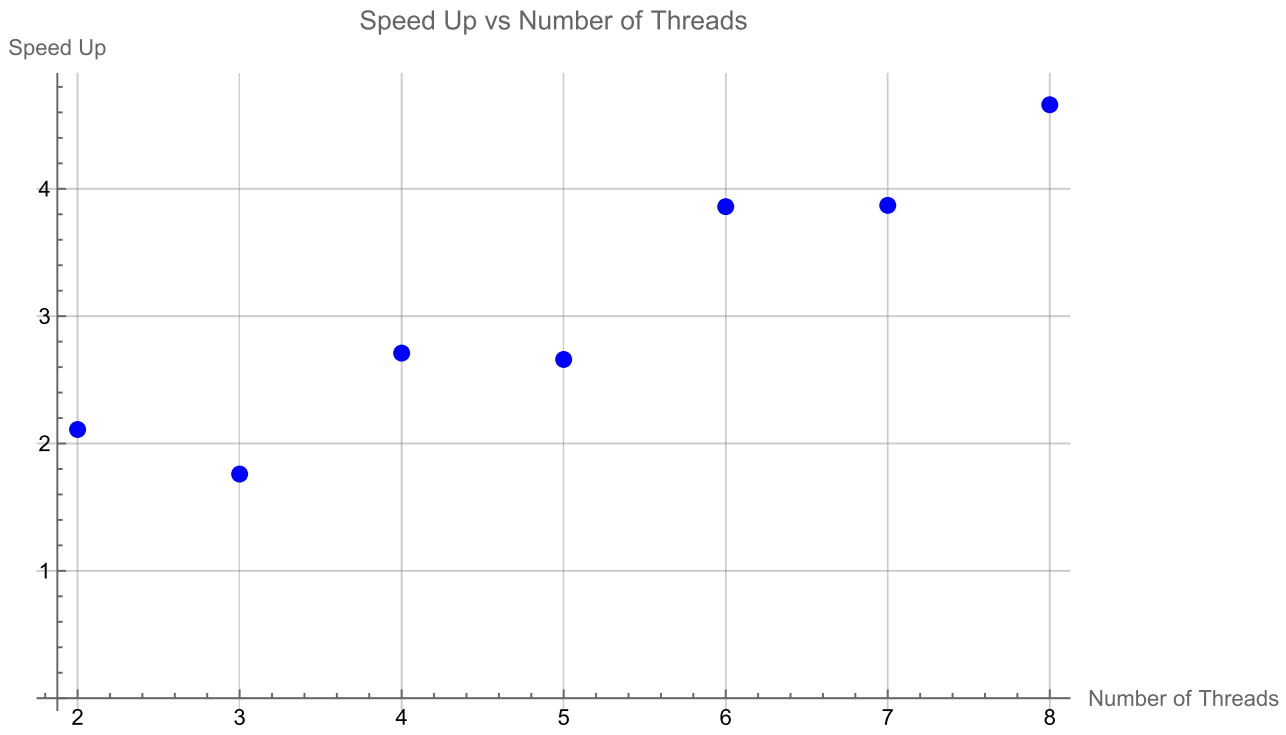
Assignment 1: Performance Analysis on a Quad-Core CPU

