Lecture 4

Programming with Message Passing: Applications and Performance
Announcements

• Quiz #1 in section on 10/13
• Midterm: evening of 10/30, 7:00 to 8:20 PM
• Wednesday’s office hours start at 1.30 (rest of the quarter)
Today’s lecture

• Interconnect

• Our first MPI applications
  – Numerical quadrature
  – Measuring communication performance
  – N-body simulation

• Reporting Performance

• Under the hood of MPI
Interconnect

• Generic interconnect hides the topology

Ring
Toroidal mesh
Performance characteristics

- **Diameter**: maximum distance between any 2 points in the network
- **Bisection bandwidth**: collective bandwidth between two “halves” of the network; split the graph into two equal parts and measure the capacity of the cut edges
Ring

- P element array with end-around connection
- Diameter is cut in half by the end around connection
- Bisection bandwidth?
- Broadcast running time?
Mesh

- \( \sqrt{P} \times \sqrt{P} \) array if square
- Diameter ?
- Bisection bandwidth?
- Broadcast algorithm?
Toroidal mesh

- End around connections on rows and columns
- Diameter ?
- Bisection bandwidth ?
- Broadcast running time ?
Crossbar

- Expensive – all points connected
- Diameter?
- Bisection bandwidth?
- Broadcast algorithm?
Multi-stage networks

- Switching is performed in stages
- Sometimes the stages are inductively constructed
- Often redundant paths
An Omega Network

- The network is constructed from switch modules.
- A module can swap the inputs, or pass them through unchanged.
Switch contention

- If two messages require the same modules at the same time, they contend for that module.
- Performance penalty, since access is serialized.
Today’s lecture

• Interconnect

• Our first MPI applications
  – Numerical quadrature
  – Measuring communication performance

• Under the hood of MPI
Numerical Quadrature

• Compute a numerical approximation to the definite integral

\[ \int_{a}^{b} f(x) \, dx \]

using the trapezoidal rule
How the trapezoidal rule works

- Divide the interval \([a,b]\) into \(n\) segments of size \(h=1/n\)
- Approximate the area under an interval using a trapezoid
- Area under the \(i^{th}\) trapezoid
  \[
  \frac{1}{2} \left( f(a+i\times h) + f(a+(i+1)\times h) \right) \times h
  \]
- Area under the entire curve
  \(\approx\) sum of all the trapezoids

\[ y = f(x) \]

\[ a \quad \frac{a}{h} \quad \frac{a}{2h} \quad \frac{a}{3h} \quad b \]

\[ a+i\times h \quad a+(i+1)\times h \]
Reference material

• For a discussion of the trapezoidal rule
  http://metric.ma.ic.ac.uk/integration/techniques/definite/numerical-methods/trapezoidal-rule

• A applet to carry out integration

• Code (from Pacheco hard copy text)
  PUB = /export/home/cs160x-public
  Serial Code
    PUB/Pacheco/ppmpi_c/chap04/serial.c
  Parallel Code
    PUB/Pacheco/ppmpi_c/chap04/trap.c
Serial code (Following Pacheco)

main() {
    float f(float x) { return x*x; } // Function we're integrating

    float h = (b-a)/n; // h= trapezoid base width
    // a and b: endpoints
    // n = # of trapezoids

    float integral = (f(a) + f(b))/2.0;

    float x; int i;

    for (i = 1, x=a; i <= n-1; i++) {
        x += h;
        integral = integral + f(x);
    }
    integral = integral*h;
}
The parallel algorithm

• Decompose the integration interval into sub-intervals, one per processor
• Each processor computes the integral on its local subdomain
• Processors combine their local integrals into a global one
First version of the parallel code

```c
local_n = n/p; // Number of trapezoids; assume p divides n evenly
float local_a = a + my_rank*local_n*h,
        local_b = local_a + local_n*h,
        integral = Trap(local_a, local_b, local_n, h);

if (my_rank == 0) { // Sum the integrals calculated by all the processes
    total = integral;
    for (source = 1; source < p; source++) {
        MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, WORLD, &status);
        total += integral;
    }
} else
    MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, WORLD);
```
Can we improve the running time?

