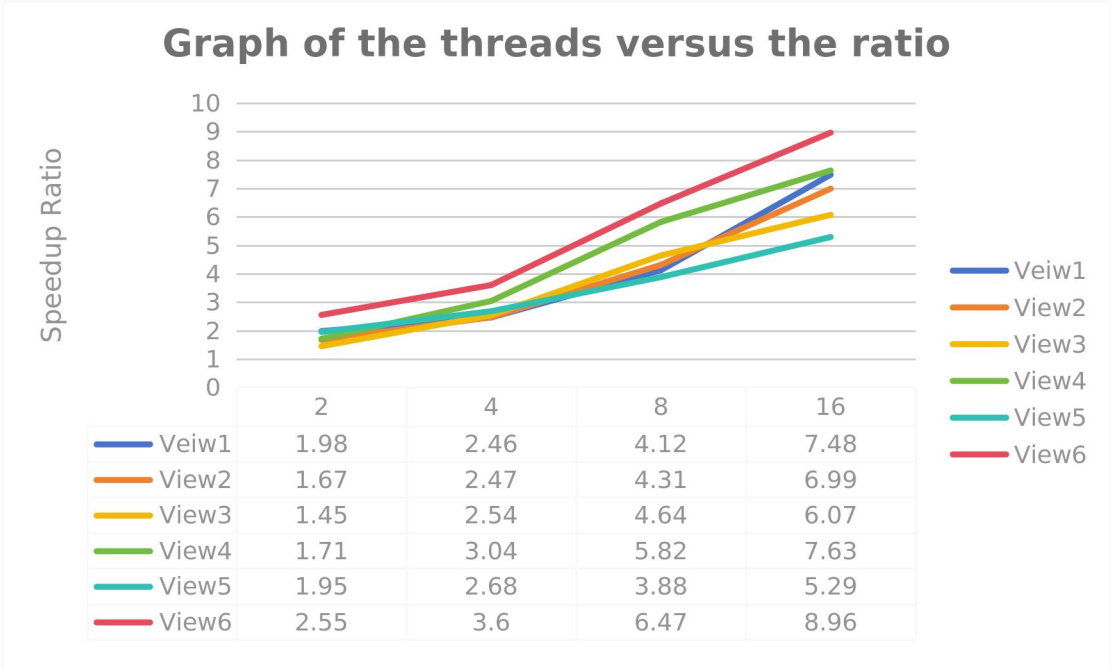
24Fall CMU 15-418 Asst1 Report

Bao Zhuhan 2024.11.07

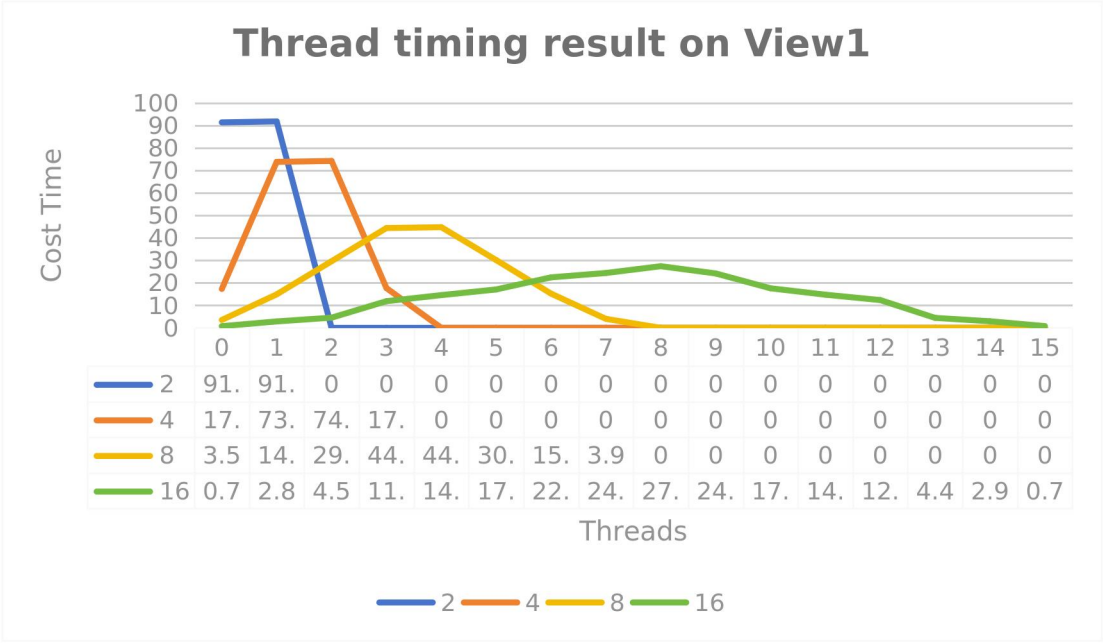
Program1

The program ran successfully. Below is a graph of the number of threads and the time acceleration ratio. For all views, the performance gains become slower as the number of threads increases. The strange thing is that for view6, when the number of threads is 2, the time acceleration ratio will exceed 2, and even reach 2.5~3.

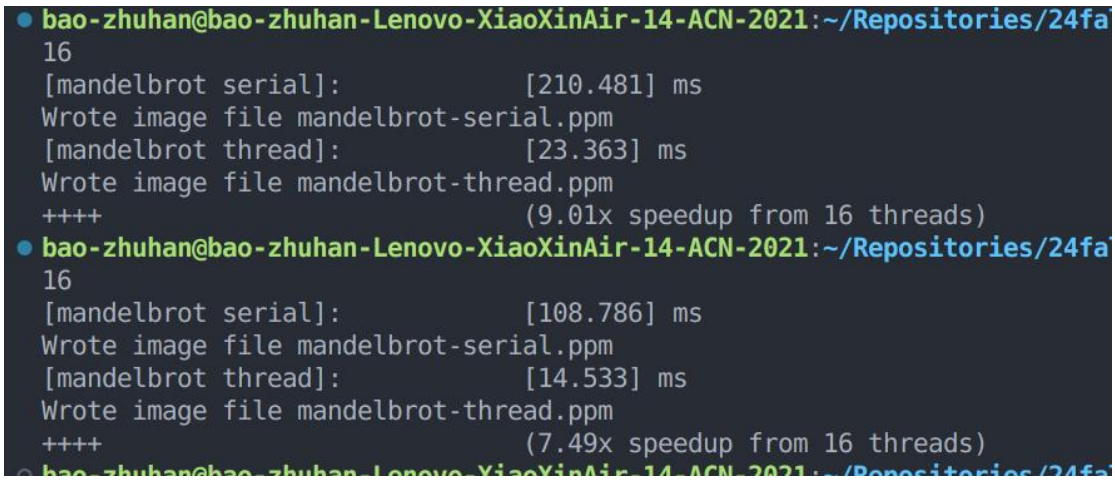


	2	4	8	16
Veiw1	1.98	2.46	4.12	7.48
View2	1.67	2.47	4.31	6.99
View3	1.45	2.54	4.64	6.07
View4	1.71	3.04	5.82	7.63
View5	1.95	2.68	3.88	5.29
View6	2.55	3.6	6.47	8.96

Each thread is assigned a different number of tasks, and in fact there is a clear normal distribution. Ideally, we want to get better performance with an equal number of tasks assigned to each thread.



I distribute the workload evenly by spacing the rows of images by the number of threads. Specifically, the number of rows processed by each thread is evenly distributed, avoiding a situation where some threads process more rows and others process fewer rows. The 16-thread parallel acceleration ratio of View1 reached 9.01, and the 16-thread parallel acceleration ratio of View2 reached 7.49.



