False Sharing

Cache Performance Issues in Parallel Programming

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Consider the Following Code

- The code finds the index at which the largest value resides
- This will have pretty bad performance because threads will be hitting the critical section continuously, leading to serialization

```
curMax = MINUS_INFINITY; maxIndex = -1;
#pragma omp parallel for
for(i=0; i<n; i++){
#pragma omp critical
  {
    if (a[i] > curMax)
      { curMax = a[i]; maxIndex = i}
  }
}
```

Maximum Index: Improved

- The problem is: in every iteration, we enter the critical section
  - What if we enter the critical section only after the test?
  - This might be incorrect because another thread might have changed the cur_ max in the meanwhile
- We repeat the test but it is correct now
- Further, we hit the critical section less often, reducing contention and serialization

```
curMax = MINUS_INFINITY; maxIndex = -1;
#pragma omp parallel for
for(i=0; i<n; i++){
  if (a[i] > curMax)
  #pragma omp critical
  {
    if (a[i] > curMax)
      { curMax = a[i]; maxIndex = i}
  }
}
```

Maximum Index: Better Strategy

- The previous code still has a potential for significant contention for the critical section
- We can avoid this by privatizing maxIndex, having each thread keep track of it in the parallel region, and selecting the maximum from among each thread's maximum at the end
    - We will explicitly privatize maxIndex, by using an array maxIndices[] indexed by thread ID

Maximum Index Using Private Variables

```
int* maxIndices = new int[p];
//p is number of threads
#pragma omp parallel
{
 int id = omp_get_thread_num();
 maxIndices[id] = 0;
#pragma omp for
  for(i=0; i<n; i++)
     if (a[i] > a[ maxIndices[id] ])
       maxIndices[id] = i;
}
maxIndex = maxIndices[0];
  for(i=1; i<p; i++)
    if (a[maxIndices[i]] > a[maxIndex])
      maxIndex = a[maxIndices[i];
// now maxIndex is the index of the
largest value in the array
curMax = a[maxIndex];
```

We expect this code to have good performance because there is no critical section in the main loop

And the second loop executed by the master is short

But we find the performance is not very good

Why?

The Problem: Cache Traffic Increase

- The problem with the code is that different threads are writing to the same cache line
  - When they write to `maxIndices[i]`
  - The cache line is continuously shuttled between caches of different cores
  - This is sometimes called “thrashing”
- This is inspite of the fact that each thread writes only to its own location – `maxIndices[id]`
  - I.e., there is no sharing of data
- This is called false sharing
- A common solution to this problem is “padding,” so that each thread writes to its own cache line

```
typedef struct{
int index;
char padding[28]
}
PaddedIndex;
```

`maxIndices[]` is now an array of these structures

`maxIndices[id].index` should be used instead of

`maxIndices[id]` in the code of the previous slide

This assumes a 32-byte cache line

Recap: False Sharing

- False sharing happens when:
  - Two or more threads access the same cache line for writes
  - Over a common time interval
  - Writing to distinct variables
    - I.e., no two threads are writing to the same variable
    - So, there is no real sharing
  - But because write access involves bringing the whole cache line to my private (say L1) cache, there is continuous traffic, leading to performance loss
  - This is "unnecessary" in the sense that if cache line size was one word, it would not happen
- One technique for eliminating false sharing is padding
  - So, data accessed by distinct threads is in different cache lines