Matlab MPI

Info

Multicore supported functions and operators.

Matlab support many multicore supported functions and operators, eg matrix multiplication. Matlab also support parallel for-loops. Drawback: It runs only on a single computer.

We have now implemented MPI for Matlab on Vilje, Fram, Maur and Idun clusters (see below). For non MPI programmers; you can see here.

All source codes are open and free, but with NTNU copy right. (Note! Matlab is not free and you need a license).

You need an account on Fram and Vilje. Send an application to Sigma2.

How to use and login to Vilje and Fram.

Support for Matlab MPI: john.floan@ntnu.no

Matlab MPI

Matlab MPI on Vilje, Fram, Maur and Idun are implemented with mex compiled c programs, with standard MPI calls (MPT MPI, IntelMPI and OpenMPI).

Matlab MPI is for running on serveral compute node. The communication between the compute nodes are via MPI calls (Message Passing Interface).

This is a Open Source code.

For non MPI programmers; you can see here.

All MPI Matlab functions have prefix NMPI_

Note! All arrays must be one dimensional.

Imlemented MPI functions are:

NMPI_Comm_size
NMPI_Comm_rank
NMPI_Init
NMPI_Finalize
NMPI_Send
NMPI_Isend
NMPI_Recv
NMPI_Sendrecv
NMPI_Scatter
NMPI_Gather
NMPI_Bcast
NMPI_Barrier
NMPI_Reduce

NMPI_Allreduce

How to use

Matlab job script (job.sh):

Example (myprogram.m): Submit 4 compute nodes and 1 MPI process each node.

FRAM:

```
#!/bin/bash
#SBATCH --account=myaccount
#SBATCH --job-name=jobname
#SBATCH --time=0:30:0
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
module purge
module load intel/2017b
module load MATLAB/2019a
module list
srun --mpi=pmi2 matlab -nodisplay -nodesktop -nojvm -r "myprogram"
```

SAGA:

```
#!/bin/bash
#SBATCH --account=myaccount
#SBATCH --job-name=jobname
#SBATCH --time=0:30:0
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --mem=185G
module purge
module load intel/2019a
module load MATLAB/2019a
module list
srun --mpi=pmi2 matlab -nodisplay -nodesktop -nojvm -r "myprogram"
```

VILJE:

```
#!/bin/bash
#PBS -N jobname
#PBS -l walltime=00:30:00
#PBS -l select=4:ncpus=32:mpiprocs=1:ompthreads=16
#PBS -q workq
#PBS -A myaccount

cd $PBS_O_WORKDIR
module load matlab/R2014a
module load intelcomp
module load mpt
mpiexec_mpt -n 2 matlab -nodisplay -nodesktop -nojvm -r "myprogram"
```

IDUN:

```
#!/bin/bash
#SBATCH -J job                    # Sensible name for the job
#SBATCH -N 2                        # Allocate 2 nodes for the job
#SBATCH --ntasks-per-node=20        # 20 ranks/tasks per node (see example: job script)
#SBATCH -t 00:10:00                 # Upper time limit for the job (HH:MM:SS)
#SBATCH -p WORKQ

module load intel/2017b
module load MATLAB/2019a
mpirun matlab -nodisplay -nodesktop -nosplash -nojvm -r "myprogram"
```

Matlab Code:

Init and Finalize

Matlab script: Hello world from each rank.

```
% For Fram only(R2019a): Add link to MPI libraries
addpath('/cluster/software/MATLAB/2019a/NMPI/version13_intel2017b/');

% For Saga only(R2019a): Add link to MPI libraries
addpath('/cluster/software/MATLAB/2019a/NMPI/version13_intel2019a/');

% For Idun only (R2019a):
addpath('/lustre1/apps/software/Core/MATLAB/2019a/NMPI/version13_intel17b');

% For Vilje only (R2017a):
addpath('/sw/sdev/Modules/matlab/R2017a/NMPI');

NMPI_Init(); % Init the MPI communication between MPI processes

num_ranks = NMPI_Comm_size(); % Get number of MPI processes.
my_rank   = NMPI_Comm_rank(); % Get my MPI process ID (from 0 to num_ranks-1)

display(['Hello world from rank ',num2str(my_rank), ' of total ', num2str(num_ranks)]);

NMPI_Finalize(); % End the MPI communication
```

Output (eg. 2 ranks):

Hello world from rank 0 of total 2

Hello world from rank 1 of total 2

Send and Receive (Point to point communication)

With NMPI_Send and NMPI_Recv; you can send and receive an one dimentional array between the Compute Nodes. Both Send and Recv are blocking operation.

