Distributed Memory Programming with Message-Passing

Pacheco. Chapter 3

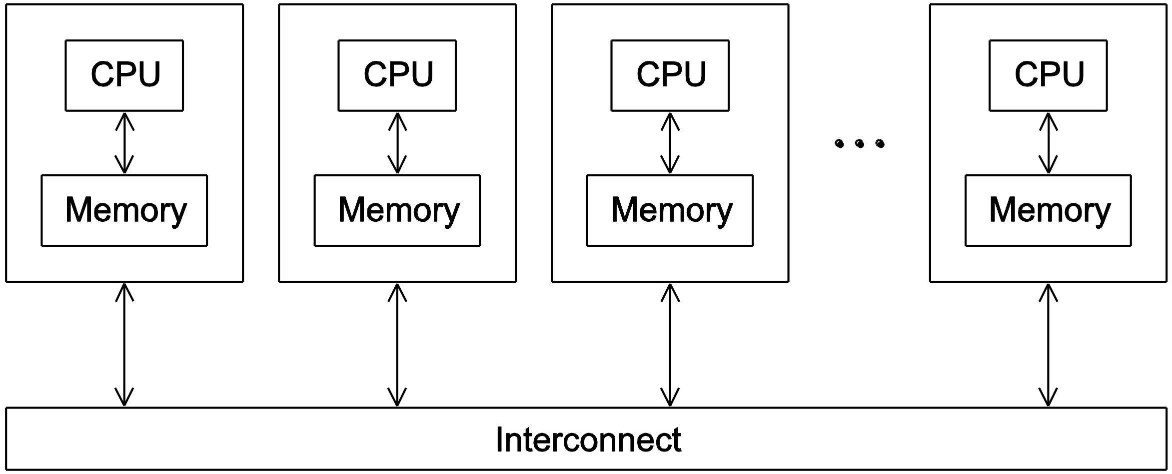
T. Yang, CS140 2014

Part of slides from the text book and B. Gropp

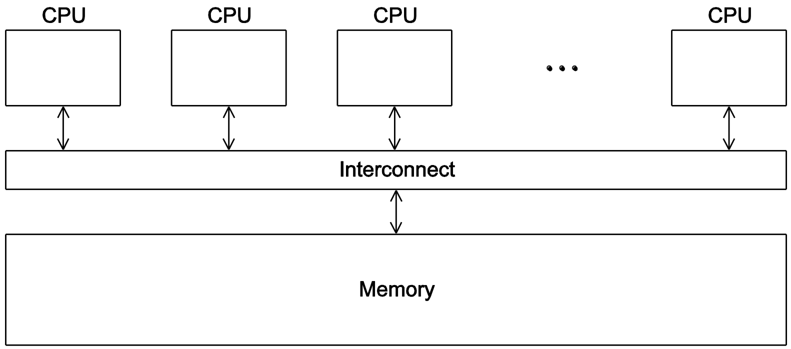
Outline

- **An overview of MPI programming**
  - Six MPI functions and hello sample
  - How to compile/run
- **More on send/receive communication**
- **Parallelizing numerical integration with MPI**

Mainly for distributed memory systems



Not targeted for shared memory machines. But can work



Message Passing Libraries

- MPI, Message Passing Interface, now the industry standard, for C/C++ and other languages
- **Running as a set of processes. No shared variables**
- **All communication, synchronization require subroutine calls**
  - Enquiries
    - How many processes? Which one am I? Any messages waiting?
  - Communication
    - point-to-point: Send and Receive
    - Collectives such as broadcast
  - Synchronization
    - Barrier

MPI Implementations & References

- **The Standard itself (MPI-2, MPI-3):**
  - at http://www.mpi-forum.org
- **Implementation for Linux/Windows**
  - Vendor specific implementation
  - MPICH
  - Open MPI
- **Other information on Web:**
  - http://en.wikipedia.org/wiki/Message_Passing_Interface
  - http://www.mcs.anl.gov/mpi MPI talks and tutorials, a FAQ, other MPI pages

MPI is Simple

- Many parallel programs can be written using just these six functions, only two of which are non-trivial:
  - `MPI_INIT`
  - `MPI_FINALIZE`
  - `MPI_COMM_SIZE`
  - `MPI_COMM_RANK`
  - `MPI_SEND`
  - `MPI_RECV`
- `To measure time:  MPI_Wtime()`

Slide source: Bill Gropp, ANL

Hello World!

```
#include <stdio.h>

int main(void) {
    printf("hello, world\n");

    return 0;
}
```

(a classic)

Mpi_hello (C)

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

int main( 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( "I am %d of %d!\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

Slide source: Bill Gropp, ANL

Compilation

wrapper script to compile

source file

mpicc -g -Wall -o mpi_hello mpi_hello.c

produce debugging information

create this executable file name (as opposed to default a.out)

turns on all warnings

Execution with mpirun or mpiexec

mpirun -n <number of processes> <executable>

mpirun -n 1 ./mpi_hello

run with 1 process

mpirun -n 4 ./mpi_hello

run with 4 processes

Execution with mpirun at TSCC

mpirun –machinefile <filename> -n <number of processes> <executable>

mpirun -machinefile $PBS_NODEFILE -n 4 ./mpi_hello

run with 4 processes on allocated machines

Execution

```
mpirun -n 1 ./mpi_hello

    I am 0 of 1 !
```

```
mpirun -n 4 ./mpi_hello

    I am 0 of 4 !
    I am 1 of 4 !
    I am 2 of 4 !
    I am 3 of 4 !
```

MPI Programs

- **Written in C/C++.**
  - Has main.
  - Uses stdio.h, string.h, etc.
- **Need to add mpi.h header file.**
- **Identifiers defined by MPI start with "MPI_".**
- **First letter following underscore is uppercase.**
  - For function names and MPI-defined types.
  - Helps to avoid confusion.
- **MPI functions return error codes or `MPI_SUCCESS`**

MPI Components

- **MPI_Init**
  - Tells MPI to do all the necessary setup.

```
int MPI_Init(
      int*      argc_p  /* in/out */,
      char***   argv_p  /* in/out */);
```

