COMP/CS 605: Introduction to Parallel Computing
Topic : MPI: Trapezoid Function

Mary Thomas

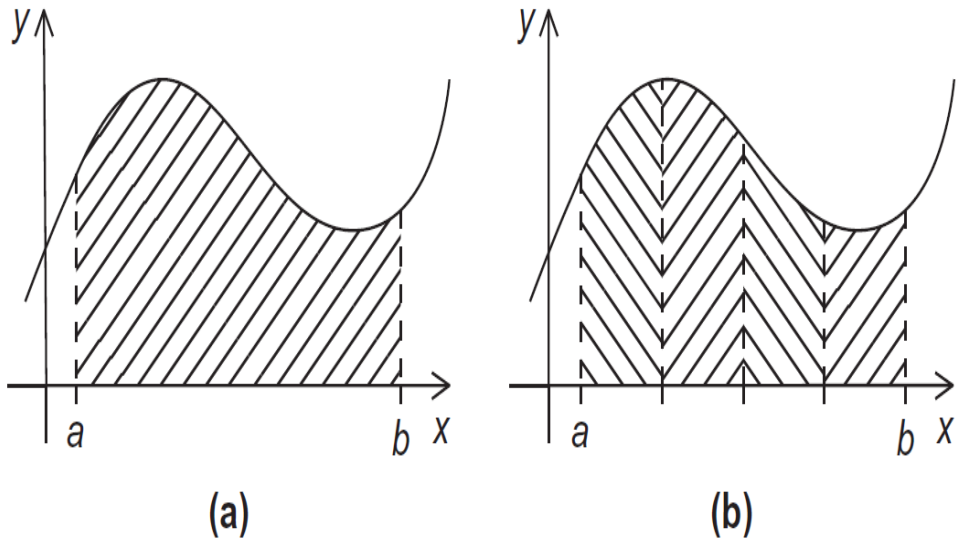
Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Presented: 02/13/16
Updated: 02/13/16

Table of Contents

The Trapezoid Rule for Numerical Integration

Solve the Integral: $\int_a^b F(x)dx$



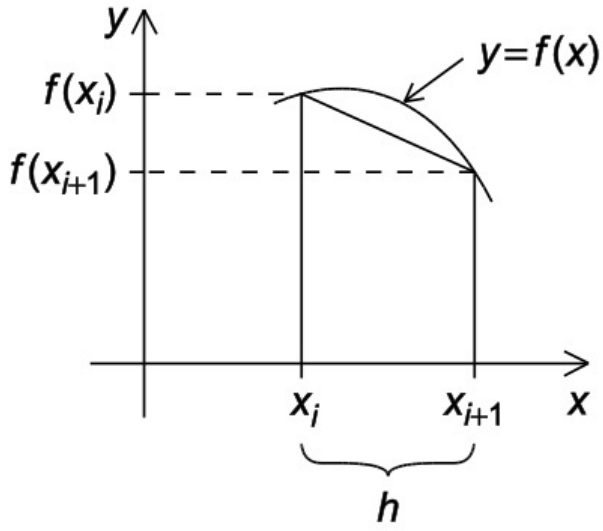
Where $F(x)$ can be any function of x: $f\left(x^2\right)$, $f\left(x^3\right)$
See Pacheco IPP (2011), Ch3.

Trapezoid Equations

Integral: $\int_a^b f(x)dx$

Area of 1 trapezoid: $= \frac{h}{2}\left\lfloor f(x_i) + f(x_{i+1})\right\rfloor$

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



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

Sum of Areas: $Area = h\left[\frac{f(x_0)}{2} + f(x_{i+1}) + f(x_{i+1}) + \ldots + f(x_{n-1})\frac{f(x_n)}{2}\right]$

