Rashmi Anil, Ishaan Gupta
15418
Final Project Checkpoint Report
30 November 2020

**Work Completed So Far:**

Over the last few weeks, our group has managed to complete implementing a serial version as well as the SIMD version of the Hough Transform Algorithm. For the fast serial version, we implemented the standard Hough Transform algorithm as described in our proposal:

```
·     Initialize accumulator H to all zeros
·     For each edge point (x,y) in the image
·            For θ = 0 to 180
                 ρ = x cos θ + y sin θ
                 H(θ, ρ) = H(θ, ρ) + 1
·     Find the value(s) of (θ, ρ) where H(θ, ρ) is a local maximum
·     The detected line in the image is given by ρ = x cos θ + y sin θ
```

We also compared our algorithm with OpenCV on different test images to ensure that it was correct. We didn't use the OpenCV implementation as our serial one since after looking closely at the OpenCV code, it wasn't completely serial.

We then implemented a SIMD implementation using AVX intrinsics. We had to research the available intrinsics and which ones would best speed up the serial implementation. We used SIMD vectors with a width of 8 so that 8 floats could be processed at a time. This was used to speed up the inner for loop of the algorithm in which we processed 8 values of theta at a time instead of just 1. As a result, we saw that the serial implementation which took 2.78 seconds was sped up to 0.71 seconds using SIMD.

**Project Schedule:**

| **Week** | **Goal** |
|---|---|
| **11/29/2020 – 12/02/2020** | Implement OpenMP version of the Hough Transform<br>-Ishaan and Rashmi |
| **12/03/2020 – 12/05/2020** | Benchmark OpenMP version<br>-Ishaan |
| **12/06/2020 – 12/09/2020** | Implement OpenMPI version of the Hough Transform (Nice to have)<br>-Ishaan and Rashmi |

| | |
|---|---|
| **12/10/2020 – 12/12/2020** | Benchmark OpenMPI version (Nice to have)<br>-Rashmi |
| **12/13/2020 – 12/16/2020** | Compare and contrast speedup on all 3 different parallelism implementations<br>-Ishaan and Rashmi |
| **12/16/2020 – 12/18/2020** | Work on poster presentation and final writeup<br>-Ishaan and Rashmi |

**Goals and Deliverables:**

We are slightly behind where we thought we would be by the end of this week in terms of goals and deliverables. This is primarily because we forgot to factor in midterm week and thanksgiving into our schedule that we initially created. Additionally, it took us much longer than anticipated to get the serial version of the code working because we found it extremely difficult to download and install openCV on the bridges machine without sudo permissions. However, now that we have a serial implementation and a SIMD version that works with reasonable correctness, we believe that we will be able to produce all the deliverables that we said we would be able to in the proposal. This includes the OpenMp version of the Hough Transform Algorithm. We anticipate that we can have this implementation completed by the end of this upcoming week. This is because we spent a lot of time writing our own serial version of the code in such a way that converting it to OpenMP. We have done some mild benchmarking with the code we have written so far using the `time` command on the command line to get a rough estimate of the speedups of our implementation. However, we plan to use actual timing code to benchmark the algorithms later this week. If we get the OpenMP implementation and the benchmarking done before November 6th, we plan to work on the “nice to have” portion of our project which involves implementing the algorithm using OpenMPI.

**Poster Session**

At the poster session, we plan to show graphs of the speedup as we increase processors for the OpenMP and OpenMPI implementations of the Hough Transform. We also will show the time taken by the SIMD implementation and our fast serial implementation. We will also include a compare and contrast between the different parallelism techniques to see which had the most speedup.

**Preliminary Results**

Currently, we have finished the fast serial implementation as well as the SIMD implementation of the Hough Transform. Our fast serial implementation took 2.78 seconds to finish running on an image while our SIMD implementation took 0.71 seconds. This shows that the SIMD implementation had significant speedup over the serial version.

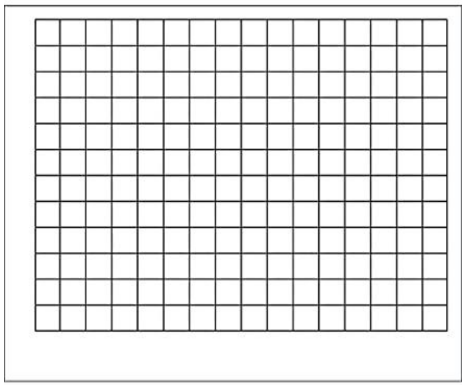
The main part of our serial code is as follows:

```
for (int i = 0; i < h; i++) {
  for (int j = 0; j < w; j++) {
    if (img_data.at<uint8_t>(i, j) != 0) {
      for (int theta = 0; theta < accum_width; theta++) {
        int rho = round(j * cos(theta * (PI / 180.0f)) + i * sin(theta * (PI / 180.0f)));
        rho += (accum_height - 1) / 2;
        int index = (rho * accum_width) + theta;
        accum[index]++;
      }
    }
  }
}
```

