Parallel Programming Basic MPI

Timothy H. Kaiser, Ph.D.
tkaiser@mines.edu

Talk Overview

- Background on MPI
- Documentation
- Hello world in MPI
- Basic communications
- Simple send and receive program

Examples at

http://geco.mines.edu/workshop

or enter the commands:

```
mkdir examples
cd examples
wget http://hpc.mines.edu/examples/examples.tgz
```

or on Mc2 or AuN

```
cp /opt/utility/examples/* .
```

Background on MPI

- MPI - Message Passing Interface
  - Library standard defined by a committee of vendors, implementers, & parallel programmers
  - Used to create parallel programs based on message passing
- Portable: one standard, many implementations
- Available on almost all parallel machines in C and Fortran
- Over 100 advanced routines but 6 basic

Documentation

- MPI home page (contains the library standard): www.mcs.anl.gov/mpi
- Books
    - "MPI: The Complete Reference" by Snir, Otto, Huss-Lederman, Walker, and Dongarra, MIT Press (also in Postscript and html)
    - "Using MPI" by Gropp, Lusk and Skjellum, MIT Press
- Tutorials
- many online, just do a search

MPI Implementations

- Most parallel supercomputer vendors provide optimized implementations
- LAM
    - www.lam-mpi.org (deprecated)
- OpenMPI
    - www.open-mpi.org (default on Mio and RA)

MPI Implementations

- MPICH:
  - http://www-unix.mcs.anl.gov/mpi/mpich1/download.html
  - http://www.mcs.anl.gov/research/projects/mpich2/index.php
- MVAPICH & MVAPICH2
  - Infiniband optimized version of MPICH
  - http://mvapich.cse.ohio-state.edu/index.shtml

Key Concepts of MPI

- Used to create parallel programs based on message passing
  - Normally the same program is running on several different processors
  - Processors communicate using message passing
- Typical methodology:

```
start job on n processors
do i=1 to j
    each processor does some calculation
    pass messages between processor
end do
end job
```

Messages

- Simplest message: an array of data of one type.
- Predefined types correspond to commonly used types in a given language
  - MPI_REAL (Fortran), MPI_FLOAT (C)
  - MPI_DOUBLE_PRECISION (Fortran), MPI_DOUBLE (C)
  - MPI_INTEGER (Fortran), MPI_INT (C)
- User can define more complex types and send packages.

Communicators

- Communicator
  - A collection of processors working on some part of a parallel job
  - Used as a parameter for most MPI calls
  - MPI_COMM_WORLD includes all of the processors in your job
  - Processors within a communicator are assigned numbers (ranks) 0 to n-1
  - Can create subsets of MPI_COMM_WORLD

Include files

- The MPI include file
  - C: mpi.h
  - Fortran: mpif.h (a f90 module is a good place for this)
- Defines many constants used within MPI programs
- In C defines the interfaces for the functions
- Compilers know where to find the include files

Minimal MPI program

- Every MPI program needs these…
  - C version

```
/* the mpi include file */
#include <mpi.h>
   int nPEs,ierr,iam;
/* Initialize MPI */
   ierr=MPI_Init(&argc, &argv);
/* How many processors (nPEs) are there?*/
   ierr=MPI_Comm_size(MPI_COMM_WORLD, &nPEs);
/* What processor am I (what is my rank)? */
   ierr=MPI_Comm_rank(MPI_COMM_WORLD, &iam);
...
   ierr=MPI_Finalize();
```

In C MPI routines are functions and return an error value

Minimal MPI program

- Every MPI program needs these…
  - Fortran version

```
! MPI include file
      include 'mpif.h'
! The mpi module can be used for Fortran 90 instead of mpif.h
!     use mpi
      integer nPEs, ierr, iam
!  Initialize MPI
      call MPI_Init(ierr)
!  How many processors (nPEs) are there?
      call MPI_Comm_size(MPI_COMM_WORLD, nPEs, ierr)
!  What processor am I (what is my rank)?
      call MPI_Comm_rank(MPI_COMM_WORLD, iam, ierr)
      ...
      call MPI_Finalize(ierr)
```

In Fortran, MPI routines are subroutines, and last parameter is an error value

Exercise 1 : Hello World

- Write a parallel "hello world" program
    - Initialize MPI
    - Have each processor print out "Hello, World" and its processor number (rank)
    - Quit MPI

Compiling

- Most everywhere including Mio and RA
  - mpif77 mpif90
  - mpicc mpiCC
- On IBM AIX
  - mpxlf, mpxlf90,
  - mpcc, mpCC
- Most MPI compilers are actually just scripts that call underlying Fortran or C compilers

Running

- Most often you will use a batch system
- Write a batch script file.
- Use the command **mpiexec** or **mpirun** to actu start the program
- You must tell the system how many copies to run
- On some systems you must tell where to run the program

IBM - runjob
Cray - aprun

A Simple PBS Run Script

```
#!/bin/bash
#PBS -q dque
#PBS -N a_long_job_name
#PBS -l nodes=2:ppn=2
#PBS -l walltime=00:5:00
#PBS -o e3d.out
#PBS -e e3d.err
#PBS -A USE300
##PBS -k eo
#PBS -V

cd /gpfs/projects/tkaiser/mpi_tests

cp $PBS_NODEFILE nodes

mpiexec -machinefile $PBS_NODEFILE -np 4 example.exe
```

