Logistics

• Don’t wholesale plagiarize code!!
• Routines okay to use as is:
  - copy_patch
  - accumulate_patch
  - mymalloc
  - tpool
• The “generic” structure of everyone’s code will be the same.
  - Style’s differ so each routine should be different.
    ▪ Variables, comments, etc.
Logistics [2]

• You may implement
  ➢ Your own stack procedure
  ➢ Your own pool code
  ➢ You must insure that no more than the specified worker threads are “active” at any given time.
    ▪ This is easy to check !!!!
      ♦ Make sure you do!!!

• Any questions ????

• MPI Homework
  ➢ Will be posted BEFORE the Pthreads homework is due
    ▪ The end of the semester is a real barrier!!
Logistics [3]

• Midterm is Graded
  - Grades Posted on the website
  - Exams back after they are copied
    ▪ Needed for the certification process 😊

• Grade Distribution
  - High 100%
  - Low 36.4%
  - Average 71.2%
    ▪ Attending 76%
    ▪ Not Attending 64%
## Grades

<table>
<thead>
<tr>
<th>University ID</th>
<th>Midterm</th>
<th>University ID</th>
<th>Midterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0486</td>
<td>63.64</td>
<td>3596</td>
<td>0.00</td>
</tr>
<tr>
<td>0772</td>
<td>40.00</td>
<td>4534</td>
<td>73.64</td>
</tr>
<tr>
<td>0998</td>
<td>74.55</td>
<td>5116</td>
<td>83.64</td>
</tr>
<tr>
<td>1177</td>
<td>90.91</td>
<td>5699</td>
<td>85.45</td>
</tr>
<tr>
<td>1282</td>
<td>100.00</td>
<td>6206</td>
<td>84.55</td>
</tr>
<tr>
<td>2370</td>
<td>36.36</td>
<td>6775</td>
<td>59.09</td>
</tr>
<tr>
<td>2506</td>
<td>63.64</td>
<td>7839</td>
<td>48.18</td>
</tr>
<tr>
<td>3259</td>
<td>78.18</td>
<td>8122</td>
<td>80.91</td>
</tr>
<tr>
<td>3387</td>
<td>84.55</td>
<td>8623</td>
<td>50.00</td>
</tr>
<tr>
<td>3420</td>
<td>73.64</td>
<td>9557</td>
<td>80.91</td>
</tr>
</tbody>
</table>
Logistics [4]

• Travel
  - First week of April sometime
    ▪ Don’t know the details yet
  - I’ve been asked to serve on the committee that will be planning the next 5 years of SciDAC
    ▪ Scientific Discovery for Advanced Computing
How does a broadcast work?
How does a Reduction work?
The Reduction problem

• If the value needs to be globally used there is a problem.
  ➢ MPI_Reduce stores the value at the root of the tree

• Option 1
  ➢ MPI_Reduce followed by
  ➢ MPI_Bcast
  ➢ Simple but not as efficient as possible

• Option 2
  ➢ MPI_Allreduce
    ▪ Does reduction
    ▪ Distributes result to all processes 😊
MPI_Allreduce

• Combines values from all processes and distribute the result back to all processes

• Synopsis
  - #include "mpi.h"
  - int MPI_Allreduce ( void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm )

• Input Parameters
  - sendbuf    starting address of send buffer (choice)
  - count      number of elements in send buffer (integer)
  - datatype   data type of elements of send buffer (handle)
  - op         operation (handle)
  - comm       communicator (handle)

• Output Parameter
  - recvbuf    starting address of receive buffer (choice)
How does a Allreduce work?
MPI_Finalize

- Terminates MPI execution environment

- Synopsis
  ```
  #include "mpi.h"
  int MPI_Finalize( )
  ```

- Notes
  - All processes must call this routine before exiting.
  - The number of processes running after this routine is called is undefined
    - it is best not to perform much more than a return after calling MPI_Finalize.
MPI_Abort

