**HPC Assignment No. 03**

**Batch:** B2

**PRN No:** 2019BTECS00033

**Name**: Teknath jha

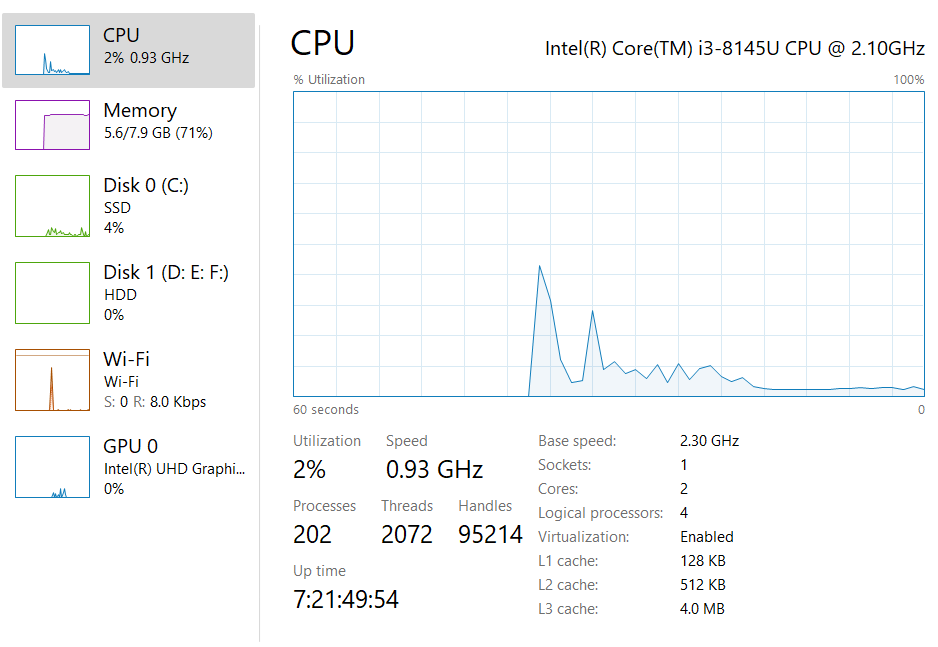
**Date**: 6th of sept 2022

* **Title:**

**openMp program for :**

**1)**

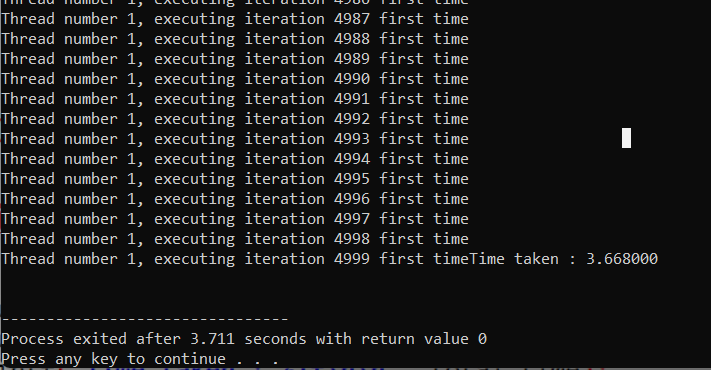
**MY SYSTEM CONFIGURATION :**



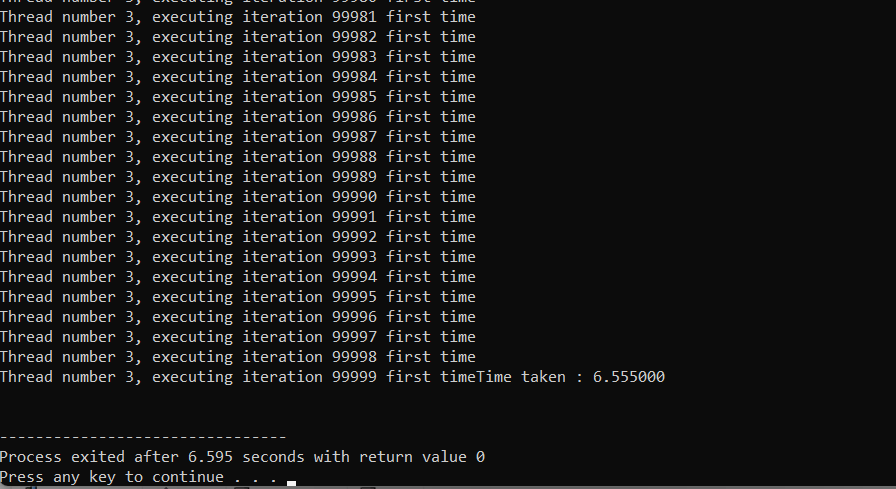
**2)** **For 1D Vector (size=200) and scalar addition, Write a OpenMP code with the following:**

