COMP 529: Parallel Programming

HW-3

Najeeb Ahmad (0059486) and Mohammad Laghari (0059487)

We have completed our implementation in 1D grid and 1D+OpenMP. Our 2D implementation is also completed and is yielding the correct image, however, the average sum is only correct up to 3 decimal places.

Introduction

The following report is divided as follows. The first section defines the parallel implementation of the program in version 1. Section 2 talks about the version 2. Section 3 discusses version 3. And Section 4 shows the results.

1. Implementation

Our implementation was done incrementally. We started with 1D process geometry, then we added OpenMP support to produce a 1D+OpenMP version. Lastly we changed the geometry to 2D with OpenMP support. The following subsections describe the working of each version

1.1 Version 1: 1-dimensional process geometry

In 1-dimensional approach, we make the master thread to do some extra work where it reads the image and it's height and width and then broadcasts it to the rest of the processes. After that the image is scattered of equal size to each of the processes so that these processes can work over their provided sub-image. After that, each image pads itself with ghost cell regions (of size one row and one column at each of its four sides) and produces a new image. This image differs from the previous one in a way that the former one only contains the divided image, whereas the latter one contains the divided image along with its ghost cells. At this point in time, each process has a separate distinct sub-image along with its ghost cells. After that, in the compute process, reduction is done to calculate the sum and sum2. Intermediate sum and sum2 are first calculated by each process and then they are reduced to produce a single value distinct for each process which is then broadcasted to rest of the process. We assert our algorithm by checking the value of sum and sum2 at this point and compare it with the values obtain through the serial execution to verify our approach's correctness.

Moving on, before executing compute loop 1, we update the ghost cells in the Y direction which is of core importance for our approach to yield accurate results. This exchanging is done at boundaries along the height. These cells are copied and the ghost cells along width are sent

and received as MPI communications. Also, the first and the last rank exchange ghost cells. After this a barrier is put to make the execution of the code consistent. Then, the compute loop 1 executes with the updated values of the ghost cells. We perform the same step before compute loop 2 in order to provide it with updated ghost cells calculated after the first loop. This process is carried out across all iterations which are defined as the command line argument.

After the noise removal algorithm is completed. We perform removal of ghost cell region at processor level and the rejoin the sub-images to form a single image. Each cell removes their ghost cells in all four direction and then an MPI_Gather command gathers all the sub-images into a single image. After this step, the master process writes back the image and computes the GFlops and verifies the accuracy of the algorithm.

1.2 Version 2: 1-dimensional processor geometry with OpenMP support

In this version, we add support to the previously implemented 1-D approach. We pick core areas that are compute intensive and add OpenMP support to them. In the compute process, all the computation areas which include Reduction, Compute loop 1 and Compute loop 2 were parallelized with OpenMP pragmas. The necessary variables which had to be private to a thread were made private to yield accurate results. Lastly, the breaking and the joining of image before and after noise removal was also parallelized by adding OpenMP directives.

1.3 Version 3: 2-dimensional process geometry with OpenMP support

This strategy was complicated to implement as it required to break down the image in both X and Y directions and then add ghost cell regions for all the sub image. We assigned the task of breaking the image to the master rank. We also take care of the cases where the image will not be evenly divisible among the processor geometry along X and Y direction. In such a case, the last rank does the extra work along both the directions. As before, the Master rank reads the image, broadcasts height and width and then makes sub-images for each of the other ranks and then sends them through MPI_Send to the rest of the ranks those receive them. After each of the process has received its' sub-image, it adds the ghost cell region to it and prepares it for noise removal.

In noise removal, the reduction of sum and sum2 is done in the same manner as it was done in 1-dimensional processor geometry. The reduction and broadcast is also performed similar to the formerly mentioned geometry.

Before progressing to Compute loop 1, ranks are calculated, as per 2D geometry, to which each process is going to send the updated values of ghost region. We also perform ghost cells data packing for the ghost cells to be sent along the height as this data is composed of discontiguous cells. This way we ensure that our destination rank receives the correct data. Unpacking is also performed in a similar fashion where each receiving process updates its image along the height. Lastly, an MPI_Barrier makes sure of the execution consistency and then Compute loop 1 executes.

The same process is done before the execution of Compute loop 2.

After all the iterations are completed and noise removal is performed, we join the image back into its original form. First all the ghost cell regions are removed from the image by each process and then each process sends its part of the subimage result to the master process. Finally, the master process rejoins all the sub-images to form a single image. This image is then written back to the disk by the master rank.

OpenMP support is added to the same regions of the code as it was in 1-dimensional version.

