Lecture 18: OpenMP and MAXLOC

William Gropp

www.cs.illinois.edu/~wgropp
More OpenMP

- Not all computations are simple loops where the data can be evenly divided among threads without any dependencies between threads
- An example is finding the location and value of the largest element in an array
Example of use – maxloc

- Find the maximum value and its location in a vector
- for (i=0; i<n; i++) {
  if (x[i] > maxval) {
    maxval = x[i];
    maxloc = i;
  }
}
Atomic Updates

- All threads are potentially accessing and changing the same values – maxloc and maxval.
- OpenMP provides several ways to coordinate access to shared values
  - `#pragma omp atomic`
    - Only one thread at a time can execute the following statement (not block)
  - `#pragma omp critical`
    - Only one thread at a time can execute the following block
- Atomic *may* be faster than critical
  - Depends on hardware
Parallelize with OpenMP

• How would you parallelize this for loop with OpenMP?
  ♦ Write down the simplest parallelization
  ♦ Look for race conditions. What’s an easy way to handle them?
  ♦ Take a few minutes to try this.
Example of use – maxloc

• First, parallelize the for loop:

```c
#pragma omp parallel for
for (i=0; i<n; i++) {
    if (x[i] > maxval) {
        maxval = x[i];
        maxloc = i;
    }
}
```
Example of use – maxloc

- Second, handle the race condition
  
  ```
  #pragma omp parallel for
  for (i=0; i<n; i++) {
    #pragma omp critical
    {
      if (x[i] > maxval) {
        maxval = x[i];
        maxloc = i;
      }
    }
  }
  ```
Measured Performance

- Blue Waters node. Dual AMD Interlagos chips; 16 integer cores/chip. 2.3-2.6GHz clock.
- 8 Threads, 114ms for n=1,000,000
- Is this good? Bad?
- How would you answer that question? Take a few minutes to think about it and write down a short answer.
Performance Estimate

- \( N^*(c+r+b) \)
- \( C = \text{float (not pipelined)} \)
- \( R = \text{Read of } x[i] \)
- \( B = \text{Branch} \)
- Saves are to registers (ignore)

For an order of magnitude estimate, what values would you use for a 2.6 GHz CPU? Assume \( N \) is \( O(10^6) \)?
Performance Estimate

- $N \times (c+r+b)$
- $C = \text{float (not pipelined)} = 10^{-9}$
- $R = \text{Read of } x[i] = 10^{-9} \text{ sec/word}$
- $B = \text{Branch} = 4 \times 10^{-9} = 10^{-8}$

For an order of magnitude estimate:

- $10^6 \times (10^{-9} + 10^{-9} + 4 \times 10^{-9}) = 6 \text{ms}$

This is a very rough estimate, but...
Performance Estimate

- To answer the original question...
- Not good – our measured performance with 8 threads and the OpenMP code was 141ms – over 20 times as slow as our estimate.
Critical Sections Can Be Costly

• The Critical Section is costly in two ways:
  ♦ Acquiring the critical section often requires a reading and writing from memory (not just cache) – and unpredictable, so cost includes full memory latency
  ♦ Only one thread at a time can be within the critical section

• Code may serialize
Comparison of Serial and OpenMP Versions

Overhead of critical section
Observations

• For 1 thread, 1,000,000 critical sections adds about 74 ms
  ♦ Each critical section really pretty fast at 74ns (a few hundred clock cycles)

• Near linear behavior as threads added
  ♦ More threads take longer

• Hypothesis: threads contenting for the same lock:
  ♦ Code serializes
  ♦ Extra overhead proportional to the number of threads
Avoiding the Critical Section

• Performance poor because we insisted on keeping track of the maxval and location during the execution of the loop.
• We don’t care about the value during the execution of the loop – just the value at the end.
• This is a common source of performance issues:
  ♦ The description of the method used to compute a value imposes additional, unnecessary requirements or properties
Remove Dependency Between Threads