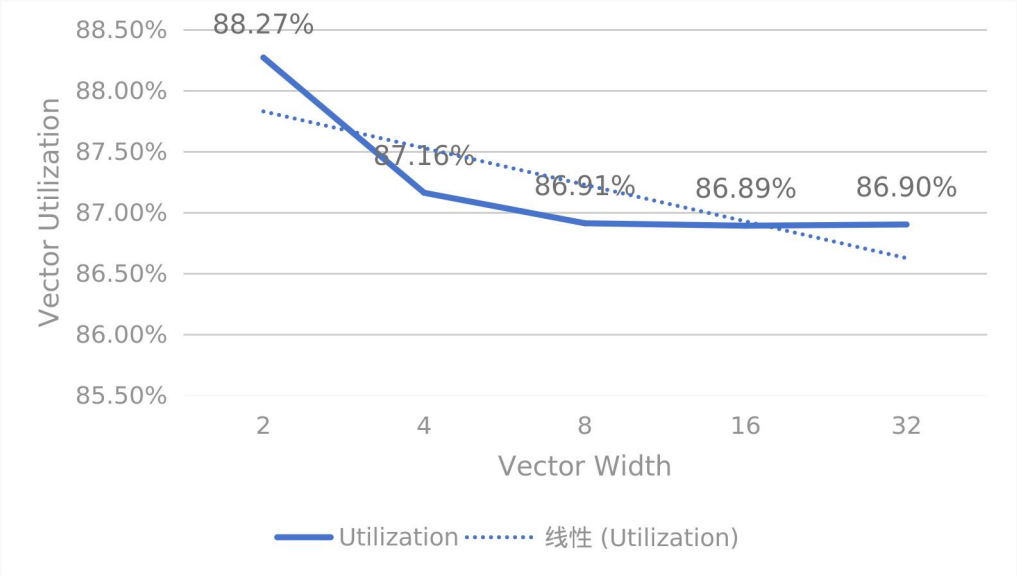
Program2

My program executed successfully and matched the expected output. The vector unit statistics show high efficiency with a vector utilization rate of 89.65%. The result verification passed, confirming the correctness of my implementation. Additionally, the array sum (bonus) test also passed, demonstrating the accuracy of my array sum function. Overall, my implementation is both correct and efficient.

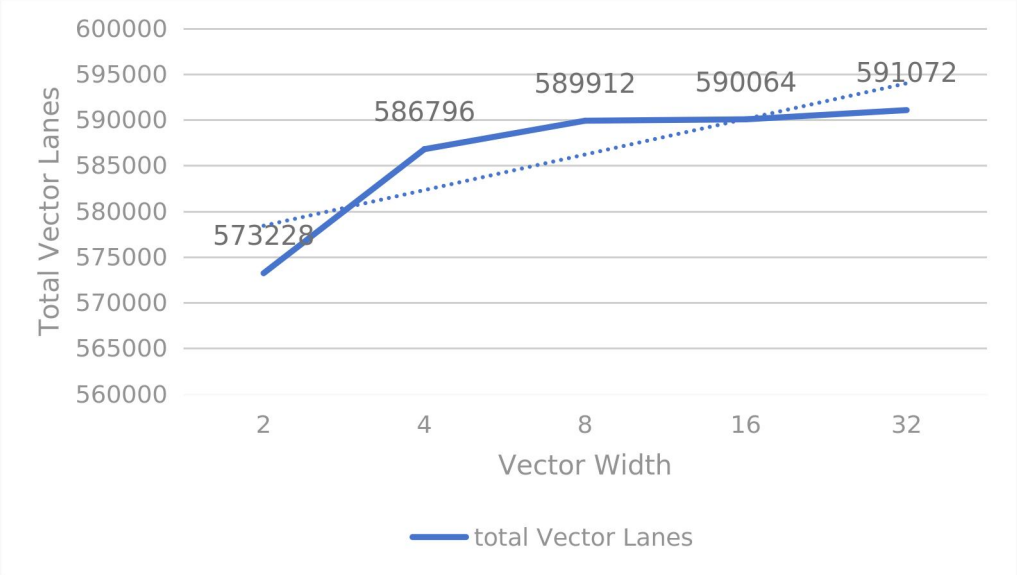
```
CLAMPED EXPONENT (required)
Results matched with answer!
****************** Printing Vector Unit Statistics ******************
Vector Width:              8
Total Vector Instructions: 122
Vector Utilization:        89.651639%
Utilized Vector Lanes:     875
Total Vector Lanes:        976
************************ Result Verification *************************
Passed!!!

ARRAY SUM (bonus)
Passed!!!
```

The decline in vector utilisation is due to factors such as data alignment and boundary effects, but the change is small (about 2%), indicating that my implementation has become relatively efficient. The utilisation rate is less sensitive to the width of the vector, which means that my data and computing tasks are more evenly distributed, and the vectorisation overhead is low.



