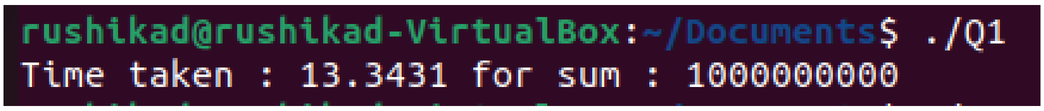
# OpenMP

| | |
|---|---|
| Name | : Jayasinghe M. A. R. D |
| Index No | : 20/ENG/061 |
| Date of Submission | : 21/02/2024 |

1. **a.** Using OpenMP functions for timing, the code first initializes an array of size ARR_SIZE with a value of 1 for each element. It then computes the sum of the array components one at a time. The outcome is then displayed on the console along with the time it took and the sum. Parallelization techniques are not used in this code.

   **b.** Time taken: 13.3431 seconds

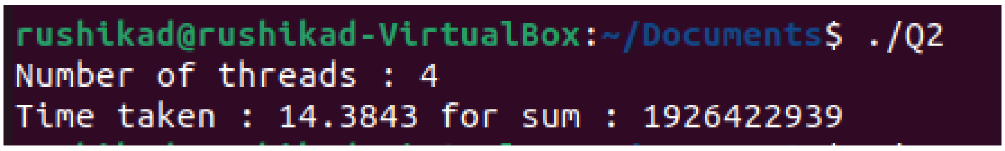


```
rushikad@rushikad-VirtualBox:~/Documents$ ./Q1
Time taken : 13.3431 for sum : 1000000000
```

2. Yes, the time taken in the second program is higher than the first one.
   The array summation is carried out progressively by a single thread in the first program. The array's elements are added together one at a time. It executes in a simple sequential fashion because there is no overhead involved in coordinating the activities of several threads.

   OpenMP is used in the second application to parallelize the array summation. Multiple threads, each in charge of a section of the array, partition out the work. But adding parallelism also means adding overhead, like creating threads, synchronizing them, and facilitating communication between them. The expense of parallelization can outweigh the advantages of a straightforward work like summing an array in this instance, especially when the array is not very large.

   The cost of maintaining parallelism for a relatively small array, such as the one used in the example, has make the parallel version run more slowly than the straightforward sequential one. That's the reason in this case why it has taken more time than the first one.

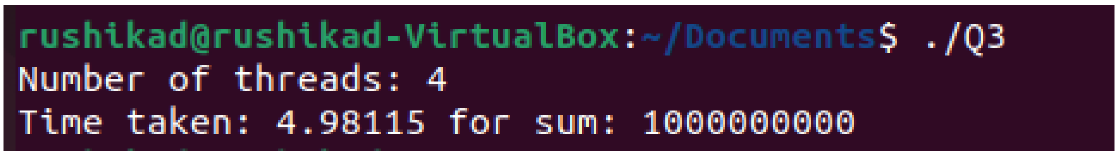


```
rushikad@rushikad-VirtualBox:~/Documents$ ./Q2
Number of threads : 4
Time taken : 14.3843 for sum : 1926422939
```

3. The array can be manually split into chunks and assigned to individual threads in order to manually share the effort over several threads.
   Modified code is as below.

```
#include <omp.h>
#include <stdio.h>
#define ARR_SIZE 1000000000
static int a[ARR_SIZE];

int main(int *argc, char *argv[])
{
   int i, sum = 0;
   int tid, numt = 0;
   double t1, t2;

   /* Initialize the array */
   for(i = 0; i < ARR_SIZE; i++)
      a[i] = 1;

   t1 = omp_get_wtime();

   #pragma omp parallel default(shared) private(i, tid)
   {
      tid = omp_get_thread_num();
      numt = omp_get_num_threads();

      // Manually distribute the workload among threads
      int chunk_size = ARR_SIZE / numt;
      int start_index = tid * chunk_size;
      int end_index = (tid == numt - 1) ? ARR_SIZE : start_index + chunk_size;

      // Calculate the sum for the assigned chunk
      int local_sum = 0;
      for(i = start_index; i < end_index; i++)
         local_sum += a[i];
```

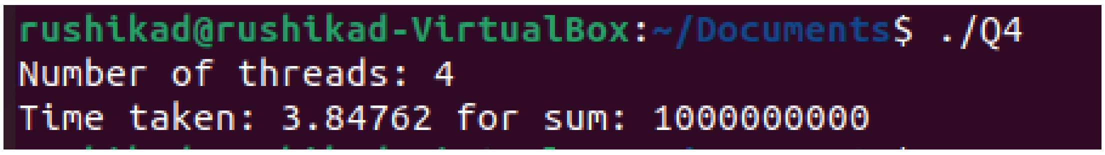
```
        // Combine local sums from all threads
        #pragma omp critical
        sum += local_sum;
    }

    t2 = omp_get_wtime();

    printf("Number of threads: %d\n", numt);
    printf("Time taken: %g for sum: %d \n", t2 - t1, sum);

    return 0;
}
```



```
rushikad@rushikad-VirtualBox:~/Documents$ ./Q3
Number of threads: 4
Time taken: 4.98115 for sum: 1000000000
```

**4.** We can use a parallelized loop to calculate the sum concurrently in order to more efficiently divide the burden over several threads using the OpenMP libraries. Modified code is as below:

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

#define ARR_SIZE 1000000000
static int a[ARR_SIZE];

int main(int *argc, char *argv[])
{
    int i, sum = 0;
    int tid, numt = 0;
    double t1, t2;

    //Initialize the array
    for (i = 0; i < ARR_SIZE; i++)
        a[i] = 1;
    t1 = omp_get_wtime();

    #pragma omp parallel default(shared) private(i, tid) reduction(+:sum)
    {
        tid = omp_get_thread_num();
        numt = omp_get_num_threads();

        #pragma omp for
        for (i = 0; i < ARR_SIZE; i++)
            sum += a[i];
    }

    t2 = omp_get_wtime();

    printf("Number of threads: %d\n", numt);
    printf("Time taken: %g for sum: %d \n", t2 - t1, sum);

    return 0;
}
```



```
rushikad@rushikad-VirtualBox:~/Documents$ ./Q4
Number of threads: 4
Time taken: 3.84762 for sum: 1000000000
```