• Terminates MPI execution environment

• Synopsis
  ➢ #include "mpi.h"
  ➢ int MPI_Abort( MPI_Comm comm, int errorcode )

• Input Parameters
  ➢ comm communicator of tasks to abort
  ➢ errorcode error code to return to invoking environment

• Notes
  ➢ Terminates all MPI processes associated with the communicator comm;
    ▪ in most systems (all to date), terminates all processes.
Trapezoid Rule Numerical Integration

- Used to compute a definite integral under a curve
- Area = $\frac{1}{2}$ base (left height + right height)
Trapezoid.c [1]

/* This is program to compute an integral under a parabola and above the x axis using the trapezoidal rule numerical integration scheme. The parabola is above the X axis in the range of 0 to 4. This component is hard coded. A mechanism to add extra work is used to scale the work to proportion of the parallelism.--------*/
#include <stdio.h>
#include <math.h>
#include "mpi.h"

/*- prototype for local integral contribution ------*/
double trapezoid(double local_left, double local_right, double base_length, int local_numTrap);

/*-- prototype for integrating function --*/
double myfunc(double x);
int main(int argc, char** argv) { … }
**Trapezoid.c [2]**

double trapezoid(double local_left, double local_right, double base_len, int local_num_traps)
/*-- function to compute the integral over a region using the trapezoidal rule --*/
{  int i;
    double my_integral, x;
    my_integral = (myfunc(local_left)+myfunc(local_right))/2.0;
    x = local_left;
    for (i = 1; i <= local_num_traps-1; i++) {
      x       += base_len;
      my_integral += myfunc(x);
    }
    my_integral *= base_len;
    return my_integral;
}
double myfunc_raw(double x)
/*- routine to compute the value of the parabola ---*/
{  double value;
    value = (double)-2.0;
    value *= (x-(double)2);
    value *= (x-(double)2);
    value += (double)8;
    return value;
} /* y = -2(x-x0) + y0 */

double myfunc(double x)
/*-- dummy routine to add work ---*/
{  double value;  int i;
    for (i=0;i<10000;i++)
      value = myfunc_raw(x);
    return value;
}
Trapezoid.c [4]

int main(int argc, char** argv)
{
    /* MPI and parallel variables */
    int nodeid = 0; /* My process rank */
    int nproc = 0; /* The number of processes */
    int master = 0; /* data sent to master proc (e.g., 0) */
    /*- timing variables */
    double mytime0, mytime1, mytime;
    double mytime_min; /* minimum time on all procs */
    double mytime_max; /* maximum time on all procs */
    double mytime_sum; /* sum of time on all procs */
    double mytime_ave; /* average time across procs */
Trapezoid.c [5]

/*--- full integral values ---*/
int num_integrations = 0;
int num_traps_init = 32; /* Initial Number of trapezoids */
int num_traps; /* Number of trapezoids for integral */
double left_end = 0.0; /* Left boundary for integral */
double right_end = 0.0; /* Right boundary for integral */
double total_integral = 0.0; /* Total integral */
double total_last = 0.0; /* last total integral */
double base_length = 0.0; /* Trapezoid base length */
double delta = 1000; /* change in area, initial value large */
double thresh = 1.0e-8; /* threshold */

/*--- Local integral information -----*/
int local_num_traps; /* Number of trapezoids local */
int extra; /* computed modulus of number of trapezoids */
double local_left; /* Left endpoint my process */
double local_right; /* Right endpoint my process */
double local_integral; /* Integral in local interval */
Trapezoid.c [6]

