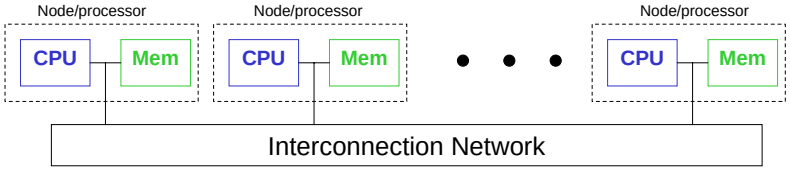
Lecture 7b – Introduction to MPI

- Some simple codes in Fortran and C
- Initialization and simple communicators
- Point to point communication
- Collective communication
- Derived datatypes
- Communicators and topology
- Examples

Message Passing Programming



- Each processor has its own private memory and address space
- The processors communicate with one another through the network
- Ideally, each node is directly connected to every other node → too expensive to build
- A compromise is to use crossbar switches connecting the processors
- Use simple topology: e.g. linear array, ring, mesh, hypercube
- Communication time is the bottleneck of message passing programming

Message Passing Programs

- Separate processors
- Separate address spaces
- Processors execute independently and concurrently
- Processors transfer data cooperatively
- Single Program Multiple Data (SPMD)
  - All processors are executing the same program, but act on different data
- Multiple Program Multiple Data (MPMD)
  - Each processor may be executing a different program
  - Ex: multi-disciplinary where some processors are running fluid, others heat conduction, others structures, etc.
- Common software tools: PVM, MPI

What is MPI?

- Message-passing library specification (IEEE Standard)
  - Message-passing model
  - Not a compiler specification
  - Not a specific product
- For parallel computers, clusters, and heterogeneous networks
- Designed to permit the development of parallel software libraries
- Designed to provide access to advanced parallel hardware for
  - End users
  - Library writers
  - Tool developers

Who Designed MPI?

- **Broad group of participants**
- **Vendors:**
  - IBM, Intel, TMC, Meiko, Cray, Convex, nCube
- **Library developers:**
  - PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda
- **Application specialists and consultants**
  - Companies: ARCO, KAI, NAG, Parasoft, Shell,…
  - Labs: ANL, LANL, LLNL, ORNL, SNL,…
  - Universities: almost 20

Why Use MPI?

- **Standardization:**
  - The only message passing library which can be considered a standard
- **Portability:**
  - There is no need to modify the source when porting codes from one platform to another
- **Performance:**
  - Vendor implementations should be able to exploit native hardware to optimize performance
- **Availability:**
  - A variety of implementations are available, both vendor and public domain, e.g. MPICH implementation by ANL, OpenMP by openmp.org
- **Functionality:**
  - It provides around 200 subroutine/function calls

Features of MPI

- **General:**
  - Communicators combine context and group for message security
  - Thread safety
- **Point to point communication:**
  - Structured buffers and derived datatypes, heterogeneity
  - Modes: standard, synchronous, ready (to allow access to fast protocols), buffered
- **Collective communication:**
  - Both built-in and user defined collective operations
  - Large number of data movement routines
  - Sub-group defined directly or by topology

Is MPI Large or Small?

- **MPI is large – around 200 functions**
  - Extensive functionality requires many functions/subroutines
- **MPI is small – 6 basic functions**
  - MPI_Init: Initialize MPI
  - MPI_Comm_size: Find out how many processes there are
  - MPI_Comm_rank: Find out which process I am
  - MPI_Send: Send a message
  - MPI_Recv: Receive a message
  - MPI_Finalize: Terminate MPI
- **MPI is just right**
  - One can use its flexibility when it is required
  - One need not master all parts of MPI to use it

Example of Large-Scale MPI Codes: TFLO, MBFLO, OVERFLOW

- **MPI Standard has around 200 functions/subroutines for just about anything that you may think of**
- **MBFLO: general conjugate heat transfer (fluid/thermal) code uses ~20 MPI subroutines**
- **MBFLO uses some of the advanced features of MPI**
- **Normal codes are likely to use only a few MPI calls**

Example: Hello, World! C-Code

- **#include "mpi.h" provides basic MPI definitions and types**
- **MPI_Init starts MPI**
- **MPI_Finalize exists MPI**
- **Note that all non-MPI routines are local; thus printf runs on each process.**

