Name : Rahul Bhati

Roll no : 2223416 , LY-AIEC

Q2. Make a shared counter int hits=0; and a loop of N trials that increments hits when a condition shared(hits) and protect updates with atomic or critical.

holds (e.g., i%2==0). Use

```
[mit108@login01 ~]$ module load gcc
Lmod has detected the following error: The following module(s) are
unknown: "gcc"

Please check the spelling or version number. Also try "module spider ..."
It is also possible your cache file is out-of-date; it may help to try:
  $ module --ignore_cache load "gcc"

Also make sure that all modulefiles written in TCL start with the string
#%Module



[mit108@login01 ~]$ gcc --version
gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-22)
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[mit108@login01 ~]$ nano hits_openmp.c
[mit108@login01 ~]$ gcc -fopenmp -02 hits_openmp.c -o hits_openmp
gcc: error: unrecognized command line option '-02'
[mit108@login01 ~]$ gcc -fopenmp -O2 hits_openmp.c -o hits_openmp
[mit108@login01 ~]$ export OMP_NUM_THREADS=4
[mit108@login01 ~]$ ./hits_openmp 10000000
N = 10000000
OMP max threads = 4
Unsync hits = 1250000 (time 0.002117 s)
Atomic  hits = 5000000 (time 0.123858 s)
Critical hits = 5000000 (time 0.467428 s)
Reduction hits = 5000000 (time 0.001431 s)
Expected (N/2) = 5000000
[mit108@login01 ~]$
```

Q3. Define int x = 42; outside. In a parallel region, use firstprivate(x), then do x += omp_get_thread_num(); and print per thread. After the region, print x.

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

int main() {
    int x = 42; // shared variable before parallel region

    printf("Before parallel region: x = %d\n", x);

    #pragma omp parallel firstprivate(x)
    {
        // Each thread has its own 'x' initialized to 42
        x += omp_get_thread_num();
        printf("Thread %d: x = %d\n", omp_get_thread_num(), x);
    }

    // After parallel region, x is unchanged
    printf("After parallel region: x = %d\n", x);

    return 0;
}
```

```
[mit108@login01 ~]$ nano firstprivate_demo.c
[mit108@login01 ~]$ gcc -fopenmp firstprivate_demo.c -o firstprivate_demo
[mit108@login01 ~]$ export OMP_NUM_THREADS=4
[mit108@login01 ~]$ ./firstprivate_demo
Before parallel region: x = 42
Thread 0: x = 42
Thread 3: x = 45
Thread 2: x = 44
Thread 1: x = 43
After parallel region: x = 42
[mit108@login01 ~]$
```

Q4. Create a parallel region with an int t = -1; declared outside the region. Inside the region, use private(t) and set t = omp_get_thread_num(); then print t. After the region, print t.

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

int main() {
    int t = -1; // declared outside the parallel region

    printf("Before parallel region: t = %d\n", t);

    #pragma omp parallel private(t)
    {
        // Inside, 't' is private & uninitialized for each thread
        t = omp_get_thread_num();
        printf("Thread %d: t = %d\n", omp_get_thread_num(), t);
    }

    // After parallel region, 't' refers to the original variable (still -1)
    printf("After parallel region: t = %d\n", t);

    return 0;
}
```

```
[mit108@login01 ~]$ nano private_demo.c
[mit108@login01 ~]$ gcc -fopenmp private_demo.c -o private_demo
[mit108@login01 ~]$ export OMP_NUM_THREADS=4
[mit108@login01 ~]$ ./private_demo
Before parallel region: t = -1
Thread 0: t = 0
Thread 2: t = 2
Thread 1: t = 1
Thread 3: t = 3
After parallel region: t = -1
[mit108@login01 ~]$ ./private_demo
Before parallel region: t = -1
Thread 0: t = 0
Thread 2: t = 2
Thread 1: t = 1
Thread 3: t = 3
After parallel region: t = -1
```

Q5. Parallelize for (int i=1;i<=N;i++) result = i*i; with #pragma omp parallel for lastprivate(result). Print result after the loop (use N=10).*

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

int main() {
    int result = -1;
    int N = 10;

    printf("Before loop: result = %d\n", result);

    #pragma omp parallel for lastprivate(result)
    for (int i = 1; i <= N; i++) {
        result = i * i;
        printf("Thread %d processing i=%d, result=%d\n",
               omp_get_thread_num(), i, result);
    }

    printf("After loop: result = %d\n", result);

    return 0;
}
```

```
[mit108@login01 ~]$ ./lastprivate_demo
Before loop: result = -1
Thread 0 processing i=1, result=1
Thread 3 processing i=9, result=81
Thread 3 processing i=10, result=100
Thread 0 processing i=2, result=4
Thread 0 processing i=3, result=9
Thread 2 processing i=7, result=49
Thread 2 processing i=8, result=64
Thread 1 processing i=4, result=16
Thread 1 processing i=5, result=25
Thread 1 processing i=6, result=36
After loop: result = 100
```

Q6. Write a parallel for that sums an array a[] into sum using reduction(+:sum). Add default(none) and explicitly list shared(a,N).

```
[mit108@login01 ~]$ gcc -fopenmp reduction_demo.c -o reduction_demo
[mit108@login01 ~]$ ./reduction_demo
Sum of array: 55
```

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

#define N 10

int main() {
    int a[N];
    int i;
    int sum = 0;

    // Initialize array
    for (i = 0; i < N; i++) {
        a[i] = i + 1; // 1, 2, 3, ... N
    }

    // Parallel sum with reduction
    #pragma omp parallel for default(none) shared(a) reduction(+:sum)
    for (i = 0; i < N; i++) {
        sum += a[i];
    }

    printf("Sum of array: %d\n", sum);
    return 0;
}
```