/*-- Initialize MPI library system-----------------------------*/
MPI_Init(&argc, &argv);
/*- Determine nodeid and number of processes --------*/
MPI_Comm_rank(MPI_COMM_WORLD, &nodeid);
MPI_Comm_size(MPI_COMM_WORLD, &nproc);
left_end    = (double)0.0;  /* doing and integral from 0 to 4 */
right_end   = (double)4.0;
num_traps   = num_traps_init;
MPI_Barrier(MPI_COMM_WORLD)) {
if(nodeid == master) {
    (void)printf("\n\nTrapizoid Integration on %d
process(es)\n",nproc);
    (void)fflush(stdout);
}
mytime0 = MPI_Wtime();
Trapezoid.c [7]

while (fabs(delta) > thresh) {
    /*- ensure total number of trapezoids is divisible by nproc --*/
    extra = num_traps % nproc;  /* umber of extra trapezoids */
    if (extra != 0) num_traps += (nproc - extra);
    local_num_traps = num_traps/nproc;  /* per process*/
    base_length = (right_end - left_end)/num_traps;
    /*-----------------------------------------*/
    Each process computes the local integral
    /*-----------------------------------------*/
    local_left  = left_end + nodeid*local_num_traps*base_length;
    local_right = local_left + local_num_traps*base_length;
    local_integral = trapezoid(local_left, local_right,
                                base_length, local_num_traps);
Now do a global sum reduction to get the total integral

```c
MPI_Allreduce(&local_integral, &total_integral, 1,
            MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD)
```

Print the result

```c
num_integrations++;
if (nodeid == master) {
    (void)printf("With %7d trapezoids, the estimated ",
                 num_traps);
    (void)printf("integral from %.2e to %.2e is %.10e
<iteration: %3d>\n",
                 left_end, right_end, total_integral,num_integrations);
    (void)fflush(stdout);
}
delta = total_last - total_integral;
num_traps *= 2;
total_last = total_integral;  }
```
Trapezoid.c [9]

/*- end of while loop. get time stamps ------*/
mytime1 = MPI_Wtime();
mytime = mytime1 - mytime0;
/*-- compute the minimum time used -----*/
MPI_Reduce(&mytime,&mytime_min,1,
    MPI_DOUBLE,MPI_MIN,master,
    MPI_COMM_WORLD);
/*-- compute the maximum time used -*/
MPI_Reduce(&mytime,&mytime_max,1,
    MPI_DOUBLE,MPI_MAX,master,
    MPI_COMM_WORLD);
/*-- compute the average time used ---*/
MPI_Reduce(&mytime,&mytime_sum,1,
    MPI_DOUBLE,MPI_SUM,master,
    MPI_COMM_WORLD);
mytime_ave = mytime_sum/nproc;
if (nodeid == master) {
    (void)printf(" Total number of integrations: %d
    required for a threshold of %.4e using %d
    process(es)\n", num_integrations, thresh, nproc);
    (void)printf(" Total Time:master: %.3f seconds\n", mytime);
    (void)printf(" Minimum time : %.3f
    seconds\n", mytime_min);
    (void)printf(" Average time : %.3f
    seconds\n", mytime_ave);
    (void)printf(" Maximum time : %.3f
    seconds\n", mytime_max);
    (void)fflush(stdout);
}

/*-- Shutdown MPI --*/    MPI_Finalize();}
Trapezoid Output

Trapezoid Integration on 8 process(es)
With 32 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1312500000e+01  <iteration: 1>
With 64 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1328125000e+01  <iteration: 2>
With 128 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1332031250e+01  <iteration: 3>
With 256 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.13332051953e+01  <iteration: 4>
With 512 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333312988e+01  <iteration: 5>
With 1024 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333332054e+01  <iteration: 6>
With 2048 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333128e+01  <iteration: 7>
With 4096 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333313e+01  <iteration: 8>
With 8192 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333328e+01  <iteration: 9>
With 16384 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333332e+01  <iteration: 10>
With 32768 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333333e+01  <iteration: 11>
With 65536 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333333e+01  <iteration: 12>
With 131072 trapezoids, the estimated integral from 0.00e+00 to 4.00e+00 is 2.1333333333e+01  <iteration: 13>

Total number of integrations: 13 required for a threshold of 1.0000e-08 using 8 process(es)
Total Time:master: 242.404 seconds
Minimum time : 242.404 seconds
Average time : 242.563 seconds
Maximum time : 242.715 seconds
Trapezoid Performance [1]

