Data Races

Course Level:
CS1/CS2

PDC Concepts Covered:

PDC Concept	Bloom Level
Data Races	C

Programming Knowledge Prerequisites:
- Know how to compile C++
- Be able to understand loops and functions

Tools Required:
C/C++: A C++ compiler that is OpenMP capable (e.g. the Gnu gcc C++ compiler)

Activity:

Let us consider a very simple situation. We want to sum the elements in an array in parallel. Look at the following code in C++:

```cpp
#include <iostream>
using namespace std;

int main()
{
        const int SIZE = 100000;
        int* array = new int[SIZE];

        for(int i = 0; i < SIZE; i++)
                array[i]=1;

        int sum = 0;

        #pragma omp parallel for num_threads(4)
        for(int i = 0; i < SIZE; i++)
        {
                sum += array[i];
        }

        cout << "sum: " << sum << endl;
}
```

This makes an array of 100,000 elements, sets each to one, and then calculates the sum using OpenMP or Pyjama. Compile and run this code. You may be surprised to see that the sum is not 100,000 like it should be. Run the program a couple more times. It doesn't even give the same wrong answer on each run.

The reason that this code is incorrect lies in the single line `sum += array[i];` When the processor runs this it is actually broken up into three major steps:

- Read `sum` and `array[i]` from memory
- Add `sum` and `array[i]` together
- Write the new value of `sum` back to memory

If we were running this with a single thread then this would not be a problem. However, we have multiple threads each doing the three steps at the same time. Look at the following two scenarios.

- Sum is 45
- Thread one reads 45
- Thread one adds 1 to 45 to get 46
- Thread one writes 46 to memory
- Thread two reads 46
- Thread two adds 1 to 46 to get 47
- Thread two writes 47 to memory
- Sum is 47

- Sum is 45
- Thread one reads 45
- Thread two reads 45
- Thread one adds 1 to 45 to get 46
- Thread one writes 46 to memory
- Thread two adds 1 to 45 to get 46
- Thread two writes 46 to memory
- Sum is 46

Both of these are possible and it is not possible to predict which will occur. This is a data race. The two threads are racing against each other to read, add, and write a single variable. Sometimes one thread wins and sometimes the other wins. This is the cause of the errors.

The way to fix this data race is to only let one thread update `sum` at a time. This can be done using the *critical* directive. For C++ simply add `#pragma omp critical` on the line above `sum += array[I;`

Add this, recompile, and run. You should now get the correct sum of 100,000.