Lab5 c5441aa

Omid Asudeh
OSUID: 500149733

To compile: make
To run: mpiexec -np 12 lab5.out <input_img> <output.bmp>

Timing and threshold summary:

image_name	serial	MPI 1thread	MPI 2thread	MPI 3thread	MPI 4thread	iterarion	CUDA
coins.bmp	0.06	0.03	1.37	1.65	1.58	50	0.04
image01.bmp	0.9	0.29	1.22	1.77	2.07	43	0.54
image02.bmp	1.11	0.25	1.16	1.78	1.67	33	0.7
image03.bmp	3.23	0.43	1.49	1.91	1.83	40	1.82
image04.bmp	6.91	0.71	4.41	6.9	6.87	157	3.68
image05.bmp	1.04	0.31	1.69	2.96	2.62	55	0.69
image06.bmp	0.3	0.09	0.75	1.27	1.7	17	0.25
image07.bmp	0.58	0.24	1.25	1.75	2	37	0.37
image08.bmp	2.08	0.29	3.04	4.13	3.91	106	1.14
image09.bmp	0.85	0.18	1.22	1.64	2.2	41	0.56
average timing	1.706	0.282	1.76	2.576	2.645		0.979
improvement	1	6.0496	0.969	0.662	0.644		

- As it is apparent in the results, **MPI has increased the performance 6 times** using 12 processes and 1 thread. However, as the number of threads increases the performance decreases substantially to make it even worse than the serial version. My justification is that as the cost of making and handling several threads ruins the performance. This matches with the results of the lab 2 when we were comparing the results of POSIX with OMP. OMP is a high level black box, easy to use, but not that good in performance.
-
    - **MPI load sharing**: My program partitions the input image between the processes. For instance, in the case that we have 12 processes, it will partition the input image to 12 chunks. Since images has sizes that may not always be divisible to the number of processes, the program gives the slave processes the same size chunks (image_size/number_of_processes) and the master thread will

handle the remaining. Here, the processes 1 to 11 each get the same amount, and the master process (process number 0) will handle the remaining.
    - **OMP load sharing**: To make it more parallelized, within each process, my program partitions the chunk of image that is assigned to the process to equal size small chunks (the chunk will be divided to the number of threads), and each thread works on one small chunk.
- When we fire one thread for MPI it beats the CUDA results. But for more threads, CUDA has a better performance (results in the table in the first page last column).
- One thing to note is that, when we request for GPU the serial version runs much slower. In this lab I did not requested any GPUs and you see that the serial performance is much better than previous serial results in the lab4. This weird result may also depends on the performance of nodes.