SpeedUp on Fatman

Number of Processes

SpeedUp

Ideal

SpeedUp

Ideal

0.00
2.00
4.00
6.00
8.00

0 2 4 6 8

3/11/2005
ComS 425
Spring 2005
28 of 51: Lecture 22
Trapezoid Performance [2]

Efficiency on Fatman

Efficiency

Number of Processes

Trapezoid Performance [3]

SpeedUp on Alice

Number of Processes

SpeedUp

Ideal

SpeedUp
Trapezoid Performance [4]

Efficiency on Alice

Number of Processes

Efficiency

Efficiency

Ideal
Trapezoid Performance [5]

Trapezoid on IBM SP

Number of processors

Speedup

ideal
speedup

0 8 16 24 32 40 48 56 64

0 8 16 24 32 40 48 56 64
Trapezoid Performance [6]

Trapezoid on IBM SP

Efficiency

number of processors
Input and Environment

• The only node that will have guaranteed access to
  ➢ stdin
  ➢ argv, argc
  ➢ Environment variables

  Is the Master node.

• Some implementations pass argv, argc

• Safest thing to do is:
  ➢ Master collects “data”
  ➢ Broadcasts “data” to all processes.
Input Parsing with MPI

if (nodeid == 0){
    printf("Enter value and number\n");
    scanf("%f %d", val_ptr, num_ptr);
    for (dest = 1; dest < nproc; dest++){
        tag = 300;  MPI_Send(val_ptr, 1, MPI_FLOAT, dest, tag,
                             MPI_COMM_WORLD);
        tag = 400;  MPI_Send(num_ptr, 1, MPI_INT, dest, tag,
                             MPI_COMM_WORLD);
    }
} else {
    tag = 300;  MPI_Recv(val_ptr, 1, MPI_FLOAT, source, tag,
                         MPI_COMM_WORLD, &status);
    tag = 400;  MPI_Recv(num_ptr, 1, MPI_INT, source, tag,
                         MPI_COMM_WORLD, &status);
}
Gather and Scatter

• **Gather**
  - Collects data from all processes to one process

• **Scatter**
  - Puts data from one process to all other processes.
## Scatter Process

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0</td>
<td>X[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 1</td>
<td>X[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 2</td>
<td>X[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 3</td>
<td>X[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gather Process
MPI_Gather

- Gathers together values from a group of processes
- Synopsis
  - #include "mpi.h"
  - int MPI_Gather ( void *sendbuf, int sendcnt, MPI_Datatype sendtype, 
    void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, 
    MPI_Comm comm )
- Input Parameters
  - sendbuf starting address of send buffer (choice)
  - sendcount number of elements in send buffer (integer)
  - sendtype data type of send buffer elements (handle)
  - recvcount number of elements for any single receive 
    (integer, significant only at root)
  - recvtype data type of recv buffer elements (only root)
  - root rank of receiving process (integer)
  - comm communicator (handle)
- Output Parameter
  - recvbuf address of receive buffer
MPI_Scatter

- sends data from one task to all other tasks in a group

Synopsis

```c
#include "mpi.h"
int MPI_Scatter (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Input Parameters

- `sendbuf`: address of send buffer
- `sendcount`: number of elements sent to each process
- `sendtype`: data type of send buffer elements
- `recvcount`: number of elements in receive buffer (integer)
- `recvtype`: data type of receive buffer elements (handle)
- `root`: rank of sending process (integer)
- `comm`: communicator (handle)

Output Parameter

- `recvbuf`: address of receive buffer (choice)
Data Gathered and Scattered.

- **Scattered Data**
  - Buffer split up into “Nproc” segments.
  - Each segment stored in process rank order.
  - First “chunk” goes to process zero, one, etc.

- **Gathered Data is stored in process rank order.**
  - First “chunk” comes from process zero!
  - “recv_count” of them from every process and they are “recv_type” data.

- **What if I want to gather to “all” nodes instead of just one?**
MPI_Allgater

• Gathers data from all tasks and distribute it to all

• Synopsis
  ➢ #include "mpi.h"
  ➢ int MPI_Allgater ( void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm )

• Input Parameters
  ➢ sendbuf starting address of send buffer (choice)
  ➢ sendcount number of elements in send buffer (integer)
  ➢ sendtype data type of send buffer elements (handle)
  ➢ recvcount number of elements from any process
  ➢ recvtype data type of receive buffer elements (handle)
  ➢ comm communicator (handle)

• Output Parameter
  ➢ recvbuf address of receive buffer (choice)
Translation Example