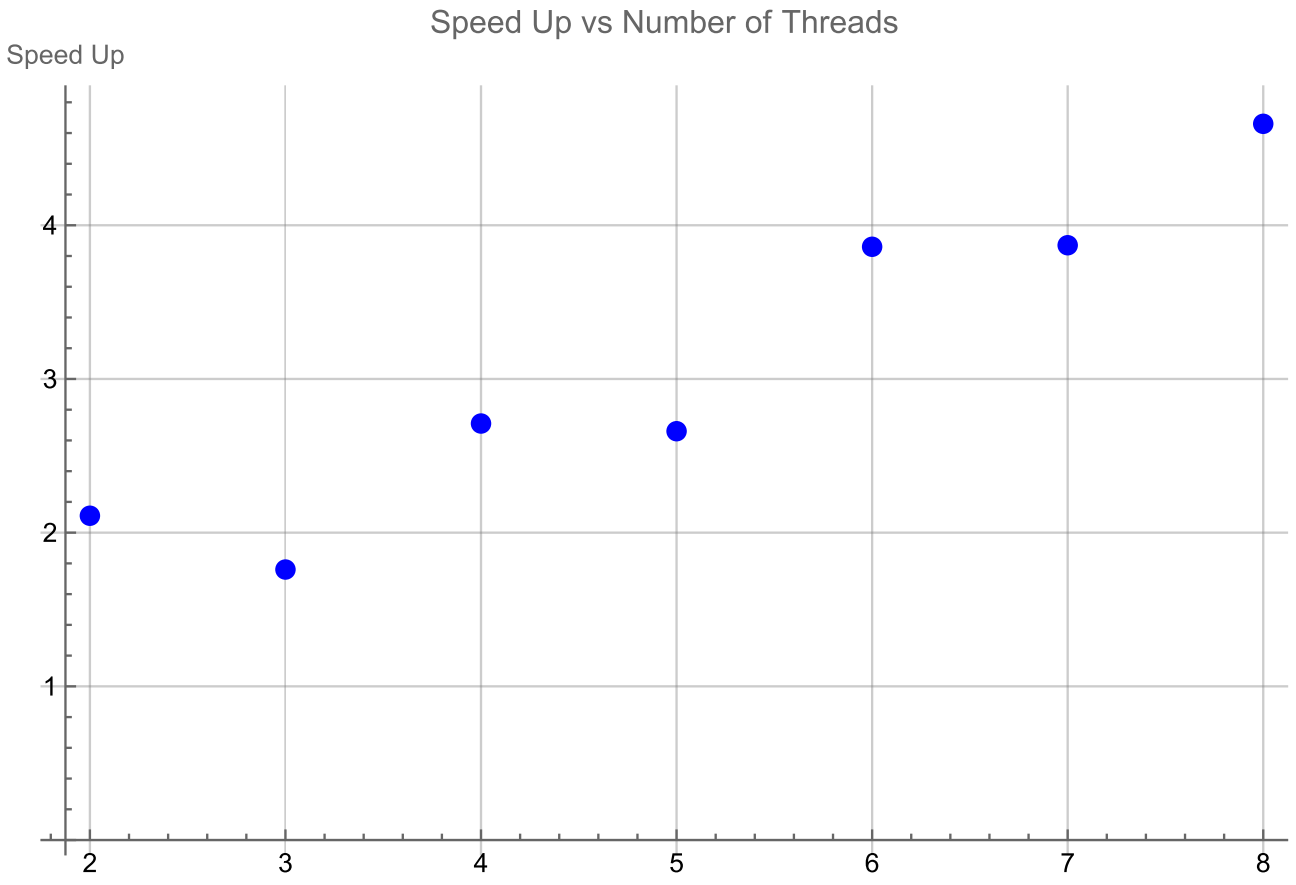
Program 1: Parallel Fractal Generation Using Threads (20 points)

Extend your code to use 2, 3, 4, 5, 6, 7, and 8 threads, partitioning the image generation work accordingly (threads should get blocks of the image).

a graph of **speedup compared to the reference sequential implementation** as a function of the number of threads used **FOR VIEW 1**.



Is speedup linear in the number of threads used? In your writeup hypothesize why this is (or is not) the case?



The computations required for the pixels in the Mandelbrot set graph are not proportional to the height of the graph. Specifically, the middle section of the graph requires much more intensive computation, while the upper and lower edges need significantly less. When the partitioning uses an odd number of threads, the thread responsible for the middle section carries a disproportionately large computational load. As a result, the overall speedup is reduced, since the total performance is constrained by the thread with the longest execution time.

