Performance Modeling of Shared Memory Systems

09/11/2020

Some content from Bill Gropp, University of Illinois

Simple Max Loop

- Let's go back to finding a maximum (or minimum) from a list

```
for (i=0; i<n; i++) {
   if (x[i] > maxval) {
      maxval = x[i];
      maxloc = i;
   }
}
```

- How do we model the serial performance?

Simple Max Loop

- Let's go back to finding a maximum (or minimum) from a list

```
for (i=0; i<n; i++) {
   if (x[i] > maxval) {
      maxval = x[i];
      maxloc = i;
   }
}
```

- n*(c+r+b)

c	Floating point operation
r	Read of x[i]
b	Branch

Blue Waters Approximate Values

- Assume n = 1e6

c	Floating point operation	1E-09
r	Read of x[i]	1E-09
b	Branch	4 x 1E-09

- n*(c+r+b) = 1e6 * (1e-9 + 1e-9 + 4*1e-9) = 6ms

OpenMP Max Loop : Critical Section

- What about if we add threads?

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

- How do we model the parallel performance?

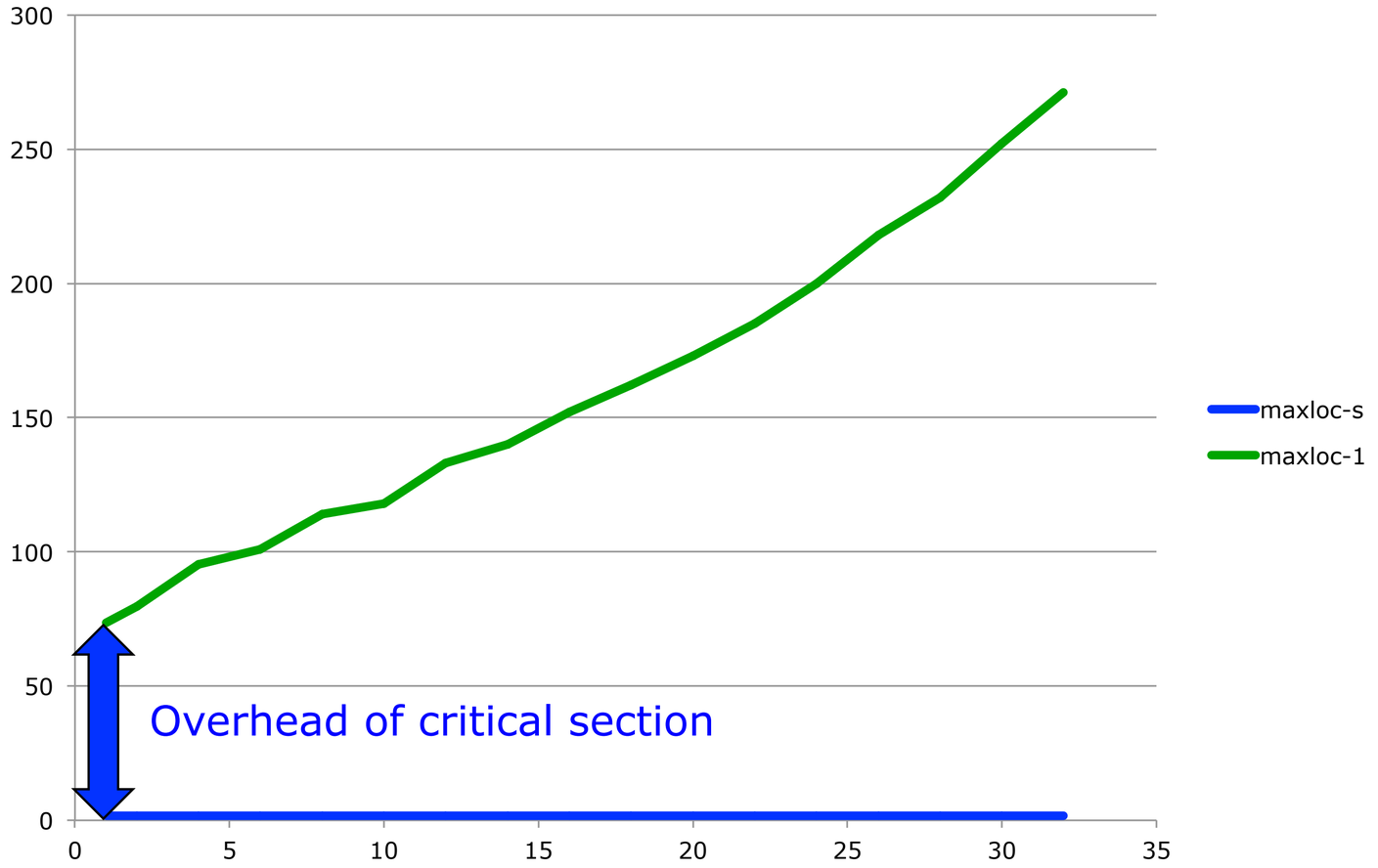
How about using the serial performance model?

