Assignment MPI: Process Symmetry and Collectives

The purpose of this assignment is for you to learn more about

- getting started with MPI by implementing a hello world
- implementing a static workload partitioning scheme in MPI on numerical integration using a reduction collective,
- data partitioning and complex collective scheme on matrix multiplication

As usual all time measurements are to be performed on the cluster.

To be able to compile and run an MPI program on mamba, **you need to add the line** `module load openmpi` **at the end of the file** `.bashrc` **located in the home directory of your account on mamba**. (log off and back in afterward.)

To compile an MPI application, use the `mpicc` compiler in C and the `mpicxx` compiler in C++. They also serve as linker. To run an MPI application using 19 processes, you can run `mpirun -n 19 ./myprogram`. But you will need to have a proper node allocation first. And if you have a proper node allocation then specifying `-n` is not necessary because the cluster scheduler does that for you.

To queue an MPI job on mamba, you will need to specify how many nodes and how many cores per node you plan on using. You could use `qsub -l nodes=2:ppn=16` to request two nodes with 16 cores each, or `qsub -l procs=32` if you only care about having 32 cores independently of where they are. You can also use `qsub -l nodes=4:ppn=4`, to have 4 cores from 4 different nodes. There is currently a cap on mamba that prevents you from requiring more than 32 processes in total. This is abstracted in the scaffolding.

1 Hello World

This problem is fairly simple. It consists in initializing the MPI application and getting each process to print a message of the form "I am process i out of P. I am running on *machine*.".

Question: Go into the `hello_world/` directory and write the code in `hello.cpp`

Question: Run the code on mamba using `make run_1x16`, `make run_2x8`, `make run_4x8` and confirm that the run happen on different machines by looking at the output generated in the `run.sh.o`xyz files.

2 Numerical Integration : static

Adapt the numerical integration application to make it use MPI in a simple way. The first MPI process should take the first N/P iterations of the loop, the second should take the next N/P iterations of the loop, etc.. The partial integration should be accumulated on rank 0 so that it can print the correct answer to `stdout` and the time it took to `stderr`.

Question: Go into the `num_int/` directory. Write the code in `mpi_num_int.cpp`.

Question: Run and time that program on mamba using many configurations. Use the `make bench` to queue all the jobs. (Note that if you run `make bench` twice, all your previously queued jobs will be deleted.)

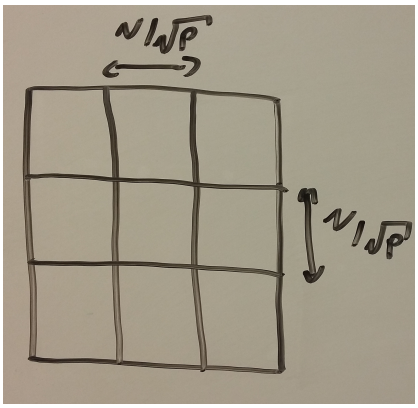
Question: Generate figures of the results using `make plot`. Do you observe speedup higher than can be achieved on a single machine?

3 Matrix Vector multiplication

The problem is to compute iterated matrix multiplication defined by $x^k = Ax^{k-1}$, where A is a given matrix of size $n \times n$ and x^k is a vector of size n. Pick x^0 as given. (There are functions in the scaffolding generating A and x_0.)

For reference, $x^k = Ax^{k-1}$ is computed using $x^k[i] = \sum_j A[i][j]x^{k-1}[j]$. Or in other words, to compute $x^k[i]$ multiply element wise the ith row of the matrix by x^{k-1} and sum the values.

You should partition the data using blocks:



blocks

Assume that the number of processors is a square number, and that n is divisible by the square root of the number of processors.

Question: Implement iterated matrix multiplication for the block decomposition partitioning scheme. Communicators are a particularly efficient way of implementing this block decomposition. Write the code in `matvec/mpi_matmul.cpp`. This file already contains a sequential implementation of matvec to tell you how to generate the matrix and how to test the multiplication works.

For information see

- `http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/`
- `man MPI_Comm_split`
- `man MPI_Comm_free`

Question: Run the code on mamba using `make bench` to queue the jobs. (Note that if you run `make bench` twice, all your previously queued jobs will be deleted.) (Note also that some of the matrices are bigger than can be stored on a single node's memory.)

Question: Generate a time table and plots using `make plot`. Do you observe that the code ran on matrices than can not fit in the memory of a single node? Do you observe performance at scale?