Discussion: MPI Parallel Programming

- Running MPI on Expanse
- Distribute data to processes and gather data from processes
- Parallel matrix multiplication with MPI

Run MPI on a CSIL machine and on Expanse

https://sites.cs.ucsb.edu/~tyang_class/140w26/install_mpi. html

- You can use your laptop or CSIL machine to quickly test and debug your MPI programs.

To compile for PA1 Q2 code:
make -f Makefile-CSIL

To run on a CSIL machine
make -f Makefile-CSIL run-mv_mult_test_mpi

- For reporting, run the job on an Expanse machine

make
make run-mv_mult_test_mpi

Compilation of MPI C code at CSIL

Install the MPI package by yourself or use pre-installed MPI

• In the attached makefile with PA1 called Makefile-CSIL, the path location prefix of the MPI package is:

MPI_PREFIX = ~tyang_class/local/bin/

- make -f Makefile-CSIL run-mv_mult_test_mpi

•Running result:

Test 1: Wall clock time = 0.000035 at Proc 0 of 2 processes
Test 2: Wall clock time = 0.001825 at Proc 0 of 2 processes
Summary: Failed 0 out of 2 tests

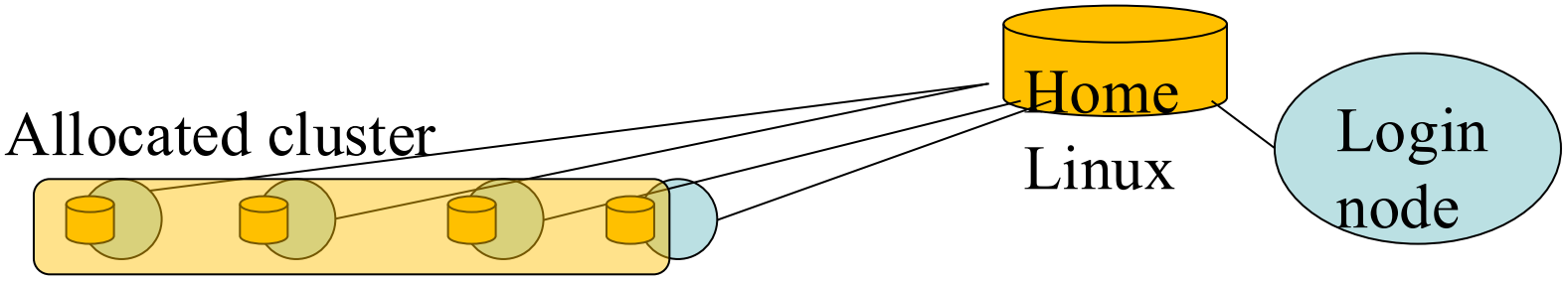
•mpicc and mpirun in ~tyang_class/local/bin/ are used to compile and run MPI programs.

Running a parallel job at Expanse

- ***To get into a login node***

ssh username@login.expanse.sdsc.edu

- ***Compile with make***
- ***Submit a job that runs a parallel job***



- Expanse cluster has 732 nodes and each node has 128 cores with AMD EPYC 7742 processors
- 256 GB memory and 1TB SSD for local scratch space.
- Attached storage: 12 petabytes of 100-200 GB/second storage

Compilation of MPI C code at Expanse

wrapper script to compile

source file

```
mpicc  -O  -o hello_mpi  mpi_hello.c
```

Compiler option

-O or –g -Wall

create this executable file name

(as opposed to default a.out)

```
sbatch -v run-hello.sh
```

Submit a parallel

job using a shell script

Job script that runs an MPI program at Expanse

```
#!/bin/bash
#SBATCH --job-name="hellompi"
#SBATCH --output="hellompi.%j.%N.out"
#SBATCH --partition=shared
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --export=ALL
#SBATCH -t 00:01:00
#SBATCH --account=csb175

module purge
module load slurm
module load cpu
module load gcc
module load openmpi
module load sdsc
ibrun -v ../hello_mpi
```

Job name you can observe when querying status

Job id in the submitted queue

Which cluster to run this program

This job runs with 2 nodes, 2 cores per node for a total of 4 cores.

Set a time limit that this job runs at most 1 minute.

ibrun in –v verbose mode will give binding detail in running the hello mpi binary

Useful Commands at Expanse

- **How to check account balance?**
  - module load sdsc
  - expanse-client user -r expanse
  - Hours shown are shared among all users in cs140!
  - Account is billed based on the nearest CPU node hour.
  - Easy to use all hours with a deadlock job. Thus set the maximum time usage: 1-2 mins.
- Use the squeue -u command to **check the job status.** Use the scancel command to cancel a job.
- **allocate a single node to execute some compute-intensive job interactively.**
- srun --export=ALL --partition=debug --account=csb175 --pty --nodes=1 --ntasks-per-node=1 -t 0:30:00 /bin/bash
  - When done, DONOT forget Ctrl-D or Cmd-D to exit.