- Critical section serializes code

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

- If n = 1e6, measured time for 8 threads is 141ms… **over 20x more expensive than our performance model estimate of 6ms**

Comparison of Serial and OpenMP Versions



OpenMP Critical Sections

- With 1 thread, 1,000,000 critical sections adds 74ms
  - 74ns per critical section, so relatively fast
- More threads take longer, linearly
- Hypothesis : threads contenting for same lock
  - Serialized code
  - Extra overhead proportional to number of threads

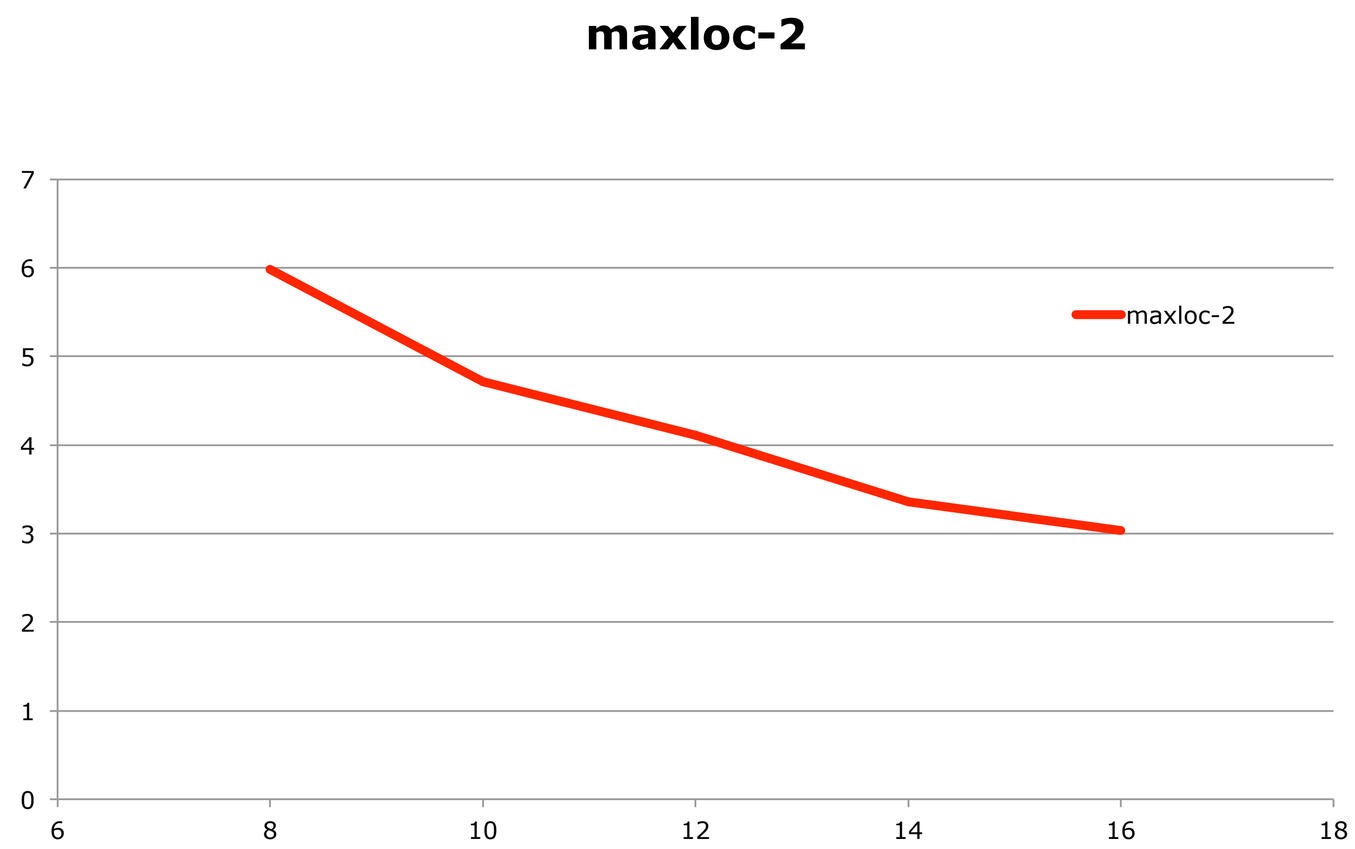
OpenMP Critical Sections

- Performance is so poor because we keep track of maximum value and locations throughout the loop
- But, we don’t care about their value during the loop! Only after…
- Common source of performance issues
  - Description of the method used to compute a value imposes additional overhead, unnecessary requirements or properties

Each Thread Finds Own Max

```
int maxloc[MAX_THREADS], mloc;
int maxval[MAX_THREADS], mval;
#pragma omp parallel shared(maxval, maxloc) {
   int id = omp_get_thread_num();
   maxval[id] = x[0];
   maxloc[id] = 0;
   #pragma omp for
   for (int i = 0; i < n; i++){
   #pragma omp critical {
       if (x[i] > maxval[id]) {
           maxval[id] = x[i];
           maxloc[id] = i;
       }
   }
   #pragma omp master {
       for (int i = 0; i < n_threads; i++) …
   }
}
```

