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）

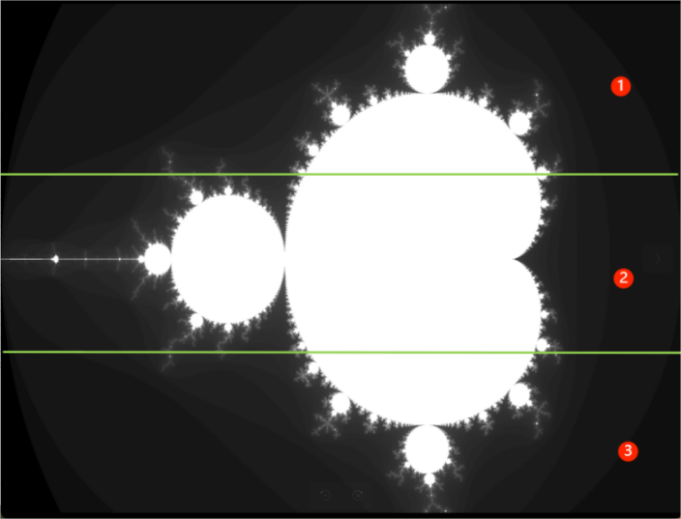
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Program 1: Parallel Fractal Generation Using Threads

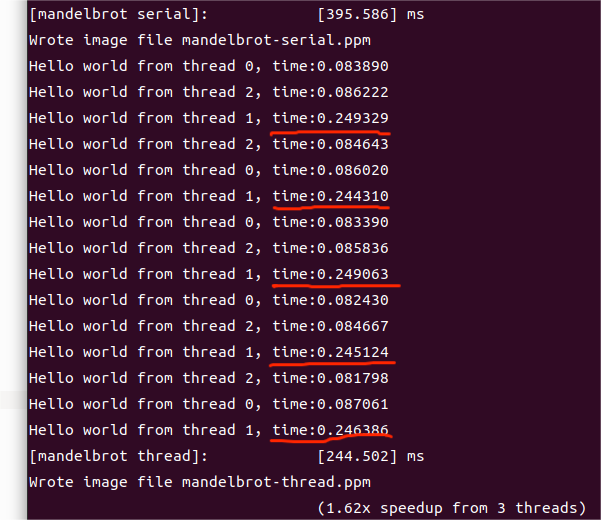
1.分区计算，线程0算上半，线程1算下半，2线程加速比1.96x。

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。



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% |

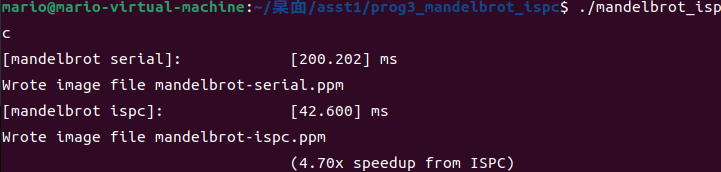
当向量长度变大时，利用率减小，因为计算时同一组中的值要等待最慢的那一个算完才能保存结果。因此每组都是取最慢的那个数计算的时长，长度越大，越可能变慢。

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

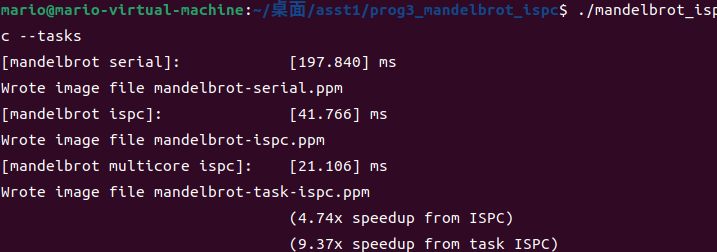
Part1：

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。



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。



2.改变mandelbrot\_ispc\_withtasks()中任务的划分，即改变tasks数。

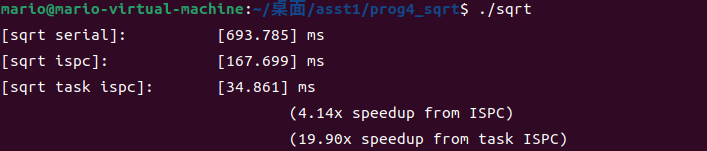
|  |  |
| --- | --- |
| Tasks数 | 加速比 |
| 2 | 9.37x |
| 4 | 12.03x |
| 8 | 18.07x |
| 16 | 28.38x |
| 20 | 20.03x |
| 32 | 23.91x |

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。

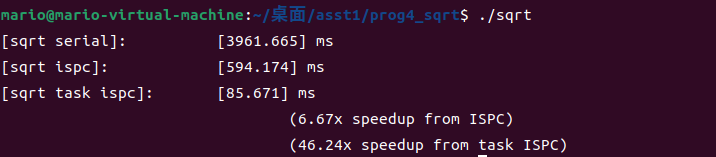
### Program 4: Iterative sqrt

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）。

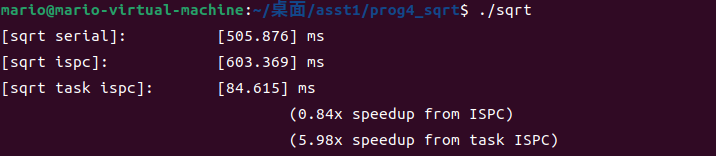


2.当数组中的值全为2.999999时，参考收敛迭代次数图可知需要的次数最多，此时ISPC相对加速比最大，理论上8x，实际单核6.67x。这种构造提升了SIMD的加速比，因为SIMD是将多组数据同时计算，会取决于最慢的一条lane，数组相同时，所有ALU同时完成计算。多核加速比也提高了，划分任务并行执行，约为单核8倍，与理论值接近。

相对最快：

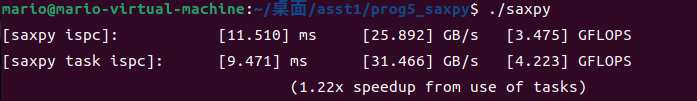


3.最容易想到的是当数组中的值全为1时，顺序执行一次即可成功，ispc由于其它开销，加速比很低。但是仔细思考发现SIMD宽度为8，构造前7位为1，第8位为2.999999的数组，这样ISPC计算时会被第8个lane拖累导致加速比最低。如图所示，单核加速比0.84x，由于ISPC有其它开销，甚至不如顺序执行。

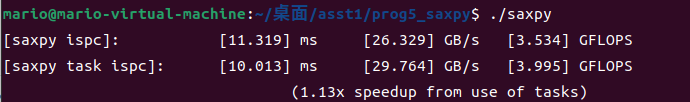


### Program 5: BLAS saxpy

1.划分任务为64task时加速比1.22x



改变tasks数为4，发现64task比4task提升很小。



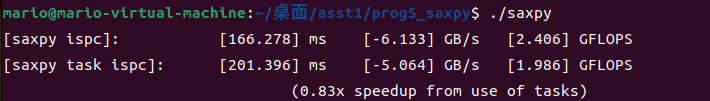
改为128task，加速比1.21x，没有提升。

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。

5io

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。