Lab 4

The critical construct

- The CRITICAL directive specifies a region of code that must be executed by only one thread at a time.
- If a thread is currently executing inside a CRITICAL region and another thread reaches that CRITICAL region and attempts to execute it, it will block until the first thread exits that CRITICAL region.
- The optional name enables multiple different CRITICAL regions to exist:
  - Names act as global identifiers. Different CRITICAL regions with the same name are treated as the same region.
  - All CRITICAL sections which are unnamed, are treated as the same section.
-

```
#pragma omp critical [(name)]
structured block
```

Example(Inefficient)

```
max = INT_MIN;
#pragma omp parallel for shared(max)
for (i = 0; i < n; i++)
{
    #pragma omp critical
    if (a[i] > max)
        max = a[i];
}
```

The reduction clause

- The reduction clause performs a reduction on the variables that appear in the list, with the operator *operator*.
- The variables must be shared scalars (scalar: a variable that contains only one value).
- A private copy for each list variable is created and initialized for each thread. At the end of the reduction, the reduction variable is applied to all private copies of the shared variable, and the final result is written to the global shared variable.

reduction(*operator*: list)

Example

```
#pragma omp parallel reduction(+:sum)
for (i = 0; i < N; i++)
{
    double x = (i + 0.5) / N;
    sum += 4.0 / (1.0 + x * x);
}
```

The flush directive

- The flush directive synchronizes copies in register or cache of the executing thread with main memory.
- It synchronizes those variables in the given list; if no list is specified, all shared variables in the region.
- A flush is executed implicitly at all synchronization points.

#pragma omp flush[(list)]

Example(Improved)

```
max = INT_MIN;
#pragma omp parallel for shared(max)
for (i = 0; i < n; i++)
{
    #pragma omp flush(max)
        if (a[i] > max)
        #pragma omp critical
            if (a[i] > max)
                max = a[i];
}
```

The barrier construct

- The barrier synchronizes all threads in a team.
- When encountered each thread waits until all threads in that team have reached this point.
- All threads then resume executing in parallel the code that follows the barrier.
- The most common use for a barrier is for avoiding a race condition.

```
#pragma omp barrier
```

Example

```
#pragma omp parallel
for (i=0; i < n; i++)
{
        a[i] = b[i] + c[i];
}
#pragma omp barrier
for (i=0; i < n; i++)
{
        d[i] = a[i] + b[i];
}
```

The atomic construct

- An atomic construct ensures that a specific memory location is updated atomically (without interference).
- It specifies that a specific memory location must be updated atomically, rather than letting multiple threads attempt to write to it. In essence, this directive provides a mini-CRITICAL section.

#pragma omp atomic

Example

```
#pragma omp parallel shared(ic, n) private(i)
for(i = 0; i < n; i++)
{
    #pragma omp atomic
        ic = ic+1;
}
```

Efficient parallel code

```
max = INT_MIN;
#pragma omp parallel shared(max)
{
    int private_max = max;
    #pragma for
    for (i = 0; i < n; i++)
        if (a[i] > private_max)
            private_max = a[i];
    #pragma omp flush(max)
    if (private_max > max)
        #pragma omp critical
        if (private_max > max)
            max = private_max;
}
```

Assignment 4

- Write an openmp program to find the maximum of n matrices using suitable clauses.
- Execute the program with different number of threads(1, 2, 4, 6, 8, 10, 12, 16, 20, 24, 28, 32, etc.)
- Draw the graph and calculate the parallel fraction.

- Last date for submission: 11-09-18(10 PM)
- Rollnumber.zip