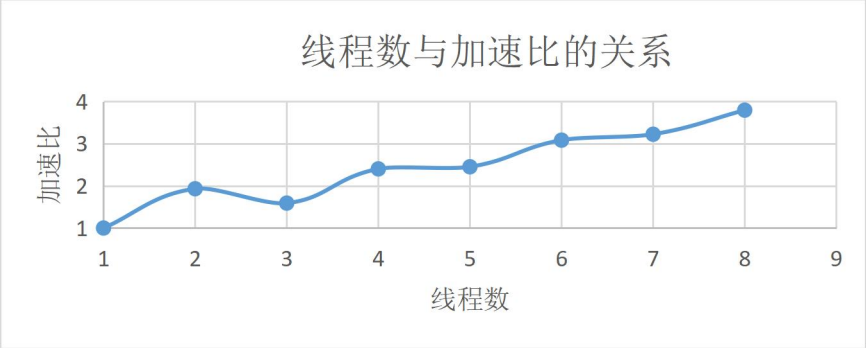
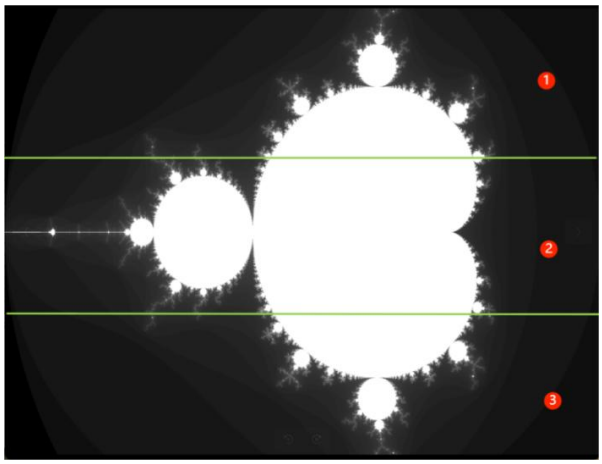
环境：
物理机：i5-11400H，6 核 12 线程，内存 16GB，支持超线程
虚拟机：Ubuntu 20.04 ，1cpu 8 内核（效果相当于 4 核 8 线程）

Program 1: Parallel Fractal Generation Using Threads

1.分区计算，线程 0 算上半，线程 1 算下半，2 线程加速比 1.96x。
2.加速比不线性，且线程数为 3 时反而下降，猜测是因为分块进行时任务分配到每个线程很不均匀导致的。仔细观察图片，中间部分计算量明显偏高。



因此 1 和 3 线程计算完后要等待 2 线程完成，耗时更长。



3.检测每个线程的时间，发现有一个线程确实耗时明显更长，与之前的猜想符合。

```
[mandelbrot serial]:            [395.586] ms
Wrote image file mandelbrot-serial.ppm
Hello world from thread 0, time:0.083890
Hello world from thread 2, time:0.086222
Hello world from thread 1, time:0.249329
Hello world from thread 2, time:0.084643
Hello world from thread 0, time:0.086020
Hello world from thread 1, time:0.244310
Hello world from thread 0, time:0.083390
Hello world from thread 2, time:0.085836
Hello world from thread 1, time:0.249063
Hello world from thread 0, time:0.082430
Hello world from thread 2, time:0.084667
Hello world from thread 1, time:0.245124
Hello world from thread 2, time:0.081798
Hello world from thread 0, time:0.087061
Hello world from thread 1, time:0.246386
[mandelbrot thread]:            [244.502] ms
Wrote image file mandelbrot-thread.ppm
                                (1.62x speedup from 3 threads)
```

4.改善的映射策略：交替并行。
例如有 3 线程时，线程 1 算 0、3、6…　线程 2 算 1、4、7…　线程 3 算 2、5、8…
因为尺寸较大，这样交替划分较为均匀，不会出现负载不均衡的情况。最后 8 线程时加速比 6.73x，明显比按照块划分更好。

