Case Study: Finding Primes
Example: List of Primes

• Consider the problem of finding the first million prime numbers
  • We want to output an array containing the first million primes

  - Note that parallel for requires that the loop bounds need to be known before the loop starts
  - So, how can we parallelize this?

Sequential Code:

```c
x = 3; j = 1; primes[0] = 2;
while(j < 1000000){
    if(isPrime(x)){
        primes[j] = x;
        j++;
    }
    x = x + 2;
}
```
Ideas for Parallelizing PrimesList

• We can have each thread explore the next unexplored odd integer beginning with 3

• Both x and j need to be protected because multiple threads want to read and write them

• We can use locks or atomic variables for this purpose

```c
x = 3; j = 1; primes[0] = 2;
while(j < 1000000){
    if(isPrime(x)){
        primes[j] = x;
        j++;
    }
    x = x + 2;
}
```
Version 1: Parallelizing PrimesList

- We protect writes to j and x using atomic
- Note that between the time a thread starts testing for x and the time it increments x, x may have been changed by other threads
  - So, after finishing testing 3, a thread may start working on testing 21, if other threads have already taken the number in between

```c
int x, j;
x = 3; j = 1; primes[0] = 2;
#pragma omp parallel
while (j < 1000000) {
    if (isPrime(x)) {
        primes[j] = x;
        #pragma omp atomic
        j++;
    }
    #pragma omp atomic
    x = x + 2;
}
```

Does this work?
Version 1: Parallelizing PrimesList

• The problem is between the
time we test the primality of a
number (x), and the time that
it executes the assignment
statement, some other thread
might have changed the value
of x

• Also, two different threads
may try to assign with the
same value of j

```c
int x,j;
x = 3; j = 1; primes[0] = 2;
#pragma omp parallel
while(j < 1000000){
   if(isPrime(x)){
      primes[j] = x;
      #pragma omp atomic
      j++;
   }
   #pragma omp atomic
   x = x + 2;
}
```

Does this work?
Version 2: Parallelizing PrimesList

- A thread saves the current value of x and increases it by 2 in a single critical section
  - No other thread can interfere
- A thread atomically increments j but saves the new value in its private variable k

```c
int x,j,myX,k;
x = 3; j = 0; primes[0] = 2;
#pragma omp parallel private(myX,k)
while(j < 1000000){
    #pragma omp atomic capture
    { myX = x; x = x + 2;}
    if(isPrime(myX)){
        #pragma omp atomic capture
        k = j++;
        primes[k] = myX;
    }
}
```

Does this work?
Version 2: Parallelizing PrimesList

- This almost works
- What are the problems?
- Is the primes array sorted?
  - No, although it is almost sorted
  - Some threads might run ahead
- How do we stop after the first one million primes?
  - While one thread is working on testing the millionth prime, another thread might finish testing the next prime and add it to the list

Fixing this is left as an exercise for you

```c
int x, j, myX, k;
x = 3; j = 0; primes[0] = 2;
#pragma omp parallel private(myX,k)
while(j < 1000000){
    #pragma omp atomic capture
    { myX = x; x = x + 2; }
    if(isPrime(myX)){
        #pragma omp atomic capture
        k = j++;
        primes[k] = myX;
    }
}
```
Parallelizing PrimesList: Ideas for Fixes

• Let the loop go further for a few more iterations
  • How many?
    • Maybe \( j < 1000000 + \text{numThreads} \)?
• Sort the array at the end?
  • Too expensive
  • And it’s mostly sorted

```c
int x, j, myX, k;
x = 3; j = 0; primes[0] = 2;
#pragma omp parallel private(myX,k)
while(j < 1000000){
  #pragma omp atomic capture
  { myX = x; x = x + 2; }
  if(isPrime(myX)){
    #pragma omp atomic capture
    k = j++;
    primes[k] = myX;
  }
}
```