Lab1

CS2305 Computer System Architecture

February 29, 2024

0 Requirements

- 最终需要提交一个 zip 压缩文件，命名方式为 “Lab1_ 学号 _ 姓名.zip”
- 压缩文件内包含一个 pdf 文档，一个 Lab1_1.c 文件，一个 Lab1_2.c 文件
- pdf 文档命名方式为 “Lab1_ 学号 _ 姓名.pdf”，需包含除代码以外的全部内容，pdf 文件首页为封面，注明 “Lab1”，学号，姓名，专业等个人信息，第 2 页开始答题，无需抄题，但需注明题号
- Lab1_1.c 文件对应 Programming 部分第 (3) 题，Lab1_2.c 文件对应 Programming 部分第 (5) 题。Programming 部分第 (1) 题无需提交代码文件
- 对于编程题，需要给出结果的，请给出代码在终端或其他工具中编译、运行的截图
- 对于思考题，体现你的探索和思考，言之有理即可，不得抄袭，表述尽可能精简直接，允许中文作答，字数限制为英文词数限制的 1.6 倍

1 Programming

1. Run the following C code blocks respectively. Give the printed results and time ticks(give the screenshot).

   (a) Fibonacci mod 10007

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
int main(int argc, char* argv[])
{
    clock_t start, end;
    start = clock();
    int mod = 10007;
    long long n = 800000001;
    long long first, second, result;
    first = 0;
    second = 1;
    result = 0;
    int i;
    for (i = 2; i <= n; ++i) {
        result = (first + second) % mod;
        first = second;
        second = result;
    }
    end = clock();
    printf("result = %d\n", result);
    printf("time ticks are %d\n", (int)((end-start)));
    return 0;
}
```

(b) $\sum(n^3+n^2) mod 10007$

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
int main(int argc, char* argv[])
{
    clock_t start, end;
    start = clock();
    int mod = 10007;
    long long n = 800000000;
    long long square, cube, sum;
    square = 0;
    cube = 0;
    sum = 0;
    int i;
    for (i = 1; i <= n; ++i) {
        square = ((i % mod) * (i % mod)) % mod;
        cube = ((square % mod) * (i % mod)) % mod;
        sum = (sum + square + cube) % mod;
    }
    end = clock();
    printf("sum = %d\n", sum);
    printf("time ticks are %d\n", (int)((end-start)));
    return 0;
}
```

2. Pick one of the code blocks in question (1) which is more appropriate for multithreading. State that why you think the other choice is not appropriate for multithreading.

3. Write your own C code to implement parallelism with OpenMP in 4 threads based on your chosen code block in question (2). Present the result and time ticks(give the screenshot). What is the speedup? Have you got a significant performance improvement? Try to explain the time ticks and the performance improvement! The answer can be flexible so feel free to share your thoughts as long as it makes sense, no more than 200 words.

4. Now we are using schedule(static,2) in your written code in question (3). Assume that the four threads are t0, t1, t2 and t3. In which thread would i=35 be calculated?

5. Rewrite the unpicked code and unroll the for loop using i+=2 instead of ++i in question (1). Present the result and time ticks(give the screenshot).