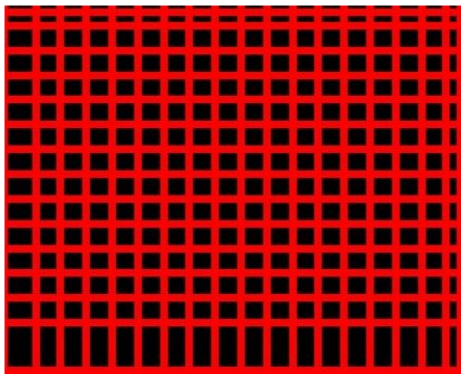
The heart of the SIMD code is as follows:

```
__m256 half_rho_height_v = _mm256_broadcast_ss(&half_rho_height);
__m256 accum_width_v = _mm256_broadcast_ss((float *)&accum_w);

for (int i = 0; i < h; i++) {
  for (int j = 0; j < w; j++) {
    if (img_data.at<uint8_t>(i, j) != 0) {

      float x_val = j * 1.0f;
      float y_val = i * 1.0f;
      x = _mm256_broadcast_ss((float *)&x_val);
      y = _mm256_broadcast_ss((float *)&y_val);

      for (int32_t theta = 0; theta < accum_width; theta += 8) {
        rho_vec = _mm256_setzero_ps();
        cos_vec1 = _mm256_loadu_ps(&(cos_theta[theta]));
        sin_vec1 = _mm256_loadu_ps(&(sin_theta[theta]));
        theta_vec1 = _mm256_set_ps(theta + 7.0f, theta + 6.0f, theta + 5.0f, theta + 4.0f,
                                   theta + 3.0f, theta + 2.0f, theta + 1.0f, theta + 0.0f);
        rho_vec = _mm256_fmadd_ps(x, cos_vec1, rho_vec);
        rho_vec = _mm256_fmadd_ps(y, sin_vec1, rho_vec);

        rho_floor = _mm256_round_ps(rho_vec, _MM_FROUND_TO_NEAREST_INT | _MM_FROUND_NO_EXC);
        rho_floor = _mm256_add_ps(half_rho_height_v, rho_floor);
        index = _mm256_cvttps_epi32(_mm256_fmadd_ps(accum_width_v, rho_floor, theta_vec1));

        accum[_mm256_extract_epi32(index, 0)]++;
        accum[_mm256_extract_epi32(index, 1)]++;
        accum[_mm256_extract_epi32(index, 2)]++;
        accum[_mm256_extract_epi32(index, 3)]++;
        accum[_mm256_extract_epi32(index, 4)]++;
        accum[_mm256_extract_epi32(index, 5)]++;
        accum[_mm256_extract_epi32(index, 6)]++;
        accum[_mm256_extract_epi32(index, 7)]++;
      }
    }
  }
}
```

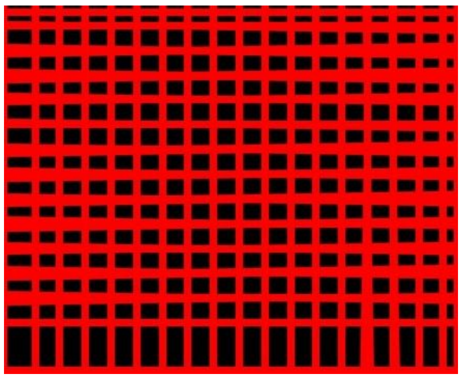
**Original Image**



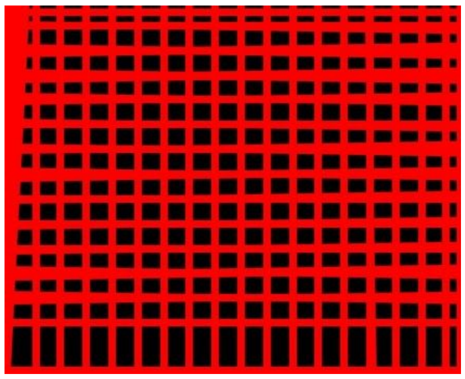
**OpenCV Detection**



**Serial Implementation**



**SIMD Implementation**



The images above show that our serial version and the SIMD version are reasonably correct when compared to the OpenCV version.

**Issues and Concerns**

While we have finished the fast serial version of our code and the SIMD version, we have not started coding the OpenMP or OpenMPI versions. However, we have a good idea of how we will use these libraries to parallelize the code and have found areas where parallelism could give significant speedup. As a result, we don’t think it will be too difficult to implement these versions in the future.