- **MPI_Finalize**
  - Tells MPI we're done, so clean up anything allocated for this program.

```
int MPI_Finalize(void);
```

Basic Outline

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
    . . .
    /* No MPI calls before this */
    MPI_Init(&argc, &argv);
    . . .
    MPI_Finalize();
    /* No MPI calls after this */
    . . .
    return 0;
}
```

Basic Concepts: Communicator

- Processes can be collected into <u>groups</u>
  - <u>Communicator</u>
  - Each message is sent & received in the same communicator
- A process is identified by its <u>rank</u> in the group associated with a communicator
- There is a default communicator whose group contains all initial processes, called `MPI_COMM_WORLD`

Slide source: Bill Gropp, ANL

Communicators

```
int MPI_Comm_size(
      MPI_Comm   comm         /* in   */,
      int*       comm_sz_p    /* out  */);
```

number of processes in the communicator

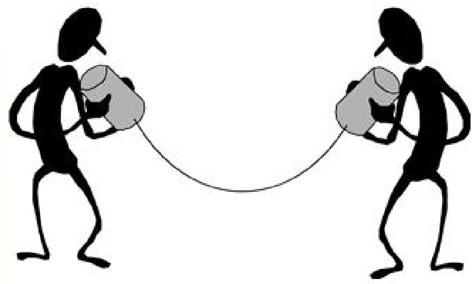
```
int MPI_Comm_rank(
      MPI_Comm   comm         /* in   */,
      int*       my_rank_p    /* out  */);
```

my rank
(the process making this call)

Basic Send

```
int MPI_Send(
    void*          msg_buf_p     /* in */,
    int            msg_size      /* in */,
    MPI_Datatype   msg_type      /* in */,
    int            dest          /* in */,
    int            tag           /* in */,
    MPI_Comm       communicator  /* in */);
```

- **Things specified:**
  - How will “data” be described?
  - How will processes be identified?
  - How will the receiver recognize/screen messages?

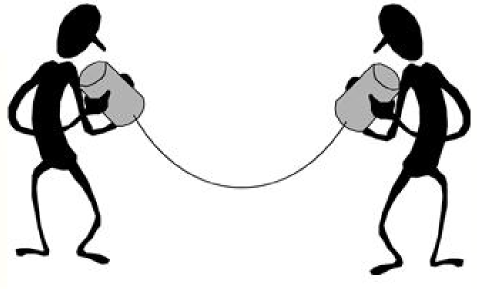


Data types

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG	signed long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Basic Receive: Block until a matching message is received

```
int MPI_Recv(
      void*          msg_buf_p      /* out */,
      int            buf_size       /* in  */,
      MPI_Datatype   buf_type       /* in  */,
      int            source         /* in  */,
      int            tag            /* in  */,
      MPI_Comm       communicator   /* in  */,
      MPI_Status*    status_p       /* out */);
```



- **Things that need specifying:**
  - Where to receive data
  - How will the receiver recognize/screen messages?
  - What is the actual message received

Message matching

```
MPI_Send(send_buf_p, send_buf_sz, send_type, dest, send_tag,
        send_comm);
```

MPI_Send

dest

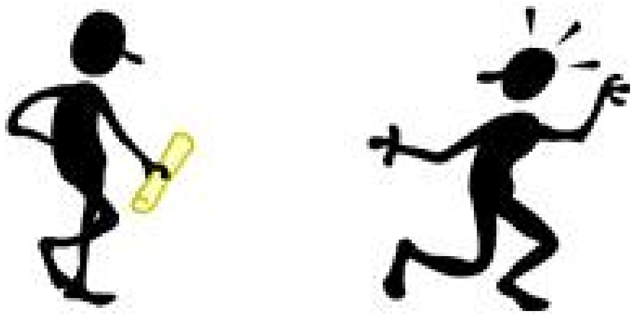
MPI_Recv

src

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,
        recv_comm, &status);
```

Receiving messages without knowing

- **A receiver can get a message without knowing:**
  - the amount of data in the message,
  - the sender of the message,
    - `Specify the source as MPI_ANY_SOURCE`
  - or the tag of the message.
    - `Specify the tag as MPI_ANY_TAG`



Status argument: who sent me and what tag is?

```
MPI_Recv(recv_buf_p, recv_buf_sz, recv_type, src, recv_tag,
         recv_comm, &status);
```

MPI_Status*

Who sent me

What tag is

Error code

Actual message length

Retrieving Further Information from status argument in C

- `Status` is a data structure allocated in the user's program.
- In C:

```
int recvd_tag, recvd_from, recvd_count;
MPI_Status status;
MPI_Recv(..., MPI_ANY_SOURCE, MPI_ANY_TAG, ..., &status )
recvd_tag  = status.MPI_TAG;
recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status, datatype, &recvd_count );
```

Slide source: Bill Gropp, ANL

Retrieving Further Information in C++

- `Status` is a data structure allocated in the user's program.
- In C++:

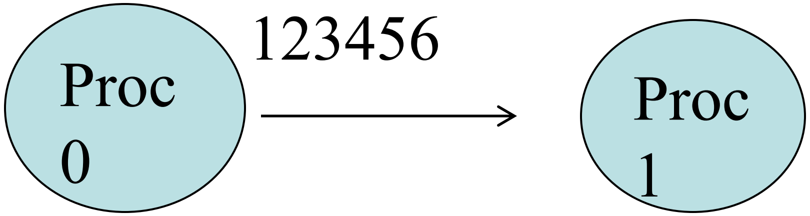
```
int recvd_tag, recvd_from, recvd_count;
MPI::Status status;
Comm.Recv(..., MPI::ANY_SOURCE, MPI::ANY_TAG, ...,
          status )
recvd_tag   = status.Get_tag();
recvd_from  = status.Get_source();
recvd_count = status.Get_count( datatype );
```

MPI Example: Simple send/recive

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[])
{
  int rank, buf;
  MPI_Status status;
  MPI_Init(&argv, &argc);
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );

  /* Process 0 sends and Process 1 receives */
  if (rank == 0) {
    buf = 123456;
    MPI_Send( &buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  }
  else if (rank == 1) {
    MPI_Recv( &buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
              &status );
    printf( "Received %d\n", buf );
  }

