MPI parallelization of convolution computation

IN3200/IN4200 Home Exam 2, Spring 2021

This mandatory project (home exam No. 2) serves as a hands-on exercise of MPI programming. Each student should independently do the coding her/himself. The detailed submission information can be found at the end of this document.

1 Introduction

Convolutional neural networks (CNNs) have become a widely used strategy of machine learning. The most important components inside a CNN are so-called convolutional layers. For a 2D scenario (such as in image analysis), the action of a convolutional layer, where a convolutional kernel is applied to an input 2D array, can be illustrated by Figure 1. (Source: https://anhreynolds.com/blogs/cnn.html.)

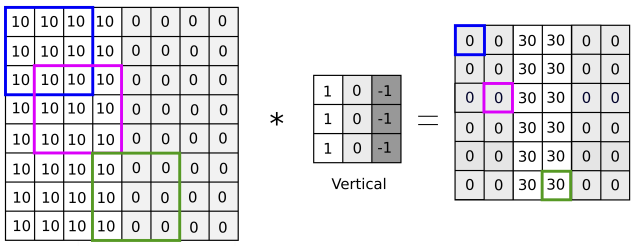

Figure 1: An example of applying a 3×3 convolutional kernel.

Accordingly, a straightforward implementation of a single-layer 2D convolution as a sequential C function is shown below:

```
void single_layer_convolution (int M, int N, float **input,
                               int K, float **kernel,
                               float **output)
{
  int i,j,ii,jj;
  double temp;
  for (i=0; i<=M-K; i++)
    for (j=0; j<=N-K; j++) {
       temp = 0.0;
```

```
        for (ii=0; ii<K; ii++)
          for (jj=0; jj<K; jj++)
            temp += input[i+ii][j+jj]*kernel[ii][jj];
        output[i][j] = temp;
    }
}
```

In the above code, the input 2D array is of dimension $M \times N$ (M rows, N columns) and the convolutional kernel is of dimension $K \times K$. The output 2D array is of dimension $(M-K+1) \times (N-K+1)$.

2 MPI programming tasks for IN3200 students (not for IN4200 students)

Task 2.1

Please write a function with the following syntax, which will be invoked inside an MPI program by *all* processes of the default `MPI_COMM_WORLD` communicator. The involved computational work should be appropriately divided among the MPI processes.

```
void MPI_single_layer_convolution (int M, int N, float **input,
                                   int K, float **kernel,
                                   float **output);
```

Remarks:

- All the function arguments, except `float **input` and `float **output`, have the same values on all the MPI processes.
- The 2D array `float **input` is *only* allocated on process 0 beforehand, with M rows and N columns. On all the other processes, `input` is just a `NULL` double pointer when function `MPI_single_layer_convolution` is called. (See Task 2.2 below for more info.)
- The 2D array `float **output` is *only* allocated on process 0 beforehand, with $M-K+1$ rows and $N-K+1$ columns. Upon return, the content of `output` on process 0 should be identical with that would have been returned by calling the sequential function `single_layer_convolution`. On all the other processes, `output` is simply a `NULL` double pointer, both before and after the function call.
- The MPI implementation should not be hard-coded for a specific number of MPI processes. Neither should M, N and K be fixed as specific values. (It can be assumed that M and N are much larger than K, as well as the total number of processes.)
- For simplicity, the division of work can adopt a 1D block-wise decomposition with respect to the rows. However, neither M nor $M-K+1$ should be assumed to be divisible by the total number of processes.

- In the beginning of function `MPI_single_layer_convolution`, the MPI collective communication function `MPI_Scatterv` is suggested to be called, with process 0 being the root, so that the content of 2D array `input` is appropriately distributed. (Note: The content of array `input` should NOT be broadcasted in its entirety from process 0 to the other processes.)

- At the end of function `MPI_single_layer_convolution`, the MPI collective communication function `MPI_Gatherv` is suggested to be called, with process 0 being the root, to collect the computed results from all the processes.