2 Supplementary Materials

2.1 安装 VS Code 并配置 C 语言编程环境

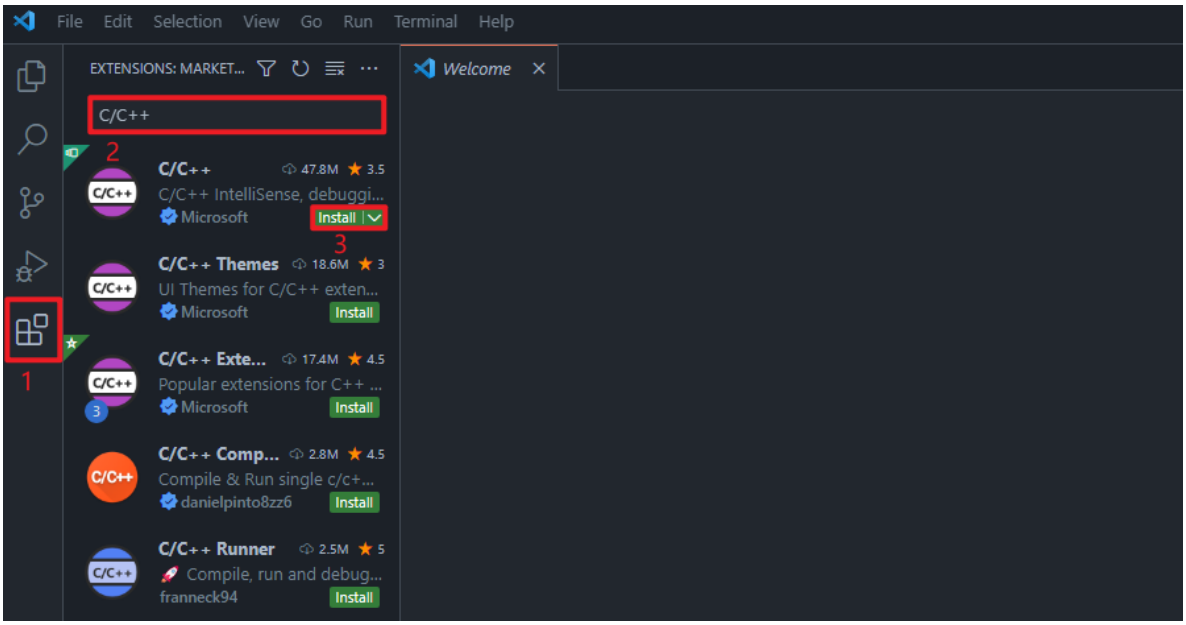
1. 安装并配置编译器。在 Windows 上，可以安装 MinGW；在 MacOS 上，可以安装 GCC。对于 macOS，使用以下命令安装 gcc：

```
$ 安装 Homebrew，若已安装可忽略
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
$ 安装 gcc
brew install gcc
$ 安装 OpenMP
brew install libomp
```

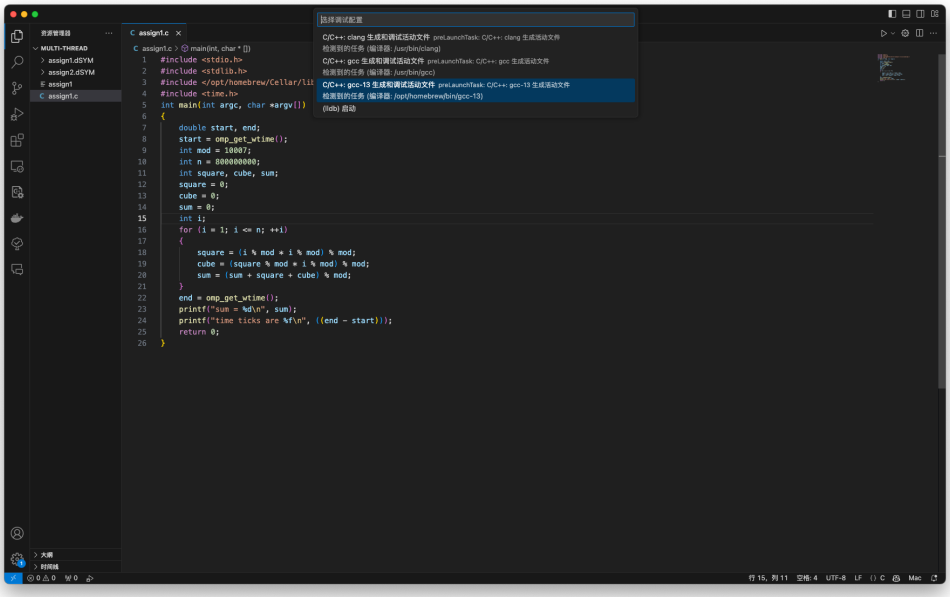
   对于 Windows，可以在https://www.mingw-w64.org/downloads/下载 MinGW 安装包，将 MinGW 安装到指定位置，并将该路径添加到系统的环境变量中。具体操作可以参考这篇博客。