1. **Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.**

**For array size 10000 with 4 threads :**

****

**For array size 100000 with 4 threads :**

****

*/\**

*Author - Teknath jha*

*PRN    - 2019BTECS00033*

*HPC-Lab- 2*

*\*/*

*// PARALLEL*

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define ARRAY\_SIZE 100000

#define NUM\_THREADS 4

int main() {

  int n = ARRAY\_SIZE;

  int a[n], c[n], flag[n];

  int i;

  for (i = 0; i < n; i++) {

    a[i] = i;

    flag[i] = 0;

  }

  int n\_per\_thread; *// elements per thread*

  int total\_threads = NUM\_THREADS; *// number of threads to use*

*// Additional work to set the number of threads.*

  omp\_set\_num\_threads(total\_threads);

*// determine how many elements each process will work on*

  n\_per\_thread = n / total\_threads;

  int x = 5;

  double start = omp\_get\_wtime();

#pragma omp parallel for shared(a, c, x) private(i)                            \

    schedule(static, n\_per\_thread)

  for (i = 0; i < n; i++) {

    if (!flag[i]) {

      c[i] = a[i] + x;

      flag[i] = 1;

      printf("\nThread number %d, executing iteration %d first time",

             omp\_get\_thread\_num(), i);

    } else {

      printf("\nThread number %d, executing iteration %d already computed",

             omp\_get\_thread\_num(), i);

    }

  }

  double end = omp\_get\_wtime();

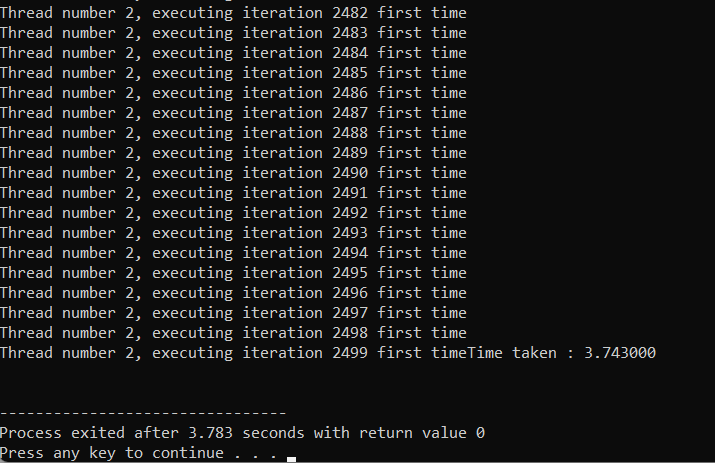
  double total\_time = (end - start);

  printf("Time taken : %lf\n\n", total\_time);

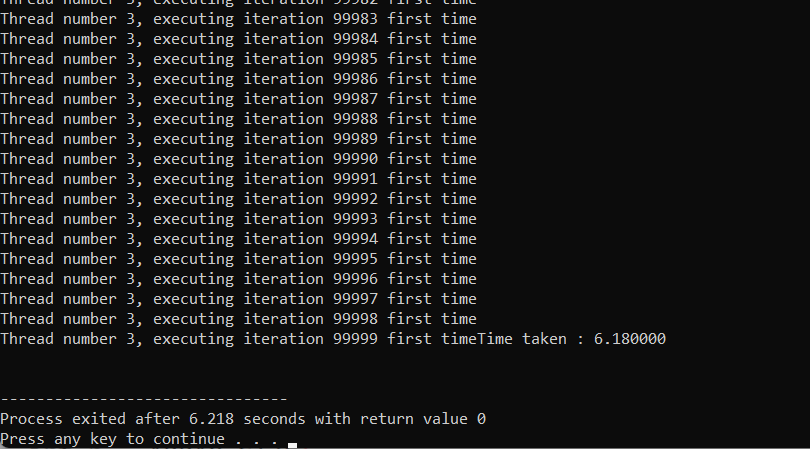
}

1. **Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup.**

**For n=10000 with 4 threads**

****

**For N= 100000 with 4 threads**

****

*/\**

*Author - Teknath jha*

*PRN    - 2019BTECS00033*

*HPC-Lab- 2*

*\*/*

*// PARALLEL*

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define ARRAY\_SIZE 100000

#define NUM\_THREADS 4

int main() {

  int n = ARRAY\_SIZE;

  int a[n], c[n], flag[n];

  int i;

  for (i = 0; i < n; i++) {

    a[i] = i;

    flag[i] = 0;