Task 2.2

Please program an MPI-parallelized `main` function using the following code skeleton:

```
int main (int nargs, char **args)
{
  int M=0, N=0, K=0, my_rank;
  float **input=NULL, **output=NULL, **kernel=NULL

  MPI_Init (&nargs, &args);
  MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);

  if (my_rank==0) {
    // read from command line the values of M, N, and K
    // allocate 2D array 'input' with M rows and N columns
    // allocate 2D array 'output' with M-K+1 rows and N-K+1 columns
    // allocate the convolutional kernel with K rows and K columns
    // fill 2D array 'input' with some values
    // fill kernel with some values
    // ....
  }

  // process 0 broadcasts values of M, N, K to all the other processes
  // ...

  if (my_rank>0) {
    // allocated the convolutional kernel with K rows and K columns
    // ...
  }

  // process 0 broadcasts the content of kernel to all the other processes
  // ...

  // parallel computation of a single-layer convolution
  MPI_single_layer_convolution (M, N, input, K, kernel, output);

  if (my_rank==0) {
    // For example, compare the content of array 'output' with that is
    // produced by the sequential function single_layer_convolution
```

```
    // ...
  }

  MPI_Finalize();
  return 0;
}
```

3 MPI programming tasks for IN4200 students (not for IN3200 students)

Task 3.1

Please write a function with the following syntax, which will be invoked inside an MPI program by *all* processes of the default `MPI_COMM_WORLD` communicator. The involved computational work should be appropriately divided among the MPI processes.

```
void MPI_double_layer_convolution (int M, int N, float **input,
                                   int K1, float **kernel1,
                                   int K2, float **kernel2,
                                   float **output);
```

The intended computational result, to be contained inside 2D array `output` of its entirety on process 0, should equal that of applying two single-layered convolutions in succession. The first convolutional kernel is of dimension $K_1 \times K_1$, and the second of dimension $K_2 \times K_2$. Consequently, the dimension of 2D array `output` will be $(M - K_1 - K_2 + 2) \times (N - K_1 - K_2 + 2)$ on process 0.

The same remarks listed in Task 2.1 are also applicable here (except the dimension of 2D array `output` on process 0).

Task 3.2

Please write an MPI-parallelized `main` function using the code skeleton shown in Task 2.2. The only differences are that a second convolutional kernel will also be needed, and that function `MPI_double_layer_convolution` should be called instead of function `MPI_single_layer_convolution` .

4 Submission

Each student should submit a single tarball (`.tar`) or a single zip file (`.zip`). Upon unpacking/unzipping it should produce a folder named `IN3200_HE2_xxx` or `IN4200_HE2_xxx`, where `xxx` is the candidate number. Inside the folder, there should be two `*.c` files (`MPI_single_layer_convolution.c` or `MPI_double_layer_convolution.c` in addition to `MPI_main.c`). These implement, respectively, Tasks 2.1 & 2.2 as required for IN3200 students, and Tasks 3.1 & 3.2 for IN4200 students. There should also be a `README.txt` (or `README.md`) file explaining how the compilation should be done, with additional comments if relevant.

There is no need to submit any report or short note. The implemented code should have comments (where necessary) for explaining the parallelization strategy and performance-wise considerations.

The submission is through the Inspera system. Please see the course's semester webpage for info about the deadline.

In case you don't have access to a computer that has MPI installed, please use a standard Linux server at Ifi (such as `login.ifi.uio.no`). The MPI-capable compiler there is `/usr/lib64/openmpi/bin/mpicc`, and the command for executing a compiled MPI program is `/usr/lib64/openmpi/bin/mpirun`. **Please only run short tests on any of Ifi's Linux servers.**

The grade of the submission will constitute 20% of the final grade of IN3200/IN4200. Grading of the submission will be based on the correctness, conformability (of filenames and function syntax), readability and speed of the implementations.