  MPI_Finalize();
  return 0;
}
```



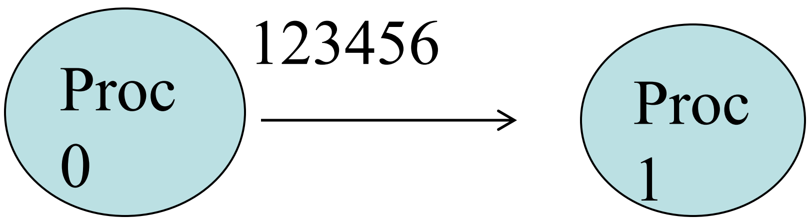
Slide source: Bill Gropp, ANL

MPI Send/Receive Example with C++

```
#include “mpi.h”
#include <iostream>
int main( int argc, char *argv[])
{
  int rank, buf;
  MPI::Init(argv, argc);
  rank = MPI::COMM_WORLD.Get_rank();

  // Process 0 sends and Process 1 receives
  if (rank == 0) {
    buf = 123456;
    MPI::COMM_WORLD.Send( &buf, 1, MPI::INT, 1, 0 );
  }
  else if (rank == 1) {
    MPI::COMM_WORLD.Recv( &buf, 1, MPI::INT, 0, 0 );
    std::cout << “Received “ << buf << “\n”;
  }