  }

  int n\_per\_thread; *// elements per thread*

  int total\_threads = NUM\_THREADS; *// number of threads to use*

*// Additional work to set the number of threads.*

  omp\_set\_num\_threads(total\_threads);

*// determine how many elements each process will work on*

  n\_per\_thread = n / total\_threads;

  int x = 5;

  double start = omp\_get\_wtime();

#pragma omp parallel for shared(a, c, x) private(i)                            \

    schedule(dynamic, n\_per\_thread)

  for (i = 0; i < n; i++) {

    if (!flag[i]) {

      c[i] = a[i] + x;

      flag[i] = 1;

      printf("\nThread number %d, executing iteration %d first time",

             omp\_get\_thread\_num(), i);

    } else {

      printf("\nThread number %d, executing iteration %d already computed",

             omp\_get\_thread\_num(), i);

    }

  }

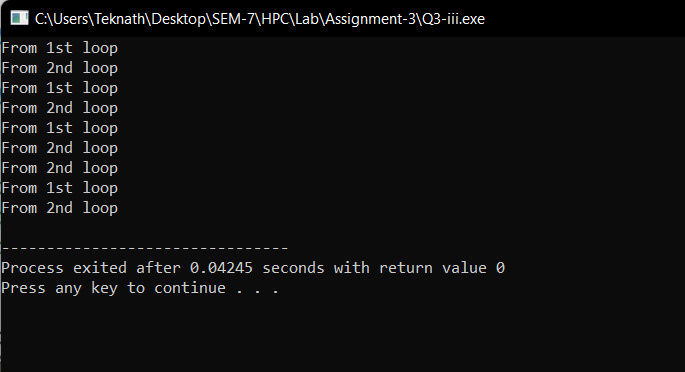
  double end = omp\_get\_wtime();

  double total\_time = (end - start);

  printf("Time taken : %lf\n\n", total\_time);

}

1. **Demonstrate the use of nowait clause**

****

*// If there are multiple independent loops within a parallel region,*

*//  you can use the nowait clause to avoid the implied barrier*

*//   at the end of the loop construct, as follows:*

#include <math.h>

#include <omp.h>

#include <stdio.h>

void nowait\_example(int n, int m, float \*a, float \*b, float \*y, float \*z) {

  int i;

#pragma omp parallel

  {

#pragma omp for nowait

    for (i = 1; i < n; i++)

      {

        b[i] = (a[i] + a[i - 1]) / 2.0;

        printf("From 1st loop\n");

      }

#pragma omp for nowait

    for (i = 0; i < m; i++)

      {

        y[i] = sqrt(z[i]);

      printf("From 2nd loop\n");

      }

  }

}

int main() {

  int n = 5, m = 5;

  float a[] = {1, 2, 3, 4, 5};

  float b[] = {1, 2, 3, 4, 5};

  float y[] = {1, 2, 3, 4, 5};

  float z[] = {1, 2, 3, 4, 5};

  nowait\_example(n, m, a, b, y, z);

  return 0;

}

**Screenshot’s OF Practical:**

1. **Vector Vector Addition :**

**Parallel program :**

*/\**

*Author - Teknath jha*

*PRN    - 2019BTECS00033*

*HPC-Lab- 2*

*\*/*

*// PARALLEL*

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define ARRAY\_SIZE 1000

#define NUM\_THREADS 4

int main() {

  printf("Parallel Program : \n");

  int n = ARRAY\_SIZE; *// number of array elements*

  int n\_per\_thread; *// elements per thread*

  int total\_threads = NUM\_THREADS; *// number of threads to use*

  int i;

  int a[n], b[n], c[n];

  for (i = 0; i < n; i++) {

    a[i] = i;

  }

  for (i = 0; i < n; i++) {

    b[i] = i;

  }

*// Additional work to set the number of threads.*

  omp\_set\_num\_threads(total\_threads);