Parallel Matrix Vector Multiplication

Collective Communication Application

Matrix-vector multiplication: y= A * x

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 1*1+2*2+3*3 \\ 4*1+5*2+6*3 \\ 7*1+8*2+9*3 \end{pmatrix} = \begin{pmatrix} 14 \\ 32 \\ 50 \end{pmatrix}$$

Problem: $y = A * x$ where A is a $n \times n$ matrix and x is a column vector of dimension n.

Sequential code:

```
for i = 1 to n do
    y_i = 0;
    for j = 1 to n do
        y_i = y_i + a_{i,j} * x_j;
    endfor
endfor
```

Partitioning and Task graph for matrix-vector multiplication

Partitioned code:

for $i = 1$ **to** n **do**

S_i : $y_i = 0;$

for $j = 1$ to n **do**

$y_i = y_i + a_{i,j} * x_j;$

endfor

endfor

S_i : Read row A_i and vector x.

Write element y_i

*yi= Row Ai * Vector x*

Task graph:

Execution Schedule and Task Mapping

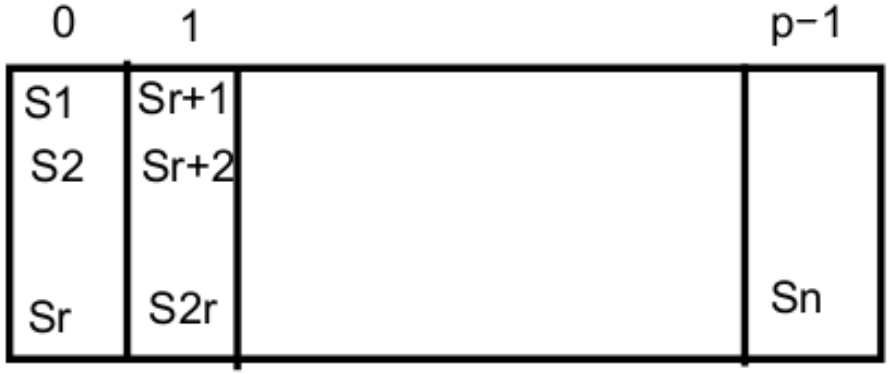
S_i : Read row A_i and vector x.

Write element y_i

*yi= Row Ai * Vector x*

Task graph:

Schedule:



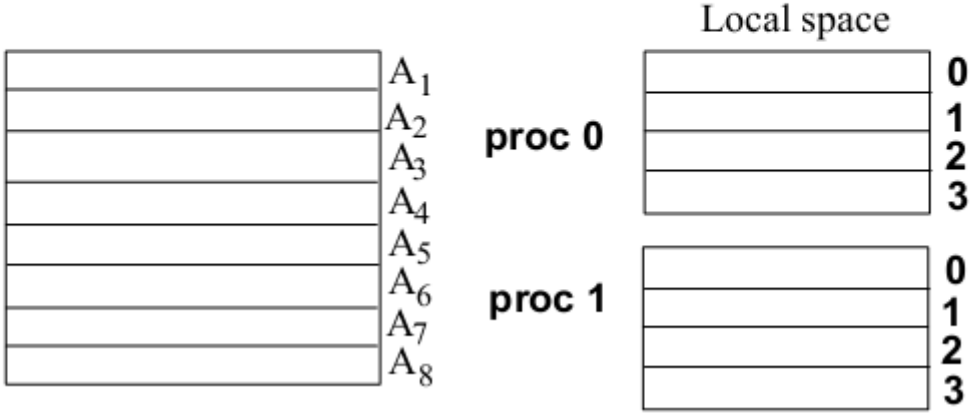
Mapping function of tasks S_i:

$proc\_map(i) = \lfloor \frac{i-1}{r} \rfloor$ where $r = \lceil \frac{n}{p} \rceil$.

Data Partitioning and Mapping for y= A*x

Data partitioning: for the above schedule:

Matrix A is divided into n rows $A_1, A_2, \cdots A_n$.



Data mapping:

Row A_i is mapped to processor $proc\_map(i)$, the same as task i. The indexing function is: $local(i) = (i-1) \bmod r$. Vectors x and y are replicated to all processors.

