Ishank Kumar
CS22B1043
HIGH PERFORMANCE COMPUTING
Tutorial – 07

Algorithm: Canny Edge Detection
(Ran the algo on 1000 images each time)

1) Write OpenMP Parallel Code

magnitude_matrix function dominates execution time, accounting for 95.35% of the total runtime.

```
void magnitude_matrix(double **pic, double **mag, double **x, double **y)
{
    int dim = 6 * sig + 1, cent = dim / 2;
    double maskx[dim][dim], masky[dim][dim];

    // Fill the mask values using the Gaussian 1st derivative formula
    #pragma omp parallel
    {

        #pragma omp for //collapse(2) schedule(dynamic)
        for (int p = -cent; p <= cent; p++)
        {
            for (int q = -cent; q <= cent; q++)
            {
                maskx[p+cent][q+cent] = q * exp(-1 * ((p * p + q * q) / (2 * sig * sig)));
                masky[p+cent][q+cent] = p * exp(-1 * ((p * p + q * q) / (2 * sig * sig)));
            }
        }

        // Scanning convolution
        #pragma omp for //collapse(2) //schedule(dynamic)
        for (int i = 0; i < height; i++)
        {
            for (int j = 0; j < width; j++)
            {
                double sumx = 0, sumy = 0;

                // Convolution
                for (int p = -cent; p <= cent; p++)
                {
                    for (int q = -cent; q <= cent; q++)
                    {
                        if ((i+p) < 0 || (j+q) < 0 || (i+p) >= height || (j+q) >= width)
                        continue;

                        sumx += pic[i+p][j+q] * maskx[p+cent][q+cent];
                        sumy += pic[i+p][j+q] * masky[p+cent][q+cent];
                    }
                }

                // Store convolution result in respective matrix
                x[i][j] = sumx;
                y[i][j] = sumy;
            }
        }

    }
```

```
    }
    // Find magnitude and maxVal
    double maxVal = 0;
    #pragma omp parallel for reduction(max:maxVal)//collapse(2) schedule(dynamic)
    for (int i = 0; i < height; i++)
    {
        for(int j = 0; j < width; j++)
        {
            mag[i][j] = sqrt((x[i][j] * x[i][j]) + (y[i][j] * y[i][j]));
            if (mag[i][j] > maxVal)
                maxVal = mag[i][j];
        }
    }
```

```
    }

    // Normalize magnitudes to the range 0-255
    #pragma omp parallel for //collapse(2) schedule(dynamic)
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            mag[i][j] = mag[i][j] / maxVal * 255;
        }
    }

    /* // for shared testing and synchronization⋯

}
```

repository link: https://github.com/ishankkumar-007/canny-parallel

2) Report Threads vs Time

With power connected:

canny-parallel > csv > performance.csv.save1000.for-2-with-power > data

```
Threads ,Time
      1 ,20.567500
      2 ,13.995522
      4 ,10.864256
      6 ,11.344192
      8 ,10.705076
     10 ,10.322820
     12 , 9.984937
     16 , 9.616416
     20 , 9.487847
     32 ,10.531266
     64 ,12.927975
```

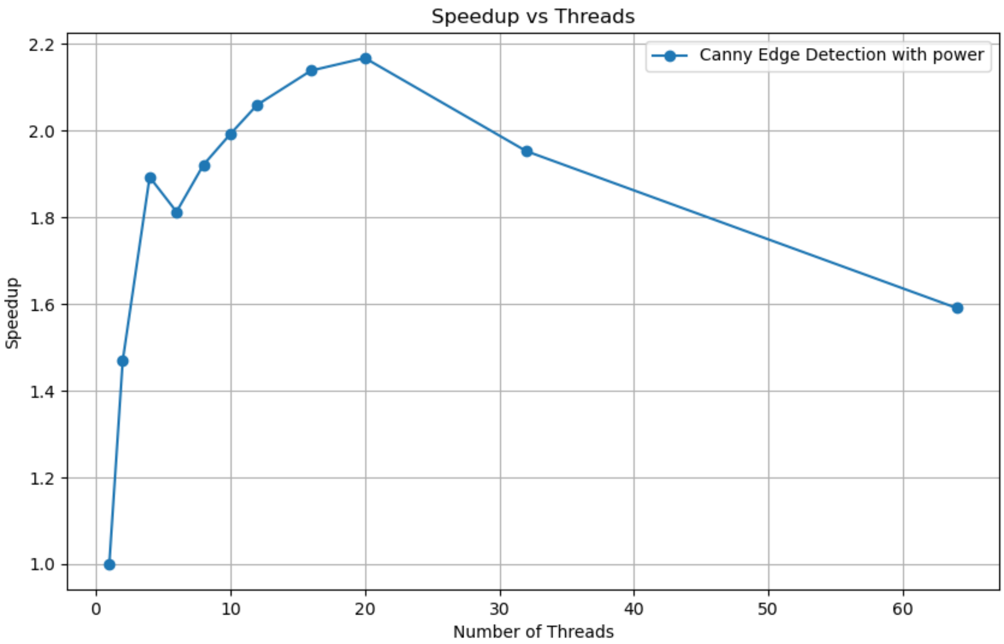
Without power connected:

canny-parallel > csv > performance.csv.save1000.for-2-without-power > data

```
Threads ,Time
      1 ,34.742106
      2 ,24.539253
      4 ,19.276993
      6 ,18.002691
      8 ,18.900125
     10 ,19.021895
     12 ,17.322950
     16 ,17.147608
     20 ,15.954623
     32 ,21.715328
     64 ,19.039861
```