To confirm (or disprove) your hypothesis, measure the amount of time each thread requires to complete its work by inserting timing code at the beginning and end of workerThreadStart(). How do your measurements explain the speedup graph you previously created?

```
[mandelbrot serial]:            [662.859] ms
Wrote image file mandelbrot-serial.ppm
[Thread 0]:             [123.480] ms
Hello world from thread 0
[Thread 2]:             [125.153] ms
Hello world from thread 2
[Thread 1]:             [398.351] ms
Hello world from thread 1
[Thread 2]:             [126.046] ms
Hello world from thread 2
[Thread 0]:             [127.629] ms
Hello world from thread 0
[Thread 1]:             [399.544] ms
Hello world from thread 1
[Thread 0]:             [123.503] ms
Hello world from thread 0
[Thread 2]:             [126.790] ms
Hello world from thread 2
[Thread 1]:             [402.205] ms
Hello world from thread 1
[Thread 0]:             [117.776] ms
Hello world from thread 0
[Thread 2]:             [120.458] ms
Hello world from thread 2
[Thread 1]:             [390.832] ms
Hello world from thread 1
[Thread 0]:             [111.925] ms
Hello world from thread 0
[Thread 2]:             [113.352] ms
Hello world from thread 2
[Thread 1]:             [389.487] ms
Hello world from thread 1
[mandelbrot thread]:            [389.773] ms
Wrote image file mandelbrot-thread.ppm
                                (1.70x speedup from 3 threads)
```

If I spawn 3 threads, thread 1 takes significantly larger time.

Modify the mapping of work to threads to achieve to improve speedup to at **about 7-8x on both views** of the Mandelbrot set

```
Wrote image file mandelbrot-serial.ppm
Hello world from thread 3
Hello world from thread 6
Hello world from thread 4
Hello world from thread 1
Hello world from thread 2
Hello world from thread 7
Hello world from thread 0
Hello world from thread 5
Hello world from thread 6
Hello world from thread 4
Hello world from thread 0
Hello world from thread 3
Hello world from thread 5
Hello world from thread 1
Hello world from thread 2
Hello world from thread 7
Hello world from thread 1
Hello world from thread 5
Hello world from thread 0
Hello world from thread 4
Hello world from thread 3
Hello world from thread 2
Hello world from thread 7
Hello world from thread 6
Hello world from thread 0
Hello world from thread 3
Hello world from thread 5
Hello world from thread 4
Hello world from thread 6
Hello world from thread 1
Hello world from thread 2
Hello world from thread 7
Hello world from thread 2
Hello world from thread 4
Hello world from thread 0
Hello world from thread 5
Hello world from thread 1
Hello world from thread 6
Hello world from thread 7
Hello world from thread 3
[mandelbrot thread]:            [90.192] ms
Wrote image file mandelbrot-thread.ppm
                                (7.42x speedup from 8 threads)
```

Instead of using a blocked mapping method, I took an interleaved approach. Each thread calculates spaced-out rows, which helps distribute the computational load more evenly.

Now run your improved code with 16 threads. Is performance noticably greater than when running with eight threads? Why or why not?

There is roughly no improvement because of two main factors: the limitations of memory latency and bandwidth, and Amdahl's Law.

Program 2: Vectorizing Code Using SIMD Intrinsics (20 points)

Implement a vectorized version of clampedExpSerial in clampedExpVector . Your implementation should work with any combination of input array size (N) and vector width (VECTOR_WIDTH).

Does the vector utilization increase, decrease or stay the same as VECTOR_WIDTH changes? Why?

```
CLAMPED EXPONENT (required)
Results matched with answer!
****************** Printing Vector Unit Statistics ******
Vector Width:               2
Total Vector Instructions: 162684
Vector Utilization:        85.1%
Utilized Vector Lanes:     276855
Total Vector Lanes:        325368
************************ Result Verification ************
Passed!!!
```

```
CLAMPED EXPONENT (required)
Results matched with answer!
****************** Printing Vector Unit Statistics ******************
Vector Width:               16
Total Vector Instructions: 26924
Vector Utilization:        76.5%
Utilized Vector Lanes:     329413
Total Vector Lanes:        430784
************************ Result Verification *************************
Passed!!!
```

```
CLAMPED EXPONENT (required)
Results matched with answer!
****************** Printing Vector Unit Statistics ******************
Vector Width:               4
Total Vector Instructions: 94532
Vector Utilization:        80.2%
Utilized Vector Lanes:     303214
Total Vector Lanes:        378128
************************ Result Verification *************************
Passed!!!
```

```
Results matched with answer!
****************** Printing Vector Unit Statistics ******************
Vector Width:               8
Total Vector Instructions: 51584
Vector Utilization:        77.7%
Utilized Vector Lanes:     320447
Total Vector Lanes:        412672
************************ Result Verification *************************
Passed!!!
```

the vector utilization decrease as VECTOR_WIDTH increase.

