MPI Workshop - II

Research Staff

Week 2 of 2
Today’s Topics

- Course Map
- Basic Collective Communications
  - MPI_Barrier
  - MPI_Scatterv, MPI_Gatherv, MPI_Reduce
- MPI Routines/Exercises
  - Pi, Matrix-Matrix mult., Vector-Matrix mult.
- Other Collective Calls
- References
# Course Map

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Week 1</strong></td>
<td><strong>Point to Point</strong></td>
<td><strong>Collective</strong></td>
<td><strong>Advanced</strong></td>
</tr>
<tr>
<td><strong>Basic Collective</strong></td>
<td><strong>Fundamental Routines</strong></td>
<td><strong>Communications</strong></td>
<td><strong>Topics</strong></td>
</tr>
<tr>
<td><strong>MPI functional routines</strong></td>
<td><strong>MPI_SEND (MPI_ISEND)</strong></td>
<td><strong>MPI_BCAST</strong></td>
<td><strong>MPI_DATATYPE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MPI_RECV (MPI_Irecv)</strong></td>
<td><strong>MPI_SCATTERV</strong></td>
<td><strong>MPI_HVECTOR</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MPI_BCAST</strong></td>
<td><strong>MPI_GATHERV</strong></td>
<td><strong>MPI_VECTOR</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MPI_SCATTER</strong></td>
<td><strong>MPI_REDUCE</strong></td>
<td><strong>MPI_STRUCT</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MPI_GATHER</strong></td>
<td><strong>MPI_BARRIER</strong></td>
<td><strong>MPI_CART_CREATE</strong></td>
</tr>
<tr>
<td><strong>Week 2</strong></td>
<td><strong>Collective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Topics</strong></td>
<td></td>
<td><strong>MPI Examples</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MPI Examples</strong></td>
<td><strong>Helloworld</strong></td>
<td><strong>Pi</strong></td>
<td><strong>Poisson Equation</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Swapmessage</strong></td>
<td><strong>Matrix/vector</strong></td>
<td><strong>Passing Structures</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Vector Sum</strong></td>
<td><strong>multiplication</strong></td>
<td><strong>/common blocks</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Matrix/matrix</strong></td>
<td><strong>Parallel topologies</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>multiplication</strong></td>
<td><strong>in MPI</strong></td>
</tr>
</tbody>
</table>
Example 1 - Pi Calculation

\[ \pi = \int_{0}^{1} \frac{4}{1 + x^2} \, dx \]

Uses the following MPI calls:

MPI_BARRIER, MPI_BCAST, MPI_REDUCE
Integration Domain: Serial

\[ f(x) \]

\[ x_0 \quad x_1 \quad x_2 \quad x_3 \quad x \quad x_N \]
Serial Pseudocode

\[ f(x) = \frac{1}{1+x^2} \]

Example:

\[ h = \frac{1}{N}, \text{sum} = 0.0 \]

\[ N = 10, h = 0.1 \]

\[ x = \{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95\} \]

\[ \text{do i = 1, N} \]

\[ x = h(i - 0.5) \]

\[ \text{sum = sum + f(x)} \]

\[ \text{enddo} \]

\[ \pi = h \times \text{sum} \]
Parallel Pseudocode

P(0) reads in N and Broadcasts N to each processor

\[ f(x) = \frac{1}{1+x^2} \]

\[ h = \frac{1}{N}, \text{ sum } = 0.0 \]

do i = rank+1, N, nprocrs

\[ x = h*(i - 0.5) \]

\[ \text{sum} = \text{sum} + f(x) \]

enddo

\[ \text{mypi} = h * \text{sum} \]

Collect (Reduce) mypi from each processor into a collective value of pi on the output processor

Example:

\[ N = 10, h=0.1 \]

Procrs: \{P(0),P(1),P(2)\}

P(0) -> \{.05, .35, .65, .95\}

P(1) -> \{.15, .45, .75\}

P(2) -> \{.25, .55, .85\}
Collective Communications - Synchronization

- Collective calls can (but are not required to) return as soon as their participation in a collective call is complete.
- Return from a call does NOT indicate that other processes have completed their part in the communication.
- Occasionally, it is necessary to force the synchronization of processes.
- MPI_BARRIER
Collective Communications - Broadcast

MPI_BCAST
Collective Communications - Reduction

- **MPI_REDUCE**
  - MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN, MPI_IAND, MPI_BAND,...
Example 2: Matrix Multiplication (Easy) in C

\[ C = AB \]

Two versions depending on whether or not the # rows of C and A are evenly divisible by the number of processes.