A Complex Loadleveler script

```
#!/usr/bin/ksh
#@environment = COPY_ALL; #AIXTHREAD_SCOPE=S; #MP_ADAPTER_USE=dedicated; \
#MP_CPU_USE=unique;#MP_CSS_INTERRUPT=no; #MP_EAGER_LIMIT=64K; \
#MP_EUIDEVELOP=min; #MP_LABELIO=yes; #MP_POLLING_INTERVAL=100000; #MP_PULSE=0; \
#MP_SHARED_MEMORY=yes; #MP_SINGLE_THREAD=yes;#RT_GRQ=ON; #SPINLOOPTIME=0; \
#YIELDLOOPTIME=0
###@account_no = your_account
#@class = normal
#@node = 1
#@tasks_per_node = 4
#@wall_clock_limit = 00:05:00
#@node_usage = not_shared
#@network.MPI = sn_all, shared, US
#@job_type = parallel
#@job_name= job.$(jobid)
#@output = LL_out.$(jobid)
#@error = LL_err.$(jobid)
#@notification = never
###@notify_user = your_email
#@initialdir = /dsgpfs/projects/tkaiser/mpi_tests
#@queue
exe=`ls *exe`
for job in $exe ; do
  date
  echo "running " $job
  runjob $job
done
```

A More Complex PBS run script

```
#!/bin/csh
#PBS -q dque
#PBS -N a_long_job_name
#PBS -l nodes=2:ppn=2
#PBS -l walltime=00:5:00
#PBS -o e3d.out
#PBS -e e3d.err
#PBS -A USE300
##PBS -k eo
#PBS -V

cd /gpfs/projects/tkaiser/mpi_tests

cp $PBS_NODEFILE nodes

setenv EXAM `ls *exe`

foreach EXE ($EXAM)
  echo time01 `date`
  echo running $EXE
  setenv OUT `echo $EXE  | sed -e "s/exe/out/"`
  mpiexec   -machinefile $PBS_NODEFILE -np 4 ./$EXE > $OUT
  echo time02 `date`
end
```

Note: we are using C shell here

Runs every *exe file in a directory.

A very simple Slurm Script

```
#!/bin/bash -x
#SBATCH --job-name="hybrid"
#comment = “glorified hello world"
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --exclusive
#SBATCH --export=ALL
#SBATCH --time=10:00:00


# Go to the directoy from which our job was launched
cd $SLURM_SUBMIT_DIR

# run an application
srun $SLURM_SUBMIT_DIR/helloc
```

```
# You can also use the following format to set
# --nodes              - # of nodes to use
# --ntasks-per-node    - ntasks = nodes*ntasks-per-node
# --ntasks             - total number of MPI tasks
#srun --nodes=$NODES --ntasks=$TASKS --ntasks-per-node=$TPN $EXE > output.$SLURM_JOBID
```

Basic Communication

- Data values are transferred from one processor to another
    - One processor sends the data
    - Another receives the data
- Synchronous
    - Call does not return until the message is sent or received
- Asynchronous
    - Call indicates a start of send or receive, and another call is made to determine if finished

Synchronous Send

- C
  - MPI_Send(&buffer, count ,datatype, destination, tag,communicator);
- Fortran
  - Call MPI_Send(buffer, count, datatype, destination,tag,communicator, ierr)
- Call blocks until message on the way

Call MPI_Send(buffer, count, datatype, destination, tag, communicator, ierr)

- **Buffer**: The data array to be sent
- **Count** : Length of data array (in elements, 1 for scalars)
- **Datatype** : Type of data, for example : MPI_DOUBLE_PRECISION, MPI_INT, etc
- **Destination** : Destination processor number (within given communicator)
- **Tag** : Message type (arbitrary integer)
- **Communicator** : Your set of processors
- **Ierr** : Error return (Fortran only)

Synchronous Receive

- C
  - MPI_Recv(&buffer,count, datatype, source, tag, communicator, &status);
- Fortran
  - Call MPI_ RECV(buffer, count, datatype, source,tag,communicator, status, ierr)
- Call blocks the program until message is in buffer
- Status - contains information about incoming message
  - C
    - MPI_Status status;
  - Fortran
    - Integer status(MPI_STATUS_SIZE)

Call MPI_Recv(buffer, count, datatype, source, tag, communicator, status, ierr)

- **Buffer**: The data array to be received
- **Count** : Maximum length of data array (in elements, 1 for scalars)
- **Datatype** : Type of data, for example : MPI_DOUBLE_PRECISION, MPI_INT, etc
- **Source** : Source processor number (within given communicator)
- **Tag** : Message type (arbitrary integer)
- **Communicator** : Your set of processors
- **Status**: Information about message
- **Ierr** : Error return (Fortran only)

Exercise 2 : Basic Send and Receive

- Write a parallel program to send & receive data
  - Initialize MPI
  - Have processor 0 send an integer to processor 1
  - Have processor 1 receive an integer from processor 0
  - Both processors print the data
  - Quit MPI

Summary

- MPI is used to create parallel programs based on message passing
- Usually the same program is run on multiple processors
- The 6 basic calls in MPI are:
  - `MPI_INIT( ierr )`
  - `MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )`
  - `MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )`
  - `MPI_Send(buffer, count,MPI_INTEGER,destination, tag, MPI_COMM_WORLD, ierr)`
  - `MPI_Recv(buffer, count, MPI_INTEGER,source,tag, MPI_COMM_WORLD, status,ierr)`
  - `MPI_FINALIZE(ierr)`