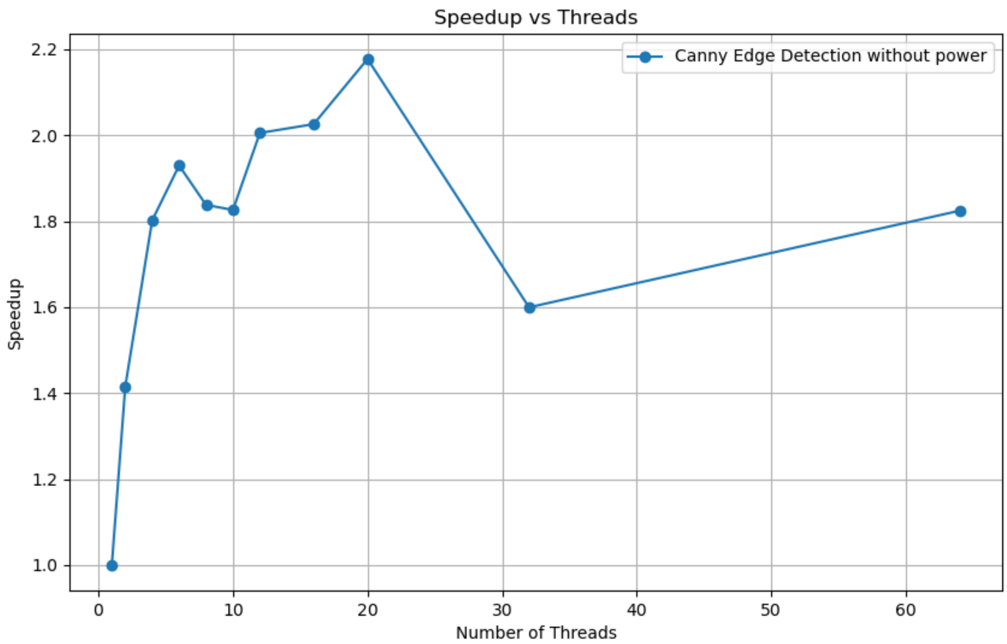
3) Plot Speed up vs Processors and estimate parallelization fraction - Inference

With power connected:




```
ishank@ishank-LOQ-15IRX9:~/Desktop/tutorials_hpc/t07/canny-parallel/python_scripts$ python per_t
    Threads       Time   Speedup  Parallel_Fraction
0         1  20.567500  1.000000           0.000000
1         2  13.995522  1.469577           0.639064
2         4  10.864256  1.893135           0.629034
3         6  11.344192  1.813042           0.538129
4         8  10.705076  1.921285           0.548017
5        10  10.322820  1.992430           0.553445
6        12   9.984937  2.059853           0.561304
7        16   9.616416  2.138791           0.567942
8        20   9.487847  2.167773           0.567050
9        32  10.531266  1.952994           0.503707
10       64  12.927975  1.590930           0.377333
ishank@ishank-LOQ-15IRX9:~/Desktop/tutorials_hpc/t07/canny-parallel/python_scripts$
```

Without power connected:




```
estimate_parallel_fraction.py           per_thread_estimate_parallel_fraction.py  plot_speed
ishank@ishank-LOQ-15IRX9:~/Desktop/tutorials_hpc/t07/canny-parallel/python_scripts$ python per
    Threads       Time   Speedup  Parallel_Fraction
0         1  34.742106  1.000000           0.000000
1         2  24.539253  1.415777           0.587348
2         4  19.276993  1.802258           0.593520
3         6  18.002691  1.929828           0.578183
4         8  18.900125  1.838195           0.521129
5        10  19.021895  1.826427           0.502759
6        12  17.322950  2.005554           0.546965
7        16  17.147608  2.026061           0.540194
8        20  15.954623  2.177557           0.569231
9        32  21.715328  1.599889           0.387052
10       64  19.039861  1.824704           0.459140
```

1. Observations

A. Performance Degradation Without Power

1. **Execution time is significantly higher without power**
    - With **1 thread**, the time increases from **20.57s (with power) to 34.74s (without power)**.
    - This is a **69% increase**, indicating **CPU throttling** due to power-saving mechanisms.
2. **Speedup**

   - Maximum speedup with power: **2.168× (20 threads)**
   - Maximum speedup without power: **2.178× (20 threads)**
   - The peak speedup is similar, but the **absolute execution time is much worse** without power.
3. **Parallel fraction is similar**
   - **With power:** Parallel fraction peaks at **0.567** (16 threads).
   - **Without power:** Parallel fraction peaks at **0.569** (20 threads).
   - **Parallel efficiency decreases faster without power**, likely due to **CPU frequency scaling** and **thermal limitations**.

B. Thread Scalability

1. **Performance improves up to ~16-20 threads, then declines**
   - Both cases show **diminishing returns beyond 20 threads** due to increased **synchronization overhead**.
2. **Performance degrades at 32+ threads**
   - With power: **Performance drops after 20 threads**.
   - Without power: **Performance drops after 20 threads, but more significantly** (possibly due to lower clock speeds).

---

2. Conclusion

A. Power Connection Significantly Improves Performance

- The **battery-only mode reduces performance by ~69%** for single-threaded execution.
- **Speedup and parallel efficiency drop faster** without power.

B. Optimal Thread Usage

- **Best performance gain is observed at 16-20 threads**.
- **Beyond 20 threads, performance degrades due to overhead and CPU limits**.