- The result does not depend on the order in which the sums are taken.
- We use a linear time algorithm to accumulate contributions, but there are other orderings.

```c
for (source = 1; source < p; source++) {
    MPI_Recv(&integral, 1, MPI_FLOAT, 
              MPI_ANY_SOURCE, tag, 
              WORLD, &status);
    total += integral;
}
```
Collective communication

• We can often improve performance by taking advantage of global knowledge about communication
• Instead of using point to point communication operations to accumulate the sum, use collective communication
Collective communication in MPI

• Collective operations are called by all processes in a communicator

• Broadcast: distribute data from a designated “root” process to all others in a communicator
  MPI_Bcast(in, count, type, root, comm)

• Reduce: combine data from all processes in communicator and returns it to one process
  MPI_Reduce(in, out, count, type, op, root, comm)
Broadcast

• One process transmits of $m$ pieces of data to all the $(p-1)$ others

• Linear algorithm performs $p-1$ sends of length $m$
  – Cost is $(p-1)(\alpha + \beta m)$

• Another approach is to use the hypercube algorithm, which has a logarithmic running time
What is a hypercube?

- A hypercube is a d-dimensional graph with $2^d$ nodes
- A 0-cube is a single node, 1-cube is a line connecting two points, 2-cube is a square, etc
- Each node has d neighbors
Properties of hypercubes

- A hypercube with p nodes has \( \lg(p) \) dimensions
- *Inductive construction*: we may construct a d-cube from two (d-1) dimensional cubes
- **Diameter**: What is the maximum distance between any 2 nodes?
- **Bisection bandwidth**: How many cut edges (mincut)
Bookkeeping

- Label nodes with a binary reflected grey code
  [Link](http://www.nist.gov/dads/HTML/graycode.html)

- Neighboring labels differ in exactly one bit position

\[ 001 = 101 \boxtimes e_2, \quad e_2 = 100 \]
Hypercube broadcast algorithm with \( p = 4 \)

- Processor 0 is the root, sends its data to its hypercube “buddy” on processor 2 (10)
- Proc 0 & 2 send data to respective buddies
Reduction

• We may use the hypercube algorithm to perform reductions as well as broadcasts
• Use variant of reduction
  \texttt{Allreduce()}  
• Everyone obtains a copy of the reduced result
• This is equivalent to a \texttt{Reduce()} + \texttt{Bcast()}
• A clever algorithm performs an Allreduce in one phase rather than having perform separate reduce and broadcast phases
Improved parallel code

```c
int local_n = n/p;

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

MPI_Allreduce( &integral, &total, count=1,
               MPI_FLOAT, MPI_SUM, WORLD)
```
How do we model communication performance?

• The simplest communication cost model is "Alpha-Beta Model: transfer time = α + βn"
  
  α = message startup time
  β = 1/ peak bandwidth (bytes per second)
  n = message length
What are startup and bandwidth?

• The startup term dominates when the message is sufficiently short
  \[ \alpha > \beta n \implies n < \frac{\alpha}{\beta} \]

• The bandwidth term dominates when the message is sufficiently long
  \[ n > \frac{\alpha}{\beta} \]

• We refer to this message threshold as the half power point \( n_{1/2} \)
Half power point

- Gives a relationship between startup and bandwidth
- $n_{1/2} =$ message size required to achieve $\frac{1}{2}$ peak bandwidth (1/β)
- In theory, this occurs when $\alpha = \beta n_{1/2}$
- But look closely!
- For NPACI Blue Horizon, the actual value of $n_{1/2} \approx 100$ KB
Evaluating communication performance with the Ring microbenchmark

- Treat the $p$ processors as if connected in a logical ring and pass messages around
- Neighbors of processor $k$ are:
  - $(k + 1) \mod p$
  - $(k + p - 1) \mod p$
- $P_0$ times the cost of sending a message around the ring
- What are we measuring?
- What are we ignoring?
Communication Bandwidth on Blue Horizon

- 390 MB/sec
- \( N = 4\text{MB} \)
- \( N_{1/2} \approx 100\text{KB} \)
Performance measurement

