Assignment 1 for CMU parallel Computer Architecture and Programming class

环境准备

linux操作系统

Intel SPMD Program Compiler（ispc）安装后添加到系统Path

make命令编译

Problem 1：Pthread

1问题的分析和策略

使用Pthread命令进行mandelbrot分形图像生成的加速

对图片进行分割，不同的部分交给不同的线程进行生成从而达到加速的效果

难点在于如何分配任务，采取的策略是在图像的高度方向上进行任务的划分（宽度方向上一样），平均分配任务，坑点在于线程的个数不能整除图像的高度，此时将多余出来的像素交给最后一个线程处理。

具体实现在mandelbrot.cpp

2 思考

1.线程个数增加时，加速效果也会变好，但在分别使用2、4、8、16、32线程时，加速效果并不是线性增加的，分别为1.88x，2.40x，3.47x，5.90x，8.01x。为什么?

以我有限的知识来看，问题可能出在

（1）生成最终图像的过程中，线程访问图片进行修改的时候会产生写者问题，所以线程之间要避免冲突不能同时写入文件，这就造成了加速效果的减弱。

（2）多线程并不意味着真正并行，线程分片实际上还是类似顺序执行。

（3）pthread_join函数会让完成的线程等待未完成的线程。

3 其他

这个问题另一个坑点是，所有的线程要基于同一个标准进行任务的执行，在(x,y)点的计算要基于全局原点的相对位置进行。这启发我在进行任务的并行解决的时候，为了达到跟顺序执行的效果相同，要有一个统一的参考标准。

Problem 2 向量化

1 问题的分析与策略

这个任务要求使用向量变量对求绝对值、求高次幂、序列求和三个任务进行加速。主要的难点在于将任意个数的数字进行向量化的计算，任务使用了自定义的向量生成函数和运算函数构建了一套类sse的向量运算体系。

在阅读了定义的所有函数后，发现任务使用“掩码”的方式形成控制流，使用8位的掩码控制对8个数组成的向量的函数的执行。

问题的坑点在于处理任务个数不能被向量宽度（Width）整除的情况,会造成代码处理了超出任务数量的任务，这种情况使用掩码在最后将要越界的时候进行控制要处理的位即可,具体实现位于functions.cpp

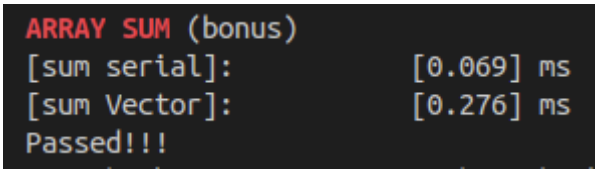
2 思考

（1）在使用不同的向量宽度运行的时候，任务会统计当前向量宽度情况下使用的向量命令个数、处理的向量对应的标量个数（比如一个8维向量其实包含了8个标量，16维依次类推）、实际处理的标量个数（一个向量里不可能所有的数都进行处理），并计算向量的利用率，即实际处理与总标量个数的比值。宽度分别取2、4、8、16、32，使用的向量个数在不断减少，基本符合线性减少；处理的实际标量和总包含的标量个数变化不大，利用率随着向量宽度的增大在降低，但变化其实不大，从91.97%降到91.63%，可以看出向量化的算法确实减少了执行的命令的数量，利用率其实没有太大的变化，这也比较符合实际，在数据确定的情况下，真正需要执行命令的数据个数是不会变的。

（2）从加速效果看，32维时加速效果最好，只用了线性顺序处理的算法一半的时间，但是在使用4维时加速效果已经很不明显，2维时甚至要花比原始算法更长的时间。初步认为维数少的时候，因为要执行的命令更多，将任务转化为并行执行付出的开销已经大于了加速效果节省的开销。

3 其他

在前两个任务中，向量计算的加速效果都不错，但在第三个任务中，即计算数列的和的时候，不管如何设定向量的宽度，计算消耗的时间都要比线性计算要多。



但从时间复杂度计算来看，线性计算复杂度为$O(N)$，向量计算的复杂度为$O(N/M+\log_2 W)$，M为向量的宽度，将最后的结果向量的各个数字相加。

这是一个问题

Problem 3 ISPC

1 问题的分析

part 1：编译并运行mandelbrot.ispc

part 2：运行./mandelbrot_ispc --tasks

```
[mandelbrot serial]:            [236.747] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [45.863] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [6.562] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (5.16x speedup from ISPC)
                                (36.08x speedup from task ISPC)
```