SPMD Code for y= A*x

int x[n], y[n], a[r][n];

me=mynode();

for $i = 1$ **to** n **do**

if $proc\_map(i) == me$, then do S_i:

S_i : $y[i] = 0;$

for $j = 1$ to n **do**

$y[i] = y[i] + a[local(i)][j] * x[j];$

endfor

endfor

Evaluation: Parallel Time

- Ignore the cost of local address calculation.

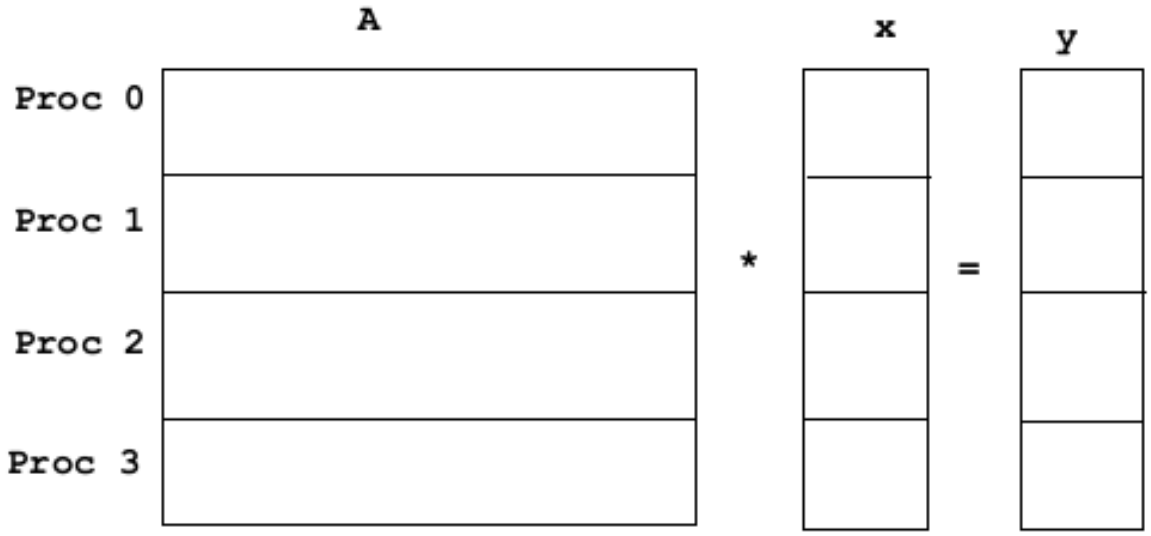
- Each task performs n additions and n multiplications.

- Each addition/multiplication costs ω

- The parallel time is approximately $\frac{n}{p} \times 2n\omega$

How is initial data distributed?

Assume initially matrix A and vector x are distributed evenly among processes

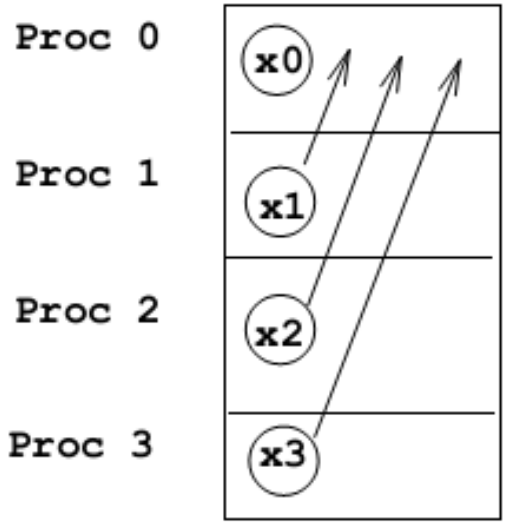


Need to redistribute vector *x* to everybody in order to perform parallel computation!

What MPI collective communication is needed?

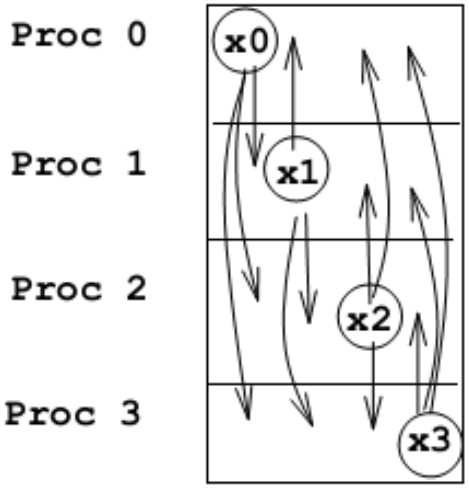
Communication Pattern for Data Redistribution

