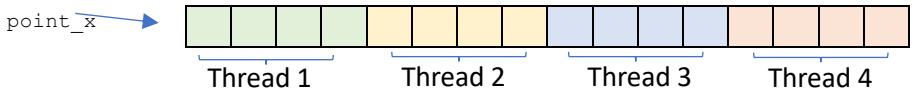
ECE1755 Assignment-1 Report

In this lab, I have implemented six optimization strategies to speed up the computation. The baseline serial version runs for approximately **17 secs**.

1. Thread-level Parallelization

This approach speeds up the execution most significantly. The speedup is $\mathbf{4.5}\times$. The workload is segmented into trunks based on the number of threads, as shown in Fig. 1.



In such a way, there is no synchronization required on the thread level. The accesses to tx data (point_x,point_y,point_z), rx_data and image are all independent.

2. Merge Tx-Rx Computation Loops and Re-use Duplicated Variables

This approach is implemented right after threads parallelization, and the speedup is on the **millisecond** scale ($\sim 0.5\ ms$), pushing the overall speedup to $\mathbf{5}\times$. Hence, it makes the memory access more compact.

3. Intel Intrinsic Vector Operation

Inspired by Piazza post and lecture note, single instruction and multiple data execution accelerated the program by $1\sim 1.5\ secs$. The overall speedup is around $\mathbf{6}\sim\mathbf{6.5}\times$.

4. GCC Ofast Compilation Flag

For this computation intense program, `-ffast-math` flag included in `GCC` is shown not causing any floating-number calculation. we assume that that *all math is finite*, no need to check `NaN` (or `zero`). It is also safe to disable `errno`, which saves one writing to a thread-local variable. Since we index `rx_data` with `int` and divide a constant, the enabled *reciprocal approximations* for division and reciprocal square root also proves to work.

5. `restrict` Keyword

Opposite to what `violatile` indicates the compiler to fetch from memory every time using a variable, the `restrict` C-only keyword tells the compiler that for the lifetime of this pointer, no other pointers will be used to access this specific object, i.e. an array of `float`.

6. Single write at the end of image shading loop

Instead of writing directly to the `image` for the approximately 1000 times at each loop iteration, we keep the temporary sum in a float and perform a single write at the end of this loop.

Approaches 4-6 jointly bring the speedup to be around $\mathbf{7}\times$, around $\mathbf{2.5}\sim\mathbf{2.6}\ \boldsymbol{secs}$.