Part1使用了ispc对Problem 1进行SIMD加速，Part2使用了tasks加速，实际上是利用了多核加速。

问题的大坑在于，分配任务时，若采用以前的策略，由于给最后的task多分配了一些任务，其他task会等待完成，实际上加速效果会与ispc一样，所以要多launch一个task专门处理不能整除时的情况，然后又会出现一个问题，实际上空出来的任务个数可能会大于一个task可以处理的量，所以要在开始计算差值

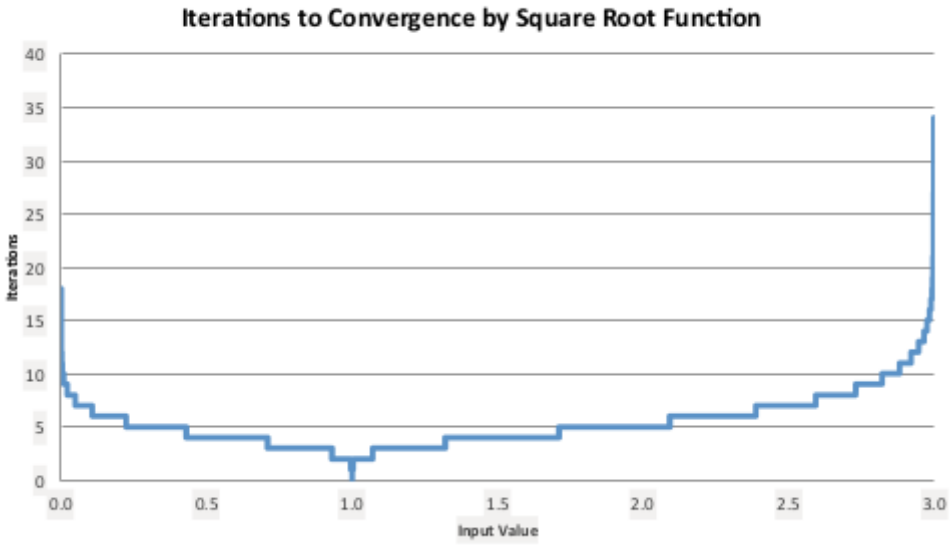
$$\frac{(\frac{\text{Height}}{\text{countThread}} - \lfloor \frac{\text{Height}}{\text{countThread}} \rfloor) \times \text{countThread}}{\lfloor \frac{\text{Height}}{\text{countThread}} \rfloor} \quad (1)$$

有趣的是，将任务分为750份，加速比也只能到达50，相同效果在任务分配为75时就差不多达到了

Problem 4 迭代平方根

1 问题分析

使用ispc加速牛顿迭代法求平方根，这个例子主要是体会初始值的设定对加速效果的影响



初始化数值靠近1时，因为需要迭代的次数较少，所以加速效果不明显，接近3时，越靠近3，需要的迭代次数接近无穷大，此时加速效果十分明显。

Problem 5 saxpy

1 问题分析

任务要求给定标量a，向量x和向量y，计算 $a\vec{x}+\vec{y}$

运行saxpy

```
[saxpy serial]:         [10.339] ms     [28.825] GB/s   [1.934] GFLOPS
[saxpy streaming]:      [10.539] ms     [28.279] GB/s   [1.898] GFLOPS
[saxpy ispc]:           [10.618] ms     [28.066] GB/s   [1.884] GFLOPS
[saxpy task ispc]:      [11.555] ms     [25.791] GB/s   [1.731] GFLOPS
                                (0.98x speedup from streaming)
                                (0.97x speedup from ISPC)
                                (0.89x speedup from task ISPC)
```

尴尬的是，所有的加速方法都不如线性计算的结果

2 思考

1.在主程序中，使用$4\times \mathrm{N}\times \mathrm{sizeof(float)}$作为总的数据吞吐量，为什么是正确的?

我的回答：虽然程序读入向量x与y并将结果存入result只使用了$3\times \mathrm{N}\times \mathrm{sizeof(float)}$的数据量，但是从汇编语言的角度想一想，中间结果不可能是直接存入result，肯定要先使用$1\times \mathrm{N}\times \mathrm{sizeof(float)}$的空间来暂时保存

2.你能否利用Intel intrinsics把空间需求降$3\times \mathrm{N}\times \mathrm{sizeof(float)}$？并实现加速?

根据代码注释里的一点点提示，看过Intel官方文档后，勉强用要求的命令实现了，但没有加速效果，也没有明白是否减少了空间要求。具体实现在saxpyStreaming.cpp

总结

总体来说，任务的基本要求不算难，但是思考部分以及额外的加分部分难度很大，需要大量的调试，问题里设置的坑也特别多，甚至3、4、5问题还需要改makefile才能编译通过……勉强算是完成了5个问题，人已经晕了……