2. Results

The final experimentation was carried out on Lufer on compute nodes with number of CPU's less than 16. The result for serial run over 100 iterations, with lambda 1 and using coffee.pgm are as follows:

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
 0.000002 s => Part I: allocate and initialize variables
 0.000017 s => Part II: parse command line arguments
 0.589978 s => Part III: read image
 0.000024 s => Part IV: allocate variables
221.260835 s => Part V: compute
 4.024369 s => Part VI: write image to file
 0.152857 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
 0.152857 s => Part VIII: deallocate variables
Total time: 226.069224 s
Average of sum of pixels: 122.594070
GFLOPS: 0.356116

2.1 1-dimensional processor geometry

MPI1 + OMP16

Time spent in different stages of the application:
 0.011270 s => Part I: allocate and initialize variables
 0.000004 s => Part II: parse command line arguments
 0.159882 s => Part III: read image
 0.002351 s => Part IV: allocate variables
171.192067 s => Part V: compute
 2.827711 s => Part VI: write image to file
 0.094799 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
 0.094799 s => Part VIII: deallocate variables
Total time: 174.291533 s
Average of sum of pixels: 121.228231

GFLOPS: 0.460270

MPI2 + OMP8

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.297903 s => Part I: allocate and initialize variables
0.000010 s => Part II: parse command line arguments
0.124370 s => Part III: read image
0.000033 s => Part IV: allocate variables
110.773624 s => Part V: compute
4.048057 s => Part VI: write image to file
0.085621 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.085621 s => Part VIII: deallocate variables
Total time: 116.338990 s
Average of sum of pixels: 122.594070
GFLOPS: 0.711312

MPI4 + OMP4

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.568650 s => Part I: allocate and initialize variables
0.000012 s => Part II: parse command line arguments
0.253856 s => Part III: read image
0.000029 s => Part IV: allocate variables
63.378831 s => Part V: compute
4.596367 s => Part VI: write image to file
0.092996 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.092996 s => Part VIII: deallocate variables
Total time: 69.896260 s
Average of sum of pixels: 122.594070
GFLOPS: 1.243231

MPI8 + OMP2

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.212248 s => Part I: allocate and initialize variables
0.000011 s => Part II: parse command line arguments
0.048222 s => Part III: read image
0.000030 s => Part IV: allocate variables
44.087022 s => Part V: compute
7.177998 s => Part VI: write image to file
0.145689 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.145689 s => Part VIII: deallocate variables

Total time: 52.675876 s
Average of sum of pixels: 122.594070
GFLOPS: 1.787251

MPI16

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.396240 s => Part I: allocate and initialize variables
0.000025 s => Part II: parse command line arguments
0.054297 s => Part III: read image
0.000041 s => Part IV: allocate variables
21.876207 s => Part V: compute
4.163712 s => Part VI: write image to file
0.081791 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081791 s => Part VIII: deallocate variables
Total time: 27.574520 s
Average of sum of pixels: 122.594070
GFLOPS: 3.601838

2.2 1-dimensional processor geometry with OpenMP support

MPI1 + OMP16

Time spent in different stages of the application:
0.234673 s => Part I: allocate and initialize variables
0.000013 s => Part II: parse command line arguments
0.075360 s => Part III: read image
0.000050 s => Part IV: allocate variables
28.406212 s => Part V: compute
4.493192 s => Part VI: write image to file
0.081015 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081015 s => Part VIII: deallocate variables
Total time: 33.309449 s
Average of sum of pixels: 121.228231
GFLOPS: 2.773849

MPI2 + OMP8

Time spent in different stages of the application:
1.164462 s => Part I: allocate and initialize variables
0.000047 s => Part II: parse command line arguments
0.061326 s => Part III: read image
0.000047 s => Part IV: allocate variables
28.628371 s => Part V: compute
4.062682 s => Part VI: write image to file

0.081612 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081612 s => Part VIII: deallocate variables
Total time: 34.009408 s
Average of sum of pixels: 122.594070
GFLOPS: 2.752324

MPI4 + OMP4

Time spent in different stages of the application:
1.163562 s => Part I: allocate and initialize variables
0.000015 s => Part II: parse command line arguments
0.067014 s => Part III: read image
0.000035 s => Part IV: allocate variables
46.355672 s => Part V: compute
6.135292 s => Part VI: write image to file
0.124728 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.124728 s => Part VIII: deallocate variables
Total time: 53.853174 s
Average of sum of pixels: 122.594070
GFLOPS: 1.699782

MPI8 + OMP2

Time spent in different stages of the application:
1.206588 s => Part I: allocate and initialize variables
0.000011 s => Part II: parse command line arguments
0.058677 s => Part III: read image
0.000038 s => Part IV: allocate variables
45.960429 s => Part V: compute
7.134253 s => Part VI: write image to file
0.145629 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.145629 s => Part VIII: deallocate variables
Total time: 54.510313 s
Average of sum of pixels: 122.594070
GFLOPS: 1.714400

MPI16

Time spent in different stages of the application:
1.371486 s => Part I: allocate and initialize variables
0.000015 s => Part II: parse command line arguments
0.175463 s => Part III: read image
0.000038 s => Part IV: allocate variables
102.915730 s => Part V: compute
7.204158 s => Part VI: write image to file
0.124478 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS

