CS420 – Lecture 10

Raghavendra Kanakagiri
Slides: Marc Snir

Spring 2023

False sharing

```
int64_t num_steps = 10000000;
int main()
{
  int i, nthreads; double pi, sum[4];
  omp_set_num_threads(4);
  auto start = high_resolution_clock::now();
#pragma omp parallel
  {
    int i, id, nthrds; double x;
    id = omp_get_thread_num();
    nthrds = omp_get_num_threads();
    if (id == 0) nthreads = nthrds;
    for (i = id, sum[id] = 0.0; i < num_steps; i = i+nthrds) {
      x = pow(-1, i) / (2 * i + 1);
      sum[id] += x;
    }
  }
  for(i = 0, pi = 0.0; i < nthreads; i++)
    pi += sum[i];
  pi = 4 * pi;
  auto stop = high_resolution_clock::now();
```

Shared memory Semantics

- Always true: local operations appear *to local thread* to execute in program order

Shared memory Semantics

- Always true: local operations appear *to local thread* to execute in program order
- *Sequential consistency:* Operations appear to all threads to execute in same order (operations from different threads can interleave arbitrarily).

Sequential consistency

```
thread 0
---
x = 1
print(y)
```

```
thread 1
---
y = 1
print(x)
```

Sequential consistency

```
thread 0
---
x = 1
print(y, z)
```

```
thread 1
---
y = 1
print(x, z)
```

```
thread 2
---
z = 1
print(x, y)
```

Sequential consistency

```
thread 0
---
x = 1
print(y, z)
```

```
thread 1
---
y = 1
print(x, z)
```

```
thread 2
---
z = 1
print(x, y)
```

- 001011

Sequential consistency

```
thread 0
---
x = 1
print(y, z)
```

```
thread 1
---
y = 1
print(x, z)
```

```
thread 2
---
z = 1
print(x, y)
```

- 001011
- 001011 : t0, t0, t1, t1, t2, t2
- 101011

Sequential consistency

```
thread 0
---
x = 1
print(y, z)
```

```
thread 1
---
y = 1
print(x, z)
```

```
thread 2
---
z = 1
print(x, y)
```

- 001011
- 001011 : t0, t0, t1, t1, t2, t2
- 101011
- 101011 : t0, t1, t1, t0, t2, t2

Sequential consistency

```
thread 0
---
x = 1
print(y, z)
```

```
thread 1
---
y = 1
print(x, z)
```

```
thread 2
---
z = 1
print(x, y)
```

- 001011
- 001011 : t0, t0, t1, t1, t2, t2
- 101011
- 101011 : t0, t1, t1, t0, t2, t2
- 000011

Compilers

```
1 X = 0
2 for i in range(100):
3   X = 1
4    print X
```

Shared memory Semantics

- Always true: local operations appear *to local thread* to execute in program order

Shared memory Semantics

- Always true: local operations appear *to local thread* to execute in program order
- *Sequential consistency:* Operations appear to all threads to execute in same order (operations from different threads can interleave arbitrarily).

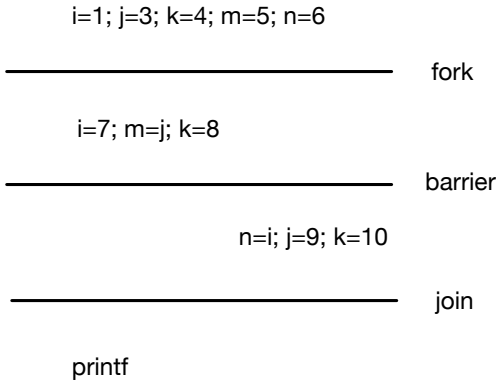
Shared memory Semantics

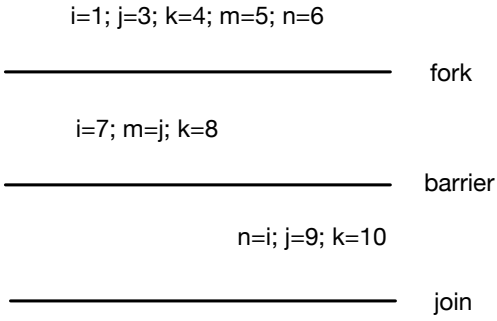
- Always true: local operations appear *to local thread* to execute in program order
- *Sequential consistency:* Operations appear to all threads to execute in same order (operations from different threads can interleave arbitrarily).
- *Most processors ARE NOT sequentially consistent*

Shared memory Semantics

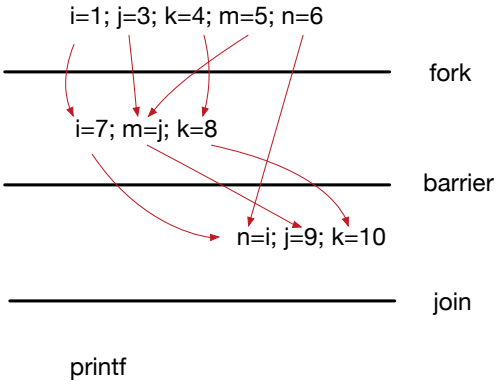
- Always true: local operations appear *to local thread* to execute in program order
- *Sequential consistency:* Operations appear to all threads to execute in same order (operations from different threads can interleave arbitrarily).
- *Most processors ARE NOT sequentially consistent*
- User should write only *race-free* code: If two threads perform conflicting accesses to a shared variable, then the accesses must be explicitly ordered by an OpenMP synchronization operation
- The OpenMP compiler and runtime will make sure that properly synchronized accesses appear to occur in the right order

```
...
int i=1,j=3,k=4,m=5,n=6;
omp_set_num_threads(2);
#pragma omp parallel
{
if(omp_get_thread_num()==0) {
i=7;m=j;k=8;
#pragma omp barrier
}
else {
#pragma omp barrier
n=i;j=9;k=10;
}
}
printf("i=%d,j=%d,k=%d,m=%d,n=%d \n",i,j,k
}
```





output is: i=7,j=9,k=10,m=3,n=7



Synchronization

- Ordering constructs: barrier, fork, join
- Mutual Exclusion constructs: atomic, critical, lock