(1) Limitations of Memory Bandwidth: As VECTOR_WIDTH increases, the memory system may struggle to keep pace with the processing speed. This bandwidth limitation can prevent the vector units from being fully utilized, as data can't be supplied fast enough.

(2) Increased Masking of Vector Lanes: As VECTOR_WIDTH grows, masking operations may cause more vector lanes to be masked off, leaving them idle. When some data elements finish their calculations earlier, these vector lanes remain unused, reducing overall utilization.

Program 3, Part 1. A Few ISPC Basics (10 of 20 points)

What is the maximum speedup you expect given what you know about these CPUs? Why might the number you observe be less than this ideal?

SIMD计算的负载不均匀性：Mandelbrot集的计算在图像的不同区域复杂性差异很大。某些区域的点会非常快地退出迭代循环，而其他区域则需要进行较多的迭代。因此，虽然SIMD向量化允许并行处理多个像素，但并不是所有像素都需要相同的计算量。这导致了SIMD计算的负载不均衡问题，使得某些向量单元空闲，从而降低了加速效果。

Program 3, Part 2: ISPC Tasks (10 of 20 points)

What speedup do you observe on view 1? What is the speedup over the version of mandelbrot_ispc that does not partition that computation into tasks?

```
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog3_mandelbrot_ispc# ./mandelbrot_ispc -t
[mandelbrot serial]:            [286.114] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [63.813] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [28.465] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.48x speedup from ISPC)
                                (10.05x speedup from task ISPC)
```

How did you determine how many tasks to create? Why does the number you chose work best?

```
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog3_mandelbrot_ispc# ./mandelbrot_isp
c -t
[mandelbrot serial]:            [302.530] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:              [65.433] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:    [9.459] ms
Wrote image file mandelbrot-task-ispc.ppm
                                (4.62x speedup from ISPC)
                                (31.98x speedup from task ISPC)
```

I created 16 tasks, which helps achieve nearly a 31x speedup. Typically, an even partition performs better than an odd partition because an odd partition distributes the workload less evenly.

The smart-thinking student's question: Hey wait! Why are there two different mechanisms (foreach and launch) for expressing independent, parallelizable work to the ISPC system? **Couldn't the system just partition the many iterations of foreach across all cores and also emit the appropriate SIMD code for the cores?**

ISPC 通过 launch 实现了多线程的task并行，再结合 foreach 实现单线程内的SIMD指令。这两种机制各自专注于并行化的不同层次，通过相互结合，ISPC 可以在多核系统上实现高效的并行计算。

Program 4: Iterative sqrt (15 points)

What is the speedup due to SIMD parallelization?

What is the speedup due to multi-core parallelization?

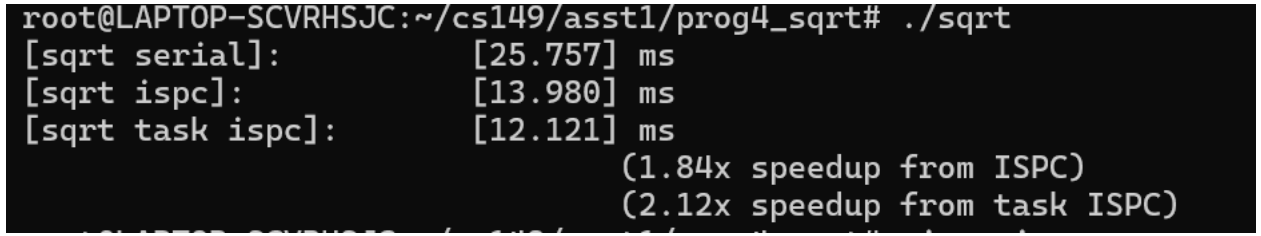
```
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog4_sqrt# ls
Makefile  main.cpp  objs  sqrt  sqrt.ispc  sqrtSerial.cpp
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog4_sqrt# ./sqrt
[sqrt serial]:          [1002.694] ms
[sqrt ispc]:            [251.816] ms
[sqrt task ispc]:       [33.590] ms
                                (3.98x speedup from ISPC)
                                (29.85x speedup from task ISPC)
```

Construct a specific input for sqrt that **minimizes speedup for ISPC (without-tasks) over the sequential version of the code**. Describe this input, describe why you chose it, and re