  MPI::Finalize();
  return 0;
}
```



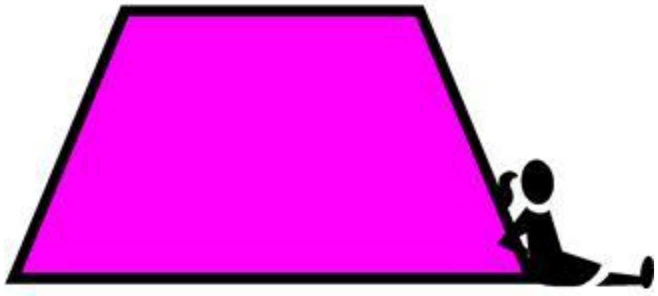
Slide source: Bill Gropp, ANL

MPI_Wtime()

- **Returns the current time with a double float.**
- **To time a program segment**
  - *Start time= MPI_Wtime()*
  - *End time=MPI_Wtime()*
  - Time spent is *end_time – start_time.*

Example of using MPI_Wtime()

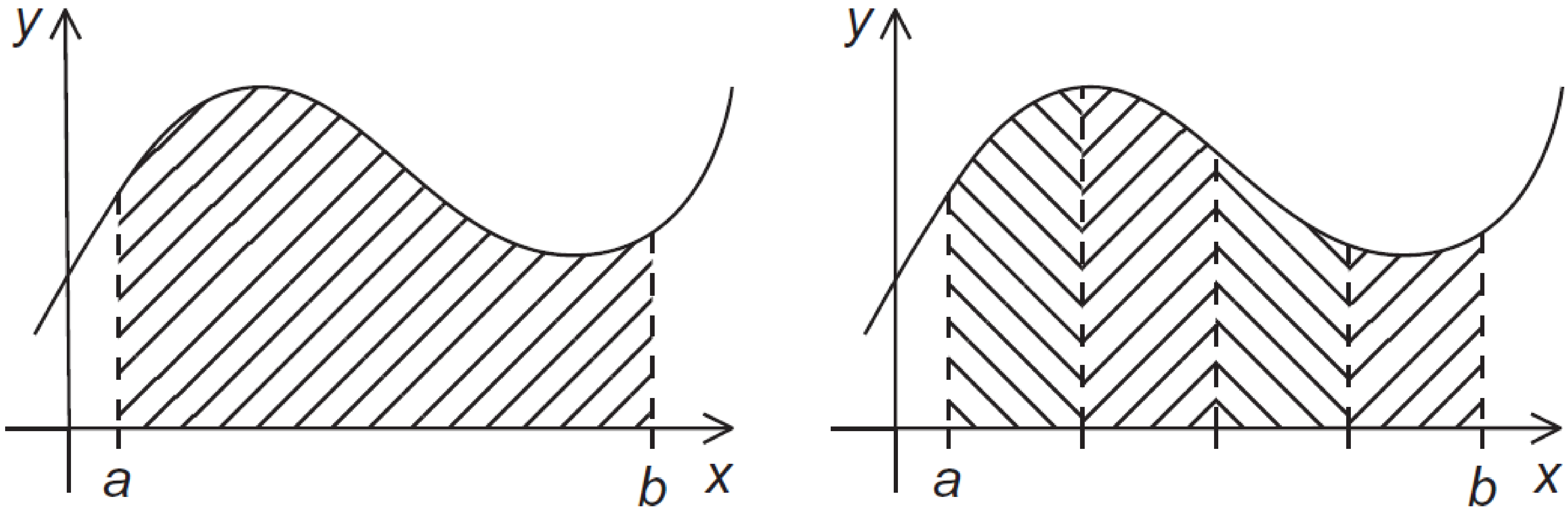
```
#include<stdio.h>
#include<mpi.h>
main(int argc, char **argv){
  int size, node;             double start, end;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &node);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  start = MPI_Wtime();
  if(node==0)  {
      printf(" Hello From Master. Time = %lf \n", MPI_Wtime() -
   start);
}
  else  {
      printf("Hello From Slave #%d %lf \n", node, (MPI_Wtime()
   - start));
    }
  MPI_Finalize();
}
```



MPI Example: Numerical Integration With Trapezoidal Rule

Textbook p. 94-101

Numerical Integration with Trapezoidal Rule



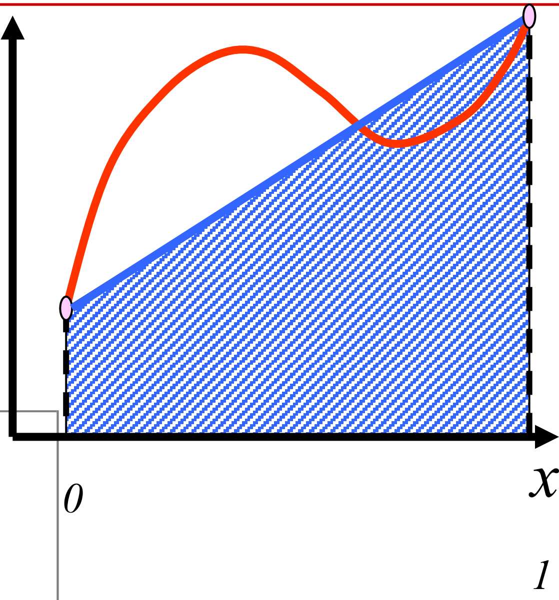
(a) (b)

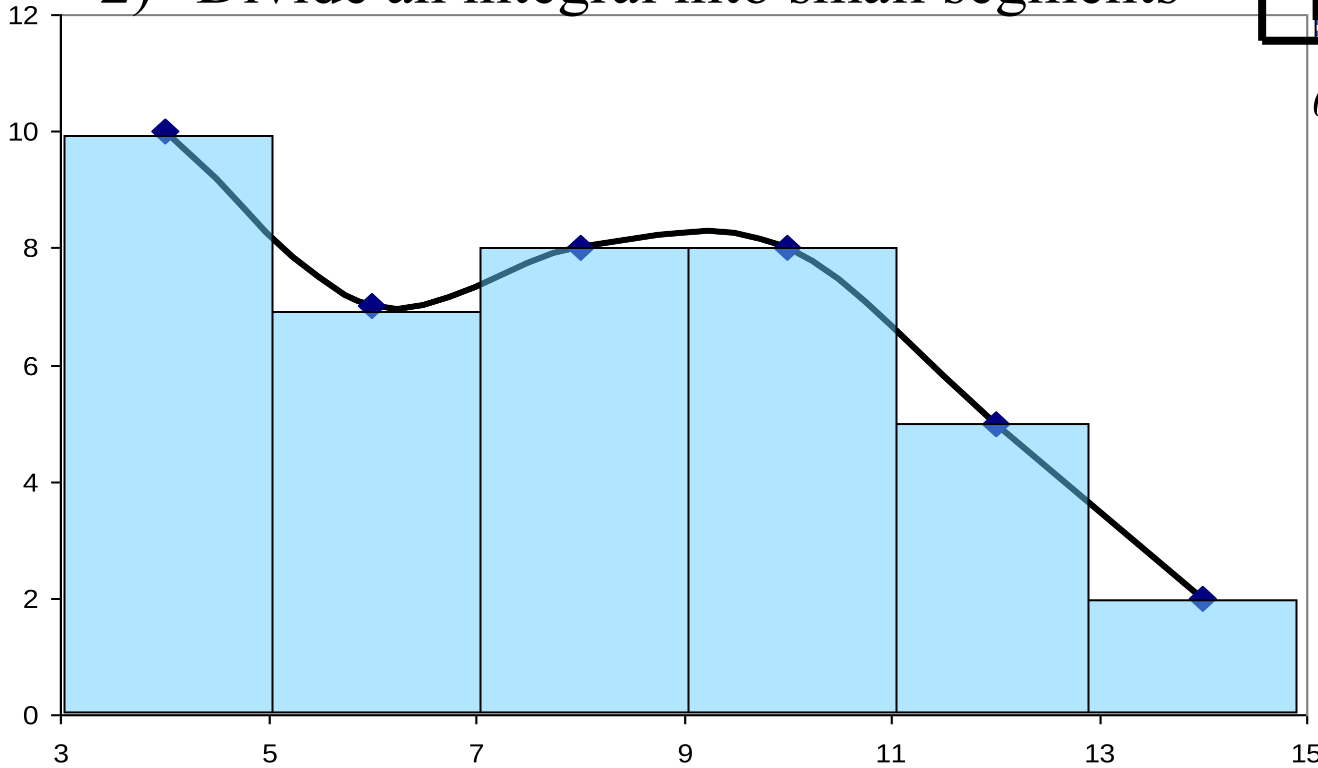
Approximation of Numerical Integration

Two ideas

1) Use a simple function to approximate the integral area.

2) Divide an integral into small segments



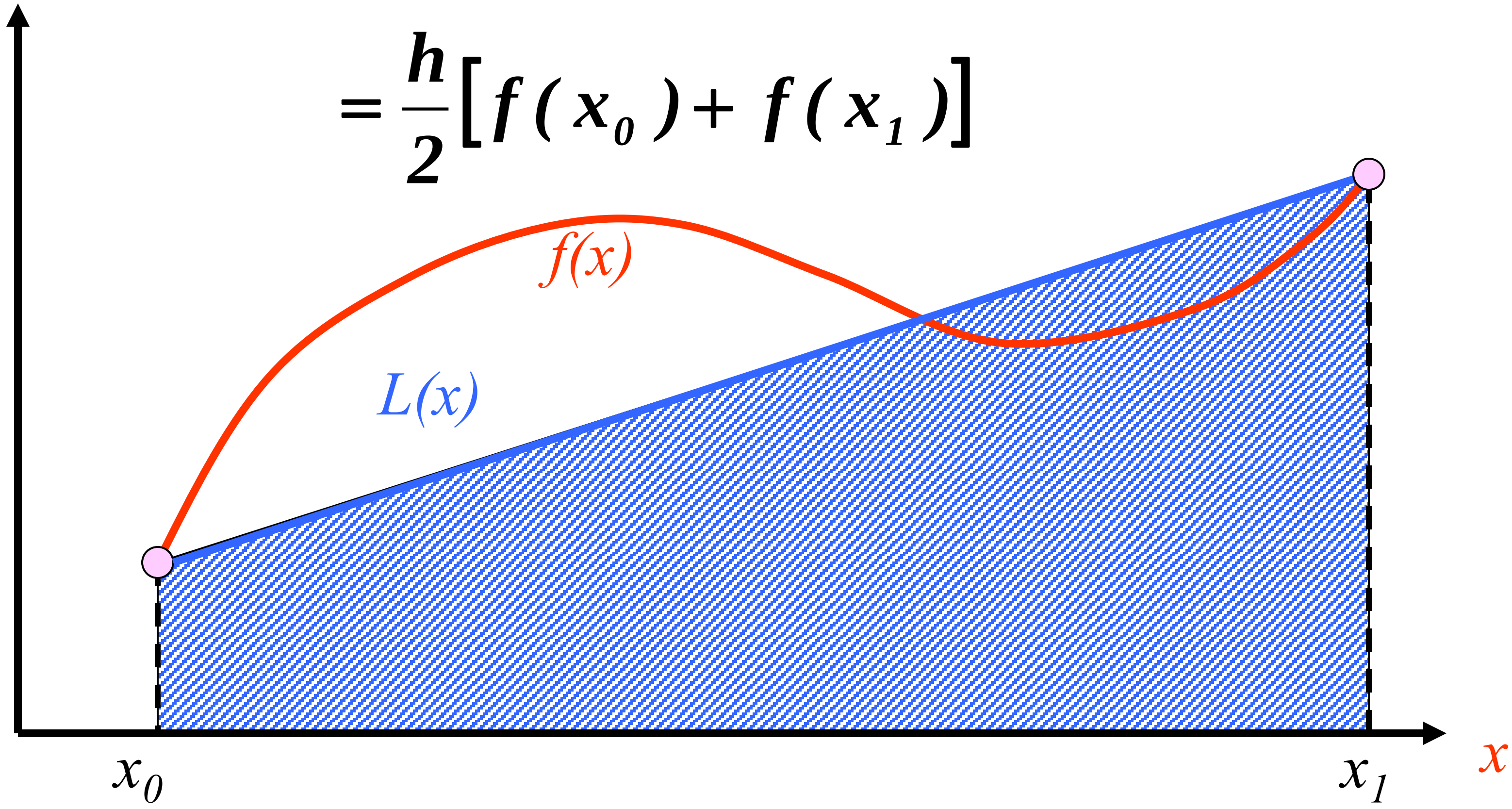


Trapezoid Rule

