Goethe-University
Frankfurt am Main

Lab Parallelization · Summer Semester 2018

Prof. Dr. Ulrich Meyer
Dipl. Inf. Gerhard Leuck

Assignment 4

Hand out: 19.6.2018
Hand in: 12.7.2018 at ppva-tut@informatik.uni-frankfurt.de

Task 1
Distributed Image Processing Filter

In this assignment you write a program that applies a given filter to an image file by using parallelized computation. Each processor should receive a part of the image only. The image file is provided as a raw image file. The image is given as an array of h rows, a row has 1280 pixels, where each pixel is one unsigned char gray value.

On a Linux system, you can create such .gray image files using

```
convert image.png -geometry 1280 -depth 8 image.gray
```

You can view a .gray image by using

```
display -depth 8 -size 1280xNNNN image.gray
```

where you insert the actual height of the image as the NNNN.

You can find an example image ffm_1280x960.gray in the directory /home/lab/2018/src/4.

During runtime the user may select the filter. The following filters shoud be provided:

a) Blur:

$$F = \frac{1}{37}\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 1 & 4 & 9 & 4 & 1 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

b) Sharpen:

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

c) Relief:

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & -1 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

d) Edge detection:

$$F = \frac{1}{4} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & -12 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The filter works as follows: given the filter matrix F of dimensions $(2k+1) \times (2k+1)$, with $k = 2$: an output pixel in position $[y][x]$ is computed as

$$\text{result}[y][x] = \sum_{v=0}^{2k} \sum_{u=0}^{2k} F[v][u] \cdot \text{source}[y+v-k][x+u-k]$$

In iteration, set $result[y][x] = 0$ if the value is < 0 and set the $result[y][x] = 255$ if the value is > 255.

For simplicity reasons, you may assume that pixels outside the image (with too high or too low values of x and y) are black (have value 0).

The user should be able to specify the strength of the filter as a program parameter m for the filter to be applied m times.

Write an MPI program. Each process reads the necessary data from a original file as a block of columns by using MPI_File_read_ordered and the datatype MPI_Type_vector. It may be accepted that the number of columns can be devided by the number of processes. Create a 1-d topology by using MPI_Cart_create(). During the iterations use persistent communication and the created communicator. At the end of the calculation write the data into a new file with a collective MPI_IO function.

Task 2
Matrix multiplication with the Cannon Algorithm

The Cannon Algorithm for matrix multiplication was presented in the course
Parallel and Distributed Algorithms.

Matrix $C = A \cdot B$ should be calculated on a grid (process ij has values A_{ij} and B_{ij}) as follows:

step 1: Shift values so that the process at position ij has values $A_{i,(i+j)\bmod\sqrt{p}}$ and $B_{(i+j)\bmod\sqrt{p},j}$
step 2: Calculate $C_{ij} = A_{ij} \cdot B_{ij}$
step 3: Shift the values of matrix A left $(i \rightarrow i-1)$ and shift the values of matrix B up $(j \rightarrow j-1)$

Steps 2 and 3 are repeated $\sqrt{p}$ times.

The matrices A and B are stored in files. The datatype of the matrices elements is double.

First start only one master process. The master process provides a command line interface with at least two options quit and e.g. start. Quit should wait (test) for open computations and after finishing terminate the program. After start the user should be able to specify two matrices (or paths to matrices) which are read in from files by the master process.

Now the master process starts worker processes. The number of worker processes is depending on the size of the matrices. The master process sends only the necessary matrix values to the workers.

After spawning the worker processes and distributing the values to the workers the master waits for further interactions (quit or start). Use the C-function select() for interaction.

Implement the Cannon Algorithm on the workers by using a 2-d topology, which can be created using `MPI_Cart_create()`.

You can assume that for p processes and n x n matrix each process has m^2 values with $m = \frac{n}{\sqrt{p}}$.

During the calculation use MPI_Sendrecv_replace and MPI_Card_shift for exchange data between processes.

After the calculation all worker processes writes the results to the same result file with a collective MPI_IO function. The master process terminates if the user type exit and all results are written.

You can find an example matrix in the directory /home/lab/2018/src/4.