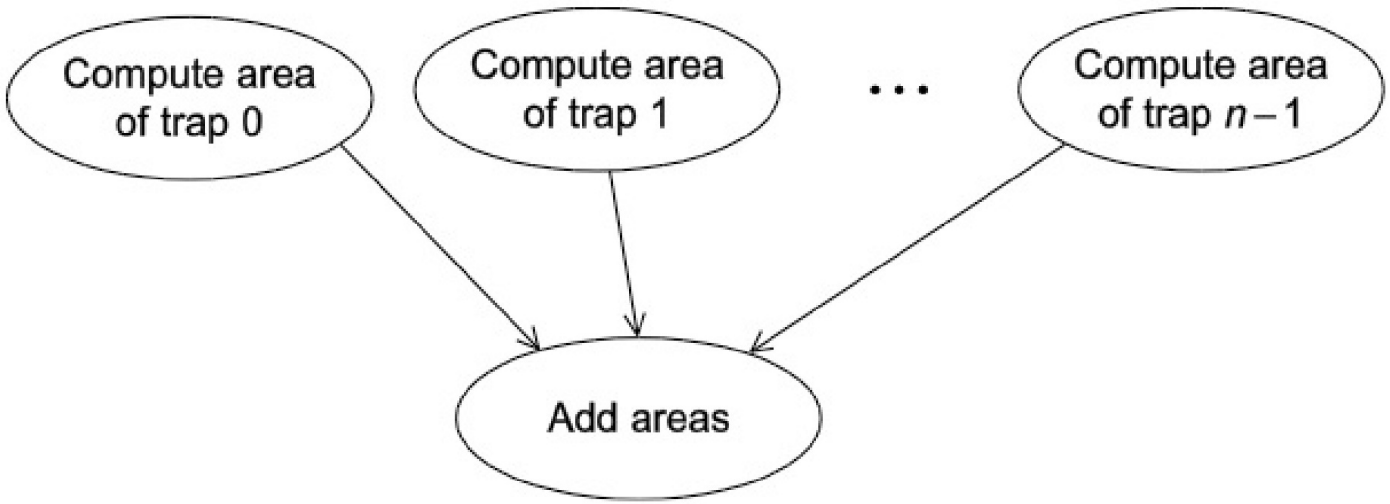
Pseudocode for a Serial Algorithm

```
/*  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
```

Paralellizing the Trapezoidal Rule

PCAM Approach

- Partition problem solution into tasks.
- Identify communication channels between tasks.
- Aggregate tasks into composite tasks.
- Map composite tasks to cores.



Two types of tasks:

- **Compute area of 1 trapezoid**
- **Compute area sums**

Parallel Pseudocode

```
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;
```

Datatype Example: Pacheco code: mpi-trap1.c

```
/* File:     mpi_trap4.c
 * Purpose:  Use MPI to implement a parallel version of the trapezoidal rule.
 *        This version uses collective communications and    MPI derived datatypes
 *        to distribute the input data and  compute the global sum.
 *
 * Input:    The endpoints of the interval of integration and the number  of trapezoids
 * Output:   Estimate of the integral from a to b of f(x) using the trapezoidal rule and n trapezoids.
 *
 * Compile:  mpicc -g -Wall -o mpi_trap4 mpi_trap.c
 * Run:      mpiexec -n <number of processes> ./mpi_trap
 *
 * Algorithm:
 *    1.  Each process calculates "its" interval of integration.
 *    2.  Each process estimates the integral of f(x) over its interval using the trapezoidal rule.
 *    3a. Each process != 0 sends its integral to 0.
 *    3b. Process 0 sums the calculations received from  the individual processes and prints the result.
 *
 * Note:  f(x) is all hardwired.
 * IPP:   Section 3.2.2 (pp. 96 and ff.)
 */
#include <stdio.h>
#include <mpi.h>     /* MPI routines, definitions, etc. */

/* Calculate local integral  */
double Trap(double left_endpt, double right_endpt, int trap_count, double base_len);

/* Function we're integrating */
double f(double x);
```

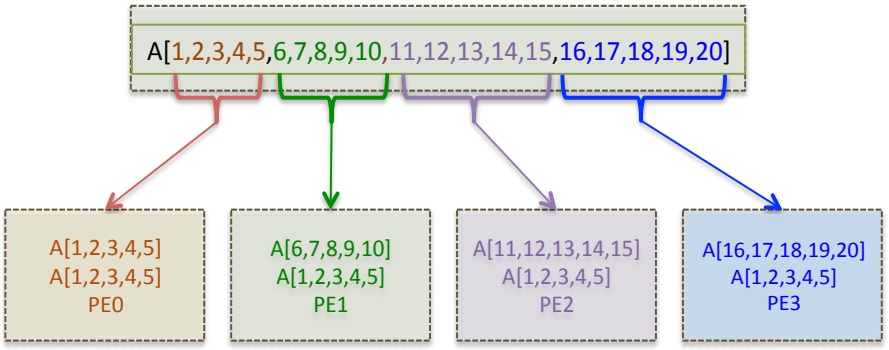
Datatype Example: Pacheco code: mpi-trap1.c

```
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;

   /* Let the system do what it needs to start up MPI */
   MPI_Init(NULL, NULL);

   /* Get my process rank */
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   /* Find out how many processes are being used */
   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  */

   /* Length of each process' interval of integration = local_n*h.  So my interval starts at: */
   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);

   /* Add up the integrals calculated by each process */
   if (my_rank != 0) {
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   } 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;
      }     }

   /* Print the result */
   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();        /* Shut down MPI */
```

