

Introduction to MPI

John Burkardt
School of Computational Science
Florida State University

.....

[http://people.sc.fsu.edu/~jburkardt/presentations/
mpi_2005_fsu.pdf](http://people.sc.fsu.edu/~jburkardt/presentations/mpi_2005_fsu.pdf)

School of Computational Science,
21 October 2005



Introduction to MPI

- Why is MPI needed?
- What is an MPI computation doing?
- What does an MPI program look like?
- How (and where) do I compile and run an MPI program?



It Looks Easy Today



Richardson's Computation, 1917



Richardson's Forecasting Factory



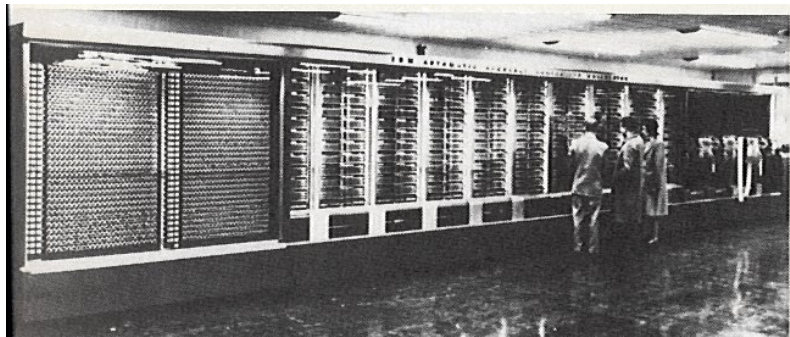
The First Computers



The Harvard College Observatory Computer Lab, 1890.



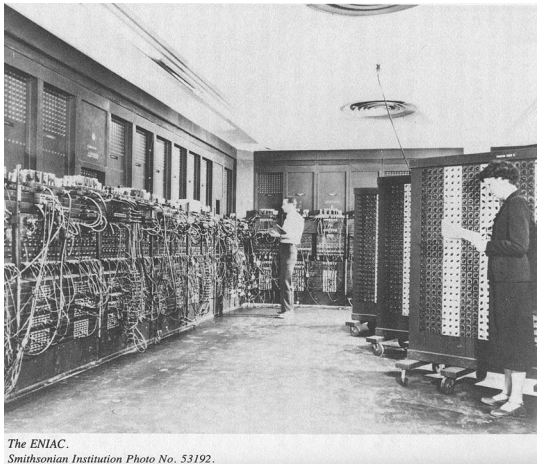
The Harvard Mark I: 1944



The first modern computers were awesome.



The ENIAC: 1942

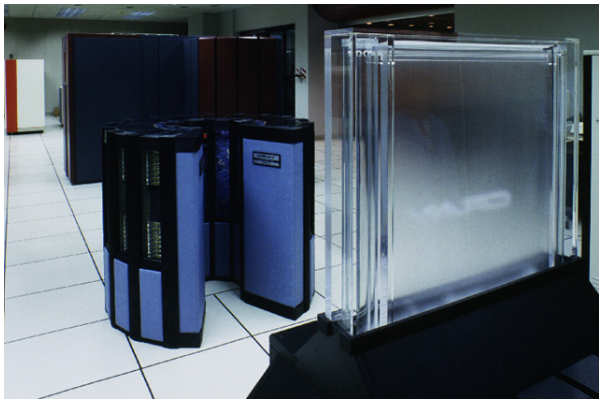


*The ENIAC.
Smithsonian Institution Photo No. 53192.*

John von Neumann wanted ENIAC for weather prediction.



The Cray YMP: 1990



As problems got bigger, supercomputers got **smaller** and **hotter**.



Grace Hopper with a Nanosecond



1 gigahertz clock cycle implies one "nanosecond" radius.



Parallel Processing to the Rescue

A single processor is exponentially expensive to upgrade.
but a *cluster* of processors is trivial to upgrade; just buy some more!

The rate of communication is an problem, so controlling the *amount* of communication is important.

MPI enables the cluster of processors to work together, and communicate.



Philosophy: MPI is just a library

- User program written in C, C++, F77 or F90;
- MPI operations invoked as calls to functions;
- MPI symbols are special constants.



Philosophy: One Program Does it All

- A single program embodies the entire task;
- This program runs on multiple processors;
- Each processor knows its ID number.