Data requirement for Process 0



MPI_Gather

Data requirement for all processes



MPI_Allgather

MPI Code for Gathering Data

Data gather for Process 0

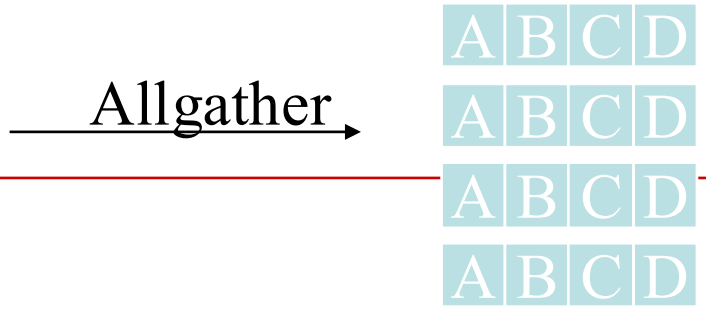
```
float local_x[]; /*local storage for x*/
float global_x[]; /*storage for all of x*/

MPI_Gather(local_x, n/p, MPI_FLOAT,
           global_x, n/p, MPI_FLOAT,
           0, MPI_COMM_WORLD);
```

Repeat for all processes

It is the same as:

```
MPI_All_gather(local_x, n/p, MPI_FLOAT,
               global_x, n/p, MPI_FLOAT,
               MPI_COMM_WORLD);
```

Allgather

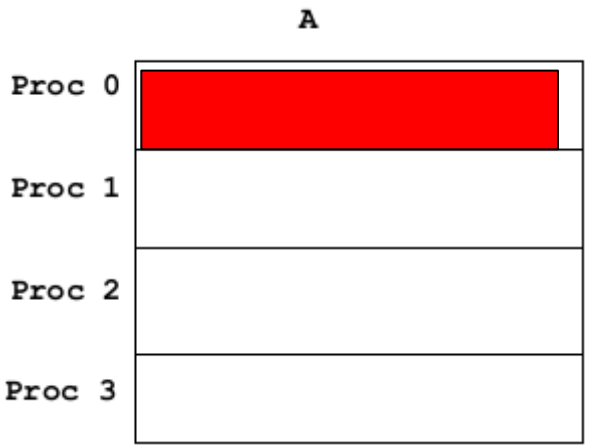


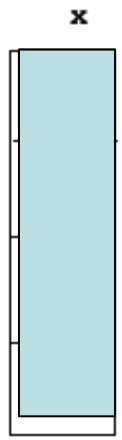
- Concatenates the contents of each process' send_buf_p and stores this in each process' recv_buf_p.
- As usual, recv_count is the amount of data being received from each process.

```
int MPI_Allgather(
      void*          send_buf_p  /* in  */,
      int            send_count  /* in  */,
      MPI_Datatype   send_type   /* in  */,
      void*          recv_buf_p  /* out */,
      int            recv_count  /* in  */,
      MPI_Datatype   recv_type   /* in  */,
      MPI_Comm       comm        /* in  */);
```

MPI SPMD Code for y=A*x

```
void Parallel_matrix_vector_prod(
        LOCAL_MATRIX_T  local_A
        int             m
        int             n
        float           local_x[]
        float           global_x[]
        float           local_y[]
        int             local_m
        int             local_n) {
 /* local_m = n/p, local_n = n/p */
 MPI_Allgather(local_x, local_n, MPI_FLOAT
               global_x, local_n, MPI_FLOAT,
                 MPI_COMM_WORLD);
```





MPI SPMD Code for y=A*x

```
    for (i = 0; i < local_m; i++) {
         local_y[i] = 0.0;
         for (j = 0; j < n; j++)
           local_y[i] = local_y[i] +
                local_A[i][j]*global_x[j];
     }
}
```

Text book solution for y=A*x

Page 114 of "An Introduction to Parallel Programming" by Peter Pacheco, 2011

$A = (a_{ij})$ is an $m \times n$ matrix

x is a vector with n components

a_{00}	a_{01}	$\cdots$	$a_{0,n-1}$
a_{10}	a_{11}	$\cdots$	$a_{1,n-1}$
$\vdots$	$\vdots$		$\vdots$
a_{i0}	a_{i1}	$\cdots$	$a_{i,n-1}$
$\vdots$	$\vdots$		$\vdots$
$a_{m-1,0}$	$a_{m-1,1}$	$\cdots$	$a_{m-1,n-1}$

x_0
x_1
$\vdots$
x_{n-1}

$=$

y_0
y_1
$\vdots$
$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1}$
$\vdots$
y_{m-1}

$$y_i = a_{i0}x_0 + a_{i1}x_1 + a_{i2}x_2 + \cdots a_{i,n-1}x_{n-1}$$

i-th component of y

Dot product of the ith row of A with x.