Datatype Example: Pacheco code: mpi-trap1.c

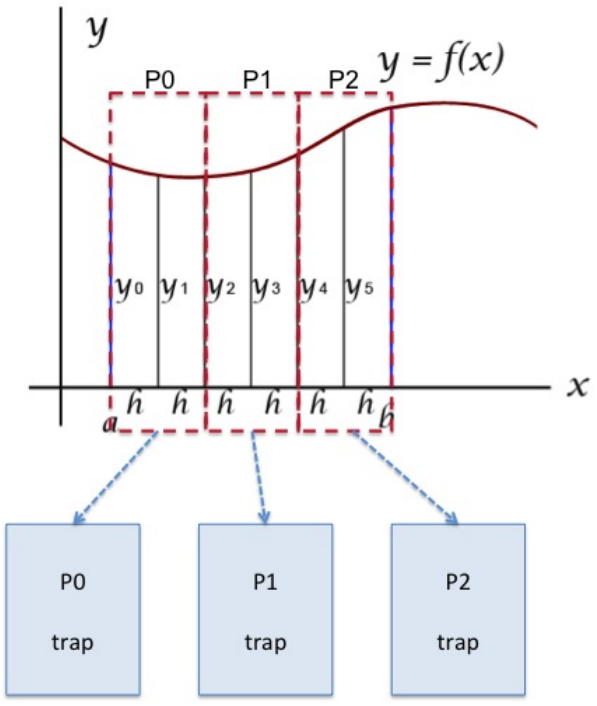
```
/*------------------------------------------------------------------
 * Function:     Trap
 * Purpose:      Serial function for estimating a definite integral
 *               using the trapezoidal rule
 * Input args:   left_endpt
 *               right_endpt
 *               trap_count
 *               base_len
 * Return val:   Trapezoidal rule estimate of integral from
 *               left_endpt to right_endpt using trap_count
 *               trapezoids
 */
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  */


/*------------------------------------------------------------------
 * Function:    f
 * Purpose:     Compute value of function to be integrated
 * Input args:  x
 */
double f(double x) {
   return x*x;
} /* f */
```