As the vector width increases, the total number of vector instructions slowly increases



Program3

Baseline

```
[mandelbrot serial]:              [142.181] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:                [28.664] ms
Wrote image file mandelbrot-ispc.ppm
                                  (4.96x speedup from ISPC)
```

3.1 Problem 3, Part 1. A Few ISPC Basics

1. Ideally, an ISPC compiler using 8-wide AVX2 vector instructions can achieve up to 8x speedup, as each SIMD instruction can process 8 data elements simultaneously.In fact, only 4.96 times the acceleration was achieved.
2. The main reasons include: I. Load imbalance causes some processors in the SIMD unit to be idle, which reduces the overall efficiency. II. Discontinuous or misaligned memory access patterns can cause cache misses and memory access delays, impacting performance.

3.2 Problem 3, Part 2: ISPC Tasks

1. Baseline : 4.97x speedup from ISPC, 9.89x speedup from task ISPC.
2. Update: By setting the number of tasks to the same as the number of cores, a performance improvement of nearly 20 times was obtained.

```
[mandelbrot serial]:              [142.247] ms
Wrote image file mandelbrot-serial.ppm
[mandelbrot ispc]:                [28.552] ms
Wrote image file mandelbrot-ispc.ppm
[mandelbrot multicore ispc]:      [7.422] ms
Wrote image file mandelbrot-task-ispc.ppm
                                  (4.98x speedup from ISPC)
                                  (19.16x speedup from task ISPC)
```

Program4

1. Build and run.

```
[sqrt serial]:          [374.137] ms
[sqrt ispc]:            [73.130] ms
[sqrt task ispc]:       [8.348] ms
                                (5.12x speedup from ISPC)
                                (44.82x speedup from task ISPC)
```

2. Generate values that are small and close to each other (Why? 生成的小且接近的数据可以更好地利用 SIMD 指令，因为 SIMD 指令在处理相似数据时效率更高。这些数据可以减少分支预测失败和缓存未命中，从而提高并行处理的效率。)

```
[sqrt serial]:          [319.493] ms
[sqrt ispc]:            [50.488] ms
[sqrt task ispc]:       [6.861] ms
                                (6.33x speedup from ISPC)
                                (46.57x speedup from task ISPC)
```

3. Generate values that are large and vary significantly(Why？生成的大且变化显著的数据会导致 SIMD 指令处理效率低下，因为这些数据可能会导致更多的分支预测失败和缓存未命中。这些数据会增加内存带宽的压力，从而降低 SIMD 的处理效率。)

```
[sqrt serial]:          [1017.782] ms
[sqrt ispc]:            [168.204] ms
[sqrt task ispc]:       [67.848] ms
                                (6.05x speedup from ISPC)
                                (15.00x speedup from task ISPC)
```

Program5(Unfinished)

1. Build and run.

```
[saxpy serial]:         [8.308] ms      [35.871] GB/s   [2.407] GFLOPS
[saxpy streaming]:      [9.386] ms      [31.753] GB/s   [2.131] GFLOPS
[saxpy ispc]:           [9.854] ms      [30.244] GB/s   [2.030] GFLOPS
[saxpy task ispc]:      [10.607] ms     [28.096] GB/s   [1.885] GFLOPS
                                (0.89x speedup from streaming)
                                (0.84x speedup from ISPC)
                                (0.78x speedup from task ISPC)
```

I. 通过符合实际硬件规模的任务划分，可以增强并行能力。

II. 通过优化数据访问模式，使得数据更好地利用 CPU 缓存，可以减少内存访问延迟。

Note:
[1]: In completing my assignments, I enlisted the help of generative AI such as Github Copilot (aka ChatGPT 4o, ChatGPT o1-preview).
[2]: Chinese-English translations are supported by Microsoft Translator.
[3]: The data of this job is completed locally, and the development environment is posted below.