- **Straight-line approximation**

$$\int_a^b f(x)dx \approx \sum_{i=0}^{1} c_i f(x_i) = c_0 f(x_0) + c_1 f(x_1)$$

$$= \frac{h}{2}\left[f(x_0) + f(x_1)\right]$$



Example:Trapezoid Rule

Evaluate the integral $\int_0^4 xe^{2x}dx$

- Exact solution

$$\int_0^4 xe^{2x}dx = \left[\frac{x}{2}e^{2x} - \frac{1}{4}e^{2x}\right]_0^4$$

$$= \frac{1}{4}e^{2x}(2x-1)\Big|_0^1 = 5216.926477$$

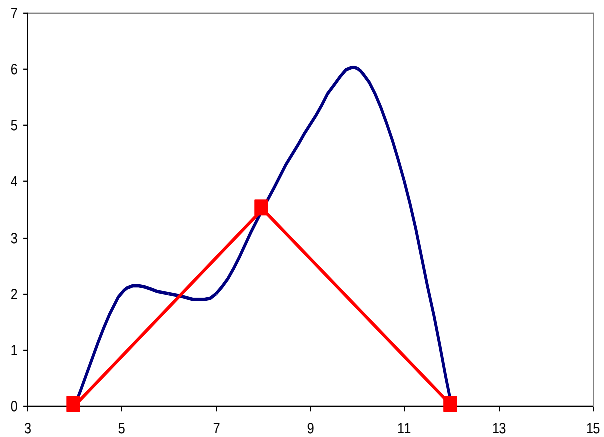
- Trapezoidal Rule

$$I = \int_0^4 xe^{2x}dx \approx \frac{4-0}{2}\left[f(0)+f(4)\right] = 2(0+4e^8) = 23847.66$$

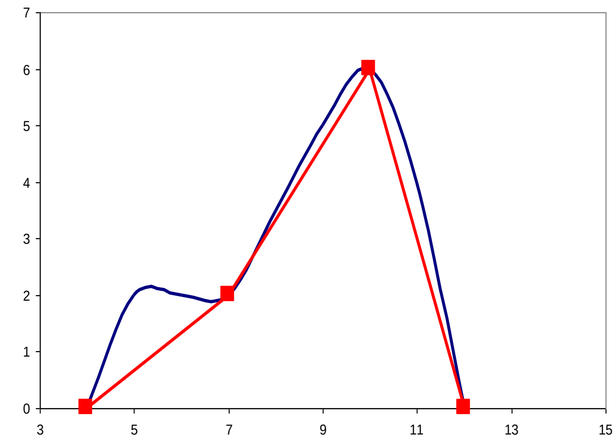
$$\varepsilon = \frac{5216.926 - 23847.66}{5216.926} = -357.12\%$$

Apply trapezoid rule to multiple segments over integration limits

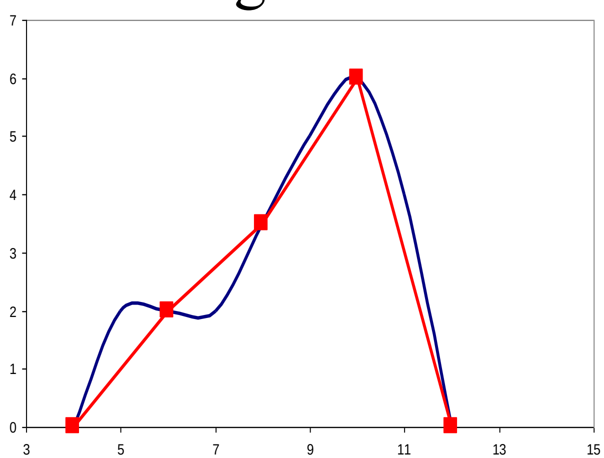
Two segments



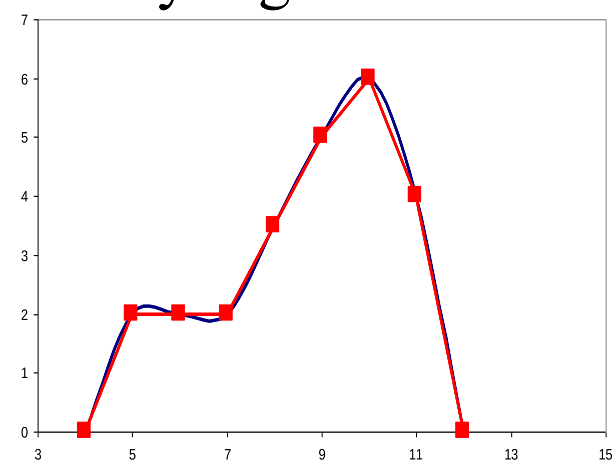
Three segments



Four segments



Many segments



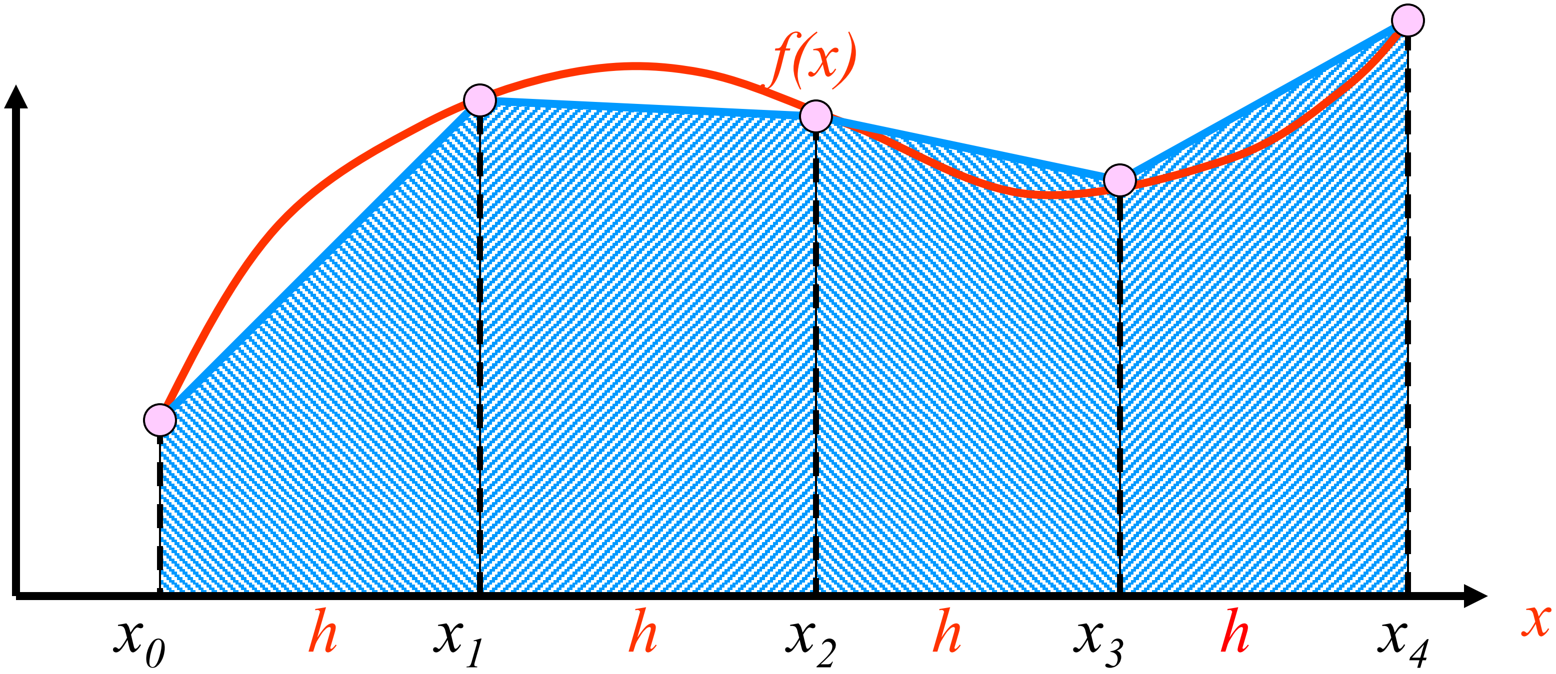
Composite Trapezoid Rule

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots\cdots + \int_{x_{n-1}}^{x_n} f(x)dx$$

$$= \frac{h}{2}\left[f(x_0) + f(x_1)\right] + \frac{h}{2}\left[f(x_1) + f(x_2)\right] + \cdots + \frac{h}{2}\left[f(x_{n-1}) + f(x_n)\right]$$

$$= \frac{h}{2}\left[f(x_0) + 2f(x_1) + \cdots + 2f(x_i) + \cdots + 2f(x_{n-1}) + f(x_n)\right]$$

