Project 3: Performance Programming

Computer Architecture I ShanghaiTech University

Project 2.2 Project 3 Project 4

IMPORTANT INFO - PLEASE READ

The projects are part of your design project worth 2 credit points. As such they run in parallel to the actual course. So be aware that the due date for project and homework might be very close to each other! Start early and do not procrastinate.

Goal

In this project, we hope you can use all knowledge about computer architecture that your have learnt in this course to optimize a very simple, yet slow program.

Getting started

Make sure you read through the entire specification before starting the project.

You will be using gitlab to collaborate with your group partner. Autolab will use the files from gitlab. Make sure that you have access to gitlab. In the group CS110_Projects you should have access to your project 3 project. Also, in the group CS110, you should have access to the p3_framework.

Obtain your files

1. Clone your p3 repository from GitLab.
2. In the repository add a remote repo that contains the framework files:
   ```
   git remote add framework
   https://autolab.sist.shanghaitech.edu.cn/gitlab/cs110/p3_framework.git
   ```
3. Go and fetch the files:
   ```
   git fetch framework
   ```
4. Now merge those files with your master branch:
   ```
   git merge framework/master
   ```
5. The rest of the git commands work as usual.

Files

The framework contains the following files:

- `Makefile` - File which records all dependencies.
- `main.cpp` - The main program.
- `calc.cpp` - The algorithm you need to optimize.

If you wish to test on your own machine, please modify the `ARCH` variable in `Makefile`. You can search your CPU's model name to find out which microarchitecture it belongs to.

For example, suppose your CPU is i5-6300HQ, the specification page for it says it belongs to the Skylake microarchitecture, thus you should change `ARCH` to `skylake` in the `Makefile`.

For a list of supported microarchitectures in GCC, please refer to x86 Options. Note: some newer microarchitectures are not supported by old versions of GCC, you can either enable the corresponding SIMD instructions manually or upgrade GCC.

Problem Setup

Given two arrays a and b of length n which contains positive integers less than or equal to 8. We assume that the array index starts at zero, calculate the c array given the following formula:

$$c_k = \sum_{i=0}^{n-k-1} a_{i+k} b_i, \quad 0 \leq k < n$$

Input

The first line contains a positive integer n. There are two positive integers in each of the next n lines representing the elements in the two arrays, i.e., the $(i+2)$-th line contains a_i, b_i. For the given input, $n \leq 2^{25}$.

Sample Input

```
5
3 1
2 4
1 1
2 4
1 4
```

Output

n lines. The $(k+1)$-th line contains one positive number which represents c_k.

Sample Output

```
24
12
10
6
1
```

Optimization Techniques

The problem is very simple, and your job is to optimize the naive $O(n^2)$ algorithm.

Algorithm

In fact, there is a faster algorithm (see below) to solve this problem. We provide you with the faster algorithm so that you can get correct result on large inputs quickly. You are not allowed to optimize the naive algorithm though, as this is not the focus of Computer Architecture. You will receive no point if we find you do that.

Compiler

There are some optimization flags that can be turned on in GCC. However, we wish you to do the optimization on your own, instead of relying on the compiler to do it for you. You will receive 0 points if you try to turn on any other optimization flags except for `-O2` specified in `Makefile`.

Multithreading

The first and the easiest approach is to use multithreading to optimize this algorithm. There are hardly any data dependencies so you do not need to consider much about synchronization. However, the computation workload for each c_k is imbalanced, you should think about how to balance it.

SIMD instructions

This algorithm is also a good candidate for SIMD instructions.

Loop unrolling

Loop unrolling works very well in combination with SIMD instructions for this algorithm, and you should think about it (Hint: there are two kinds of load: one require alignment and one do not).

Cache Blocking

For each computation of c_k, we just stride across the data. If n is very large, this wastes cache lines and may slow down your program. One way to solve this is to reuse the cache line for next loop (e.g. c_{k+1}).

Grading Policy

Your grade will be divided into two parts:

1. We will first run your code on small test cases on Autolab. If you program produces the correct result, you receive 20% points.
2. After the due, we will test your code on large test cases. Your grade on this part depends on the execution time of your code. All groups run slower than 1 minute but faster than 2 minutes get 40% points and the fastest 15 groups receive 80% points. The execution time of 15th group is 80% line and 1 minute is 40% line. The grades for the remaining groups are linearly distributed according to their execution time.
3. For your reference, if you only add a single line of `#pragma omp parallel for`, the running time will be between 1 minute and 2 minutes.

4. Of course, your code should not crash on either stage, or you will receive no point in that stage.

Server Configuration

Inspur NF5280M5

- CPU: 2x Intel(R) Xeon(R) Gold 6240M (details)
- Memory: 192 GB
- Operating System: Ubuntu 18.04 with GCC 9.3.0.

Submission

When your project is done, please submit all the files including the framework to your remote GitLab repo by running the following commands.

```
$ git commit -a
$ git push origin master:master
```

Autolab will discard all other files except for `calc.cpp`, so please put all your code in that file.

How to Autolab

Similar to previous projects, upload your `autolab.txt` to Autolab to submit your project.

Submission Time Announcement

The last time of your submission to the git repo will count towards your submission time - also with respect to slip days. So do not commit to this git after the due date, unless you want to use slip days or are OK with getting fewer points.

Collaborative Coding and Frequent Pushing

You have to work at this project as a team. We invite you to use all of the features of gitlab for your project, for example branches, issues, wiki, milestones, etc.

We require you to push very frequently to gitlab. In your commits we want to see how the code evolved. We do NOT want to see the working code suddenly appear - this will make us suspicious.

We also require that all group members do substantial contributions to the project. This also means that one group member should not finish the project all by himself, but distribute the work among all group members!

At the end of Project 3 we will interview all group members and discuss their contributions, to see if we need to modify the score for certain group members.

Appendix

Actually, there is a faster algorithm to solve this problem.

We can do a simple transform on a and c. Let $a'_k = a_{n-k-1}$, $c'_k = c_{n-k-1}$, the equation becomes

$$c'_k = \sum_{i=0}^{k} a'_i b_{k-i}, \quad 0 \le k < n$$

This is the first half of polynomial multiplication, and we can use Fast Fourier Transform to solve it in $O(n \log n)$ time.

TAs: *Yanjie Song, Anqi Pang*
Last Modified: May 17th, 2020