

CS380P Parallel Systems

Calvin Lin
January 14, 2013

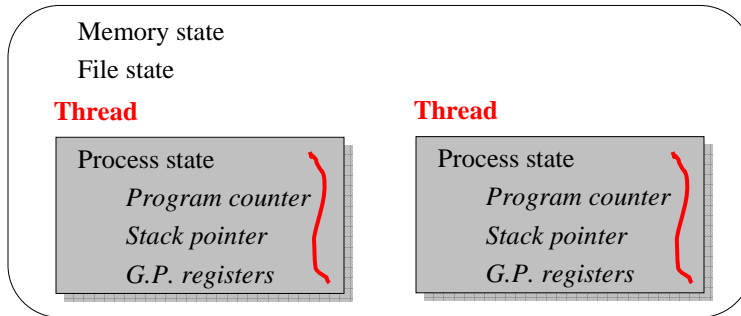
What Are We Getting Ourselves Into?

Let's look at a parallel program

- Consider a simple problem
- Use what many people consider the most convenient programming model because of its similarity to sequential programming, namely, threads

Multithreading

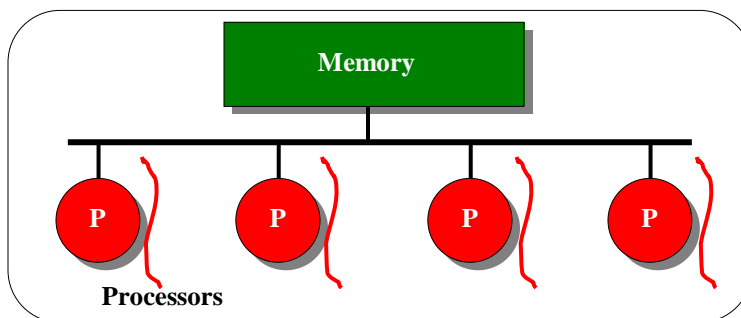
Process



Threads

- Each thread has its own process state, but threads share memory and file state

Symmetric Multiprocessors

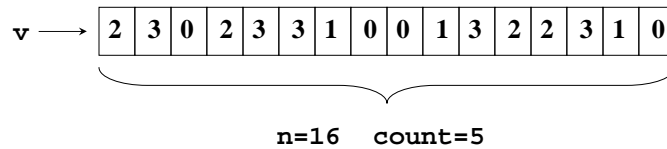


Symmetric Multiprocessor

- Processors share physical memory, and all physical memory is equidistant from all processors
- Multiple threads can execute in parallel on multiple processors, and threads can communicate through shared memory

Simple Example

Count the number of 3's in an array.



Serial code:

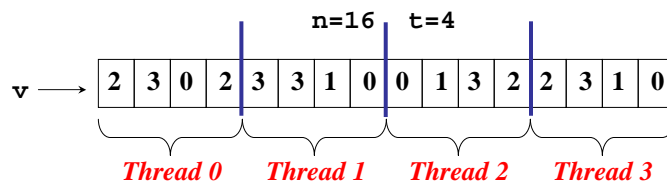
```
int *v;
int n;
int count;

void count3s () {
    count = 0;
    for (int i=0; i<n; i++)
        if (v[i]==3)
            count++;
}
```

Simple Example: Parallelization

Each thread is responsible for counting the 3's in some portion of the array.

With n elements and t threads, each thread is responsible for n/t array elements.



Simple Example—First Try

```
int t;

void count3s() {
    count = 0;

    /* Create t threads */
    for (i=0; i<t; i++)
        /*
         * Each thread calls count3s_thread with
         * parameter i
         */
        thread_create(count3s_thread, i);

    /* Wait for threads to terminate */
    for (i=0; i<t; i++)
        thread_join();
}
```

CS

Simple Example—First Try (cont)

```
void count3s_thread(int id) {

    /* Determine portion of array to work on */
    int n_per_thread = n/t;
    int start = id * n_per_thread; } extra
                                        } work

    /* Count the 3's in my portion of array */
    for (i=start; i<start+n_per_thread; i++)
        if (v[i]==3)
            count++;
}
```

Are there any problems with this code?

This code will not work because of a data race at the increment of count

Data Races

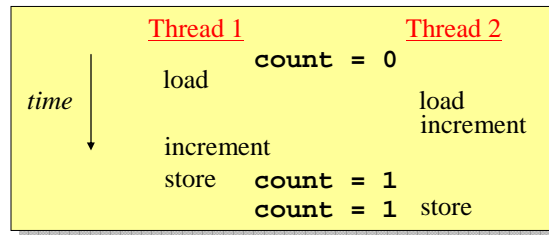
Definition

A **data race** occurs when two or more threads can modify the same memory location at the same time

Example

The statement `count++` is actually translated into 3 instructions:

1. Load `count` in register
2. Increment register contents
3. Store register in `count`



Mutual Exclusion

Solution

- To prevent the data race, we must ensure that at all times at most one thread is executing the `count++` statement
- We refer to such a condition as **mutual exclusion**, and we refer to such a statement as a **critical section**
- Mutual exclusion is achieved with a data object called a **mutex**

Mutexes

Definition

- A **mutex** (also called a **lock**) is a data object with
 - 2 states: **locked** and **unlocked**
 - 2 methods: **lock** and **unlock**
- When a thread locks a mutex, it must wait until the mutex is unlocked, and then it sets the mutex to the locked state

```
mutex m;  
  
mutex_lock(m);  
  
} critical  
section  
  
mutex_unlock(m);
```