5.没有提升，已经达到最大线程数 8，多余线程会进入等待，16 线程加速比 6.06x。

Program 2: Vectorizing Code Using SIMD Intrinsics

1.有关实现见代码文件“..\prog2_vecintrin\main.cpp”

2.利用率如下表所示：

vector width	vector utilization
2	79.8%
4	72.1%
8	68.1%
16	66.3%

当向量长度变大时，利用率减小，因为计算时同一组中的值要等待最慢的那一个算完才能保存结果。因此每组都是取最慢的那个数计算的时长，长度越大，越可能变慢。

3.思路：每次读入一个向量，设数组长 N，向量长度 n，对读入的值连续使用$\sqrt{n}$次 hadd 函数和 interleave 函数，最后取第一个值作为本向量中所有值的和 temp。循环 N/n 次，每次累加 temp 即可。

Program 3: Parallel Fractal Generation Using ISPC

Part1：
1.实际加速比为 4.7x，虚拟机系统为 4 核 8 线程，理论上最高为 8 倍。解释：图片不同部分的计算量是不均匀的，程序 1 中使用多线程因此需要负载均衡才能让每个线程差不多同时算完。而 SIMD 是单指令多数据，如果任务分配过于均衡会导致一部分 lane 先算完，另一部分等待。最好能够将相同的指令放在一起，一组中全是简单的指令，另一组全是复杂的指令，也就是不均衡分配才能充分利用。

```
mario@mario-virtual-machine:~/桌面/asst1/prog3_mandelbrot_ispc$ ./mandelbrot_isp
c
[mandelbrot serial]:            [200.202] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [42.600] ms
Wrote image file mandelbrot-ispc.ppm
                                (4.70x speedup from ISPC)
```

Part2：

1.加速比 9.37x，相比不把计算划分为任务的方式（4.7x），快了一倍。

```
mario@mario-virtual-machine:~/桌面/asst1/prog3_mandelbrot_ispc$ ./mandelbrot_isp
c --tasks
[mandelbrot serial]:            [197.840] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [41.766] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [21.106] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.74x speedup from ISPC)
                                (9.37x speedup from task ISPC)
```

2.改变 mandelbrot_ispc_withtasks()中任务的划分，即改变 tasks 数。

Tasks 数	加速比
2	9.37x
4	12.03x
8	18.07x
16	28.38x
20	20.03x
32	23.91x

tasks 数为 16 时加速比到达最大值 28.38x，可能因为物理机内核之间数据通信的一些限制没有到达 32x。虚拟机为 4 核 8 线程（虚拟机设置 8 内核），假设每个内核两个 simd 单元（因为查不到 i5 11400H 芯片每个核心的 simd 数量），所以每个内核超过两个 task 就不能同时数据并行了。因此 16 个 task 正好可以完全并行，而 32 个 task 就需要等待排队执行了。尝试更多的 task，都没有提升，反而可能因为任务启动开销导致更慢了。

3.ISPC 任务利用 SIMD 实现数据并行，流水线模式，将类似指令放在一起可以更好的效果，因此不需要负载均衡。多线程是创建多任务利用 CPU 连续执行产生宏观上的并行，每个线程分配任务量一致更好。当启动 10000 个 ISPC 任务时，每个 task 都尽可能分配 ALU，task 内部同时计算，task 之间类似线程。当启动 10000 个线程时，这些线程抢夺 CPU 执行权，充分利用 CPU，宏观上是并行的。

Program 4: Iterative sqrt

1.单核时加速比 4.14x，多核加速比 19.90x，由于 SIMD 并行产生的加速比即为单核加速比（4.14x），多核加速比为相比单核进一步提升加速比（加速 4 倍）。

```
mario@mario-virtual-machine:~/桌面/asst1/prog4_sqrt$ ./sqrt
[sqrt serial]:          [693.785] ms
[sqrt ispc]:            [167.699] ms
[sqrt task ispc]:       [34.861] ms
                                (4.14x speedup from ISPC)
                                (19.90x speedup from task ISPC)
```

