



F. Wermelinger
Office: Pierce Hall 211

Homework 1

Caches, access patterns, race conditions, OpenMP

Issued: January 31, 2023
Due: February 14, 2023 11:59pm

Note that problems 2, 3 and 4 involve very little knowledge of OpenMP. We will touch what is necessary for these problems in lecture 4.

Submission instructions: please see the homework submission section in the syllabus.^a

^a<https://harvard-iacs.github.io/2023-CS205/pages/syllabus.html#homework-submission>

Problem 1: Cache Efficiency (30 points)

3M decides to make Post-its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 1024-byte fully associative data cache with 64-byte blocks. You are given the following definitions in a C code:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[32][32];
int i, j;
```

Assume the following:

- `sizeof(int) = 4`.
- `square` begins at memory address 0.
- The cache is initially empty for each task below.

- The only memory accesses are to the entries of the array square. Variables i and j are stored in registers.

Hint: A cache line can store one block. A cache line additionally stores other information such as the valid bit and tag bits which are not of interest in this exercise.

a) **10 points**

Determine the cache performance of the following code:

```
for (i = 0; i < 32; ++i) {
    for (j = 0; j < 32; ++j) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

- What is the total number of writes?
- What is the total number of writes that hit in the cache?
- What is the cache hit rate?
- What is the cache hit rate if the cache was twice as large?

b) **10 points**

Determine the cache performance of the following code:

```
for (i = 0; i < 32; ++i) {
    for (j = 0; j < 32; ++j) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

- What is the total number of writes?
- What is the total number of writes that hit in the cache?
- What is the cache hit rate?
- What is the cache hit rate if the cache was twice as large?

c) **10 points**

Determine the cache performance of the following code:

```
for (i = 0; i < 32; ++i) {  
    for (j = 0; j < 32; ++j) {  
        square[i][j].y = 1;  
    }  
}
```

```
for (i = 0; i < 32; ++i) {  
    for (j = 0; j < 32; ++j) {  
        square[i][j].c = 0;  
        square[i][j].m = 0;  
        square[i][j].k = 0;  
    }  
}
```

- i) What is the total number of writes?
- ii) What is the total number of writes that hit in the cache?
- iii) What is the cache hit rate?
- iv) What is the cache hit rate if the cache was twice as large?

Problem 2: OpenMP Bug Hunting (20 points)

Assume you are given the following code and all necessary headers are included:

```
1  #define N 1000
2
3  extern struct data member[N]; // array of structures, defined elsewhere
4  extern int is_good(int i); // returns 1 if member[i] is "good", 0 otherwise
5
6  int good_members[N];
7  int pos = 0;
8
9  void find_good_members()
10 {
11     #pragma omp parallel for
12     for (int i = 0; i < N; ++i) {
13         if (is_good(i)) {
14             good_members[pos] = i;
15
16             #pragma omp atomic
17             pos++;
18         }
19     }
20 }
```

a) 10 points

Identify and explain any bugs in the given shared memory OpenMP code.

Hint: `omp parallel for` spans a team of threads that execute the for-loop in parallel.

Hint: `omp atomic` increments `pos` and ensures that only one thread can do the increment at a time.

b) 10 points

Propose a solution if you have identified any bugs. You may describe your approach in words or propose a solution with OpenMP. In both cases you must be clear about *how* you are going to isolate the problem.

Problem 3: Loop Parallelization with OpenMP (20 points)

State whether the following for-loops can be parallelized by prefixing the loop with:

```
#pragma omp parallel for
```

In case the answer is *no*, please state the reason. Assume there is no aliasing between different arrays.

Hint: `omp parallel for` spans a team of threads that execute the for-loop in parallel.

Hint: Look closely at the indices used to access data in the arrays.

a) 5 points

```
for (int i = 0; i < 20; ++i) {  
    result[i+20] = result[i] + B[i];  
}
```

b) 5 points

```
for (int i = 1; i < N-1; ++i) {  
    result[i] = result[i+1] + result[i-1];  
}
```

c) 5 points

Assume that the array `idx` has been filled with a random permutation of numbers between 0 and $N - 1$.

```
for (int i = 0; i < N; ++i) {  
    result[i] = B[idx[i]];  
}
```

d) 5 points

```
for (int i = 1; i < N; ++i) {  
    result1[i] = 2 * A[i];  
    result2[i] = result1[i-1] + B[i];  
}
```

Problem 4: Approximation of π with OpenMP (30 points)

The value of π can be computed by the infinite sum

$$\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}. \quad (1)$$

We approximate this sum by truncating it after n terms. A simple serial code may look like this:

```
double serial_pi(const int n)
{
    double fac = 1.0;
    double sum = 0.0;
    for (int k = 0; k < n; ++k) {
        sum += fac / (2 * k + 1);
        fac = -fac;
    }
    return 4.0 * sum;
}
```

Note that the convergence of this formula is slow and that there are better ways to approximate π . Since we are going to parallelize the loop, we are not too worried about some more iterations.

Implement a parallel OpenMP version in the function `parallel_pi(const int n)` inside the given C++ skeleton code `openmp_pi.cpp`. To compile your code with OpenMP support, invoke the compiler with the `-fopenmp` option:

```
$ g++ -fopenmp openmp_pi.cpp
```

and run the executable with

```
$ ./a.out
```

It is important that you compile your code with the `-fopenmp` flag to enable shared memory parallelism. If you omit it, your code will be sequential and *may* produce a correct result; while it is possible (due to reasons we talk about shortly) that your result is incorrect when the `-fopenmp` is added. Therefore, you must test your code *with* the flag to be eligible for full points. Make sure you clean out binary files like `a.out` before you submit your homework.

Hint: Be careful with global variables and race conditions.

Hint: See Section 17.3 in Parallel Programming for Science and Engineering¹ by V. Eijkhout

Work in directory: `./code/p4`

¹<https://web.corral.tacc.utexas.edu/CompEdu/pdf/pcse/EijkhoutParallelProgramming.pdf>