*// determine how many elements each process will work on*

  n\_per\_thread = n / total\_threads;

  double start = omp\_get\_wtime();

#pragma omp parallel for shared(a, b, c) private(i)                            \

    schedule(static, n\_per\_thread)

  for (i = 0; i < n; i++) {

    c[i] = a[i] + b[i];

    printf("Thread %d works on element %d\n", omp\_get\_thread\_num(), i);

  }

  double end = omp\_get\_wtime();

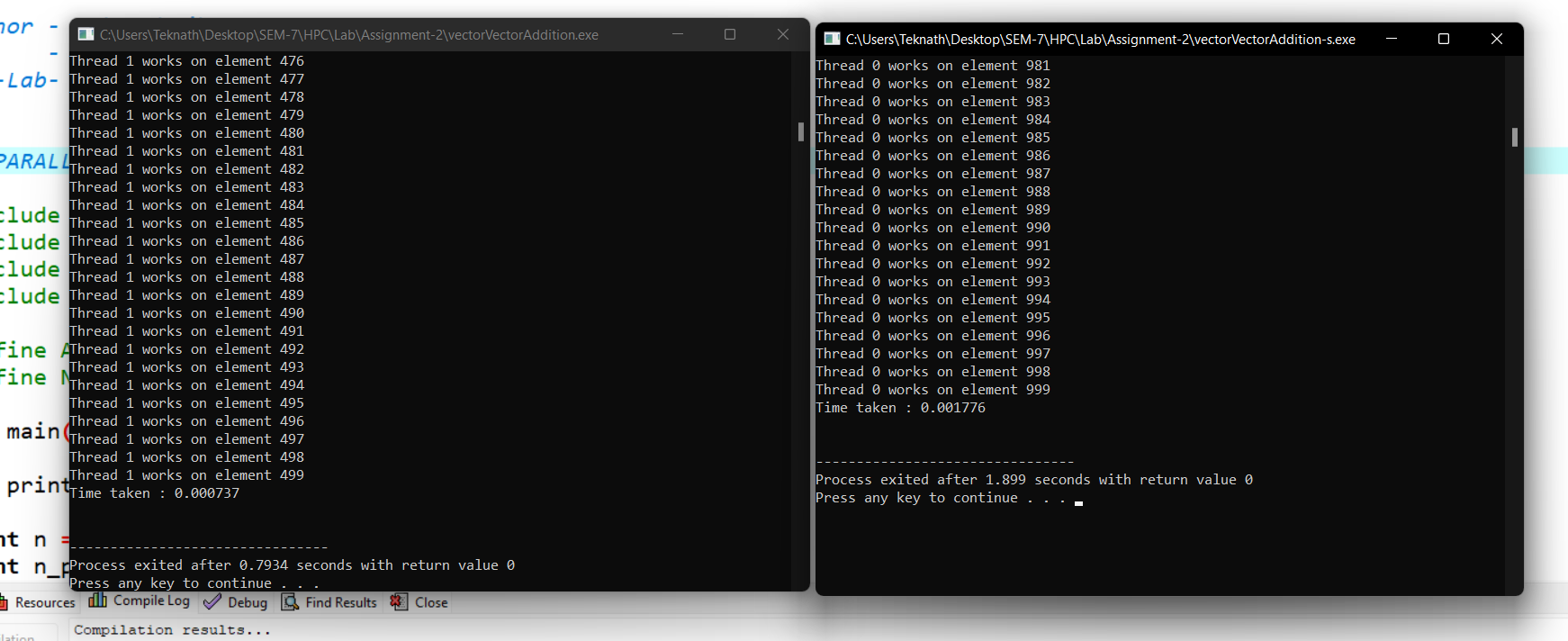
  double total\_time = (end - start) / CLOCKS\_PER\_SEC;

  printf("Time taken : %lf\n\n", total\_time);

  return 0;

