Project #3

False Sharing

70 Points

Due: May 9

Introduction

Cache issues can be insidious. As we discussed in class, False Sharing is a way that multicore programs can significantly lose performance even though it looks like you've done nothing wrong.

In False Sharing, two (or more) cores are writing to variables that live in the same cache line. As soon as each one writes to the cache line, it invalidates the contents of the other, requiring a time-consuming re-load.

Requirements

- You have the following multicore application:

```
#include <stdlib.h>

struct s
{
	float value;
	int pad[NUMPAD];
} Array[4];

	. . .

	omp_set_num_threads( NUMT );

	const int SomeBigNumber = 100000000;  // keep < 2B

	#pragma omp parallel for
	for( int i = 0; i < 4; i++ )
	{
		unsigned int seed = 0;		// automatically private
		for( unsigned int j = 0; j < SomeBigNumber; j++ )
		{
			Array[ i ].value = Array[ i ].value + (float)rand_r(
&seed );
		}
	}
```

- Apply Fix #1 from the notes, i.e., pad the rest of the structure with a certain number, NUMPAD, of integers. Set NUMPAD to 0, 1, 2, 3, ..., 15 (and more if you want).
- Use a variety of numbers of threads. At least use 1, 2, and 4. You can use more if you'd like.
- If you are doing this from a script, note that: `int pad[NUMPAD];` is a legal C statement, even when NUMPAD is `#define`'d as 0 (at least in g++). It works like you think it should, i.e., it allocates zero array elements..
- Use OpenMP's timer calls to time the length of execution. As always, change that elapsed time into a sensible unit of performance.
- After you are done with all the Fix #1 combinations, then apply Fix #2 from the notes, i.e., accumulate the sum in a variable that is private to each thread.
- Plot, on the same set of axes, Performance vs. NUMPAD for 1 thread, 2 threads, etc.
- Plot, also on that same set of axes, horizontal lines showing what performance resulted from Fix #2. These can each just be single horizontal lines because the performance won't depend on NUMPAD.
- Your commentary write-up (turned in as a PDF file) should include:

1. Tell what machine you ran this on
2. Create a table with your results.
3. Draw a graph. The X axis will be NUMPAD, i.e., the amount of integers used to pad the structure. The Y axis will be the performance in whatever units you sensibly choose. There should be at least ***6 curves*** shown together on those axes:
   1-3: Using padding with 1, 2, and 4 threads. (Fix #1)
   4-6: Using a private variable with 1, 2, and 4 threads. (Fix #2)
4. What patterns are you seeing in the performance?
5. Why do you think it is behaving this way?

Warning!

Turn all compiler optimization flags off. We don't want the compiler to figure out we are doing busy-work, and optimize the entire loop away.