- Idea – Have each thread find the maxloc in its own data, then combine
  - Use temporary arrays indexed by thread number to hold the values found by each thread
Part 1: Finding the maxloc for each thread

```c
int maxloc[MAX_THREADS], mloc;
double maxval[MAX_THREADS], mval;
#pragma omp parallel shared(maxval,maxloc)
{
    int id = omp_get_thread_num();
    maxval[id] = -1.0e30;
#pragma omp for
    for (int i=0; i<n; i++) {
        if (x[i] > maxval[id]) {
            maxloc[id] = i;
            maxval[id] = x[i];
        }
    }
}
```
Part 1: Finding the maxloc for each thread

```c
int maxloc[MAX_THREADS], mloc;
double maxval[MAX_THREADS], mval;
#pragma omp parallel shared(maxval,maxloc)
{
    int id = omp_get_thread_num();
    maxval[id] = -1.0e30;
#pragma omp for
    for (int i=0; i<n; i++) {
        if (x[i] > maxval[id]) {
            maxloc[id] = i;
            maxval[id] = x[i];
        }
    }
```

Part 2: Combining the values from each thread

```c
#pragma omp flush (maxloc,maxval)
#pragma omp master
{
    int nt = omp_get_num_threads();
    mloc = maxloc[0]; mval = maxval[0];
    for (int i=1; i<nt; i++) {
        if (maxval[i] > mval) {
            mval = maxval[i];
            mloc = maxloc[i];
        }
    }
}
```
Part 2: Combining the values from each thread

```c
#pragma omp flush (maxloc,maxval)
#pragma omp master
{
    int nt = omp_get_num_threads();
    mloc = maxloc[0]; mval = maxval[0];
    for (int i=1; i<nt; i++) {
        if (maxval[i] > mval) {
            mval = maxval[i];
            mloc = maxloc[i];
        }
    }
}```
Scaling with Number of Threads

maxloc-2
Performance Evaluation

• Is our solution good?
  ♦ Nice scaling between 8-16 threads
  ♦ Time at 8 threads about 6ms, comparable to our performance estimate

• This is a good time to discuss the limits of “back of the envelope” performance models
  ♦ Lets compare with
    • A wider range of thread numbers (1,2-32)
    • The serial code (no OpenMP) – Always good to compare with the non-parallel, simple code
Performance for Maxloc
N=1,000,000
Observations

• Serial code is about 4x faster than our simple estimate
  • Not bad, but
• Parallel code has high overhead for parallelism (1-8 threads)
• Parallel code *never* faster than serial code
What Went Wrong?

• The code is simple; each thread is referencing different elements in x and in the maxloc and maxval arrays.

• The code to combine the final results only has 32 elements or less to look at.

• But there is a dependency – something *is* shared. What is it?
False Sharing

• Consider this code:

Thread 0
N=100000;
While (N--) a++;

Thread 1
M = 100000;
While (M--) b++;

How many cache misses occur?
1 Model: 4: N, M, A, B.
False Sharing (2)

- Consider this case
  - A, B, N, M are all in the same cache line
  - A processor may only write to a value if it is in that cores L1 cache
  - A and B are written to memory (store), not just updated in register

- Then instead of 4 cache misses, there are as many as 200000 (one for each access to either A or B)

- This is not a correctness problem; it is a performance problem
  - The programming language *hides* the hardware-defined associating between variables
Ensure that each thread accesses a different cache line

typedef struct { double val; int loc; char pad[128]; } tvals;
#pragma omp parallel shared(maxinfo)
{
    int id = omp_get_thread_num();
    maxinfo[id].val = -1.0e30;
#pragma omp for
    for (int i=0; i<n; i++) {
        if (x[i] > maxinfo[id].val) {
            maxinfo[id].loc = i;
            maxinfo[id].val = x[i];
        }
    }
}
Ensure that each thread accesses a different cache line

typedef struct { double val; int loc; char pad[128]; } tvals;
#pragma omp parallel shared(maxinfo)
{
    int id = omp_get_thread_num();
    maxinfo[id].val = -1.0e30;
#pragma omp for
    for (int i=0; i<n; i++) {
        if (x[i] > maxinfo[id].val) {
            maxinfo[id].loc = i;
            maxinfo[id].val = x[i];
        }
    
    }
}
Performance for Maxloc
N=1,000,000
Questions for Discussion

• What other ways could you ensure that each thread updated data on a separate cache line?
• What if the number of threads was 1024? How would you parallelize the second loop over the values found by each thread?