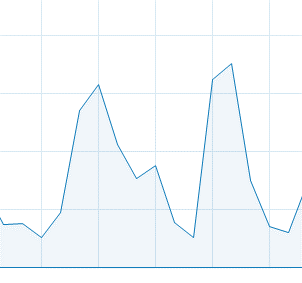
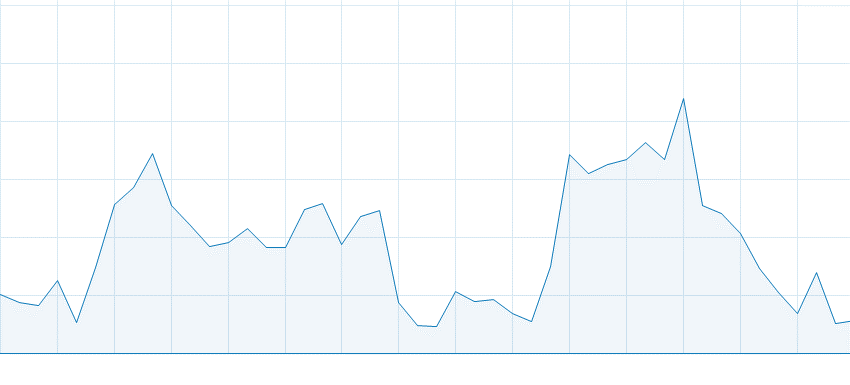
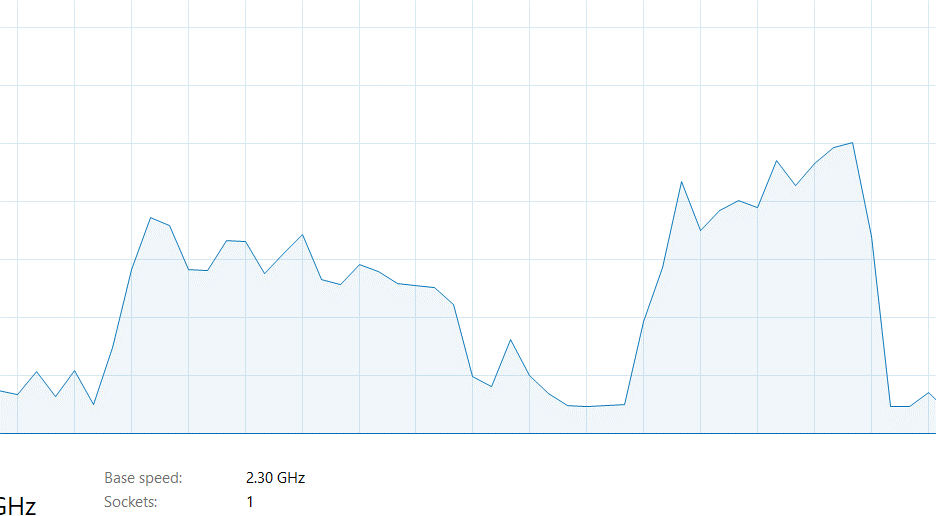
}