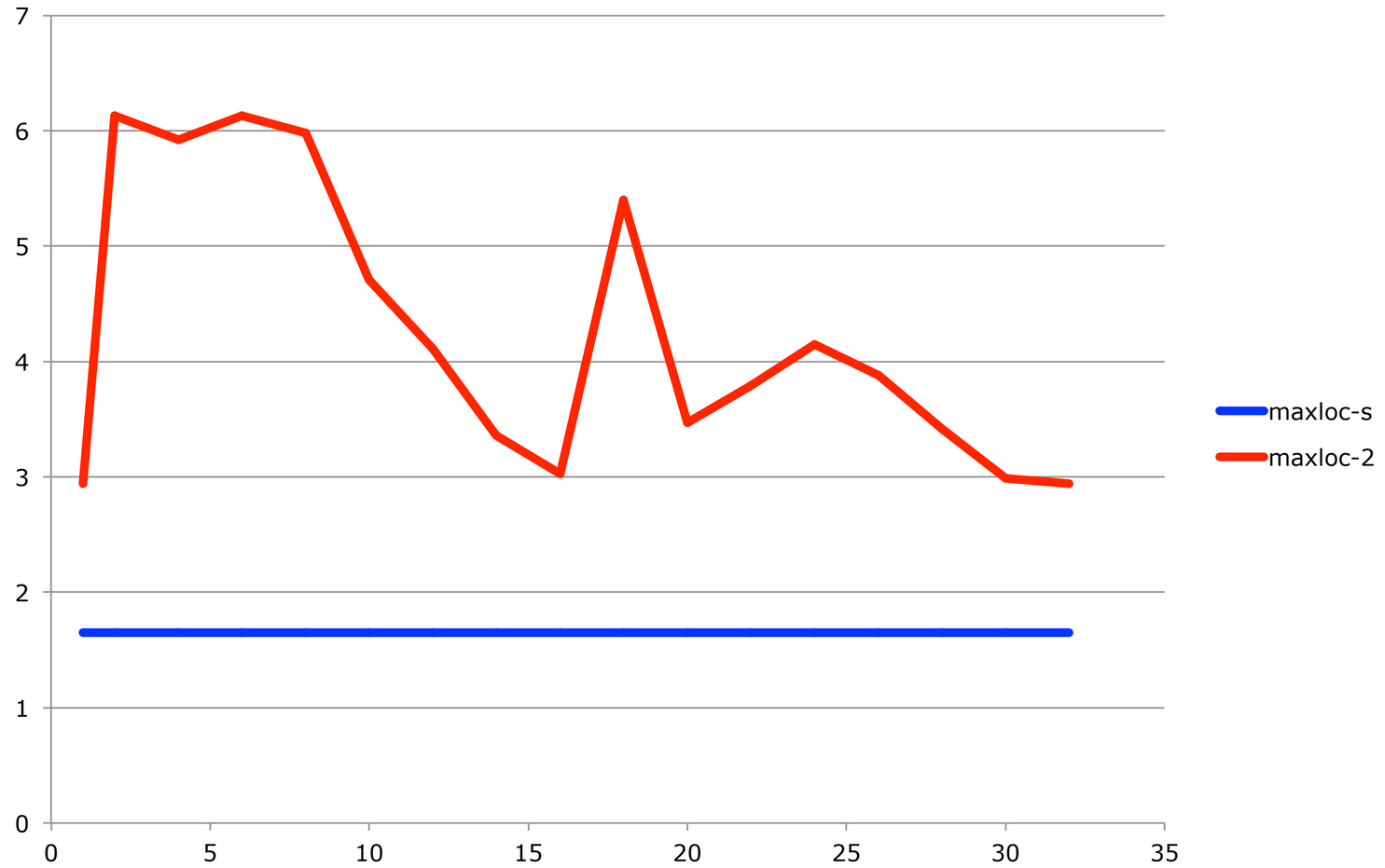
Scaling with Number of Threads



Does this method have good performance?

- Is this a good implementation?
  - At 8 cores, takes about 6ms… same as single core performance model
  - Good scaling between 8 and 16 cores
- Limits of “Back of the envelope” performance models
  - Let’s look at more threads (1-32) and compare with serial code

Performance for Maxloc N=1,000,000



Does this method have good performance?

- Serial code is 4x faster than our simple performance estimate
- Parallel code has high overhead
  - Look at 1-8 threads
- Parallel code is *never* faster than serial code

False Sharing!

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

False Sharing Example

- Consider the following code:

```
Thread 0 : N = 100000; While (N--) a++;
Thread 1 : M = 100000; While (M--) b++;
```

- How many cache misses occur? A, B, N, M (4)