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

int main(argc, argv)
int argc;
char **argv;
{
    MPI_Init(&argc, &argv);
    printf("Hello, world!\n");
    MPI_Finalize();
    return 0;
}
```


Example: "Advanced" Hello, World! C-Code

- **MPI_Comm_rank determines the proc id (0 to nproc-1)**
- **MPI_Comm_size determines the # of procs**
- **Note: for some parallel systems, only a few designated procs can do I/O. MPI-2 Standard defines API for parallel I/O**
- **What does the output look like?**

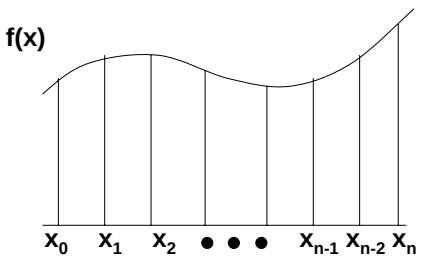
```
#include "mpi.h"
#include <stdio.h>

int main(argc, argv)
int argc;
char **argv;
{
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, world! I am %d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```


Example: Calculate π

- **Well-known formula:** $$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$
- **Numerical integration (trapezoidal rule):**



$$\int_a^b f(x)\,dx \approx h\left[\frac{1}{2}f(x_0)+f(x_1)+\cdots f(x_{n-1})+\frac{1}{2}f(x_n)\right]$$

$$x_i = a+ih,\quad h=\frac{(b-a)}{n},\quad n = \#\text{ of intervals}$$

Calculate π Serial C-Code

- **A sequential function Trap(a,b,n) approximates the integral from a to b of f(x) using the trapizoidal rule with n sub-intervals:**

```
 int n;
double a,b,integral,pi;
a = 0.0;  /*DEFINE INTERVAL START*/
b = 1.0;  /*DEFINE INTERVAL STOP */
/*READ THE NUMBER OF SUB-INTERVALS */
printf("INPUT THE NUMBER OF SUB-INTERVALS");
scanf("%i",&n);
  integral = trapc(a,b,n)
  printf("PI = %d",integral)
return 0
```

```
double Trap(a,b,n) {
   f = 4./(1.+x*x);
   h = (b-a)/n;
   integral = (f(a)+f(b))/2;
   for (i=1; i<=n-1; i++) {
      x = a+i*h;
      integral = integral + f(x);
   }
   integral = h*integral;
   return integral;
}
```

Code is on web: HOMEWORK/calcpi

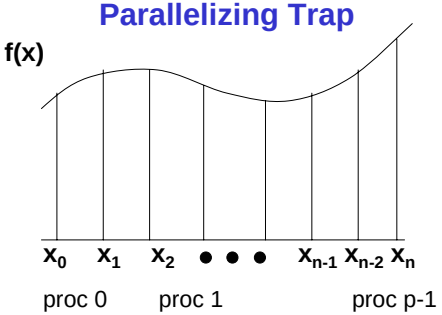
Calculate π Serial Fortran-Code

```
PROGRAM CALCPI
IMPLICIT NONE
REAL*8 :: A,B,PI
INTEGER :: N
A = 0.0   !DEFINE INTERVAL START
B = 1.0   !DEFINE INTERVAL STOP
!READ THE NUMBER OF SUB-INTERVALS
PRINT *,'INPUT THE NUMBER OF SUB-INTERVALS'
READ(*,*) N
CALL TRAP(A,B,N,PI)
PRINT *,'PI = ',PI
STOP
END
```

```
 SUBROUTINE TRAP(A,B,N,INTEGRAL)
!CALCULATE INTEGRAL OF FUNCTION WITH &
 EVEN INTERVAL
IMPLICIT NONE
REAL*8  :: A,B,F,H,INTEGRAL,X
INTEGER :: N,I
F(X) = 4.0/(1.+X**2)            !FUNCTION DEFINITION
H(A,B,N) = (B-A)/REAL(N)   !INTERVAL DEFINITION
INTEGRAL = (F(A)+F(B))/2. !INITIALIZE INTEGRAL
DO I = 1,N-1
  X = A + REAL(I)*H(A,B,N)
  INTEGRAL = INTEGRAL + F(X)
END DO
INTEGRAL = H(A,B,N)*INTEGRAL
RETURN
END
```

Code is on web: HOMEWORK/calcpi

Parallelizing Trap



- **Divide the interval [a,b] into p equal sub-intervals**
- **Each processor calculates the local approximate integral using the Trap routine simultaneously**
- **Finally, combine the local values to obtain the total integral.**