0.124478 s => Part VIII: deallocate variables
Total time: 111.793837 s
Average of sum of pixels: 122.594070
GFLOPS: 1.765622

2.3 2-dimensional processor geometry with OpenMP support

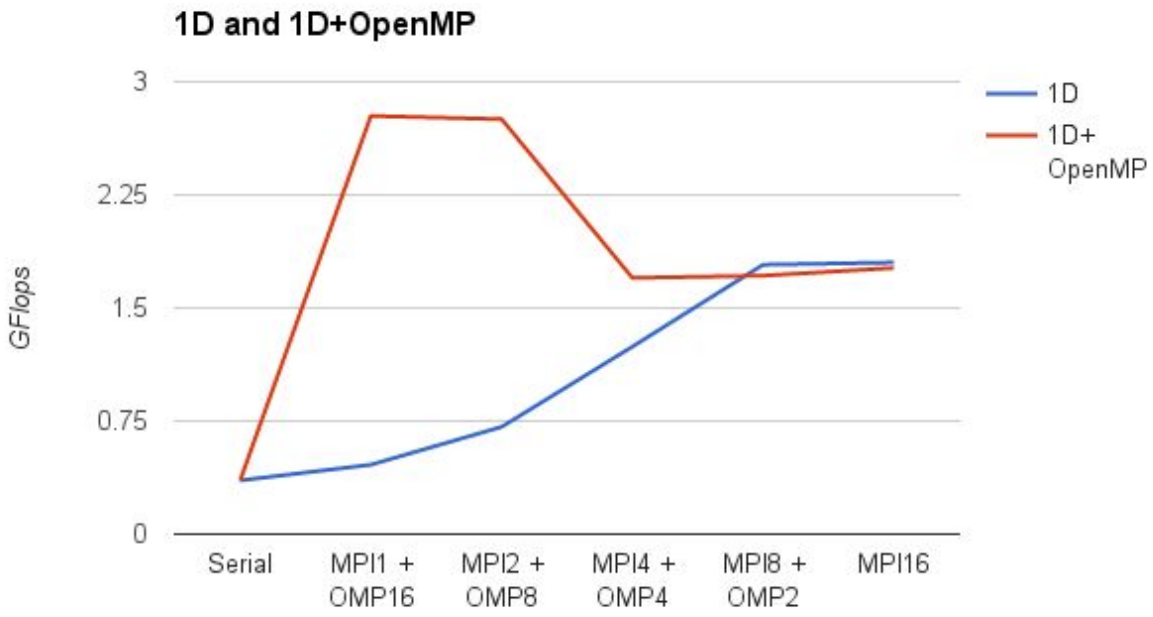
MPI4 + OMP4

Time spent in different stages of the application:
1.189475 s => Part I: allocate and initialize variables
0.000013 s => Part II: parse command line arguments
0.288011 s => Part III: read image
0.000040 s => Part IV: allocate variables
27.169115 s => Part V: compute
4.156204 s => Part VI: write image to file
0.081106 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081106 s => Part VIII: deallocate variables
Total time: 32.889727 s
Average of sum of pixels: 122.758115
GFLOPS: 2.900152

2.3 Graphical representation of data and comparison with CUDA

We achieved a GFlops rate of 62.7 on CUDA while on MPI our GFlops rate is only limited to 2.77 which occurs in 1D+OpenMP setup.
CUDA wins, because of the following reasons

1. CUDA has more threads to work simultaneously
2. Each thread in CUDA operates on a single data point
3. Inter-process communication between processes in MPI bring a significant overhead because the process may lie on a separate compute node



2.4 Measuring communication overhead

This test was done using only a single node, therefore the communication overhead is not as significant as expected.

Without Communication

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.397928 s => Part I: allocate and initialize variables
0.000011 s => Part II: parse command line arguments
0.054138 s => Part III: read image
0.000043 s => Part IV: allocate variables
21.789327 s => Part V: compute
4.158981 s => Part VI: write image to file
0.081073 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081073 s => Part VIII: deallocate variables
Total time: 27.483315 s
Average of sum of pixels: 117.423599
GFLOPS: 3.616200

With Communication

Image Read. Width : 5184, Height : 3456, nComp: 1
Time spent in different stages of the application:
1.396240 s => Part I: allocate and initialize variables
0.000025 s => Part II: parse command line arguments
0.054297 s => Part III: read image
0.000041 s => Part IV: allocate variables
21.876207 s => Part V: compute
4.163712 s => Part VI: write image to file
0.081791 s => Part VII: get average of sum of pixels for testing and calculate GFLOPS
0.081791 s => Part VIII: deallocate variables
Total time: 27.574520 s
Average of sum of pixels: 122.594070
GFLOPS: 3.601838