Simple Example—Second Try

```
mutex m;  
  
void count3s_thread(int id) {  
    /* Count 3's in portion of array */  
    for (i=start; i<start+n_per_thread; i++)  
        if (v[i]==3) {  
            mutex_lock(m);  
            count++;  
            mutex_unlock(m);  
        }  
}
```

Does this code work correctly? **Yes.**

Simple Example—Second Try

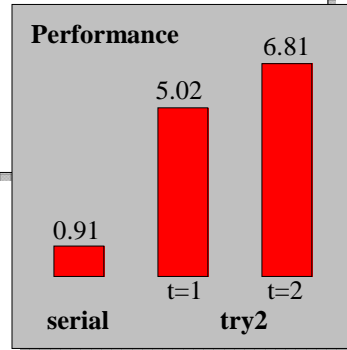
```

mutex m;

void count3s_thread(int id) {
    /* Count 3's in portion of array */
    for (i=start; i<start+n_per_thread; i++)
        if (v[i]==3) {
            mutex_lock(m);
            count++;
            mutex_unlock(m);
        }
}
    
```

Locking overhead is killing performance

Execution time in seconds
on SPARCstation 20
n=8 million, count=2 million



CS380P Lecture 1

Introduction

13

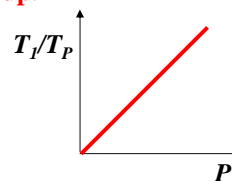
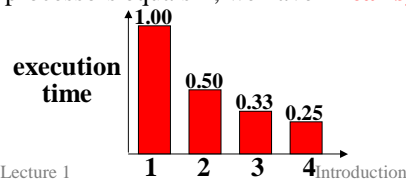
Performance Metrics

Notation

- T_s denotes the execution time of the serial program
- T_p denotes the execution time of the parallel program running on P processors

Definition

- T_s/T_p is the **efficiency** of the parallel program. Efficiency is always less than 1.0. We want efficiency close to 1.0
- For any number of P processors, T_s/T_p is the **speedup** of the parallel program on P processors. We want speedup close to P . If the speedup on P processors equals P , we have **linear speedup**.



CS380P Lecture 1

Introduction

14

Simple Example—Third Try

Each thread counts in a private counter, then combines at the end

```
int private_count[16];



void count3s_thread(int id) {
    for (i=start; i<start+n_per_thread; i++)
        if (v[i]==3)
            private_count[id]++;

    mutex_lock(m);
    count += private_count[id];
    mutex_unlock(m);
}
```

What's going on? *We have false sharing*

Execution time in seconds
on SPARCstation 20
n=8 million, count=2 million

Performance

0.91	0.91	1.15
		
serial	t=1	t=2

CS380P Lecture 1

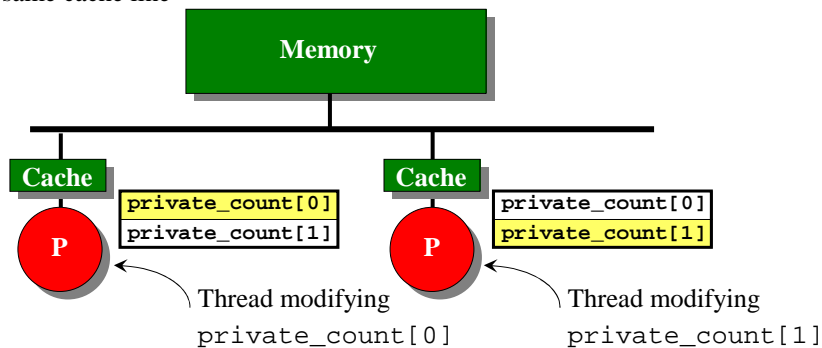
Introduction

15

False Sharing

Cache consistency

- On SMP's, caches are kept consistent
- False sharing occurs when 2 or more threads modify different data on the same cache line



The effort expended to maintain consistency can hurt performance

CS380P Lecture 1

Introduction

16

Simple Example—Success at Last




Allocate padding between private counters

```
struct padded_int {
    int value;
    char padding[32];
} private_count[16];

void count3s_thread(int id) {
    for (i=start; i<start+n_per_thread; i++)
        if (v[i]==3)
            private_count[id].value++;
    mutex_lock(m);
    count += private_count[id].value;
    mutex_unlock(m);
}
```

Execution time in seconds
on SPARCstation 20
n=8 million, count=2 million

Performance

0.91	0.91	0.51
		
serial	t=1	t=2

CS380P Lecture 1

Introduction

17

Lessons

Parallel programming is difficult

- The parallel code can be considerably more complicated than its sequential counterpart
- There's often more work to do in the parallel solution
- Getting things right can be tricky
- Getting good performance can be trickier
- Getting good performance can require knowledge of low-level details

Next class

- Architecture and micro-architecture

CS380P Lecture 1

Introduction

18

Next Class

No Free Lunch

- Read Chapter 1 of the text book for Wednesday
- Read No Free Lunch paper for Wednesday
 - Assignment:
 - Discuss anything that you found interesting or surprising
 - What are the main differences between Chapter 1 and the paper?
- Read Chapter 6 (pp.145-187) for Monday
- Sign up for TACC account