Uses the following MPI calls:

MPI_BCAST, MPI_BARRIER, MPI_SCATTERV, MPI_GATHERV
Serial Code in C/C++

for(i=0; i<nrow_c; i++)
  for(j=0; j<ncol_c; j++)
    c[i][j]=0.0e0;

for(i=0; i<nrow_c; i++)
  for(k=0; k<ncol_a; k++)
    for(j=0; j<ncol_c; j++)
      c[i][j]+=a[i][k]*b[k][j];

Note that all the arrays accessed in row major order. Hence, it makes sense to distribute the arrays by rows.
Matrix Multiplication in C
Parallel Example

\[ C = A \times B \]
Collective Communications - Scatter/Gather

MPI_GATHER, MPI_SCATTER, MPI_GATHERV, MPI_SCATTERV
Flavors of Scatter/Gather

- Equal-sized pieces of data distributed to each processor
  - MPI_SCATTER, MPI_GATHER

- Unequal-sized pieces of data distributed
  - MPI_SCATTERV, MPI_GATHERV
  - Must specify arrays of sizes of data and their displacements from the start of the data to be distributed or collected.
  - Both of these arrays are of length equal to the size of communications group
Scatter/Scatterv Calling Syntax

int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

int MPI_Scatterv(void *sendbuf, int *sendcounts, int *offsets, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
Abbreviated Parallel Code (Equal size)

```c
ierr=MPI_Scatter(*a,nrow_a*ncol_a/size,...);
ierr=MPI_Bcast(*b,nrow_b*ncol_b,...);
for(i=0; i<nrow_c/size; i++)
    for(j=0; j<ncol_c; j++)
        cpart[i][j]=0.0e0;
for(i=0; i<nrow_c/size; i++)
    for(k=0; k<ncol_a; k++)
        for(j=0; j<ncol_c; j++)
            cpart[i][j]+=apart[i][k]*b[k][j];
ierr=MPI_Gather(*cpart,(nrow_c/size)*ncol_c, ...);
```
Abbreviated Parallel Code (Unequal)

```c
ierr=MPI_Scatterv(*a,a_chunk_sizes,a_offsets,...);
ierr=MPI_Bcast(*b,nrow_b*ncol_b, ...);
for(i=0; i<c_chunk_sizes[rank]/ncol_c; i++)
    for(j=0;j<ncol_c; j++)
        cpart[i][j]=0.0e0;
for(i=0; i<c_chunk_sizes[rank]/ncol_c; i++)
    for(k=0; k<ncol_a; k++)
        for(j=0;j<ncol_c; j++)
            cpart[i][j]+=apart[i][k]*b[k][j];
ierr=MPI_Gatherv(*cpart, c_chunk_sizes[rank], MPI_DOUBLE, ...);
```

Look at C code to see how sizes and offsets are done.
Fortran version

- F77 - no dynamic memory allocation.
- F90 - allocatable arrays, arrays allocated in contiguous memory.
- Multi-dimensional arrays are stored in memory in column major order.

Questions for the student.
- How should we distribute the data in this case? What about loop ordering?
- We never distributed B matrix. What if B is large?
Example 3: Vector Matrix Product in C

Illustrates MPI_Scatterv, MPI_Reduce, MPI_Bcast

\[ C = A \times B \]
Main part of parallel code

```c
ierr=MPI_Scatterv(a,a_chunk_sizes,a_offsets,MPI_DOUBLE,
               apart,a_chunk_sizes[rank],MPI_DOUBLE,
               root, MPI_COMM_WORLD);
ierr=MPI_Scatterv(btmp,b_chunk_sizes,b_offsets,MPI_DOUBLE,
                 bparttmp,b_chunk_sizes[rank],MPI_DOUBLE,
                 root, MPI_COMM_WORLD);

... initialize cpart to zero ...
for(k=0; k<a_chunk_sizes[rank]; k++)
  for(j=0; j<ncol_c; j++)
    cpart[j]+=apart[k]*bpart[k][j];
ierr=MPI_Reduce(cpart, c, ncol_c, MPI_DOUBLE, MPI_SUM, root,
                 MPI_COMM_WORLD);
```
Collective Communications - Allgather

MPI_ALLGATHER
Collective Communications - Alltoall

MPI_ALLTOALL
References - MPI Tutorial

CS471 Class Web Site - Andy Pineda
http://www.arc.unm.edu/~acpineda/CS471/HTML/CS471.html

MHPCC
http://www.mhpcc.edu/training/workshop/html/mpi/MPIIIntro.html

Edinburgh Parallel Computing Center

Cornell Theory Center
http://www.tc.cornell.edu/Edu/Talks/topic.html#mess
References - IBM Parallel Environment

- **POE - Parallel Operating Environment**

- **Loadleveler**
  - [http://www.mhpcc.edu/training/workshop/html/loadleveler/LoadLeveler.html](http://www.mhpcc.edu/training/workshop/html/loadleveler/LoadLeveler.html)
Exercise: Vector Matrix Product in C

Rewrite Example 3 to perform the vector matrix product as shown.

\[ C = A \ast B \]