Calculate π Parallel Fortran Code

```
PROGRAM CALCPIP
IMPLICIT NONE

include "mpif.h"
REAL*8  :: A,AK,B,BK,H,PI,SUBPI
INTEGER :: K,MYID,N,NK,NPROCS
INTEGER :: IERROR,TAG,STATUS

! INITIALIZE MPI
CALL MPI_Init(IERROR)

! DETERMINE MY PROCESSOR ID
! ARGUMENTS: COMM, MYID, IERROR
CALL MPI_Comm_rank(MPI_COMM_WORLD,MYID,IERROR)

! FIND OUT HOW MANY PROCESSORS ARE USED
! ARGUMENTS: COMM, NPROCS, IERROR
CALL MPI_Comm_size(MPI_COMM_WORLD,NPROCS,IERROR)

IF(MYID == 0) THEN
  !READ THE NUMBER OF SUB-INTERVALS
  PRINT *,'INPUT THE NUMBER OF SUB-INTERVALS'
  READ(*,*) N
  IF(N < NPROCS) GO TO 1000
END IF
```

Calculate π Parallel Fortran Code

```
! BROADCAST THE NUMBER OF SUB-INTERVALS
! ARGUEMENTS: BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR
CALL MPI_Bcast(N,1,MPI_INTEGER,0,MPI_COMM_WORLD,IERROR)

A = 0.0   !DEFINE INTERVAL START
B = 1.0   !DEFINE INTERVAL STOP
H  = (B-A)/REAL(N)
! N INTERVALS MUST BE EVENLY DIVISIBLE BY NPROCS
NK = N/NPROCS
AK = A + REAL(MYID)*REAL(NK)*H
BK = AK + REAL(NK)*H

! COMPUTE LOCAL INTEGRAL
CALL TRAP(AK,BK,NK,SUBPI)

! SET UP A MASTER-SLAVE RELATIONSHIP WHERE THE MASTER
! IS RESPONSIBLE FOR ACCUMULATING THE SUB-INTEGRALS
! AND WRITING OUT THE ANSWER
```


Calculate π Parallel Fortran Code

```
 IF(MYID == 0) THEN
  ! SUM UP THE INTEGRALS FROM THE OTHER PROCESSORS
  PI = SUBPI
  ! ADD THE SUBPI'S FROM THE OTHER PROCESSORS
  ! ARGUMENTS: BUFFER, COUNT, DATATYPE, SOURCE, TAG,
  !                    COMM, STATUS, IERROR
  DO K = 1,NPROCS-1
    CALL MPI_Recv(SUBPI,1,MPI_DOUBLE_PRECISION,K,TAG,
&                       MPI_COMM_WORLD,STATUS,IERROR)
    PI = PI + SUBPI
  END DO
  PRINT *,'PI = ',PI
 ELSE
  ! SEND THE INTEGRAL TO THE MASTER
  ! ARGUMENTS: BUFFER, COUNT, DATATYPE, DEST, TAG,
  !                    COMM, IERROR
  CALL MPI_Send(SUBPI,1,MPI_DOUBLE_PRECISION,0,TAG,
&                    MPI_COMM_WORLD,IERROR)
 END IF

    ! TERMINATE MPI
1000 CALL MPI_Finalize(IERROR)
    STOP
    END
```

Code and makefiles are on web: HOMEWORK/calcpip

Calculate π Parallel Fortran Code

- **We could replace the MPI_Send, MPI_Recv and the subsequent sum with another MPI routine that gather/adds the sub-pi's: MPI_Reduce**
- **The last slide then can be replaced with:**

```
    ! GATHER/ADD THE SUB-PI'S
    ! ARGUMENTS: SENDBUF, RECVBUF, COUNT, DATATYPE, OP,ROOT, COMM, ERR
    CALL MPI_Reduce(PI,SUBPI,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
&                       MPI_COMM_WORLD,IERROR)
    ! TERMINATE MPI
1000 CALL MPI_Finalize(IERROR)
    STOP
    END
```

- **Embarrassingly parallel – no communication needed during the computations of the local integrals**

Timing

- **MPI_Wtime() returns the wall-clock time**

```
double start, finish, time;

MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();
.
.
.
MPI_Barrier(MPI_COMM_WORLD);
finish = MPI_Wtime();
time = finish – start;
```