CSE 5441: Program 5 Report
Harsh Gupta

SECTION: 2:20 pm – 3:55pm Tu Thu

Contents

Objective

The objective of this lab is to use MPI and OpenMP to design a large scale parallel version of Sobel Edge detection and contrast the execution performance of Serial, CUDA, and MPI version.

MPI API's are straight forward and easy to use. Designing the workload distribution in a master-slave arrangement was a new and challenging task, especially designing the convergence loop for processes. It also took me time to fully appreciate the message passing between master and slave along with ReduceAll operation.

Parallelizing the code

Sobel transform
This program is divided in a number of threads where one acts as a master and the rest as slaves. The master is responsible to collect the result and check if convergence has occurred. For each process the Image file is read and its attributes are stored in a global variable. A multiplication factor constant is calculated by dividing the height of the image with total number of processes including the master process. All processes but last is the assigned the constant number of rows in the image. The last process is given whatever number of rows are left. Sobel operation is performed on the respective elements by the assigned processes and the **black cell count for each process assigned shared is calculated.** Once all the processes has computed their respective black cell count then a reduceAll operation is performed to see if convergence is met.

When convergence is achieved, each of the slave node sends their calculated pixels to the master node. Master process stiches the chunks together by determining the slave node id and saves the final image in an output file.

Synchronization is achieved by using MPI_Send and MPI_Recv and by using MPI_Barrier.

I have used **openMP** to parallelize the outer “for loop” of the sobel operation. Since all the threads will calculate black_cell_count separately I used **reduction on black_cell_count**. Optimal performance was observed on using **128 openMP threads**.
```
#pragma omp parallel for num_threads(128) reduction( + : black_cell_count )
```

Summary of execution time

Sobel transform

Threshold remained same for serial, CUDA and MPI versions. Therefore, making a single column for threshold.

Time is given in seconds. Time is given for Serial, CUDA, MPI and MPI with OpenMp. MPI with OpenMp is used where OpenMp is used only in the outer loop of the sobel operation as it gave the best performance. OpenMp when used in the inner loop of the sobel operation didn't give a good performance hence the result for it is not shown.

In the table X means times faster. Here Serial time is compared against and MPI's and MPI with OpenMp's time and it shows how much faster is the parallel program is against the serial program.

CUDA's time is also reported in the table. For CUDA the time which is given in the table is the time considering the memory allocation time along with memcpy time as well. This result is taken from the Lab 4.

Image	**Serial Time**	**Cuda Time**	**MPI**		**MPI with OpenMP 1 layer**		**Threshold**
			Time	**X**	**Time**	**X**	
Coins.bmp	0.153040	0.276493	0.032190	5	0.341087	0.44	49
Test_sample_1.bmp	2.258575	0.287067	0.073660	30	0.240242	9.4	36
Test_sample_2.bmp	3.551521	0.294186	0.105340	33	0.261821	13.5	54
Test_sample_3.bmp	10.005480	0.348766	0.236509	42	0.611491	16.3	39

Test_sample_4.bmp	2.710359	0.285489	0.082798	32	0.302989	8.9	42

Analysis and Observations

From the table, we can see that MPI alone gives better result than MPI with OpenMp. I did some experimentation on OpenMp with MPI and according to me doing a parallel for makes the overhead a little bit more as there are more memory access from the array using which sum1 and sum2 is calculated in the code and the reduction used for percent_black_cell_local is also making it a little slower but still the performance is not getting extremely slow compared to MPI.

Cuda's value is little slower than MPI's value. This is because the time taken for cuda considers the time taken for memcpy and malloc allocations, and not only the time taken for the sobel operation. These operations are overhead and takes time.

Another observation is if the size of the image is larger than the performance is of the parallel code is outperforming the performance of the serial code.

Were there any surprises you encountered in this exercise?
The most surprising thing which I encountered was that MPI with OpenMp was performing slower than only MPI.