Paleti Krishnasai
CED18I039

HPC LAB - Sum of N Numbers

Name: Paleti Krishnasai
Roll No: CED18I039
Programming Environment: OpenMP
Problem: Sum of N numbers
Date: 26th 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@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

Serial Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define n 100000

int main()
{
   double a[n], random_a;
   double sum=0;
   float startTime, endTime,execTime;
   int i;
   srand(time(0));
   startTime = omp_get_wtime();

       for(i=0;i<n;i++)
       {
           random_a = rand();
           a[i] = i * random_a;
           for(int j=1;j<n;j++)
           sum = sum + a[i];
       }


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

Paleti Krishnasai
CED18I039

Parallel code : [REDUCTION]

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define n 100000

int main()
{
   double a[n], random_a;
   double sum=0;
   float startTime, endTime,execTime;
   int i;
   srand(time(0));
   startTime = omp_get_wtime();
   #pragma omp parallel private (i) shared (a) reduction(+:sum)
   {
       #pragma omp for
       for(i=0;i<n;i++)
       {
           random_a = rand();
           a[i] = i * random_a;
           for(int j=1;j<n;j++)
           sum = sum + a[i];
       }

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

Paleti Krishnasai
CED18I039

Compilation and Execution: [REDUCTION]

To enable OpenMP environment, use -fopenmp flag while compiling using gcc.
-O0 flag is used to disable compiler optimizations.

gcc -O0 -fopenmp Sum_N_numbers_mp_reduction.c

Then,
export OMP_NUM_THREADS= no of threads for parallel execution.

./a.out

Observations : [REDUCTION]

Threads (n)	Runtime	Speedup (s)	Parallelization Fraction
1	40.613281	1	
2	20.453125	1.985676076	0.9927863745
4	10.594727	3.833348514	0.9855086238
6	8.431641	4.816770662	0.9508704308
8	7.074219	5.741026819	0.943788722
10	7.421875	5.472105229	0.9080610847
12	7.783203	5.218067806	0.8818452896
16	7.644531	5.312723698	0.8658908072
20	7.144531	5.684527228	0.867456711
32	7.839844	5.180368512	0.8329946218
64	7.162109	5.670575664	0.8367248877
128	7.15625	5.675218306	0.8302819018

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)

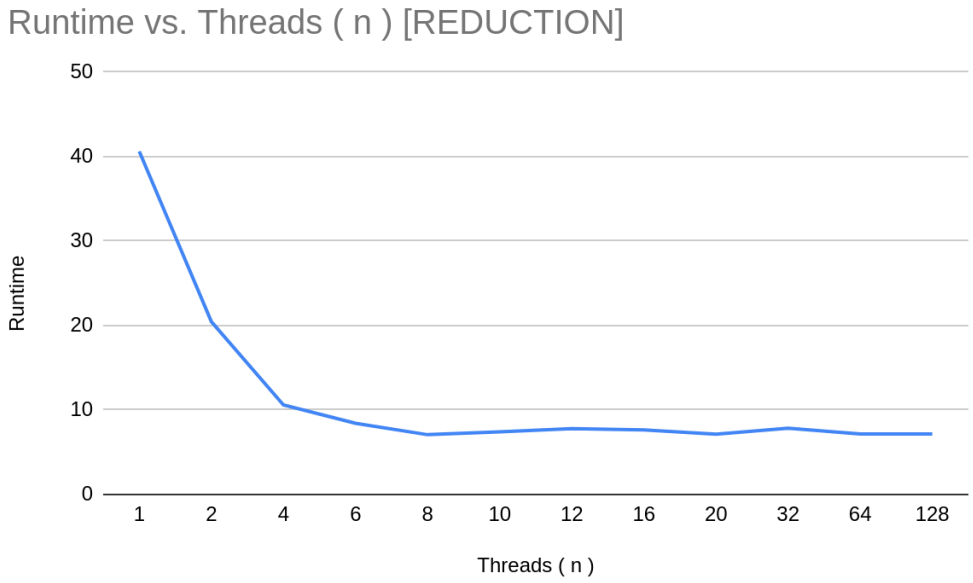
Parallelization Fraction can be found using the following formula,
S(n)=1/((1 - p) + p/n)
where, **S(n)** = Speedup for thread count ‘n’
n = Number of threads
p = Parallelization fraction