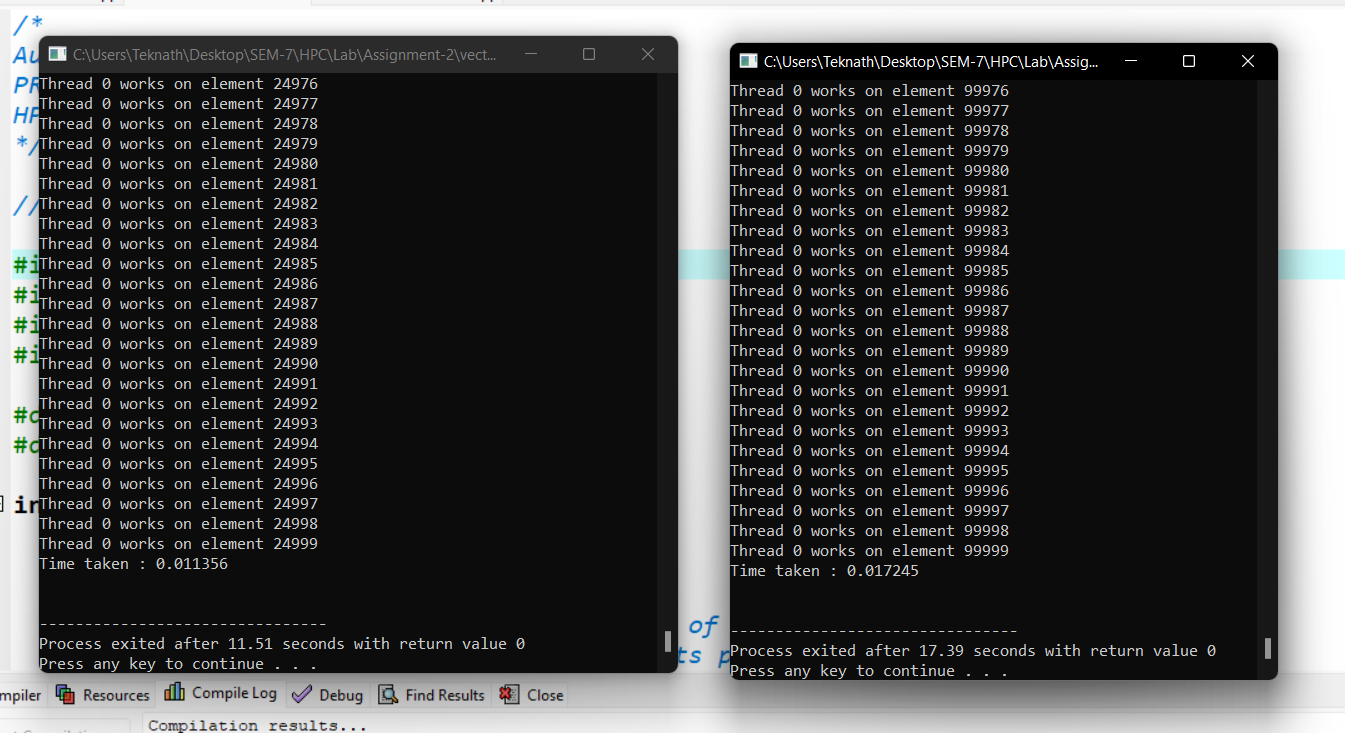
**Comparison with sequential :**



**In below images 1st peak is of sequential and later is of parallel program :**

**Images from CPU Utilization Task Manager :**

1. **for n=1000**
2. **for n=10000**
3. **for n= 100000**

****

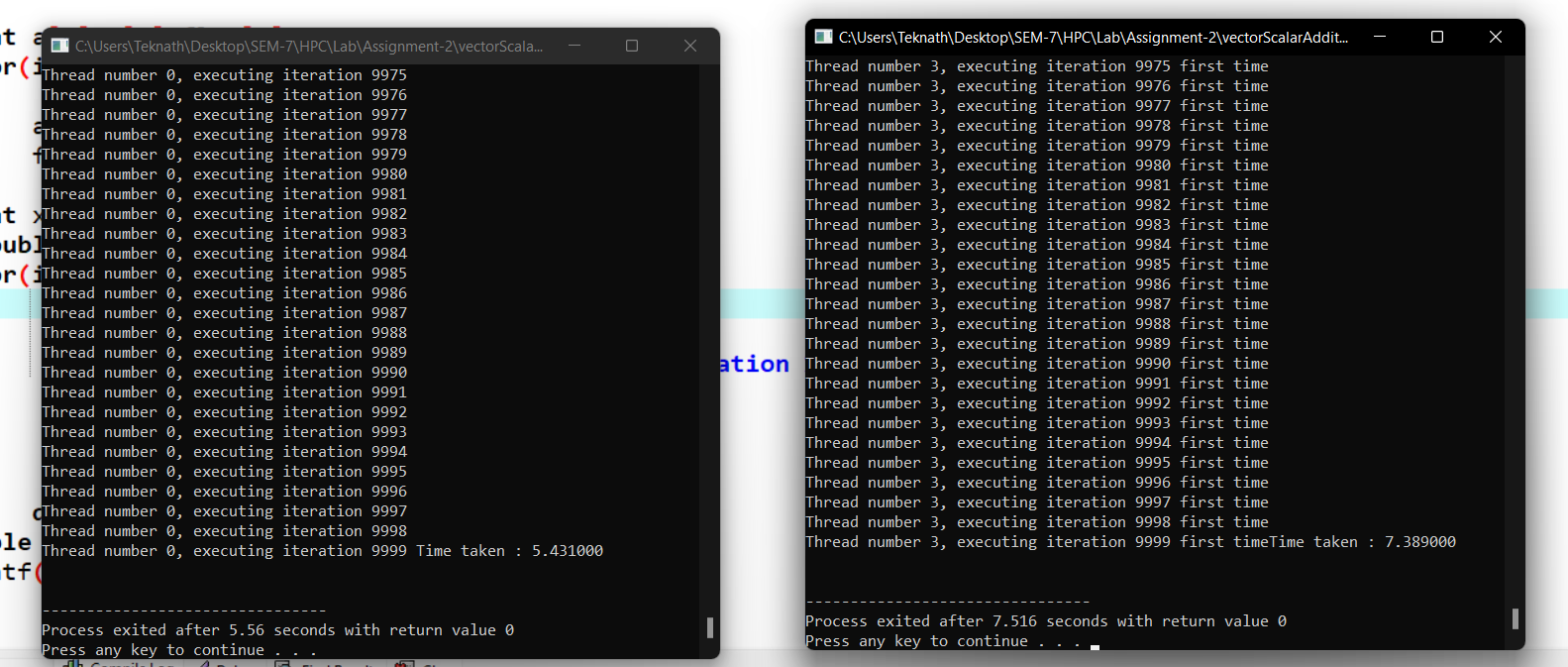
**Here sequential takes 0.01725 and parallel takes 0.011356 clock ticks**