HELLO HELLO HELLO HELLO in C

```
# include <stdlib.h>
# include <stdio.h>
# include "mpi.h"

int main ( int argc, char *argv[] )
{
    int ierr;
    int num_procs;
    int my_id;

    ierr = MPI_Init ( &argc, &argv );
    ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
    if ( my_id == 0 )
    {
        ierr = MPI_Comm_size ( MPI_COMM_WORLD, &num_procs );
        printf ( "\n" );
        printf ( "HELLO.WORLD--Master_process:\n" );
        printf ( "--A_simple_C_program_using_MPI.\n" );
        printf ( "\n" );
        printf ( "--The_number_of_processes_is_%d.\n", num_procs );
        printf ( "\n" );
    }
    else
    {
        printf ( "--Process_%d_says_'Hello ,_world!'\n", my_id );
    }
    ierr = MPI_Finalize ( );
    return 0;
}
```



HELLO HELLO HELLO HELLO in FORTRAN77

```
program main

include 'mpif.h'

integer error
integer my_id
integer num_procs

call MPI_Init ( error )

call MPI_Comm_rank ( MPLCOMM_WORLD, my_id, error )

if ( my_id == 0 ) then
  call MPI_Comm_size ( MPLCOMM_WORLD, num_procs, error )
  print , ' '
  print , 'HELLO.WORLD - Master process:'
  print , ' A FORTRAN77 program using MPI.'
  print , ' '
  print , ' The number of processes is ', num_procs
  print , ' '
else
  print , ' '
  print , ' Process ', my_id, ' says "Hello ,_world!"'
end if

call MPI_Finalize ( error )

stop
end
```



HELLO HELLO HELLO HELLO in C++

```
# include <cstdlib>
# include <iostream>
# include "mpi.h"
using namespace std;

int main ( int argc, char *argv[] )
{
    int my_id;
    int num_procs;

    MPI::Init ( argc, argv );
    my_id = MPI::COMM_WORLD.Get_rank ( );

    if ( my_id == 0 )
    {
        num_procs = MPI::COMM_WORLD.Get_size ( );
        cout << "\n";
        cout << "HELLO_WORLD--Master_process:\n";
        cout << "--A_simple_C++_program_using_MPI.\n";
        cout << "--The_number_of_processes_is--" << num_procs << "\n";
    }
    else
    {
        cout << "--Process--" << my_id << "--says--'Hello ,_world!'\n";
    }

    MPI::Finalize ( );

    return 0;
}
```



The output from *HELLO*⁴

Process 2 says "Hello, world!"

HELLO WORLD - Master Process:
A simple FORTRAN90 program using MPI.
The number of processes is 4

Process 3 says "Hello, world!"

Process 1 says "Hello, world!"



Philosophy: Each Process(or) Has Its Own Data

MPI data is not shared, but can be *communicated*.

- Each process has its own data;
- To communicate, one process may send some data to another;
- Basic routines **MPI_Send** and **MPI_Recv**.



MPI_Send (data, count, type, to, tag, channel)

- **data**, the address of the data;
- **count**, number of data items;
- **type**, the data type (use an MPI symbolic value);
- **to**, the processor ID to which data is sent;
- **tag**, a message identifier;
- **channel**, the channel to be used.



MPI_Recv (data, count, type, from, tag, channel, status)

- **data**, the address of the data;
- **count**, number of data items;
- **type**, the data type (use an MPI symbolic value);
- **from**, the processor ID from which data is received;
- **tag**, a message identifier;
- **channel**, the channel to be used;
- **status**, warnings, errors, etc.



Communication: An example algorithm

Compute $A * x = b$.

- a "task" is to multiply one row of A times x ;
- we can assign one task to each processor. Whenever a processor is done, give it another task.
- each processor needs a copy of x at all times; for each task, it needs a copy of the corresponding row of A .
- processor 0 will do no tasks; instead, it will pass out tasks and accept results.



