Lecture 4

Communication performance
An application of MPI
Collective Communication

Announcements

• Assignment #1 due on Tuesday
• 10% bonus for handing in today in class
Communication cost model

- Communication performance is a major factor in determining the overall performance of an application
- The simplest communication cost model is
  \[ \text{transfer time} = \alpha + \beta n \]
  - \( \alpha \) = message startup time
  - \( \beta = 1/\text{peak bandwidth (bytes per second)} \)
  - \( n \) = message length
- LogP model (Culler et al, 1993), is more precise, but the \( \alpha - \beta \) model is often good enough
- Note that these models ignore some important effects: switch and processor contention

Startup and bandwidth

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

- Formally $n_{1/2}$ is the message size required to achieve $1/2$ peak bandwidth ($1/\beta$)
- This occurs when $\alpha = \beta n_{1/2}$
- In practice, $n_{1/2}$ may deviate from the formula
- For NPACI Blue Horizon, $N_{1/2} \approx 100$ KB
- In HW#2, you’ll measure $n_{1/2}$ for Valkyrie

Communication Bandwidth on Blue Horizon

![Graph showing communication bandwidth on Blue Horizon with peak bandwidth at 390 MB/sec for $N = 2-4$ MB and $n_{1/2} \approx 100$ KB]
More on communication cost

- Buffering a message is costly
- Modify the Ring program to copy the message a specified number of times before sending it and note the effect

Performance measurement

- To measure times in MPI, use MPI_Wtime()
- Measures wall clock time
- We often need to eliminate 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
  - But note any outliers
Measurement technique with Ring

for (int len = 1, l=0; len <= maxSize; len *= 2, l++)
    if (myid == 0) {
        // (Execute code for warm up)
        const double start = MPI_Wtime();
        for (int i = 0; i < trips; i++) {
            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);
        }
        const double delta = MPI_Wtime() - start;
        const long bw = (long)((trips*len*nodes)/delta/1000.0);
    }

The Ring program continued

for (int len = 1, l=0; len <= maxSize; len *= 2, l++)
    if (myid == 0) {
        // Send side code
        const double delta = MPI_Wtime() - start;
        const long bw = (long)((trips*length*nodes)/delta/1000.0);
    } else {
        // (WARM UP CODE)
        for (int i = 0; i < trips; i++) {
            MPI_Status status;
            MPI_Recv(buffer, len, MPI_CHAR, MPI_ANY_SOURCE,
                      tag, MPI_COMM_WORLD, &status);
            MPI_Send(buffer, len, MPI_CHAR, (rank+1)%p,
                      tag, MPI_COMM_WORLD);
        }
    }
Another application

- Integration using the trapezoidal rule
- Code follows Pacheco text (hard copy)
- For a discussion of the trapezoidal rule:
  http://metric.ma.ic.ac.uk/integration/techniques/definite/
  numerical-methods/trapezoidal-rule/

Another application

- Compute a numerical approximation to the definite integral
  \[ \int_{a}^{b} f(x) \, dx \]
- Use the trapezoidal rule

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

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

![Diagram of trapezoidal rule](image)

Serial code (Following Pacheco)

```c
main() {
    float integral;  /* Store result in integral */
    float x;
    int i;

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

    h = (b-a)/n; /* n = # of trapezoids, h = trapezoid base width; a & b: endpoints */

    integral = (f(a) + f(b))/2.0;
    for (i = 1, x=a; i <= n-1; i++) {
        x += h;
        integral = integral + f(x);
    }
    integral = integral*h;
}
```

First version of 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) { /* Add up the integrals calculated by each process */
    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);
```

Improvements

- Use wildcard specifier for sending process
- Avoid linear running time algorithm for computing the global sum (see discussion on collective communication)

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

Collective communication

- As we saw in the Trapezoidal rule computation, some communication operations are expensive
- We can sometimes improve performance by taking advantage of global knowledge about communication
- Such communication is called collective
Collective communication in MPI

- Collective operations are called by all processes in a communicator
  - Broadcast: distribute data from one process (the root) to all others in a communicator.
    \[
    \text{MPI\_Bcast(start, count, datatype, source, comm);} \\
    \]
  - Reduce: combine data from all processes in communicator and returns it to one process.
    \[
    \text{MPI\_Reduce(in, out, count, datatype, operation, dest, comm);} \\
    \]
- Information flow in Reduce is the reverse of the information flow in a broadcast

Broadcast

- In a broadcast, one processor has m pieces of data to send to the p-1 other processors
- A straightforward approach is to let this processor perform 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
  - A 1-cube is a line connecting two points
  - A 2-cube is a square, etc
Node numbering

- We label node using a binary reflected grey code; see http://www.nist.gov/dads/HTML/graycode.html
- Each node has d neighbors
- A neighbor’s label differs in exactly one bit position

Hypercube broadcast algorithm

- 2-cube case
- Processor 0 is the root
Hypercube exchange

• Processor 0 sends its data to its hypercube “buddy:” processor 2
• We repeat on the other hypercube dimension

Hypercube exchange

• Processor 0 and 2 send data to its their respective “buddies”
Reduction

• We can use the hypercube algorithm to perform reductions as well
• Use variant of reduction
  Allreduce( )
• Everyone obtains a copy of the reduced result
• This is equivalent to a Reduce( ) + Bcast( )
• A clever algorithm can be used to perform an Allreduce in one step

Interconnection network 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
Hamiltonian path

- A Hamiltonian path visits each node exactly once
- Used to embed a ring interconnect

Mesh

- $\sqrt{P} \times \sqrt{P}$ array
- Diameter is $2\sqrt{P}$
- Bisection bandwidth?
- Broadcast algorithm?
Linear array

- P element array
- What is the diameter?
- Bisection bandwidth?
- Broadcast running time

Ring

- P element array with end-around connection
- We can add end around connections to a mesh (toroidal mesh)
- Diameter is cut in half
- Bisection bandwidth doubles
- Broadcast running time?
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
- There are usually redundant paths
- Diameter?
- Bisection bandwidth?
- Broadcast algorithm?

An Omega Network

- The network is built 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

Modeling performance of an application

- We’ll consider two applications
  - Trapezoidal rule
  - Compare exchange sort