction

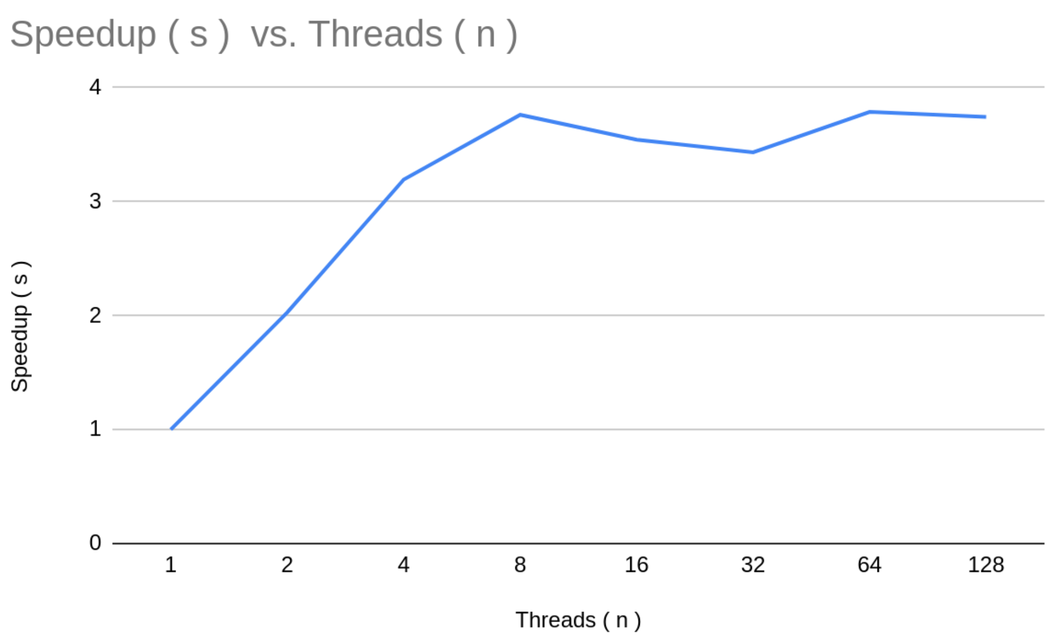
Paleti Krishnasai
CED18I039

Assumption:

Following extra for loop is added to increase the number of operations in the parallel region to visualize the effect of multi-threading in vector addition.

for(int j=0;j<m;j++)
*c[i] = a[i] * b[i];*





Paleti Krishnasai
CED18I039

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

- At thread count 8,64,128 maximum speedup is observed.
- The Runtime is least at thread count is 8, but it fluctuates from there on rising slightly. This may be due to the fact that there are 4 cores and 2 threads on each core are active at a moment for this task.