CSC4005 Project 1

Jiarui Chen 120090361

Part A: RGB to Grayscale

In this part, we are provided with 1 sequential program and 6 parallel programs, which can turn RGB pictures into Grayscale pictures.

Compile and run

We can compile the programs by

```
cd ./project1
mkdir build && cd build
cmake ..
make -j4
```

The programs can be run by `sbatch` command on the cluster.

```
cd ./project1
sbatch ./src/scripts/sbatch_PartA.sh
```

Output of the programs

Example Lena-Gray image generated by simd program

Experiment results

Here is the time spent by 7 programs to handle 20K image.

NUMBER OF PROCESSES / CORES	SEQUENTIAL	SIMD (AVX2)	MPI	PTHREAD	OPENMP	CUDA	OPENACC
1	561	228	552	627	505	27	28
2	N/A	N/A	578	592	504	N/A	N/A
4	N/A	N/A	377	310	267	N/A	N/A
8	N/A	N/A	243	185	162	N/A	N/A
16	N/A	N/A	212	97	117	N/A	N/A
32	N/A	N/A	224	76	60	N/A	N/A

Part B: Image Filtering

In this part, 6 programs have been implemented using 6 different parallel programming models which can do image filtering. Also, the provided sequential implementation has been optimized.

Compile and run

We can compile the programs by

```
cd ./project1
mkdir build && cd build
cmake ..
make -j4
```

The programs can be run by `sbatch` command on the cluster.

```
cd ./project1
sbatch ./src/scripts/sbatch_PartB.sh
```

Output of the programs

Example Lena-Smooth image generated by simd program

Experiment results

Here is the time spent by 7 programs to handle 20K image.

NUMBER OF PROCESSES / CORES	SEQUENTIAL	SIMD (AVX2)	MPI	PTHREAD	OPENMP	CUDA	OPENACC
1	4708	1642	5189	5574	5343	182.5	370
2	N/A	N/A	5403	5181	4969	N/A	N/A

NUMBER OF PROCESSES / CORES	SEQUENTIAL	SIMD (AVX2)	MPI	PTHREAD	OPENMP	CUDA	OPENACC
4	N/A	N/A	3035	2619	2520	N/A	N/A
8	N/A	N/A	1855	1316	1372	N/A	N/A
16	N/A	N/A	1286	728	739	N/A	N/A
32	N/A	N/A	1022	373	434	N/A	N/A

Optimization have done

1. We optimize the algorithm in sequential program by unrolling the inner 2 loops and modifying the 2-dimension array of filter to 9 constants. The program take 10,000ms to handle the 20K image before optimization, it has speeded up more than 100%.
2. The other programs use the same way to do optimization, they are also speeded up a lot, but some of them still cannot reach the baseline.
3. After first optimization, we use shared memory of the blocks to optimize the speed of reading variables in CUDA program. The former program take about 360ms to finish the job, there is an 100% speed-up, but still cannot reach the baseline.

Conclusion

In this project, we use 6 different parallel models to implement image filtering.

PARALLEL MODEL	PARALLELISM TYPE	HARDWARE TARGET
SIMD	DLP	CPU
MPI	TLP	CPU
Pthread	TLP	CPU
OpenMP	TLP	CPU
CUDA	DLP	GPU
OpenACC	DLP	GPU