$$h = \frac{b-a}{n}$$



Composite Trapezoid Rule: Example

Evaluate the integral

$$I = \int_0^4 xe^{2x}dx$$

$$n = 1, h = 4 \Rightarrow I = \frac{h}{2}\left[f(0) + f(4)\right] = 23847.66 \qquad \varepsilon = -357.12\%$$

$$n = 2, h = 2 \Rightarrow I = \frac{h}{2}\left[f(0) + 2f(2) + f(4)\right] = 12142.23 \qquad \varepsilon = -132.75\%$$

$$n = 4, h = 1 \Rightarrow I = \frac{h}{2}\left[f(0) + 2f(1) + 2f(2) + 2f(3) + f(4)\right] = 7288.79 \qquad \varepsilon = -39.71\%$$

$$n = 8, h = 0.5 \Rightarrow I = \frac{h}{2}\left[f(0) + 2f(0.5) + 2f(1) + 2f(1.5) + 2f(2) + 2f(2.5) + 2f(3) + 2f(3.5) + f(4)\right] = 5764.76 \qquad \varepsilon = -10.50\%$$

$$n = 16, h = 0.25 \Rightarrow I = \frac{h}{2}\left[f(0) + 2f(0.25) + 2f(0.5) + \cdots + 2f(3.5) + 2f(3.75) + f(4)\right] = 5355.95 \qquad \varepsilon = -2.66\%$$

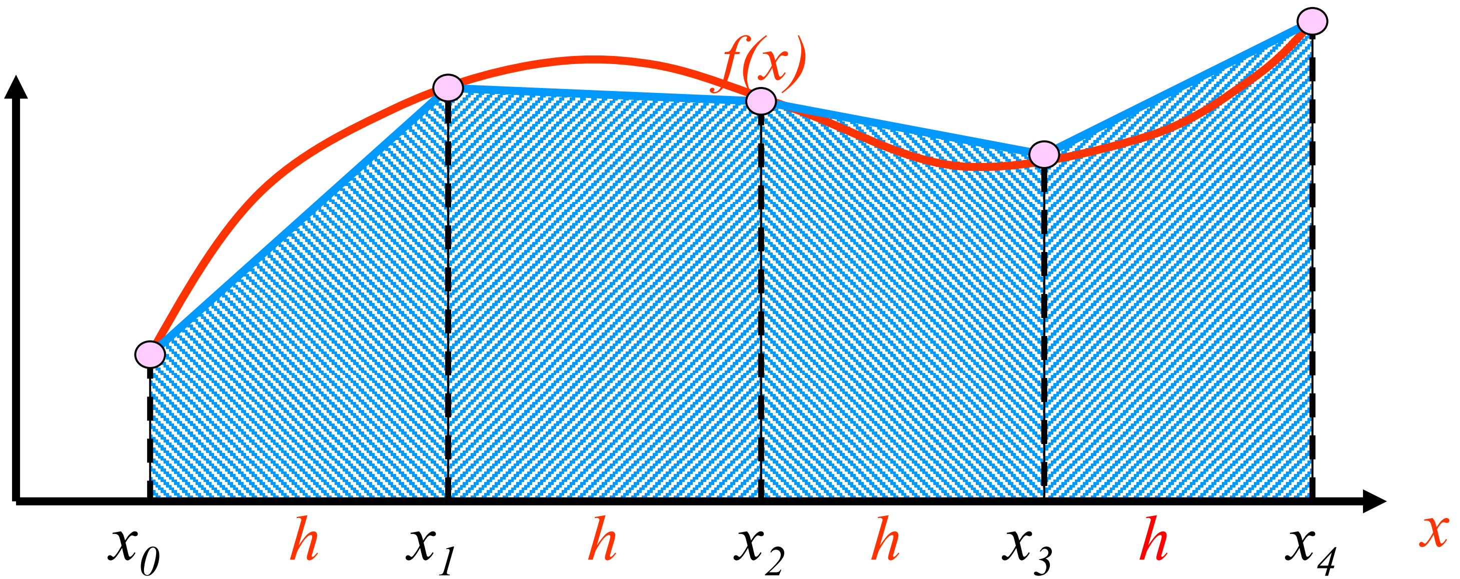
Implementing Composite Trapezoidal Rule

$$\text{Area of one trapezoid } = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$x_0 = a,\ x_1 = a + h,\ x_2 = a + 2h,\ \ldots,\ x_{n-1} = a + (n-1)h,\ x_n = b$$

$$\text{Sum of trapezoid areas } = h[f(x_0)/2 + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)/2]$$

$$h = \frac{b - a}{n}$$



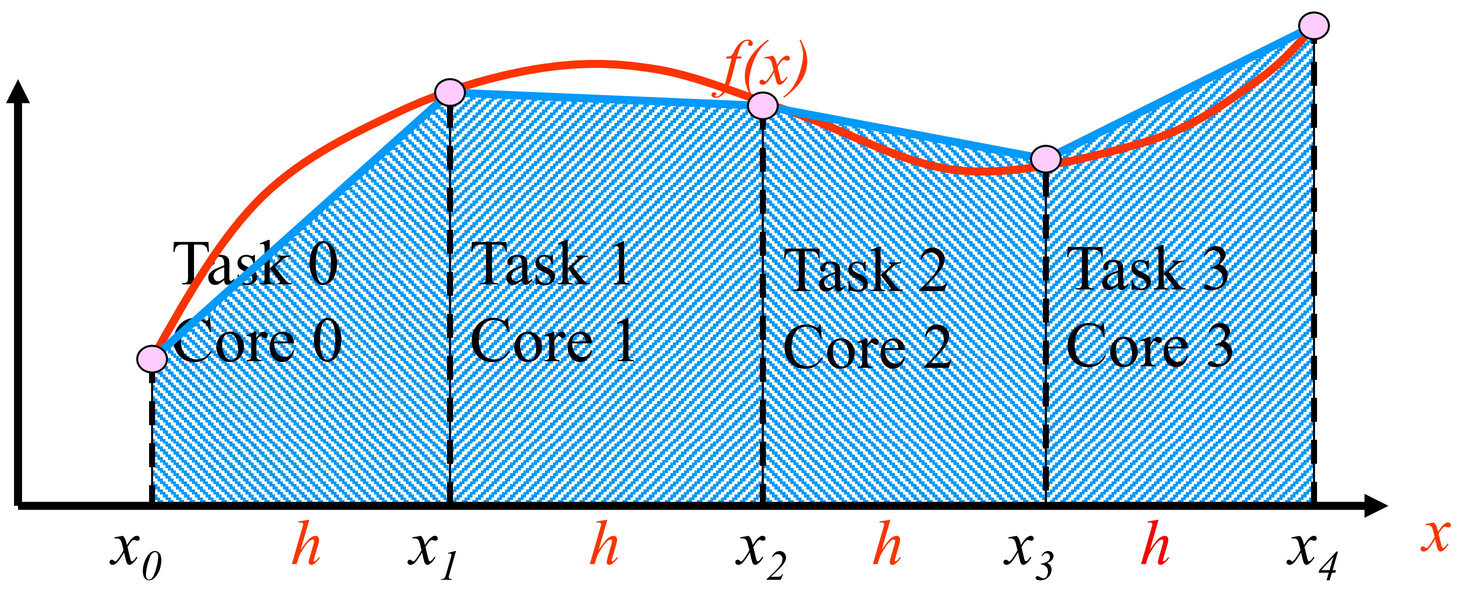
Pseudo-code for a serial program

