Critical Region

In exercise P3, you probably used an array to create space for each thread to store its partial sum.

If array elements happen to share a cache line, this leads to false sharing.

- Non-shared data in the same cache line so each update invalidates the cache line … in essence “sloshing independent data” back and forth between threads.

The critical construct is a directive that contains a structured block. The construct allows only a single thread at a time to execute the structured block (region). Multiple critical regions may exist in a parallel region, and may act cooperatively (only one thread at a time in all critical regions), or separately (only one thread at a time in each critical region when a unique name is supplied on each critical construct). An optional (lock) hint clause may be specified on a named critical construct to provide the OpenMP runtime guidance in selection of a locking mechanism.

Q1. Modify your “pi program” from P4Q1 to avoid false sharing due to the sum array by using the following steps:

1. Create a scalar local to each thread to accumulate partial sums.
2. No array, so no false sharing.
3. Sum goes “out of scope” beyond the parallel region … so you must sum it in here. Must protect summation into pi in a critical region so updates don’t conflict. (#pragma omp critical)

Output:

shared Specifies that one or more variables should be shared among all threads.

OpenMP Clauses | Microsoft Docs - #pragma omp parallel shared(var)

critical Specifies that code can only be executed on one thread at a time.

OpenMP Directives | Microsoft Docs - #pragma omp critical(var)

P4 - Concurrency Control (Part 1)

Q2. Compute the average of array A with a vector of MAX = 100000 in parallel. Below is the serial program which you can download here:

```
double ave=0.0, A[MAX];
int i;
for (i=0;i<MAX;i++){
ave += A[i];
}
ave = ave/MAX;
```

1st Method: #pragma omp parallel, #pragma omp critical
Reference:
https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170#critical

2nd Method: #pragma omp parallel for reduction(+:ave)

Question (discussion):
Why the sequential (without #pragma omp critical) run faster / same speed as running the code in threads using #pragma omp critical ?

Producer Consumer

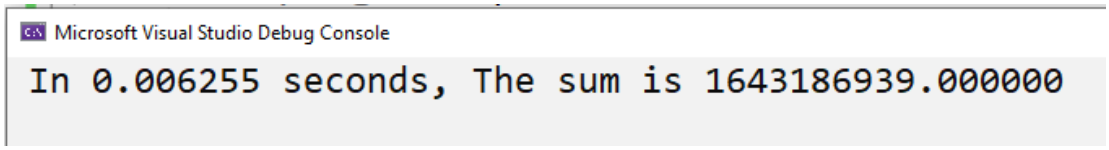
Q3. Parallelize the P4Q3.c program. This is a well known pattern called the producer consumer pattern. Using #pragma omp sections and #pragma omp flush

- One thread produces values that another thread consumes.
- Often used with a stream of produced values to implement “pipeline parallelism”.
- The key is to implement pairwise synchronization between threads.
- Producer needs to tell consumer that the data is ready
- Consumer needs to wait until data is ready
- Producer and consumer need a way to communicate data
    - output of producer is input to consumer
    - Producer and consumer often communicate through First-in-first-out (FIFO) queue

#pragma omp sections - Identifies code sections to be divided among all threads.
https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170#sections-openmp

#pragma omp flush - Specifies that all threads have the same view of memory for all shared objects.
https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170#flush-openmp

Output:



The *atomic* Construct

The following example avoids race conditions (simultaneous updates of an element of x by multiple threads) by using the atomic construct.

Atomic = ~~**one at a time**~~ **in one go**

The advantage of using the atomic construct in this example is that it allows updates of two different elements of x to occur in parallel. If a critical construct were used instead, then all updates to elements of x would be executed serially (though not in any guaranteed order).
Note that the atomic directive applies only to the statement immediately following it.
As a result, elements of y are not updated atomically in this example.

Q4. Modify the P4Q4.c and add atomic construct (#pragma omp atomic) inside the parallel construct to ensure the respective final sum of array of x and y are consistent.

#pragma omp atomic - Specifies a memory location that will be updated atomically.
https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170#atomic