

## 1. General Remarks

This assignment is an introduction to MPI programming. You will be using the ece mpi cluster composed of 6 nodes:

`en-openmpiXX.ece.cornell.edu`

where `XX = 02, 03, 04, 05, 06, 07` Each node is a Ubuntu single socket server with 16 cores. Servers are connected by a 10GB router.

You will submit your jobs to SLURM scheduler from the head node `en-openmpi00.ece.cornell.edu`.

You should be able to login to any of the nodes using your Cornell netid. When you login your directory will be visible from any of the nodes so there is no need to copy files among the nodes.

**IMPORTANT:** The cluster is for ece5720 work only. You are not allowed to run any jobs on the cluster which are not ece5720 jobs.

## 2. Compile and execute

SLURM requires two files `your_jobname.sub` and `your_jobname.sh`. Examples for these files are given below and also can be found on Canvas.

`your_jobname.sub:`

```
#!/bin/bash
#SBATCH -J mpi_hello           # Job name
#SBATCH -o output/mpi_hello.o%j # stdout output file(%j expands to jobId)
#SBATCH -e output/mpi_hello.e%j # stderr output file(%j expands to jobId)
#SBATCH --nodes=6              # number of nodes requested
#SBATCH --ntasks=12            # number of tasks to be configured for.
#SBATCH --tasks-per-node=2     # number of tasks to run on each node.
#SBATCH --cpus-per-task=1      # number of cpus needed by each task
#SBATCH --get-user-env         # retrieve the users login environment.
#SBATCH -t 00:10:00           # Run time (hh:mm:ss)
#SBATCH --mem-per-cpu=1000     # memory required per allocated CPU
#SBATCH --partition=six       # Which queue it should run on.
```

`/home/your_main_directory/mpi/mpi_hello.sh`

You need to create a subdirectory where your output errors (if any) will be written to. In the case above it is the `output` subdirectory.

`nodes` is the number of servers that you want to use. In our case the maximal number of servers is 6.

`ntasks` is the size returned by `MPI_Comm_size` in `MPI_COMM_WORLD`

`tasks-per-node` times `nodes` should equal `ntasks`

`cpus_per_task` usually we want only one MPI process per core.

`partition` at the moment there are 3 partitions:

- two nodes `en-openmpi02` and `en-openmpi03` which are identical
- gpu nodes `en-openmpi04` to `en-openmpi07` which are identical and equipped with GPUs
- six nodes `en-openmpi02` to `en-openmpi07` which is a default partition (that is if you do not specify a partition the system will assume `six` partition)

You also need to create `your_jobname.sh`: file

```
#!/bin/bash
mpirun -np 12 ./mpi_hello --mca opal_warn_on_missing_libcuda 0
```

Note that `-np xx` (in the example given `xx = 12`) must be equal to `ntasks`.

The flag `--mca opal_warn_on_missing_libcuda 0` discards error related to CUDA code, for this assignment we do not need CUDA support.

Before you submit a job for execution you have to compile it:

```
mpicc mpi_hello.c -o mpi_hello
```

plus all flags that are needed for compilation.

The next step is to submit the job

```
sbatch mpi_hello.sub
```

The system will assign a number to your job. You can find the output in the directory `output` where you identify the output by the number given to the job.

### 3. One sided Jacobi for computing the SVD

For a given  $m \times n$  matrix  $A$ ,  $m \geq n$ , we want to find orthogonal  $m \times m$   $U$  and  $n \times n$   $V$ , and diagonal  $m \times n$   $\Sigma$  such that

$$AV = U\Sigma \tag{1}$$

The equation (1), after transposition, becomes

$$V^T A^T = \Sigma^T U^T \tag{2}$$

We prefer (2) as in that form Givens rotations are applied to rows (in "C" arrays are stored by rows).

We approximate the decomposition (2) by performing so called **sweeps**. A sweep is a product of  $\frac{n(n-1)}{2}$  Givens rotations applied to the working matrices on the left.

A sweep in turn is a product of **compound** rotations. A compound rotation is a product of  $\frac{n}{2}$  independent rotations. These are rotations that operate on pairs of consecutive rows of working matrices.

