Faruk KAPLAN
18 May, 2024

Parallel Programming
Project

1 Very short pseudocode of parallel algorithm

Algorithm 1: Parallel Edge Detection Pseudocode

```
Load the image
Allocate necessary memory for variables, Define block and grid dimensions for CUDA
Start measure time
Choose one of the below methods, put command line in front of unused method:
    - Edge detection with OpenMP
        Initialize the edges of output image to zero
        Apply x mask and y mask to calculate gx and gy, then compute the gradient magnitude
        Clamp the result to [0, 255] and store in output image
    - Edge detection with CUDA
        Compute gx, gy using x mask and y mask for each pixel in the block's stride
        Calculate gradient magnitude, clamp to [0, 255], and store in output image
End measure time, print elapsed time and put the image
```

2 Explanation with Foster Methodology

2.1 Decomposition

- In this step, the problem is divided into smaller sub-problems that can be solved concurrently.
- Image splitted and assigned to different number of threads and blocks by OpenMP or CUDA according to our input

2.2 Communication

- Communication involves the exchange of data between different processing elements.
- Communication provided by OpenMP and CUDA

2.3 Aggregation

- Aggregation combines the results of individual computations to produce the final output.
- Results from different threads are summed inside methods

2.4 Mapping

- Mapping assigns computational tasks to processing elements, determining how the algorithm is executed on the underlying hardware.
- CUDA threads are mapped to available GPU resources (CUDA cores) to execute image processing tasks concurrently. Load balancing techniques ensure that computational workload is evenly distributed among threads and blocks.

2.5 Discussion

- The code utilizes both task and data parallelism. Task parallelism is employed through OpenMP for CPU-based edge detection, distributing independent pixel computations across multiple threads. Data parallelism is utilized through CUDA for GPU-based edge detection, where each thread handles a distinct block of pixels simultaneously, exploiting parallel processing power.

3 Tables for local PC results

3.1 papagan.jpg - CUDA

→ Here we kept the number of threads constant and increased the number of blocks.

Table 1: Time

# of threads	# of blocks	Sequential	Parallel
4	4	1.66	0.033
4	8	-	0.028
4	16	-	0.012
4	32	-	0.011

Table 2: Speed Up

# of threads	# of blocks	Parallel
4	4	50.3
4	8	59.29
4	16	138.33
4	32	150.91

Table 3: Efficiency

# of threads	# of blocks	Parallel
4	4	3.14
4	8	1.85
4	16	2.16
4	32	1.18

3.2 papagan.jpg - CUDA

→ Here we kept the number of blocks constant and increased the number of threads.

Table 4: Time

# of threads	# of blocks	Sequential	Parallel
4	4	1.66	0.033
8	4	-	0.012
16	4	-	0.007
32	4	-	0.004

Table 5: Speed Up

# of threads	# of blocks	Parallel
4	4	50.3
8	4	138.3
16	4	237.1
32	4	415

Table 6: Efficiency

# of threads	# of blocks	Parallel
4	4	3.14
8	4	4.32
16	4	3.7
32	4	3.24

3.3 papagan.jpg - OpenMP

Table 7: Time

# of threads	Sequential	Parallel
2	1.66	0.49
4	-	0.25
8	-	0.23
16	-	0.24

Table 8: Speed Up

# of threads	Parallel
2	3.39
4	6.64
8	7.22
16	6.92

Table 9: Efficiency

# of threads	Parallel
2	1.69
4	1.66
8	0.9
16	0.43

3.4 papagan.jpg - OpenMP

→ Here we use same number of threads for all schedulers (8).

Table 10: Static

Chunk Size	Time	Speed Up
1	0.239	6.945
100	0.231	7.186

Table 11: Dynamic

Chunk Size	Time	Speed Up
1	0.251	6.614
100	0.255	6.509

Table 12: Guided

Chunk Size	Time	Speed Up
100	0.24	6.917
1000	0.233	7.124

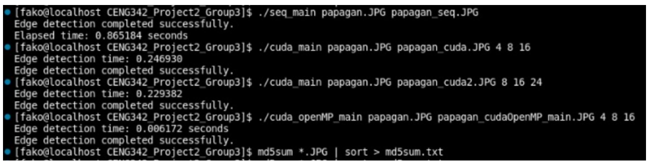
Table 13: Default

# of threads	Time	Speed Up
8	0.23	7.22

→ According to this measurements, the best options are default, (static, 100) and (guided, 1000)

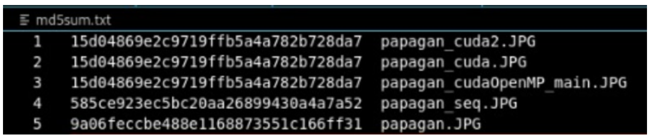
4 Checksum

→ The reason behind sequential (gcc) and CUDA (nvcc) image's hash values different is compiler.



```
[fako@localhost CENG342_Project2_Group3]$ ./seq_main papagan.JPG papagan_seq.JPG
Edge detection completed successfully.
Elapsed time: 0.865184 seconds
[fako@localhost CENG342_Project2_Group3]$ ./cuda_main papagan.JPG papagan_cuda.JPG 4 8 16
Edge detection time: 0.246930
Edge detection completed successfully.
[fako@localhost CENG342_Project2_Group3]$ ./cuda_main papagan.JPG papagan_cuda2.JPG 8 16 24
Edge detection time: 0.229382
Edge detection completed successfully.
[fako@localhost CENG342_Project2_Group3]$ ./cuda_openMP_main papagan.JPG papagan_cudaOpenMP_main.JPG 4 8 16
Edge detection time: 0.006172 seconds
Edge detection completed successfully.
[fako@localhost CENG342_Project2_Group3]$ md5sum *.JPG | sort > md5sum.txt
```

Figure 1: Creating images with different files



```
md5sum.txt
1  15d04869e2c9719ffb5a4a782b728da7  papagan_cuda2.JPG
2  15d04869e2c9719ffb5a4a782b728da7  papagan_cuda.JPG
3  15d04869e2c9719ffb5a4a782b728da7  papagan_cudaOpenMP_main.JPG
4  585ce923ec5bc20aa26899430a4a7a52  papagan_seq.JPG
5  9a06feccbe488e1168873551c166ff31  papagan.JPG
```

Figure 2: Hash values