```
/* Input: a, b, n */
h = (b-a)/n;
approx = (f(a) + f(b))/2.0;
for (i = 0; i <= n-1; i++) {
    x_i = a + i*h;
    approx += f(x_i);
}
approx = h*approx;
```

Parallelizing the Trapezoidal Rule

1. **Partition problem solution into tasks.**
2. **Identify communication channels between tasks.**
3. **Aggregate tasks into composite tasks.**
4. **Map composite tasks to cores.**

$$h = \frac{b-a}{n}$$



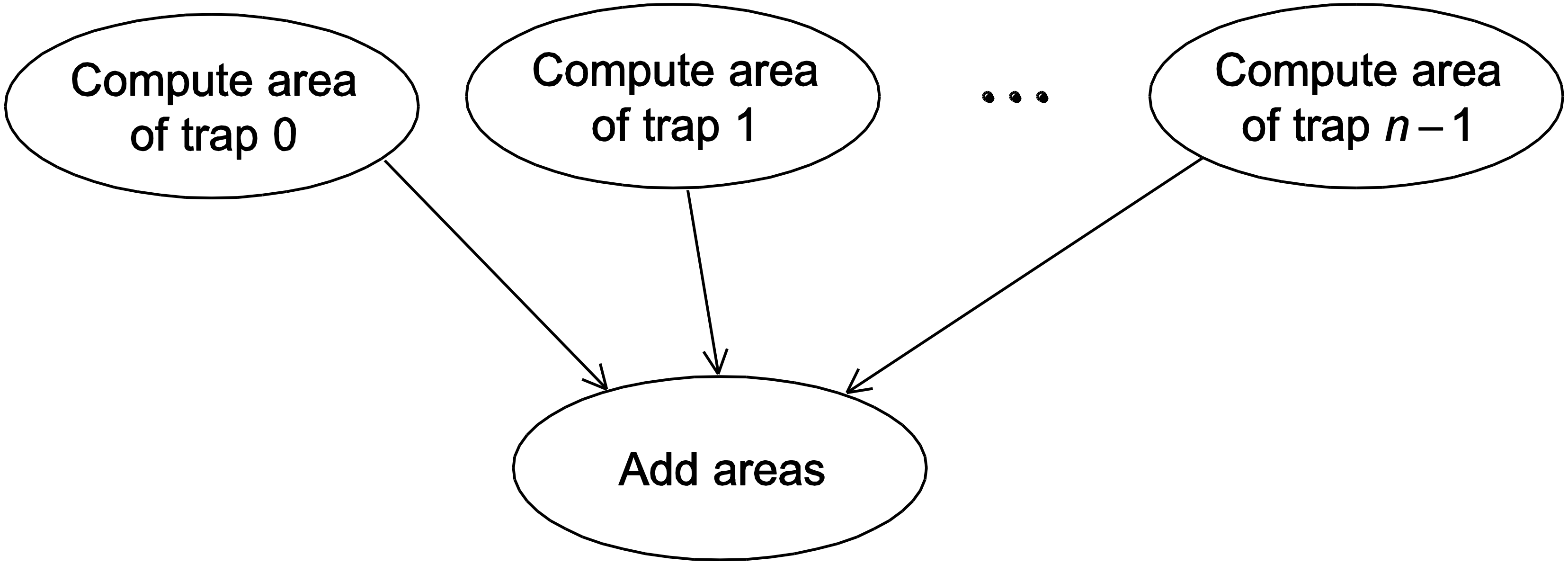
Parallel pseudo-code