Use one dimensional C array to represent 2D matrix

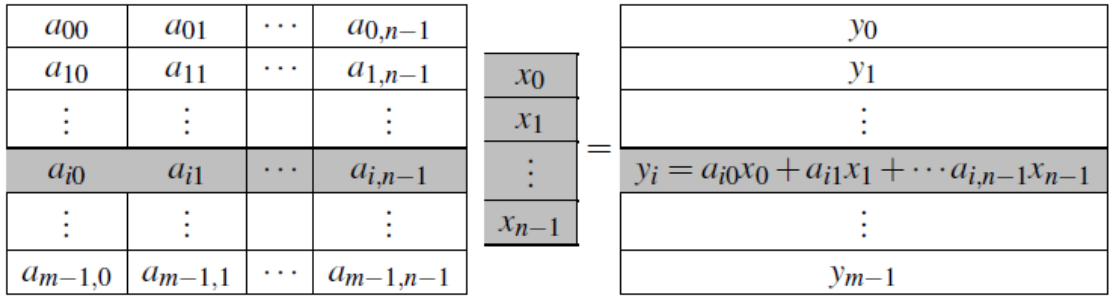
$A = (a_{ij})$ is an $m \times n$ matrix

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \end{pmatrix}$$

stored as

0 1 2 3 4 5 6 7 8 9 10 11

Sequential code for y=A*x



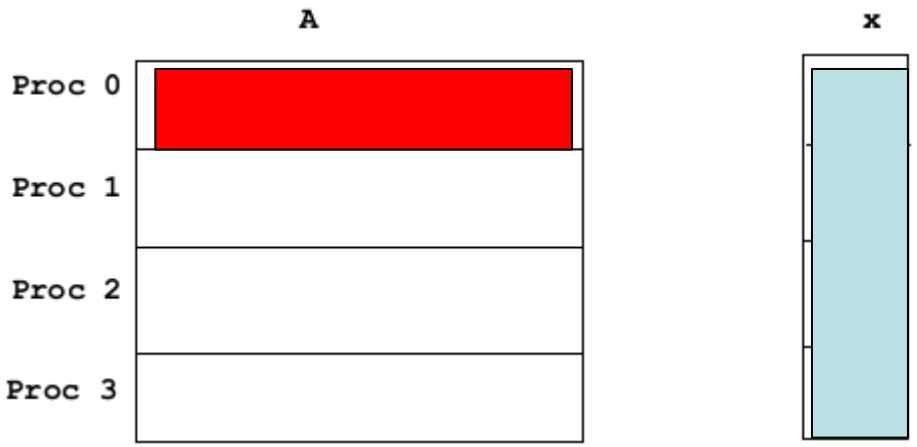

```
void Mat_vect_mult(
        double  A[]   /* in  */,
        double  x[]   /* in  */,
        double  y[]   /* out */,
        int     m     /* in  */,
        int     n     /* in  */) {
    int i, j;

    for (i = 0; i < m; i++) {
        y[i] = 0.0;
        for (j = 0; j < n; j++)
            y[i] += A[i*n+j]*x[j];
    }
}  /* Mat_vect_mult */
```

Textbook MPI code for matrix-vector multiplication

```
void Mat_vect_mult(
      double     local_A[]  /* in  */,
      double     local_x[]  /* in  */,
      double     local_y[]  /* out */,
      int        local_m    /* in  */,
      int        n          /* in  */,
      int        local_n    /* in  */,
      MPI_Comm   comm       /* in  */) {
   double* x;
   int local_i, j;
   int local_ok = 1;
```

Textbook MPI code for y=A*x

```
x = malloc(n*sizeof(double));
MPI_Allgather(local_x, local_n, MPI_DOUBLE,
      x, local_n, MPI_DOUBLE, comm);

for (local_i = 0; local_i < local_m; local_i++) {
   local_y[local_i] = 0.0;
   for (j = 0; j < n; j++)
      local_y[local_i] += local_A[local_i*n+j]*x[j];
}
free(x);
}
```



Speedups of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5

Efficiencies of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.00	1.00	1.00	1.00	1.00
2	0.89	0.94	0.97	0.96	0.98
4	0.51	0.78	0.89	0.96	0.98
8	0.30	0.61	0.82	0.94	0.98
16	0.15	0.39	0.68	0.89	0.97