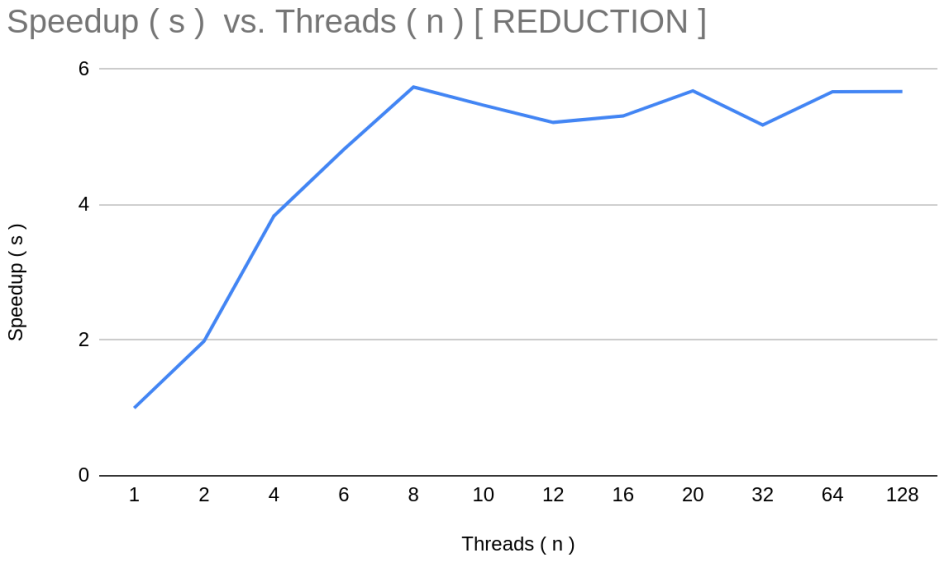
Paleti Krishnasai
CED18I039

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 the sum of N numbers.

for(int j=0;j<n;j++) , where n = 100000





Paleti Krishnasai
CED18I039

Parallel Code : [CRITICAL SECTION]

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define n 100000

int main()
{
   double a[n], random_a;
   double sum=0, privatesum;
   float startTime, endTime,execTime;
   int i;
   srand(time(0));
   startTime = omp_get_wtime();
   #pragma omp parallel private (i,privatesum) shared (a, sum)
   {
       privatesum=0;
       #pragma omp  for
       for(i=0;i<n;i++)
       {
           random_a = rand();
           a[i] = i * random_a;
           for(int j=1;j<n;j++)
           privatesum = privatesum + a[i];
       }
       #pragma omp critical
       {
           sum = sum + privatesum;
       }

   }
   endTime = omp_get_wtime();

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

Paleti Krishnasai
CED18I039

Compilation and Execution: [CRITICAL SECTION]

To enable OpenMP environment, use -fopenmp flag while compiling using gcc.
-O0 flag is used to disable compiler optimizations.

gcc -O0 -fopenmp Sum_N_numbers_mp_CS.c

Then,

export OMP_NUM_THREADS= no of threads for parallel execution.

./a.out

Observations : [CRITICAL SECTION]

Threads (n)	Runtime	Speedup (s)	Parallelization Fraction
1	40.595703	1	
2	20.421875	1.987853858	0.9938898213
4	10.40332	3.902187283	0.9916446247
6	8.405273	4.829789943	0.9515419896
8	7.329102	5.53897367	0.9365270155
10	7.480469	5.426892752	0.9063694363
12	7.439453	5.456812887	0.8909921955
16	7.299805	5.561203758	0.8748616703
20	7.599609	5.341814691	0.8555765256
32	7.293945	5.565671663	0.8467893328
64	7.136719	5.688286592	0.8372826795
128	7.111328	5.708596622	0.8313202877

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)