A general form of a compound rotations for  $n = 8$  is the following

$$J^T = \left( \begin{array}{cc|cc|cc|cc} c_0 & s_0 & & & & & & \\ -s_0 & c_0 & & & & & & \\ \hline & & c_1 & s_1 & & & & \\ & & -s_1 & c_1 & & & & \\ \hline & & & & c_2 & s_2 & & \\ & & & & -s_2 & c_2 & & \\ \hline & & & & & & c_3 & s_3 \\ & & & & & & -s_3 & c_3 \end{array} \right)$$

The parameters  $s_i$  and  $c_i$  are such that  $s_i^2 + c_i^2 = 1$  (thus, these are sines and cosines of a common angle).

We start iteration from  $V^{(0)} = I_n$  and  $A^{(0)} = A^T$  (thus  $A^{(0)}$  is an  $n \times m$  matrix).

Say that after  $k$  sweeps we have matrices  $V^{(k)}$  and  $A^{(k)}$ . We want to get  $V^{(k+1)}$  and  $A^{(k+1)}$ . For that we compute another sweep composed of compound rotations.

Let  $a_i^{(k)}$  denote row  $i$  of  $A^{(k)}$ . For a compound rotation we select pairs of consecutive rows  $a_{2p}^{(k)}$  and  $a_{2p+1}^{(k)}$ ,  $p = 0, \dots, \frac{n}{2} - 2$ , and form a  $2 \times 2$  matrix

$$B = \begin{pmatrix} b_{2p,2p} & b_{2p,2p+1} \\ b_{2p+1,2p} & b_{2p+1,2p+1} \end{pmatrix} = \begin{pmatrix} a_{2p}^{(k)} \\ a_{2p+1}^{(k)} \end{pmatrix} \begin{pmatrix} a_{2p}^{(k)} & a_{2p+1}^{(k)} \end{pmatrix}$$

Thus

$$b_{2p,2p} = a_{2p}^{(k)}(a_{2p}^{(k)})^T, \quad b_{2p,2p+1} = a_{2p}^{(k)}(a_{2p+1}^{(k)})^T = b_{2p+1,2p}, \quad b_{2p+1,2p+1} = a_{2p+1}^{(k)}(a_{2p+1}^{(k)})^T$$

Now we find cosine and sine  $c_p$  and  $s_p$  so  $B$  is diagonalized,

$$\begin{pmatrix} c_p & s_p \\ -s_p & c_p \end{pmatrix}^T \begin{pmatrix} b_{2p,2p} & b_{2p,2p+1} \\ b_{2p+1,2p} & b_{2p+1,2p+1} \end{pmatrix} \underbrace{\begin{pmatrix} c_p & s_p \\ -s_p & c_p \end{pmatrix}}_{G_p} = \begin{pmatrix} \hat{b}_{2p,2p} & 0 \\ 0 & \hat{b}_{2p+1,2p+1} \end{pmatrix}$$

(Formulas for  $c_p$  and  $s_p$  can be found in the notes `jacobi.pdf`.)

Now pairs of rows of  $V^{(k)}$  and  $A^{(k)}$  are updated by multiplying them by Givens matrices  $G_p^T$ . This completes execution of a compound rotation. All PEs must be done before moving to the next step, thus some form of synchronization is needed here.

The next thing we want to do is to permute the working matrix so another set of row pairs is created. This is done by the Brent-Luk scheme. In the scheme, rows are shifted along a ring as illustrated in Figure 1 (and in the notes `jacobi.pdf`).

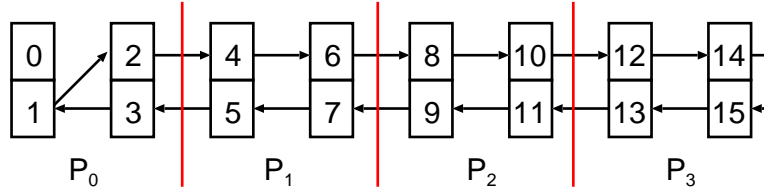


Figure 1.

