并行与分布式作业

“数据级并行-线程级并行”
第一次作业

姓名：谷正阳
班级：行政一班
学号：18308045

一、 问题描述

我们在第一次课程中已经讲到，早期单节点计算系统并行的粒度分为：Bit 级并行，指令级并行和线程级并行。现代处理器如 Intel、ARM、AMD、Power 以及国产 CPU 如华为鲲鹏等，均包含了并行指令集合，❶请调查这些处理器中的并行指令集，并选择其中一种进行编程练习，计算两个各包含 10^6 个整数的向量之和。

此外，❷现代操作系统为了发挥多核的优势，支持多线程并行编程模型，请将问题❶用多线程的方式实现，线程实现的语言不限，可以是 Java，也可以是 C/C++。

二、 解决方案

针对问题❶，我选用 AVX 指令集。AVX 指令集是 SIMD（Single Instruction, Multiple Data）指令集的一种。SIMD 指令级通过把 64 位寄存器拆成多个低位寄存器，从而能同时完成多个操作，提升计算效率。AVX 指令集是 SIMD 指令集中相对新的一种，在 C++中通过<immintrin.h>库实现。

针对问题❷，我选用 C++实现。多线程通过提供 CPU 利用率来提高效率。数据库访问、磁盘 IO 等操作的速度比 CPU 执行代码速度慢很多，单线程环境下，这些操作会阻塞程序执行，导致 CPU 空转，因此对于会产生这些阻塞的程序来说，使用多线程可以避免在等待期间 CPU 的空转，提高 CPU 利用率。多线程是指从软件或者硬件上实现多个线程并发执行的技术，可以。在 C++

中通过 thread 库实现。

三、 实验结果

```
Sequential:
verify_res:     1
time:   2600746.500000

AVX:
verify_res:     1
time:   826086.187500
speedup:        3.148275

4threads:
verify_res:     1
time:   1205387.250000
speedup:        2.157603

5threads:
verify_res:     1
time:   1274598.375000
speedup:        2.040444

8threads:
verify_res:     1
time:   1414182.000000
speedup:        1.839046

4threads with AVX:
verify_res:     1
time:   1347905.500000
speedup:        1.929472

5threads with AVX:
verify_res:     1
time:   1222862.375000
speedup:        2.126770

8threads with AVX:
verify_res:     1
time:   1474038.250000
speedup:        1.764368
```

步骤：

1. 编写函数：序列求和，AVX 求和
2. 编写函数：4 线程求和，5 线程求和，8 线程求和，每个线程可以选择序列求和或 AVX 求和
3. 分别对两个数组中的 10^6 个数分别初始化为 0-99999
4. 每个函数运行 1000 次，记录一次的运行结果和总运行时间
5. 检查运行结果每个数是否是原来的两倍，若是则打印 1，否则打印 0
6. 打印运行时间，计算并加速比

结果分析：

1. 可以看到 AVX 和多线程还有两者结合都有一定的加速
2. 加速比和预期值有偏差：预期分别是 8，4，5，8，32，40，64
3. 多线程往往线程数越多加速比越差

四、遇到的问题及解决方法

1. AVX 加速比不尽如人意，可能数据移动时间是无法并行的。去掉数据移动：

```
Sequential_without_move:
time:   2059806.000000
```

根据 Amdahl's law

理论 speedup = 2600746 / (2059806 / 8 + (2600746.5 - 2059806))

$$speed = \frac{2600746}{\frac{2059806}{8} + (2600746.5 - 2059806)}$$

```
PS C:\Users\guzy0\Desktop\18308045-谷正阳-并行分布式计算作业1-v1> python calculate.py
3.257381096639754
```

结果接近 3.148275

2. 多线程加速比不尽如人意，经过上网搜索，可能线程没有分配在不同的核心上，会出现上下文切换现象。特此，进行了以下实验：计算数组内所有数字的和，使用第一次前一半和后一半并发求和存入前一半，后面对前一半进行如上求和……如果是理想情况下的并行将得到$\frac{\log_2 n}{n}$的加速比，然而在 n 为 1000 时得到加速比如下：

```
Sequential:
res:    49995000
time:   4671749040117907456

AVX:
res:    49995000
time:   4664638498421080064
speedup:        3.041667

Thread0:
res:    49995000
time:   4737304910515666944
speedup:        0.000043

Thread1:
res:    49995000
time:   4737336287399247872
speedup:        0.000043
```

Thread0，Thread1 是递归和循环两种实现方式，结果发现多线程耗时远多于其他方法。猜测主要是多线程占用同一个核心的结果。