```
for (root = 0; root < nproc; root++)
    MPI_Gather(local_vec, n/p, MPI_DOUBLE,
               global_vec, n/p, MPI_DOUBLE,
               root, MPI_COMM_WORLD)
```

Is equivalent to:

```
MPI_Allgather (local_vec, n/p, MPI_DOUBLE,
               global_vec, n/p, MPI_DOUBLE,
               MPI_COMM_WORLD)
```
Derived Data Types

• **C does not guarantee that:**
  - Elements of a structure are contiguous
  - Subsequent “malloc” calls represent data that is contiguous.

• **Need to derive a data type for a message.**

• **Example.**
  - `left_end` (double)
  - `right_end` (double)
  - `num_traps` (int)
What needs to be known?

- Three elements will be sent/received.
- The three elements have the types
  - double
  - double
  - int
- The address of each can be determined
  - &left_end
  - &right_end
  - &num_trap
The MPI interface needs to know

- There are three elements to be transmitted
- The elements are a double, a double, and an int, respectively
- The first element is displaced 0 bytes from the beginning of the message buffer.
- The second element is displaced X bytes from the first element or beginning of the message.
- The third element is displaced Y bytes from the first element or beginning of the message.
A MPI derived datatype

- $[(t_0, d_0), (t_1, d_1), (t_2, d_2) \ldots (t_{n-1}, d_{n-1})]$  
  - A sequence of 2 element tuples.
    - The first is the MPI datatype
    - The second element is the displacement from the beginning of the message.
- This is can get ugly and is a lot of work.
  - Necessary in some cases!
- See why I suggested “linear” arrays
- Building derived datatypes is like building the structure for the argument to a pthreads routine.
  - It is overhead.
Example Derived Datatype

void Build_derived_type(
    float* a_ptr     /* in */,
    float* b_ptr     /* in */,
    int* n_ptr       /* in */,
    MPI_Datatype* mesg_mpi_t_ptr  /* out */) {
    /* pointer to new MPI type */
    /* The number of elements in each "block" of the new type. For us, 1 each. */
    int block_lengths[3];
    /* Displacement of each element from start of new type. The "d_i's." */
    /* MPI_Aint ("address int") is an MPI defined C type. Usually an int. */
    MPI_Aint displacements[3];
Example Derived Datatype [2]

/* MPI types of the elements. The "t_i's." */
MPI_Datatype typelist[3];

/* Use for calculating displacements */
MPI_Aint start_address;
MPI_Aint address;


/* Build a derived datatype consisting of */
/* two floats and an int */
typelist[0] = MPI_FLOAT;
typelist[1] = MPI_FLOAT;
typelist[2] = MPI_INT;
Example Derived Datatype [3]

/* First element, a, is at displacement 0 */
displacements[0] = 0;

/* Calculate other displacements relative to a */
MPI_Address(a_ptr, &start_address);

/* Find address of b and displacement from a */
MPI_Address(b_ptr, &address);
displacements[1] = address - start_address;

/* Find address of n and displacement from a */
MPI_Address(n_ptr, &address);
displacements[2] = address - start_address;
Example Derived Datatype [4]

/* Build the derived datatype */
MPI_Type_struct(3, block_lengths, displacements,
    typelist, msg_mpi_t_ptr);

/* Commit it -- tell system we'll be using it for */
/* communication. */
MPI_Type_commit(msg_mpi_t_ptr);
}