

Pratik Dhimal

2407779

```
completion terminated.  
→ Week7 git:(master) x pwd && ./1  
/Users/pratikdhimal/Developer/sem_5_hpc/Week7  
Average = 499.500000  
→ Week7 git:(master) x █
```

The numbers above 0 to 999 are entered into an array in this program and then the total of these numbers is calculated to obtain the average. Lastly, it gives the total divided by 1000 and values of the average are printed.

```
Average = 499.500000  
⊗ → Week7 git:(master) x pwd && ./2  
/Users/pratikdhimal/Developer/sem_5_hpc/Week7  
Average = 499.500000  
○ → Week7 git:(master) x █
```

In this program, an array of values 0-999 is filled and an attempt is made to calculate the average with an OpenMP parallel loop. The mean is however distributed among threads without reducing it resulting in a race condition and wrong conclusion.

```
→ Week7 git:(master) x pwd && ./3  
/Users/pratikdhimal/Developer/sem_5_hpc/Week7  
Average = 499.500000  
→ Week7 git:(master) x █
```

In this program, OpenMP parallelism is used to compute the average of the numbers ranging 0 to 999 using the reduction clause to safely add the numbers of threads. By the reduction, the race conditions are eliminated and therefore the end average is calculated appropriately.

```
⊗ → Week7 git:(master) x pwd && ./4
    /Users/pratikdhimal/Developer/sem_5_hpc/Week7
    Average = 499.500000
○ → Week7 git:(master) x █
```

This program averages numbers between 0 and 999 by using the OpenMP technique with the critical section that is used to make sure only a single thread is updating the average at any given time. This is better in avoiding race conditions but decreases the performance because of thread synchronization overhead.

```
Average = 499.500000
→ Week7 git:(master) x pwd && ./5
    /Users/pratikdhimal/Developer/sem_5_hpc/Week7
→ Week7 git:(master) x █
```

This program executes parallel matrix multiplication with the help of OpenMP to calculate matrix C by the use of A ( $4 \times 3$ ) and B ( $3 \times 4$ ) matrices. The outer loop is made parallel where several rows of matrix C can be calculated at a time thus enhancing performance. No results are obtained since the matrices have not been initialized and printed.

```
→ Week7 git:(master) x pwd && ./6
/Users/pratikdhimal/Developer/sem_5_hpc/Week7
38.000000 17.000000 23.000000 29.000000
83.000000 44.000000 59.000000 74.000000
128.000000 71.000000 95.000000 119.000000
15.000000 9.000000 12.000000 15.000000
→ Week7 git:(master) x
```

This program uses OpenMP to compute parallel matrix multiplication of A (4x3) and B (3x4), to compute C (4x4). Note: omp is missing in the blank line containing the pragma parallel for in the printing loop and therefore it is not considered and printing is a serial process.

```
→ Week7 git:(master) x pwd && ./7
/Users/pratikdhimal/Developer/sem_5_hpc/Week7
27.000000 3.000000 6.000000 9.000000
54.000000 6.000000 12.000000 18.000000
81.000000 9.000000 18.000000 27.000000
9.000000 1.000000 2.000000 3.000000
→ Week7 git:(master) x █
```

This program is parallel matrix multiplication implemented with the help of OpenMP, however, the collapse(3) is wrong since only two nested loops (I and j) can be collapsed. Although this error (which can create an error in compilation or be overlooked) exists, the logic will compute matrix C correctly in case it is run.

```
9.000000 1.000000 2.000000 3.000000
⊗ → Week7 git:(master) x pwd && ./8
/Users/pratikdhimal/Developer/sem_5_hpc/Week7
Thread 1: C0unter=0
Thread 0: C0unter=1
Thread 0: C0unter=2
Thread 0: C0unter=3
Thread 0: C0unter=4
Thread 7: C0unter=5
Thread 5: C0unter=6
Thread 5: C0unter=7
Thread 5: C0unter=8
Thread 5: C0unter=9
Thread 5: C0unter=10
Thread 5: C0unter=11
Thread 5: C0unter=12
Thread 3: C0unter=13
Thread 4: C0unter=14
Thread 3: C0unter=15
Thread 4: C0unter=16
Thread 3: C0unter=17
Thread 4: C0unter=18
Thread 3: C0unter=19
Thread 7: C0unter=20
Thread 3: C0unter=21
Thread 7: C0unter=22
Thread 1: C0unter=23
Thread 4: C0unter=24
Thread 4: C0unter=25
Thread 1: C0unter=26
Thread 1: C0unter=27
Thread 1: C0unter=28
```

This program employs an OpenMP lock to make sure that only one thread is updating the shared counter at once to eliminate race conditions. The counter is incremented 100 times by each thread safely hence the synchronization is correct yet not synchronized.