**Conclusion : my sequential program uses less CPU and parallel program uses more UPU for same program and same number of instruction Sets.**

**Although time calculation is negligible as it is small program .**

**B) Vector Scalar Addition**

**Output Screenshot’s :**

****

**Parallel :7.3 sec Sequential: 5.4sec**

**So here sequential is faster than parallel.**

*/\**

*Author - Teknath jha*

*PRN    - 2019BTECS00033*

*HPC-Lab- 2*

*\*/*

*// PARALLEL*

#include <omp.h>

#include <stdio.h>

#include<stdlib.h>

#define ARRAY\_SIZE 10001

#define NUM\_THREADS 4

int main() {

int n = ARRAY\_SIZE;

    int a[n],c[n],flag[n];

    int i;

    for( i=0;i<n;i++)

    {

      a[i]=i;

      flag[i]=0;

  }

  int n\_per\_thread; *// elements per thread*

  int total\_threads = NUM\_THREADS; *// number of threads to use*

*// Additional work to set the number of threads.*

  omp\_set\_num\_threads(total\_threads);

*// determine how many elements each process will work on*

  n\_per\_thread = n / total\_threads;

  int x=5;

  double start = omp\_get\_wtime();

  #pragma omp parallel for shared(a, c, x) private(i)   schedule(static, n\_per\_thread)

  for( i = 0;i < n;i++) {

    if(!flag[i])

    {

      c[i]=a[i]+x;

      flag[i]=1;

      printf("\nThread number %d, executing iteration %d first time",omp\_get\_thread\_num(),i);

    }

    else

    {

      printf("\nThread number %d, executing iteration %d already computed",omp\_get\_thread\_num(),i);

    }

  }

  double end = omp\_get\_wtime();

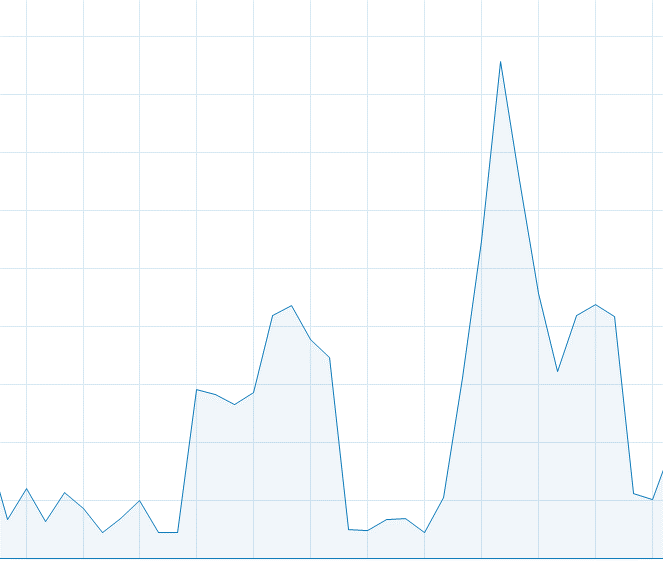
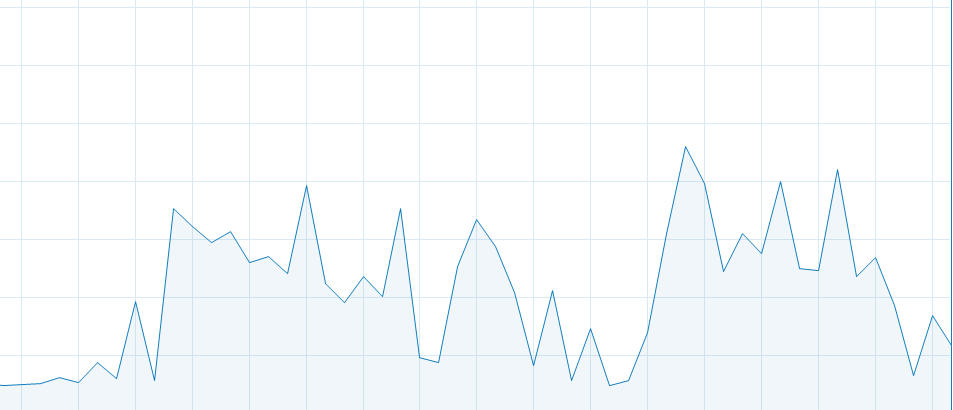
  double total\_time = (end - start);