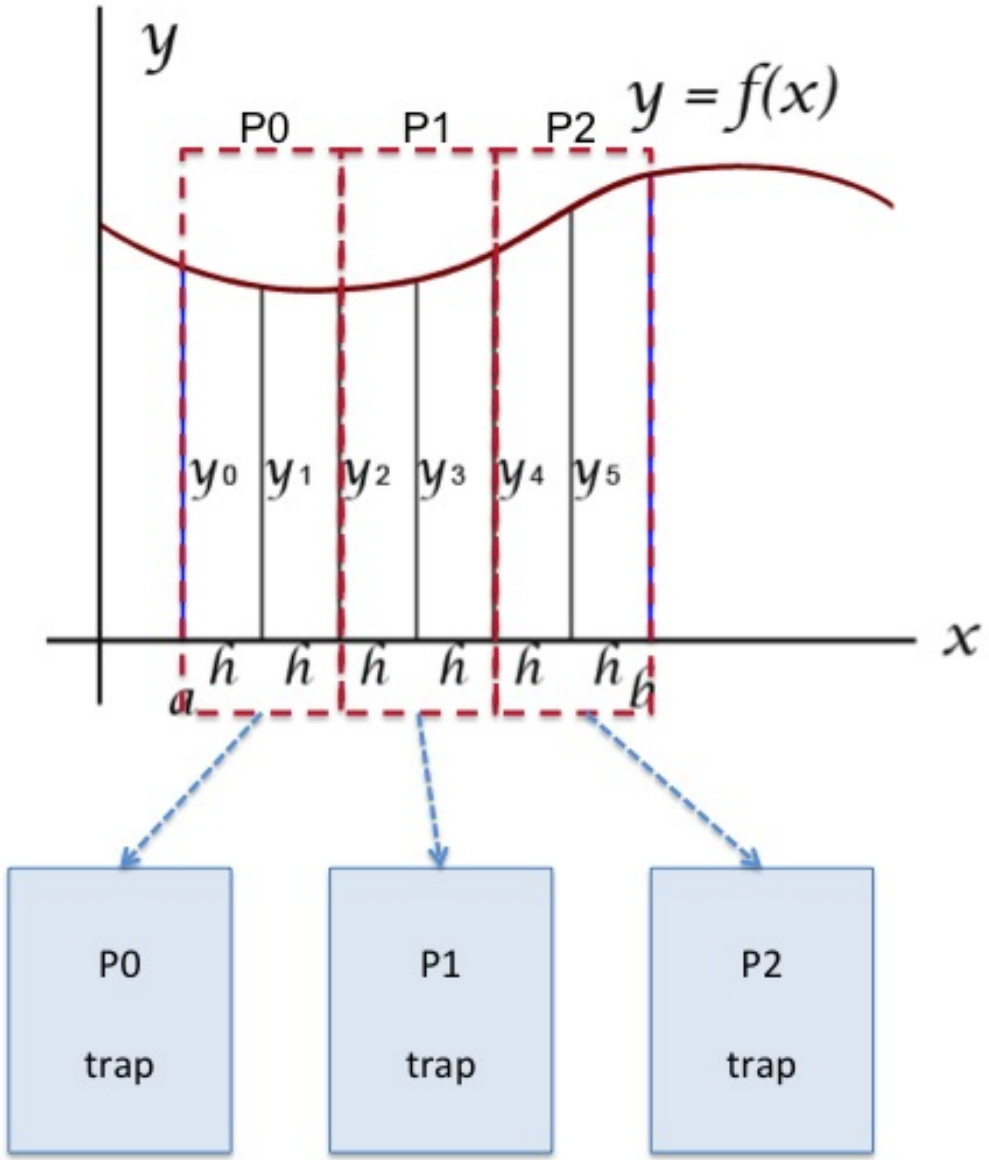
Data Distribution for Trap Function



1D Vector Data Distribution

- Parallel *trap* distributes $[x_i]$ data points to different processors.
- Each processor runs the same code but solves for different $[x_i]$ data
- This is a *1-D* data decomposition
- This is a *1-D* (virtual) processor arrangement




y
y = f(x)
P0
P1
P2
y0
y1
y2
y3
y4
y5
x
h
h
h
h
h
h
a
b
P0
trap
P1
trap
P2
trap

Job Example: Serial Hello World

```
#include <stdio.h>
#include <string.h>  /* For strlen             */
#include <mpi.h>     /* For MPI functions, etc */

const int MAX_STRING = 100;

int main(void) {
   char     greeting[MAX_STRING];  /* String storing message */
   int        comm_sz;               /* Number of processes    */
   int        my_rank;               /* My process rank        */

   /* Start up MPI */
   MPI_Init(NULL, NULL);

   /* Get the number of processes */
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

   /* Get my rank among all the processes */
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

   /* Print my message */
   printf("Greetings from process %d of %d!\n", my_rank, comm_sz);

   /* Shut down MPI */
   MPI_Finalize();

   return 0;
}  /* main */
```

Job Example: Serial Hello World

```
[tuckoo] mthomas% mpirun -np 8 ./simp_mpi_hi
Greetings from process 2 of 8!
Greetings from process 0 of 8!
Greetings from process 1 of 8!
Greetings from process 3 of 8!
Greetings from process 6 of 8!
Greetings from process 5 of 8!
Greetings from process 7 of 8!
Greetings from process 4 of 8!
```

Output is random, or non-deterministic

To build more flexible code, you need to read in variable data. One approach is to use the STDIN.

- 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.

Datatype Example: Pacheco code: mpi-trap2.c

```
/* File:     mpi_trap4.c
 * Purpose:  Use MPI to implement a parallel version of the trapezoidal rule.
 *        This version uses collective communications and    MPI derived datatypes
 *        to distribute the input data and  compute the global sum.
 *
 * Input:    The endpoints of the interval of integration and the number  of trapezoids
 * Output:   Estimate of the integral from a to b of f(x) using the trapezoidal rule and n trapezoids.
 *
 * Compile:  mpicc -g -Wall -o mpi_trap4 mpi_trap.c
 * Run:      mpiexec -n <number of processes> ./mpi_trap
 *
 * Algorithm:
 *    1.  Each process calculates "its" interval of integration.
 *    2.  Each process estimates the integral of f(x) over its interval using the trapezoidal rule.
 *    3a. Each process != 0 sends its integral to 0.
 *    3b. Process 0 sums the calculations received from  the individual processes and prints the result.
 *
 * Note:  f(x) is all hardwired.
 * IPP:   Section 3.2.2 (pp. 96 and ff.)
 */
#include <stdio.h>
#include <mpi.h>     /* MPI routines, definitions, etc. */

/* Get the input values */
void Get_input(int my_rank, int comm_sz, double* a_p, double* b_p, int* n_p);

/* Calculate local integral  */
double Trap(double left_endpt, double right_endpt, int trap_count, double base_len);

/* Function we're integrating */
double f(double x);
```

