OpenMP

Dr. William McDoniel and Prof. Paolo Bientinesi

HPAC, RWTH Aachen
mcdoniel@aices.rwth-aachen.de

WS17/18

Synchronization

OpenMP:

- Shared-memory programming model
- Unintended sharing of data causes race conditions
- Protect data conflicts with synchronization

Main constructs/tools:

- Critical sections:
  - `critical`
  - `atomic`
- Barriers: `barrier`
- Locks (low level)

Synchronization - Critical sections

- Syntax:

```
#pragma omp critical
```

- When?
  - Every thread must execute a section of the code
  - They can execute it in any order
  - Mutual exclusion is required

Synchronization - Critical sections

Don't do this at home! Why? (See next slide :))

```
double sum = 0.0, pi;
double step = 1.0/NUM_STEPS;
double x_i;
int i;

#pragma omp parallel for private(x_i)
for ( i = 0; i < NUM_STEPS; i++ )
{
    x_i = (i + 0.5) * step;
    #pragma omp critical
    sum = sum + 4.0 / (1.0 + x_i * x_i);
}
pi = sum * step;
```

Synchronization - Critical sections

To keep in mind...

- Minimize the size of critical sections
  - Refactor and pull heavy work outside the region
  - Have local copies, then reduce in a critical region
- Make sure you don’t hit the critical section too often
- Otherwise, the overhead will kill performance

Synchronization - Atomic

- Syntax:

```
#pragma omp atomic
```

- The `atomic` construct is a very restricted form of `critical`
- Often, hardware provides support to perform quick updates of memory locations
- If those hardware instructions are available, `atomic` tells the compiler to use them
- Otherwise, acts as a critical region

Synchronization - Atomic

- Restricted critical section, applied to a single statement. Valid statements:
  - `x++`, `x-`, `++x`, `-x`
  - `x binop= expr`
  - `x = x binop expr`
  - `x = expr binop x`

  where
  - `x` is of scalar type
  - `expr` is an expression of scalar type (which does not include `x`)
  - `binop` is one of `+`, `-`, `*`, `/`, `&`, `^`, `|`, `«`, `»`

Synchronization

Timings for the computation of π using multiple threads and different approaches for the mutually exclusive update of `sum`.

# threads	Sequential	`critical`	`atomic`	`reduction`
1	0.027 secs	0.031	0.026	0.030
2	0.027 secs	0.295	0.108	0.015
4	0.027 secs	0.572	0.117	0.008

Critical and atomic: Caveats

- The `critical` directive may be labeled with a name:

```
#pragma omp critical (name)
```

- Critical regions labeled with the same name are considered one single critical region
- Unnamed critical regions are considered to have the same name
- Do not mix `critical` and `atomic` to protect regions modifying the same storage location (considered different regions)

Synchronization - Barriers

- Syntax:

```
#pragma omp barrier
```

- Synchronizes all threads in the enclosing parallel region
- Ensures that all the code before the barrier has been executed by all threads before proceeding with the code beyond the barrier

Synchronization - Barriers

```
#pragma omp parallel
{
    int id = omp_get_thread_num();
    // A is an array of size n
    lot_of_work_inside( A, n, id );
    // No thread continues until all
    // done building pieces of A
    #pragma omp barrier
    B = more_work( A, n, id );
}
```

Synchronization - Barriers

- Many constructs imply a barrier (e.g., at the end of a parallel region)
- Thus, explicit barriers are often unnecessary

Careful:

- Each barrier must be encountered by all threads in the team (or none at all)
- The sequence of barriers encountered must be the same for every thread in the team