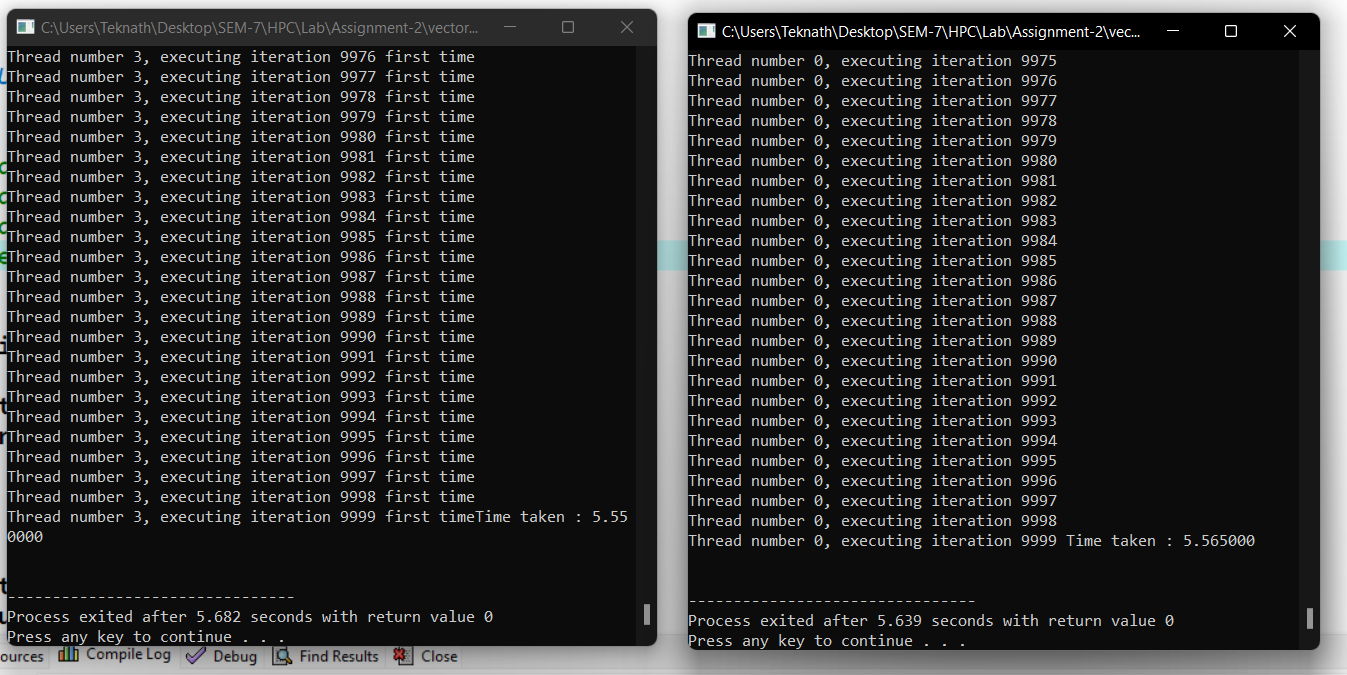
  printf("Time taken : %lf\n\n", total\_time);

}

**In below images 1st peak is of sequential and later is of parallel program :**

**CPU Graphs :**

1. 
2. ****

****

**Here most of time width of sequential is more than parallel which shows time difference.**

**Conclusion :**

**In execution : sequential taken 5.56 while parallel taken 5.55 which is considerable difference , further observation of CPU cycles also proves this that parallel is faster than sequential .**