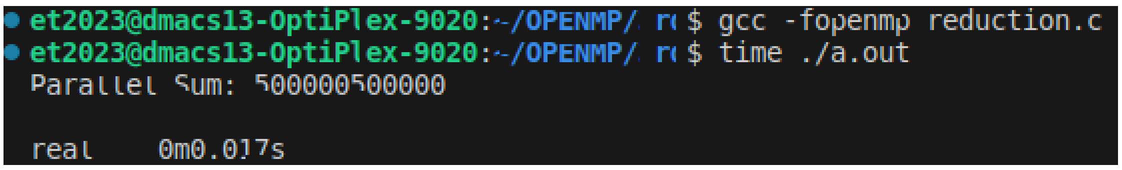
Name: Nitesh Sharma
Reg No: 23361
MTCS-103(P)
Parallel Processing Practicals
ASSignment 11

==
<u>reduction(operator:list):</u>

The reduction(operator:list) clause in OpenMP is used to perform accumulation operations on a list of variables, using a specified associative operator. This clause creates a private copy of each variable in the list for every thread, and each thread accumulates its partial result into its private copy. After the parallel region, the private copies are combined with the original variables using the specified associative operator. This approach ensures both thread-level parallelism and correct accumulation of results without data races. The final result is the accumulated value of the original variables after the reduction operation is applied.

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

#define ARRAY_SIZE 1000000

int main() {
  long long int array[ARRAY_SIZE];

  for (int i = 0; i < ARRAY_SIZE; ++i) {
     array[i] = i + 1;
  }

  long long int sum = 0;
  #pragma omp parallel for reduction(+:sum)
  for (int i = 0; i < ARRAY_SIZE; ++i) {
     sum += array[i];
  }

  printf("Parallel Sum: %lld\n", sum);

  return 0;
}
```

Output:

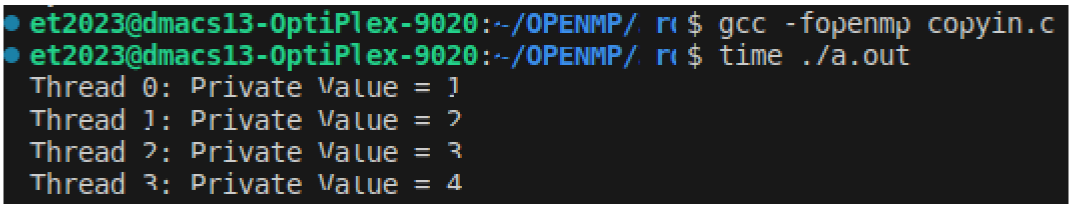


```
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r$ gcc -fopenmp reduction.c
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r$ time ./a.out
Parallel Sum: 500000500000

real    0m0.017s
```

Explanation:
The array is first initialized with consecutive values. Then, a parallel loop distributes the work across multiple threads, where each thread computes a portion of the sum. The reduction clause ensures that each thread maintains a private copy of the sum and accumulates its local sum into a shared sum variable. Once all threads complete their computation, the shared sum variables are combined using the specified reduction operation (in this case, addition), resulting in the final parallel sum.

copyin(list):
The copyin clause in OpenMP copies the value of a master thread's threadprivate variable to the threadprivate variable of every other thread within the team executing the parallel region. This ensures that each thread has its own private copy of the variable initialized with the value from the master thread, maintaining thread-specific data consistency and isolation. This mechanism helps avoid unintended data sharing and synchronization issues that might arise from concurrent access to shared variables across multiple threads.

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

int thread_private_value = 0;
#pragma omp threadprivate(thread_private_value)

int main() {
  #pragma omp parallel copyin(thread_private_value)
  {
      int thread_id = omp_get_thread_num();
      thread_private_value = thread_id + 1;

      printf("Thread %d: Private Value = %d\n", thread_id, thread_private_value);
  }

  return 0;
}
```

Output:

```
● et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r $ gcc -fopenmp copyin.c
● et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r $ time ./a.out
  Thread 0: Private Value = 1
  Thread 1: Private Value = 2
  Thread 2: Private Value = 3
  Thread 3: Private Value = 4
```

Explanation:
In the above code, the copyin directive is used to manage threadprivate variables in an OpenMP parallel region. The thread_private_value is declared as a threadprivate variable using #pragma omp threadprivate, which means each thread will have its private copy of this variable. The copyin clause copies the value of the master thread's thread_private_value to the corresponding private copy of each thread. As the program enters the parallel region, each thread prints its private value based on its thread ID, demonstrating that the private value is successfully assigned and shared among the threads.