2. 在官网 https://code.visualstudio.com 下载 VSCode 安装包，依照提示安装到本地。

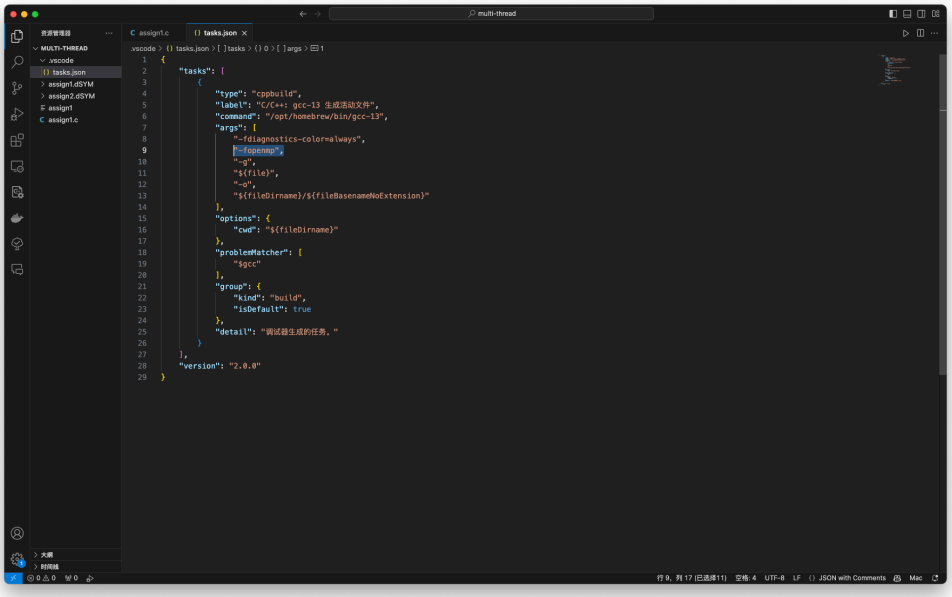
3. 在插件管理中安装 C/C+ 插件。



4. 新建一个任意以.c 作为后缀的文件，在 VSCode 中打开。点击右上角的三角标识，为当前文件夹配置编译信息。在弹出的窗口中选择之前安装的编译器。注意应该选择 GCC 而不是 Clang（MacOS 自带的 Clang 不包含 OpenMP 的内容，无法完成本次作业）。



5. 点击后，文件夹内会新建一个名为.vscode 的文件夹，打开文件夹中的 tasks.json，在 args 中添加”-fopenmp”，从而在编译时能够调用 OpenMP 中的函数。配置完成后，即可在 C 语言中包含 omp.h，调用 OpenMP 中的函数与宏进行多线程计算。



2.2 OpenMP

OpenMP（Open Multi-Processing）是一种并行计算的编程模型，用于在共享内存系统中实现多线程并行化。它提供了一组指令和库函数，使得开发者可以通过简单地插入指令来将串行代码转换为并行代码。使用 OpenMP，开发者可以通过在适当的位置插入 #pragma omp 指令来标识出需要进行并行化处理的代码段。这些指令告诉编译器如何将代码分解成多个任务，并将这些任务分配给不同的线程执行。同时，OpenMP 还提供了一系列库函数，用于管理线程、同步数据访问以及性能优化等操作你可以通过以下示例检查之前的环境配置是否正确：

```
#include <stdio.h>
#include <omp.h>

int main()
{
    int sum = 0;
    #paragma omp parallel for num_threads(4)
    for(int i = 0;i < 100;i++)
        sum += i;
    printf("%d", sum);
    return 0;
}
```

上述代码实现了用 4 个线程并行地计算 1-99 的和，但存在竞争、写冲突的问题，可能无法输出正确的答案。为了完成本次作业给出的任务，你可能需要查阅 OpenMP 的基本使用方法、多线程计算的基本概念等相关内容。