```
Get a, b, n;
h = (b-a)/n;
local_n = n/comm_sz;
local_a = a + my_rank*local_n*h;
local_b = local_a + local_n*h;
local_integral = Trap(local_a, local_b, local_n, h);
if (my_rank != 0)
   Send local_integral to process 0;
else /* my_rank == 0 */
   total_integral = local_integral;
   for (proc = 1; proc < comm_sz; proc++) {
      Receive local_integral from proc;
      total_integral += local_integral;
   }
}
if (my_rank == 0)
   print result;
```

Compute the local area

Summation of local values

Tasks and communications for Trapezoidal Rule



First version (1)

```
int main(void) {
   int my_rank, comm_sz, n = 1024, local_n;
   double a = 0.0, b = 3.0, h, local_a, local_b;
   double local_int, total_int;
   int source;

   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

   h = (b-a)/n;          /* h is the same for all processes */
   local_n = n/comm_sz;  /* So is the number of trapezoids  */

   local_a = a + my_rank*local_n*h;
   local_b = local_a + local_n*h;
   local_int = Trap(local_a, local_b, local_n, h);

   if (my_rank != 0) {
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
            MPI_COMM_WORLD);
```

Use send/receive to sum

First version (2)

```
   } else {
      total_int = local_int;
      for (source = 1; source < comm_sz; source++) {
         MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         total_int += local_int;
      }
   }

   if (my_rank == 0) {
      printf("With n = %d trapezoids, our estimate\n", n);
      printf("of the integral from %f to %f = %.15e\n",
          a, b, total_int);
   }
   MPI_Finalize();
   return 0;
} /*  main  */
```

Use send/receive to sum

First version: Trapezoidal Rule of local area

```
double Trap(
        double left_endpt   /* in */,
        double right_endpt  /* in */,
        int    trap_count   /* in */,
        double base_len     /* in */) {
    double estimate, x;
    int i;

    estimate = (f(left_endpt) + f(right_endpt))/2.0;
    for (i = 1; i <= trap_count-1; i++) {
        x = left_endpt + i*base_len;
        estimate += f(x);
    }
    estimate = estimate*base_len;

    return estimate;
} /* Trap */
```

I/O handling in trapezoidal program

- **Most MPI implementations only allow process 0 in MPI_COMM_WORLD access to stdin.**
- **Process 0 must read the data (scanf) and send to the other processes.**

```
. . .
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

Get_data(my_rank, comm_sz, &a, &b, &n);

h = (b-a)/n;
. . .
```

Function for reading user input

```
void Get_input(
      int      my_rank    /* in  */,
      int      comm_sz    /* in  */,
      double*  a_p        /* out */,
      double*  b_p        /* out */,
      int*     n_p        /* out */) {
   int dest;

   if (my_rank == 0) {
      printf("Enter a, b, and n\n");
      scanf("%lf %lf %d", a_p, b_p, n_p);
      for (dest = 1; dest < comm_sz; dest++) {
         MPI_Send(a_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(b_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(n_p, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
      }
   } else { /* my_rank != 0 */
      MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
   }
}  /* Get_input */
```

Process 0 inputs parameters

Broadcast parameters