Problem set 4, Part 1, Theory

TDT4200, Fall 2016

Deadline: 12.10.2016 at 20.00 Contact course staff if you cannot meet the deadline.

Evaluation: Graded, counts 5% of final grade.

Delivery: Use It's Learning. Deliver exactly one file:

- *yourusername_ps4.pdf*, with answers to the theory questions

Cooperation: This problem set is to be done **INDIVIDUALLY, no cooperation of any kind is allowed**. Cooperation will be regarded as cheating on an exam, for details see `https://innsida.ntnu.no/wiki/-/wiki/English/Cheating+on+exams`

General notes: Code must compile and run on the course servers. Do not add third-party code or libraries. Make any reasonalbe assumptions neccessary, and state them.

Problem 1, Memory and Caching

a) For each of the three types of cache misses, describe it, and ways of avoiding such misses.

b) For each of the following code snippets, determine if the access pattern exhibits spatial locality, temporal locality, both, or neither, and explain why (`a`, `b` and `c` are in all cases apropriately sized `int` arrays):

I)

```
for(int i = 0; i < 10000; i++){
    a[i] = b[i] + c[i];
}
```

II)

```
for(int i = 0; i < 100; i++){
    for(int j = 0; j < 100; j++){
        a[j] = b[j] + c[j];
    }
}
```

III)

```
for(int i = 0; i < 100; i++){
    for(j = 0; j < 10; j++){
        a[j*1000] += b[j * 2000] + c[j * 3000];
    }
}
```

Problem 2, Branching

a) What is branch prediction? How can it improve performance?

b) Consider the following code:

```
for(int i = 0; i < n; i++){

    if(special(a[i]){
        fast(a[i]);
    }
    else{
        slow(a[i]);
    }
}
```

The code processes all the elements in an array. If the elements satisfy a special condition, the can be processed with the fast `fast()` funciton. If not, they must be processed with the slow `slow()` function.

The exection times of `slow()`, `fast()` and `special()` are s, f and p respectively. The delay casued by a mispredicted branch is b. Furthermore, $f + p < s$ and $f + p + b > s$, that is, using the fast function is only beneficial when it does not cause a branch missprediction delay. The fraction of elements satisfying the special condition is r, where $r < 0.5$. The branch predictor will allways predict the same branch as in the previous iteration, in the first iteration, it will predict the first branch.

At what value of r will it be beneficial to remove the branch, and use `slow()` for all the elements?

I) If all the special elements are placed at the start of `a`.

II) If the special elements are distributed evenly.

III) If the special elements are distributed randomly.

Show your work. Make any reasonable assumptions neccessary.

Problem 3, Vectorization

Consider the following for loop:

```
for(int i = 0; i < 1024; i++){
    a[i] = b[i] + c[i];
}
```

a) What are SIMD instructions (on x86 processors)?

b) How much faster would the loop above execute if if was vectorized using:

I) 128 bit vector instructions?

II) 512 bit vector instructions?

Show your work. Make any reasonable assumptions neccessary.

Problem 4, Optimization

The following code snippet calculates $\sin(x)$ using the Taylor series expansion

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots.$$

It uses an additional function that calculates $n!$ (the factorial of n, defined as $n! = 1 \cdot 2 \cdot \ldots \cdot n$).

```
 /* calculate n! */
double factorial(int n) {
    if(n<2)
        return 1.0;
    else
        return (double)n*factorial(n-1);
}

/* calculate sin(x) using Taylor series expansion */
double slow_sin(double x) {
    double r=0; int i;
    for(i=0;i<100;i++) {
        if(i%2==0) {
            r += pow(x, i*2+1) / factorial(i*2+1);
        }
        else {
            r -= pow(x, i*2+1) / factorial(i*2+1);
        }
    }
    return r;
}
```

Mention 4 different ways to speed up the above code. Don't answer with code only, but give explanations.

Problem 5, OpenMP schedules

Consider the following code:

```
for(int i = 0; i < 1024; i++){
    int arg = ...
    compute(arg);
}
```

Where the execution time of `compute()` is proportional to the value of its argument. We want to parallelize this loop using OpenMP, with one of the schedules static, dynamic or guided. Find an example of an expression for `arg` (i.e. complete line 2) so that the best schedule to use would be:

a) static.

b) dynamic.

c) guided.

Explain your answers.