2.当数组中的值全为 2.999999 时，参考收敛迭代次数图可知需要的次数最多，此时 ISPC 相对加速比最大，理论上 8x，实际单核 6.67x。这种构造提升了 SIMD 的加速比，因为 SIMD 是将多组数据同时计算，会取决于最慢的一条 lane，数组相同时，所有 ALU 同时完成计算。多核加速比也提高了，划分任务并行执行，约为单核 8 倍，与理论值接近。
相对最快：

```
mario@mario-virtual-machine:~/桌面/asst1/prog4_sqrt$ ./sqrt
[sqrt serial]:          [3961.665] ms
[sqrt ispc]:            [594.174] ms
[sqrt task ispc]:       [85.671] ms
                                (6.67x speedup from ISPC)
                                (46.24x speedup from task ISPC)
```

3.最容易想到的是当数组中的值全为 1 时，顺序执行一次即可成功，ispc 由于其它开销，加速比很低。但是仔细思考发现 SIMD 宽度为 8，构造前 7 位为 1，第 8 位为 2.999999 的数组，这样 ISPC 计算时会被第 8 个 lane 拖累导致加速比最低。如图所示，单核加速比 0.84x，由于 ISPC 有其它开销，甚至不如顺序执行。

```
mario@mario-virtual-machine:~/桌面/asst1/prog4_sqrt$ ./sqrt
[sqrt serial]:          [505.876] ms
[sqrt ispc]:            [603.369] ms
[sqrt task ispc]:       [84.615] ms
                                (0.84x speedup from ISPC)
                                (5.98x speedup from task ISPC)
```

Program 5: BLAS saxpy

1.划分任务为 64task 时加速比 1.22x

```
mario@mario-virtual-machine:~/桌面/asst1/prog5_saxpy$ ./saxpy
[saxpy ispc]:           [11.510] ms     [25.892] GB/s   [3.475] GFLOPS
[saxpy task ispc]:      [9.471] ms      [31.466] GB/s   [4.223] GFLOPS
                                (1.22x speedup from use of tasks)
```

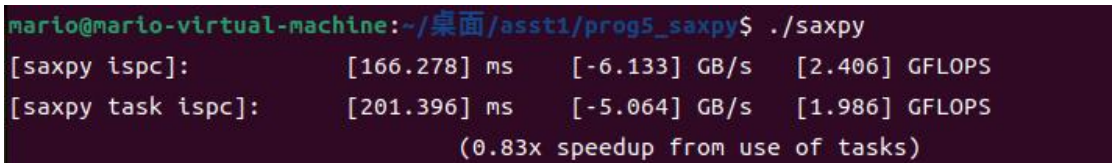
改变 tasks 数为 4，发现 64task 比 4task 提升很小。

```
mario@mario-virtual-machine:~/桌面/asst1/prog5_saxpy$ ./saxpy
[saxpy ispc]:           [11.319] ms     [26.329] GB/s   [3.534] GFLOPS
[saxpy task ispc]:      [10.013] ms     [29.764] GB/s   [3.995] GFLOPS
                                (1.13x speedup from use of tasks)
```

改为 128task，加速比 1.21x，没有提升。
因此认为不会实质性改进了，主要限制是带宽。
测试磁盘读写速度，顺序读速度 4.6GB/s，顺序写速度 2.3GB/s。

```
> Disk  Random 16.0 Read                    923.46 MB/s          8.7
> Disk  Sequential 64.0 Read                4555.71 MB/s         9.5
> Disk  Sequential 64.0 Write               2340.87 MB/s         9.1
```

将数据大小增加 10 倍，发现确实是受内存读写限制。



2.读入 X 和 Y 数据时，每次 cache 载入发生内存 IO 一次，写数据时，需要先更新 cache 行再写入内存，IO 两次，总共 4 次 IO 操作，因此内存带宽计算式*4 是正确的。