- Use MPI_Wtime() to measure wall clock time
- Ignore transient behavior
  - Measure sufficiently long periods of representative steady state behavior
  - "Warm" up the program by running it first without collecting timing information
  - Repeat the measurements several times, and report the shortest times
  - Note any outliers
Measurement technique with Ring

for (int len = 1, l=0; len <= maxSize; len *= 2, l++)
if (myid == 0) {
  // (WARM UP CODE)
  const double start = MPI_Wtime();
  for (int i = 0; i < trips; i++) {
    PROCESSOR 0 CODE
  }
  const double delta = MPI_Wtime() - start;
  Bandwidth = (long)((trips*len*nodes)/ delta /1000.0);
} else { // myid != 0
  // (WARM UP CODE)
  for (int i = 0; i < trips; i++) {
    ALL OTHER PROCESSORS
  }
}
The Ring program

Processor 0:

```
MPI_Request req;
MPI_Irecv(buffer, len, MPI_CHAR, (rank + p - 1)%p, 
tag, MPI_COMM_WORLD, &req);
MPI_Send(buffer, len, MPI_CHAR, (rank + 1) % p, 
tag, MPI_COMM_WORLD);
MPI_Status status;
MPI_Wait(&req,&status);
```

All others:

```
MPI_Status status1;
MPI_Recv(buffer, len, MPI_CHAR, (rank + p - 1)%p, 
tag, MPI_COMM_WORLD, &status1);
MPI_Send(buffer, len, MPI_CHAR, (rank+1)%p, 
tag, MPI_COMM_WORLD);
```
Reporting and Displaying Performance

• Give the viewer sufficient information to…
  – Draw their own conclusions
  – Reproduce your results

• Tabulate and display the results fairly
  – Avoid misleading techniques
  – See the Bailey paper for examples of how not to display and report performance data
Challenges to measuring performance

• Reproducibility
  – Transient system operating conditions
  – Differing systems or program configuration

• Measurements are imprecise
  – “Heisenberg uncertainty principle:” measurement technique may affect performance
  – Variations in performance are inevitable; OK if we can explain and tolerate them
  – Overheads and inaccuracy

• Explain anomalous behavior, but ignore anomalies that are not significant
Complications

• Cost of measuring a full run is prohibitive
  – Ignore startup code if you plan to run for a much longer time in production

• Transient behavior
  – Repeat your measurements
  – “Warm up” the coded before collecting measurements
  – Ignore outliers unless their behavior is important to you
  – Average time, maximum time, minimum time?
Measurement collection

• Report the *best* timings
  ▶ Repeat results 3 to 5 times until at least 2 measures agree to within… 5%, 10%
  ▶ Report the minimum time

• Also report outliers

• A scatter plot or error bar can be useful
Timing collection

• Measures of time
  ▶ Elapsed, or “wall clock” time
  ▶ CPU time = system + user time
  ▶ Overhead, resolution, and quantization effects

• Measurement tools
  ▶ Unix time command does a reasonable job for long-running programs
  ▶ Hardware performance monitors
  ▶ System clocks
    ▪ Often platform dependent, especially library routines
    ▪ MPI provides MPI_Wtime( ), elapsed or “wall clock” time
Qualifying measurements

• Specify version and options
  – Compiler
  – Operating system
  – Numerical libraries

• Establish appropriate operating conditions
  – Program inputs
  – System environment variables
  – Dedicated system access

uname -a
Linux valkyrie.ucsd.edu 2.6.9-5.0.5.ELsmp #1 SMP Wed Apr 20 00:16:40 BST 2005 i686 i686 i386 GNU/Linux

g++ -v
Reading specs from /usr/lib/gcc/i386-redhat-linux/3.4.5/specs
Configured with: ....
host=i386-redhat-linux
Thread model: posix
gcc version 3.4.5 20051201 (Red Hat 3.4.5-2)
Under the hood of MPI

- If there is not a pending receive, then an incoming message is placed in an anonymous system buffer.
- When the receive gets posted, the message is moved into the user specified buffer.
- Ring uses non-blocking communication to avoid extra copy.
- For more information see “Buffering and Safety” in *MPI: The Complete Reference*, by Marc Snir et al.

Rendezvous protocol

- When a long message is to be sent, MPI first checks if the recipient has sufficient storage to receive the message.
- If so, then it sends the message.
Eager limits

• In an *eager* implementation, we just send the message
• In practice, MPI implementations switch between the two modes
• The *eager limit* is the longest message that can be sent in eager mode