Synopsis:

NMPI_Send (buffer , size_of_buffer , destination);

buffer = NMPI_Recv (size_of_buffer , source);

Example code: Rank 0 send to Rank 1

```
...
n=3;m=3; % Size of the matrix
if my_rank==0
   dest=1;                                        % Destination rank
   data=rand(n,m);                           % Create a 2 dim array
   d=reshape(data,n*m,1);         % Reshape the 2 dim array to one dim
   NMPI_Send(d,n*m,dest);   % Send array to rank 1
else
   source=0;                                   % Source rank
   d=NMPI_Recv(n*m,source); % Receive array from rank 0
   data = reshape(d,[n,m]); % Reshape the received one dim array to two dim array.
end
...
```

Sendrecv (NMPI_Sendrecv is a non blocking operation)

Synopsis:

recvbuffer = NMPI_Sendrecv (sendbuffer, size_of_sendbuffer , dest , size_of_recvbuffer , source);

Gather and Scatter (Collective communcation)

Scatter: Root sending array to all ranks and the receiver receive a chunks of the array.

Gather: All ranks sending its chunked array to the root rank, which gather this to one array.

Example Scatter (4 ranks)

Sending buffer (root): (1, 2 , 3 , 4 , 5 , 6 , 7 , 8)

Receiving buffer : rank 0: (1,2) rank 1: (3,4) rank 2: (5,6) rank 3: (7,8)

Example Gather (4 ranks)

Sending buffer : rank 0: (1,2) rank 1: (3,4) rank 2: (5,6) rank 3: (7,8)

Receiving buffer (root): (1, 2 , 3 , 4 , 5 , 6 , 7 , 8)

Synopsis:

local_buffer = NMPI_Scatter(buffer , size_of_local_buffer , size_of_local_buffer , root);

buffer = NMPI_Gather (local_buffer , size_of_local_buffer , size_of_buffer , root);

size_of_buffer = size_of_local_buffer * num_ranks;

Scatter: root is the sender rank, all other receiving from the root.

Gather: root is the receiver rank, all other sending to the root.

Gatherv and Scatterv (Collective communication)

Scatterv: Root sending array to all ranks (stride lenght) and the receiver receive a chunks of the array.

Gatherv: All ranks sending its chunked array to the root rank, which gather this to one array (in stride length).

(Stride must be equal or larger than receive buffer size)

Example Scatterv (4 ranks)

Sending buffer size = stride * num_ranks;

Example: stride=3, receive_buffer_size = 2, send_buffer_size = stride * num_ranks = 3 * 4 = 12

Sending buffer (root): (1, 2 , 0, 3 , 4 , 0 , 5 , 6 , 0 , 8 , 9, 0)

Receiving buffer : rank 0: (1,2), rank 1: (3,4), rank 2: (5,6), rank 3: (7,8)

Example Gatherv (4 ranks)

Receiving buffer size = stride * num_ranks;

Example: stride=3, sending_buffer_size = 2, receive_buffer_size = stride * num_ranks = 3 * 4 = 12

Sending buffer : rank 0: (1,2), rank 1: (3,4), rank 2: (5,6), rank 3: (7,8)

Receiving buffer (root): (1, 2 , 0, 3 , 4 , 0 , 5 , 6 , 0 , 7 , 8, 0)

```
NMPI_Scatterv:
root=0;
stride=11;
local_buffer_size=10;
buffer_size=stride*num_ranks;
buffer=rand(buffer_size,1);
local_buffer=NMPI_Scatterv( buffer, buffer_size, stride, local_buffer_size, root);

NMPI_Gatherv:
root=0;
stride=11;
local_buffer_size=10;
local_buffer=rand(local_buffer_size,1);
buffer_size=stride*num_ranks;
buffer=NMPI_Gatherv( local_buffer, local_buffer_size, buffer_size, stride root);
```

Synopsis:

local_buffer = NMPI_Scatterv (buffer , buffer_size, stride, local_buffer_size , root);

buffer = NMPI_Gatherv (local_buffer , local_buffer_size, buffer_size, stride , root);

size_of_buffer = stride * num_ranks; size_of_local buffer is <= stride

Scatter: root is the sending rank, all other receiving from the root.

Gather: root is the receiving rank, all other sending to the root.

Broadcast (Collective communication)

The root-rank broadcast a buffer to all ranks.

Synopsis:

buffer=NMPI_Bcast(buffer , size_of_buffer , root);

root is the sender rank.

Reduce (Collective communication)

Reduce the content of sending buffer (element for element) for all ranks, to the root-rank.

Synopis:

buffer_out = NMPI_Reduce (buffer_in , size_buffer , operator , root);

operator: Summation : '+', Multiplication : '*', Maximum: 'M', Minimum : 'N', logical AND: '&', logical OR: '|'.

Note! Max and Min returns a variable with max or min value.

root is the receiving rank.

Allreduce (Collective communication)

Reduce the content of sending buffer (element for element) for all ranks, result back to all ranks.

Synopis:

buffer_out = NMPI_Allreduce (buffer_in , size_buffer , operator);

operator: Summation : '+', Multiplication : '*', Maximum: 'M', Minimum : 'N', logical AND: '&', logical OR: '|'.

Note! Max and Min returns a variable with max or min value.

Barrier (Synchronization)

NMPI_Barrier block all processes, to all MPI ranks have called the MPI_Barrier.

Synopsis:

NMPI_Barrier();