Datatype Example: Pacheco code: mpi-trap2.c

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

   /* Let the system do what it needs to start up MPI */
   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);      /* Get my process rank */
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);       /* Find out how many processes are being used */

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

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

   /* Length of each process' interval of integration = local_n*h.  So my interval starts at: */
   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);

   /* Add up the integrals calculated by each process */
   if (my_rank != 0) {
      printf("PE[%d] SEND: For %d trapezoids, local estimate=%f\n",my_rank, local_n, local_int);
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   }
   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);
         printf("PE[%d] RECV from PE[%d]: local estimate=%f \n", my_rank,source, local_int);
         total_int += local_int;
      }     }

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

   MPI_Finalize();        /* Shut down MPI */
   return 0;
   } /*  main  */
```

MPI_Send: Performs a blocking send

```
int MPI_Send(
        const void *sendbuf,
        int count,
        MPI_Datatype datatype,
        int dest,
        int tag,
        MPI_Comm comm )
```

Input Parameters:

- sendbuf: address of send buffer (choice)
- count: number of entries in buffer (integer)
- datatype: data type of buffer (handle)
- dest: rank of destination (integer)
- tag: message tag (integer)

MPI_Recv: Blocking receive for a message

```
int MPI_Recv(
        const void *recvbuf,
        int count,
        MPI_Datatype datatype,
        int source,
        int tag,
        MPI_Comm comm,
        MPI_Status *status )
```

Input Parameters:

- count: number of entries in buffer (integer)
- datatype: data type of buffer (handle)
- dest: rank of destination (integer)
- tag: message tag (integer)

Output Parameters:

- recvbuf: address of receive buffer (choice)
- status: status object (Status)

Datatype Example: Pacheco code: mpi-trap2.c

```
/*---------------------------------------------------------------------
 * Function:     Get_input
 * Purpose:      Get the user input:  the left and right endpoints
 *               and the number of trapezoids
 * Input args:   my_rank:  process rank in MPI_COMM_WORLD
 *               comm_sz:  number of processes in MPI_COMM_WORLD
 * Output args:  a_p:  pointer to left endpoint
 *               b_p:  pointer to right endpoint
 *               n_p:  pointer to number of trapezoids
 */
void Get_input(int my_rank, int comm_sz, double* a_p, double* b_p, int* n_p)
 {
   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 */
```

What happens if you are using the batch queuing system?
There is no way to capture *stdin*, so you have to read in data as command line args or from a file.

Datatype Example: Pacheco code: mpi-trap2.c

```
main(int argc, char *argv[]) {
  long cmdvar;
  char cptr[100];
  int  comm_sz;               /* Number of processes    */
  int  my_rank;               /* My process rank        */

   /* Start up MPI */
   MPI_Init(NULL, NULL);

   /* Get the number of processes */
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

   /* Get my rank among all the processes */
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  /* get a number from the command line */
  if (argc != 2) {
      printf("usage: hello-arg <integer number>\n");
      return 0;
  }
  cmdvar = strtol(argv[1], NULL, 10);

  /* Print my message */
  printf("Greetings from process %d of %d, cmd arg val=%ld\n", my_rank, comm_sz, cmdvar);

   /* Shut down MPI */
   MPI_Finalize();

   return 0;
}  /* main */
"hello-arg.c" 41L, 980C written
```

Datatype Example: Pacheco code: mpi-trap2.c

```
[mthomas@tuckoo:~/pardev/hello] mpirun -np 5 ./hello-arg 837298
Greetings from process 4 of 5, cmd arg val=837298
Greetings from process 0 of 5, cmd arg val=837298
Greetings from process 1 of 5, cmd arg val=837298
Greetings from process 2 of 5, cmd arg val=837298
Greetings from process 3 of 5, cmd arg val=837298

[mthomas@tuckoo:~/pardev/hello] mpirun -np 8 ./hello-arg 10000000000000000000
Greetings from process 6 of 8, cmd arg val=9223372036854775807
Greetings from process 7 of 8, cmd arg val=9223372036854775807
Greetings from process 0 of 8, cmd arg val=9223372036854775807
Greetings from process 3 of 8, cmd arg val=9223372036854775807
Greetings from process 4 of 8, cmd arg val=9223372036854775807
Greetings from process 5 of 8, cmd arg val=9223372036854775807
Greetings from process 1 of 8, cmd arg val=9223372036854775807
Greetings from process 2 of 8, cmd arg val=9223372036854775807
```