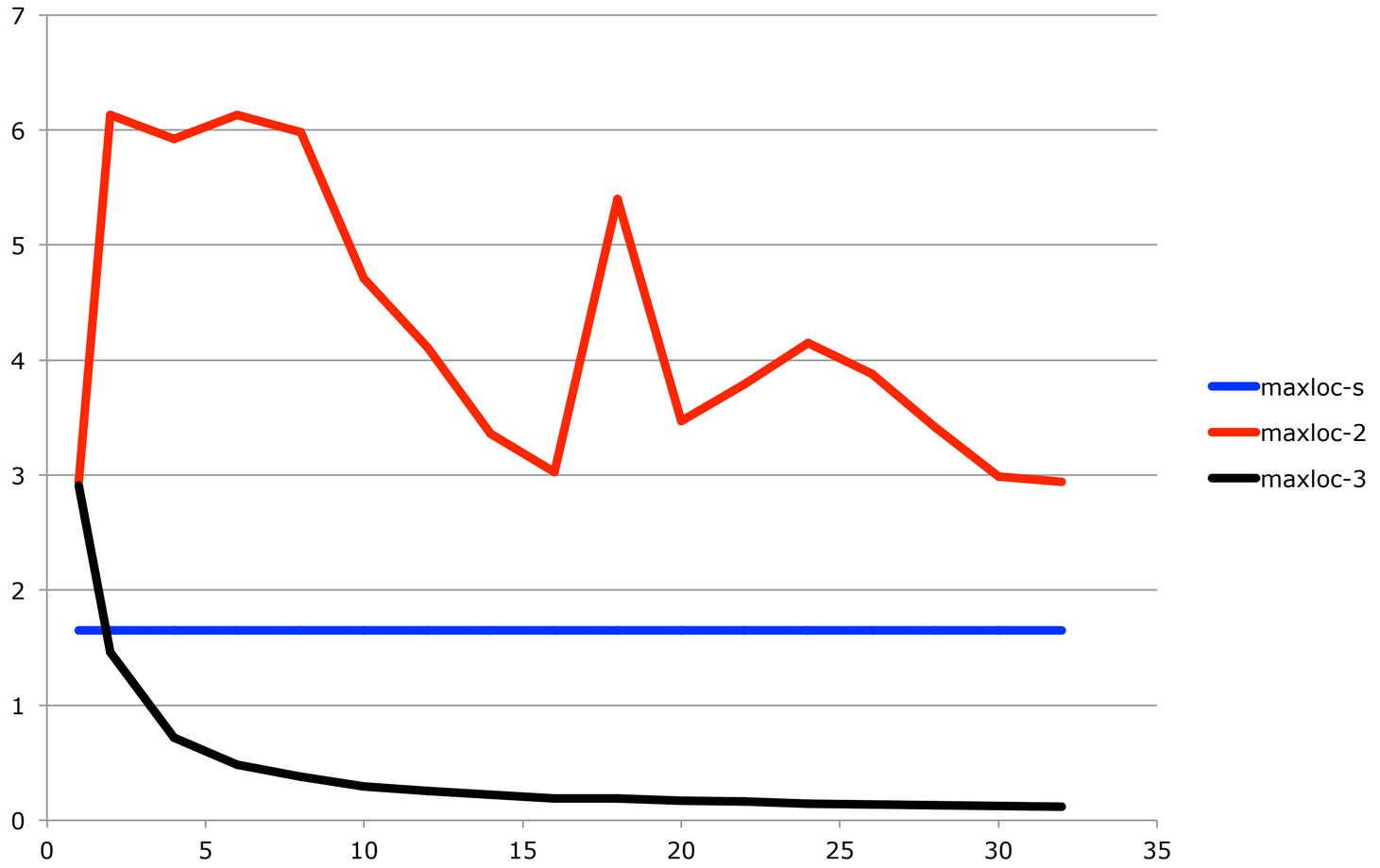
False Sharing

- Consider the following:
  - A, B, N, M all on same cache line
  - Processor may only write to a value if it is in that core's L1 cache
  - A and B are written to memory, not just updated in register
- Instead of 4 cache misses, there are as many as 200,000 (one for each access to either A or B)
- Not a correctness problem
- Performance problem

Removing False Sharing

```
typedef struct { double val; int loc; char pad[128]; } tvals;
#pragma omp parallel shared(maxinfo) {
   int id = omp_get_thread_num();
   maxinfo[id].val = x[0]; maxinfo[id].loc = 0;
   #pragma omp for
   for (int i = 0; i < n; i++){
      if (x[i] > maxval[id].val) {
         maxval[id].val = x[i];
         maxloc[id].info = i;
       }
   }
   #pragma omp master {
       for (int i = 0; i < n_threads; i++) …
   }
}
```

Performance for Maxloc N=1,000,000



Parallel Performance Model

- How would you change the performance model for our parallel code?

```
for (i=0; i<n; i++) {
   if (x[i] > maxval) {
      maxval = x[i];
      maxloc = i;
   }
}
```

c	Floating point operation
r	Read of x[i]
b	Branch

- Serial model : n*(c+r+b)