Construct a specific input for sqrt that minimizes speedup for ISPC (without-tasks) over the sequential version of the code. Describe this input, describe why you chose it, and report the resulting relative performance of the ISPC implementations. What is the reason for the loss in efficiency?

setting every element in the array to 1 greatly speeds up the sequential version of the code

```
29      for (unsigned int i=0; i<N; i++)
30      {
31          // TODO: CS149 students. Attempt to change the values in the
32          // array here to meet the instructions in the handout: we want
33          // to you generate best and worse-case speedups
34
35          // starter code populates array with random input values
36          values[i] = 1.f;
37      }
```



虽然 ISPC 的并行化技术能显著提升程序性能，但根据 Amdahl's Law，并行化的加速受限于程序中无法并行化的部分。即使我们将并行部分优化到极致，最终的加速效果还是会受串行部分的影响。

当串行执行时间很短（仅 25.757 毫秒），这意味着即使使用 ISPC 并行化，程序的瓶颈仍然存在于那些无法并行化的部分，导致并行加速的提升不明显（1.84x 和 2.12x 的加速）。

相比之下，当串行执行时间（421.684 毫秒）较长，因此并行化带来了显著的性能提升。通过 ISPC 的 SIMD 指令和任务并行，分别实现了 6.34x 和 30.83x 的加速。

Modify the contents of the array values to improve the relative speedup of the ISPC implementations. Construct a specifc input that **maximizes speedup over the sequential version of the code** and report the resulting speedup achieved (for both the with- and without-tasks ISPC implementations). Does your modification improve SIMD speedup? Does it improve multi-core speedup (i.e., the benefit of moving from ISPC without-tasks to ISPC with tasks)? Please explain why.

```
25      for (unsigned int i=0; i<N; i++)
24      {
23          // TODO: CS149 students.  Attempt to change the values in the
22          // array here to meet the instructions in the handout: we want
21          // to you generate best and worse-case speedups
20
19          // starter code populates array with random input values
18          values[i] = 2.998f;
17  /*
16          if (rand() % 2 == 0) {
15            // 从 [0, 0.1] 中取值
14            values[i] = .001f + static_cast<float>(rand()) / RAND_MAX * 0.1f;
13          } else {
12          // 从 [2.9, 3.0] 中取值
11            values[i] = 2.998f - static_cast<float>(rand()) / RAND_MAX * 0.1f;
10          }
 9  */
 8      }
 7
```

```
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog4_sqrt# ./sqrt
[sqrt serial]:          [2415.256] ms
[sqrt ispc]:            [436.510] ms
[sqrt task ispc]:       [59.735] ms
                                (5.53x speedup from ISPC)
                                (40.43x speedup from task ISPC)
```

Program 5: BLAS saxpy (10 points)

What speedup from using ISPC with tasks do you observe? Explain the performance of this program.

```
root@LAPTOP-SCVRHSJC:~/cs149/asst1/prog5_saxpy# ./saxpy
[saxpy ispc]:           [17.845] ms     [16.700] GB/s   [2.241] GFLOPS
[saxpy task ispc]:      [20.224] ms     [14.736] GB/s   [1.978] GFLOPS
                                (0.88x speedup from use of tasks)
```

Overhead of Task Parallelism:

While dividing the workload into smaller tasks seems reasonable, creating and managing tasks introduces overhead. For saxpy, which performs a simple operation (a multiplication and addition per element), the overhead from task creation, scheduling, and context switching can outweigh the benefits of parallel execution.

The performance is very good with non-task vectorized exection.
The benefit of creating task is trivial.

What speedup from using ISPC with tasks do you observe? Explain the performance of this program.

Cache Efficiency Issues:

Since saxpy accesses three large arrays (X[], Y[], result[]), dividing the data across multiple tasks may result in inefficient use of CPU caches, causing cache misses and slower memory access.

Do you think it can be substantially improved?

No, achieving near-linear speedup is unlikely. Even with task optimizations, the overhead from task management, scheduling, and cache inefficiencies would still hinder achieving linear speedup. The fundamental problem is that the cost of task management becomes significant relative to the amount of work being done in each task. For such a simple computation, the gains from parallelism will be limited by these overheads, and thus linear scaling is not feasible.