CS105 – Computer Systems Spring 2023

Week 7: Optimization

SOLUTION

March 6-8, 2023

1. **Optimization.** You have just joined a new startup that is trying to develop the world's fastest factorial routine. Starting with recursive factorial, they converted to the code to use iteration:

```
int fact(int n){
    int i;
    int result = 1;

    for (i=n; i > 0; i--){
        result = result * i;
    }

    return result;
}
```

By doing so, they have reduced the number of cycles per element (CPE) for the function from around 63 to around 4 (really!). Still, they would like to do better.

(a) One of the programmers heard about loop unrolling. He generated the following code:

```
int fact_u2(int n){
    int i;
    int result = 1;

    for (i = n; i > 0; i-=2){
        result = (result *i) * (i-1);
    }

    return result;
}
```

Is this a valid optimization that a compiler might perform? If so, justify why the two functions are equivalent. If not, state which values of n will return different values and show how to fix it.

No, optimization does not behave the same as original code. Will return 0 whenever n is odd.

```
int fact_u2(int n){
    int i;
```

```
    int result = 1;

    for (i = n; i > 1; i-=2){
        result = (result *i) * (i-1);
    }

    return result;
}
```

(b) You modify the line inside the loop to read: `result = result * (i * (i-1));` To everyone's astonishment, the measured performance now has a CPE of 2.5. How do you explain this improved performance?

The multiplication i * (i-1) can overlap with the multiplication by result from the previous iteration.

(c) Name two further changes might you make to try to further improve the performance of your factorial function.

unroll loop, separate accumulators

(d) Show how to modify the code to improve the performance using the techniques identified in Part1c.

```
int fact_u2(int n){
    int result = 1;
    int result2 = 1;
    int result3 = 1;
    int result4 = 1;

    for (int i = n; i > 3; i-=4){
        result = result * i;
        result2 = result2 * (i-1);
        result3 = result3 *(i-2);
        result4 = result4 * (i-3);
    }
    result *= result2 * result3 * result4;

    for(;i > 0; i--){
        result *= i;
    }

    return result;
}
```

2. **Optimization with Caches.**

After graduation, you start working for a company that wants 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. They ask you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32 byte blocks.

You are given the following definitions:

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

struct point_color square[16][16];
register int i, j;
```

Assume:

- `sizeof(int) = 4`
- `square` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

(a) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: `12.5` %

(b) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

Miss rate for writes to `square`: 25 %

(c) What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].y = 1;
    }
}
for (i=0; i<16; i++) {
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: 25 %

(d) Which implementation would you recommend they use?

I'd go with (a). Fewer cache misses than (b) or (c), so we would expect it to have better performance.