Shifts are executed by the following pseudo code

```
temp = A[1]; index = 1;

// Shift all elements in bottom row to the left
for (i = 0; i < n/2 - 1; i++) {
    A[index] = A[index + 2];
    index += 2;
}

// Shift the top right to bottom right
A[index] = A[index - 1];
index -= 1;

// Shift top row from left to right
for (i = 0; i < n/2 - 1; i++) {
    A[index] = A[index - 2];
    index -= 2;
}

// Fill in the temp value
A[2] = temp;

// Make sure that the element A[1] moves to A[2]
```

Note that there are "external" swaps of rows among neighboring PEs and, if there are more than 2 rows per PE, there are "internal" swaps of rows within PEs.

All PEs must be done before moving to the next step, thus some form of synchronization is needed here.

After the shifts are executed another compound rotation is generated. This is repeated  $n - 1$  times (as there are  $\frac{N(n-1)}{2}$  rotations in a sweep).

Sweeps are repeated until a (user defined) maximum number of sweeps are executed, or when desired accuracy is achieved.

There are number of ways to check for accuracy. One way is to compute

$$\text{error} = \text{off}(A^{(k)}(A^{(k)})^T)$$

where `off` was defined in Lecture 12 in notes `jacobi.pdf`.

If error is smaller than a desired threshold, the iterations will stop. However the computation of `off` requires matrix-matrix multiplication and is expensive. Rather than performing matrix-matrix multiplication for the entire matrix, one can cheat a bit by performing matrix-matrix multiplication on submatrices that reside in all PEs. The local `offs` can be accumulated by `MPI_Allreduce` on all nodes and compared to the threshold. Also, we will start computing `off` after  $s = \min(8, \log_2(n))$  sweeps have been performed.

At the termination the matrix  $A^{(k)}$  has all rows (approximately) orthogonal to each other. Norms of rows of  $A^{(k)}$  are taken as (approximate) singular values. Rows of  $U^T$  are rows of  $A^{(k)}$  scaled by the corresponding singular values.

The last step is to order singular values and store them in PE 0.

## 5 Format.

We assume a 1D arrangement for  $P$  MPI processes, and a block row distribution of the data matrix  $A$ .

You are asked to run your algorithm on

1. the matrix from `MyMatrix.txt`, please use the code `cread_mat.c` to read the dimensions and entries of the test matrix, the code can be found in the directory `/classes/assignments/hw3`,
2. square matrices generated randomly as explained below.

For randomly generated matrices, your algorithm should accept the following arguments:

- `n` - dimension of the matrix
- `max_sweeps` - maximal number of sweeps allowed
- `tol` - measure of orthogonality of  $A^{(k)}$ .

Populate the  $n \times n$  matrix  $A$  with random numbers using `drand48` random numbers generator.

- Your code must be well documented so any person with limited knowledge of C could understand what the code is doing.
- For a fixed number of PEs you need to run your code for progressively larger matrices.

- For a fixed size of matrix  $A$  you need to run your code for progressively larger number of PEs.
- Your timing data must include at least the following cases:  $n = 512, 1024, P = 8, 32, 64$  (do not include `en-openmpi00.ece.cornell.edu` in your experiments).
- For each case record speed-up (or slow down) over a sequential algorithm ( $P = 1$  and no MPI). If there is a slow down, try to identify the parts of the code that contribute to the slow down.
- Your Jacobi code must be saved in the file `your_netid_hw3.c`. You also need to submit `your_netid_jacobi.sub` and `your_netid_hw3.sh` scripts.
- Write a document that describes how your programs work.
- Sketch the key elements of your internode communication strategy and describe how synchronization is achieved.
- Your findings, a discussion of results, graphs and tables should be saved in a file `your_netid_hw3.pdf`. (NO \*.docx files please).
- Please present your tables and graphs in a way so they are easily readable. Use the scientific format (`x.xxeyy` which show 3 most significant decimal digits and 2 digits of exponent).
- The codes, files `.sh` and `.sub` and the writeup must be archived as `your_netid.zip` or `your_netid.tar` files.
- NOTE: When we unzip your submission we want to see only the files

```
your_netid_hw3.pdf,
your_netid_hw3.c,
your\_netid\_hw3.sh},
your\_netid\_hw3.sub
```

Please **NO** other files or subdirectories. If there are additional files or subdirectories present in the archive, points will be **subtracted** from the final score.

- If you relay on resources outside lecture notes but publicaly available, please cite sources in your write-up.
- You can discuss the homework with your classmates but the final work must be your own (no code sharing).