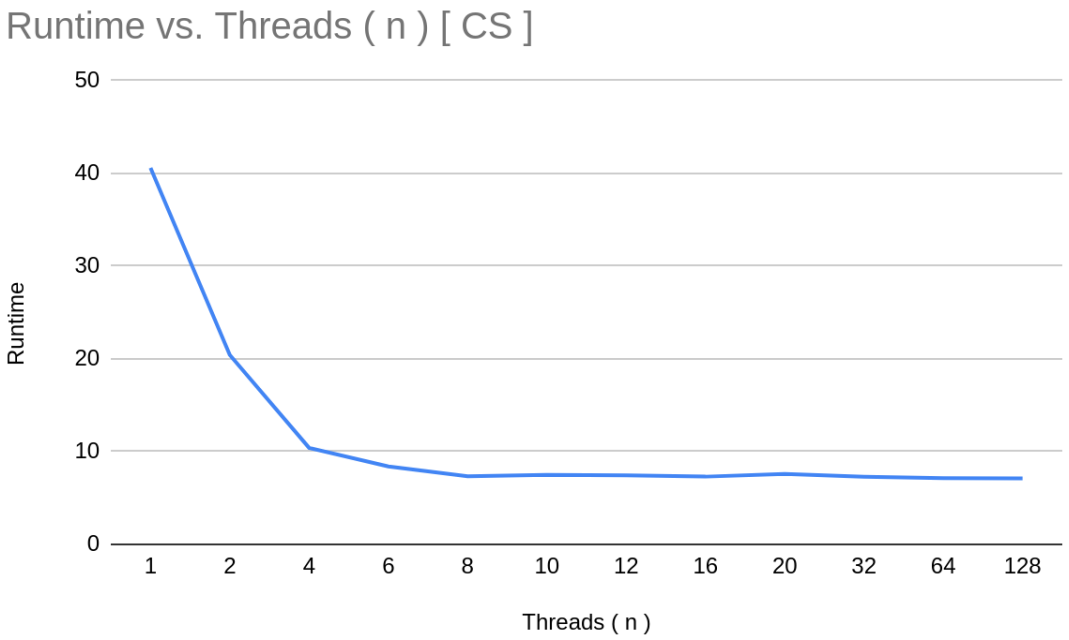
Parallelization Fraction can be found using the following formula,
S(n)=1/((1 - p) + p/n)
where, **S(n)** = Speedup for thread count ‘n’
n = Number of threads
p = Parallelization fraction

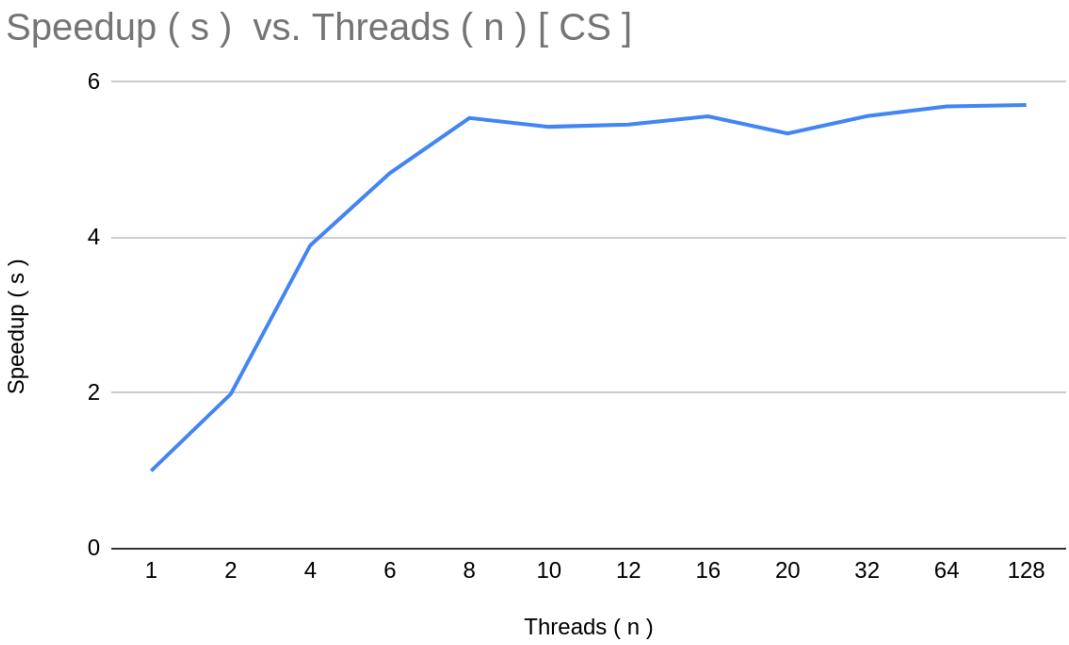
Paleti Krishnasai
CED18I039

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 the sum of N numbers.

for(int j=0;j<n;j++) , where n = 100000





Paleti Krishnasai
CED18I039

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

- At thread count 8 maximum speedup is observed in both methods and has since flatlined from 8 to 128 with small changes.
- The Runtime is least at thread count 8 for both methods, 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 that are active for this task.