Matrix * Vector in FORTRAN77 (Page 1)

```
      if ( my_id == master )  
          numsent = 0  
c  
c BROADCAST X to all the workers.  
c  
      call MPI_BCAST ( x, cols, MPI.DOUBLE.PRECISION, master,  
          & MPI.COMM_WORLD, ierr )  
  
c  
c SEND row l to worker process l; tag the message with the row number.  
c  
      do i = 1, min ( num-procs-1, rows )  
  
          do j = 1, cols  
              buffer(j) = a(i,j)  
          end do  
  
          call MPI_SEND ( buffer, cols, MPI.DOUBLE.PRECISION, i,  
              & i, MPI.COMM_WORLD, ierr )  
  
          numsent = numsent + 1  
  
      end do
```



Matrix * Vector in FORTRAN77 (Page 2)

```
c
c  Wait to receive a result back from any processor;
c  If more rows to do, send the next one back to that processor.
c
      do i = 1, rows

          call MPI_RECV ( ans, 1, MPI.DOUBLE.PRECISION,
&             MPI.ANY_SOURCE, MPI.ANY_TAG,
&             MPI.COMM.WORLD, status, ierr )

          sender = status(MPI_SOURCE)
          anstype = status(MPI_TAG)
          b(anstype) = ans

          if ( numsent .lt. rows ) then

              numsent = numsent + 1

              do j = 1, cols
                  buffer(j) = a(numsent,j)
              end do

              call MPI_SEND ( buffer, cols, MPI.DOUBLE.PRECISION,
&             sender, numsent, MPI.COMM.WORLD, ierr )

          else

              call MPI_SEND ( MPI_BOTTOM, 0, MPI.DOUBLE.PRECISION,
&             sender, 0, MPI.COMM.WORLD, ierr )

          end if

      end do
```



Matrix * Vector in FORTRAN77 (Page 3)

```
c
c Workers receive X, then compute dot products until
c done message received
c
  else
    call MPI_BCAST ( x, cols, MPI_DOUBLE_PRECISION, master,
& MPI_COMM_WORLD, ierr )
90  continue
    call MPI_RECV ( buffer, cols, MPI_DOUBLE_PRECISION, master,
& MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr )
    if ( status(MPI_TAG) .eq. 0 ) then
      go to 200
    end if
    row = status(MPI_TAG)
    ans = 0.0
    do i = 1, cols
      ans = ans + buffer(i) * x(i)
    end do
    call MPI_SEND ( ans, 1, MPI_DOUBLE_PRECISION, master,
& row, MPI_COMM_WORLD, ierr )
    go to 90
200  continue
  end if
```



Running MPI: on Phoenix

At SCS, there is a public cluster called Phoenix.

Any SCS user can log in to **phoenix.csit.fsu.edu**.

Compile your MPI program with **mpicc**, **mpiCC**, **mpif77**.
(**mpif90** is not working yet!)

Run your job by writing a condor script:
condor_submit job.condor



Running MPI: on Phoenix

A sample Condor script:

```
universe = MPI
initialdir = /home/u8/users/burkardt
executable = matvec
log = matvec.log
output = output$(NODE).txt
machine_count = 4
queue
```



Running MPI: on Teragold

Two IBM clusters, Teragold and Eclipse.

To get an account requires an application process.

Log in to **teragold.fsu.edu**.

Run your job by writing a LoadLeveler script:
lsubmit job.ll



Running MPI: on Teragold

A sample LoadLeveler script:

```
# job_name = matvec
# class = short
# wall_clock_limit = 100
# job_type = parallel
# node = 1
# tasks_per_node = 4
# node_usage = shared
# network.mpi = css0,shared,US
# queue
mpcc_r matvec.c
mv a.out matvec
matvec > matvec.out
```



- Peter Pacheco, **Parallel Programming with MPI** ;
- Stan Openshaw+, **High Performance Computing+**;
- Scott Vetter+, **RS/600 SP: Practical MPI Programming**;
- William Gropp+, **Using MPI**;
- Marc Snir, **MPI: The Complete Reference**.



With prefix **http://www.csit.fsu.edu/**

- **this talk:** `~jburkardt/pdf/mpi_intro.pdf`
- **MPI + C:** `~jburkardt/c_src/mpi/mpi.html`
(or `cpp_src`, `f77_src`, `f_src`)
- **Condor:** `~jburkardt/f_src/condor/condor.html`
or
`twiki/bin/view/TechHelp/UsingCondor`
- **LoadLeveler:** `supercomputer/sp3_batch.html`



As today's master process...

I BROADCAST the following MESSAGE:

Happy Parallel Trails to You!

MPI_Finalize()!

