Lecture 7

Revisiting MPI performance & semantics
Strategies for parallelizing an application
Word Problems
Announcements

• Quiz #1 in section on Friday
• Midterm Room: SSB 106
• Monday 10/30, 7:00 to 8:20 PM
• No lectures 11/13 and 11/15
• Makeup on 11/17 10AM to 11:20 AM, room TBA
• Section will be held on Weds 11/15 in our class room at usual meeting time
K-ary d-cubes

- **Definition** - A k-ary d-cube is an interconnection network with $k^d$ nodes
  - There are $k$ nodes along all $d$ axis
  - End around connections
  - A generalization of a mesh and hypercube
  - The hypercube is a special case with $k=2$
K-ary d-cubes

• Derive the diameter, number of links, & bisection bandwidth of a k-ary d-cube with p nodes
• Diameter: \( d\lfloor k/2 \rfloor \)
• Bisection Bandwidth: \( 2k^{d-1} \)
Hamiltonians

- A Hamiltonian cycle is a path in an undirected graph that visits each node exactly once.
- Any Hamiltonian circuit on a labeled hypercube defines a Gray code [Skeina].
- Map a 1D ring onto a hypercube, mesh, k-ary d-cube.

www.cs.sunysb.edu/~skiena/combinatorica/animations/ham.html
mathworld.wolfram.com/HamiltonianCircuit.html
Prefix sum

- The prefix sum (also called a sum-scan) of a sequence of numbers \( x_k \) is in turn a sequence of running sums \( S_k \) defined as follows
  \[ S_0 = 0, \quad S_k = S_{k-1} + x_k \]
- Thus, scan \((3,1,4,0,2)\) = \((3,4,8,8,10)\)
- Design an algorithm for prefix sum based on the array interconnect (w/o end around connections), and give the accompanying performance model
- Repeat for the hypercube
- Use the gray coding to define the layout of values across processors
Prefix sum algorithm

\[ S = \text{PrefixSum}(x) \{
\text{if } (\text{rank}==0) \{ \ S = x; \ Send(S,1); \}\}
\text{for i=1 to P-1}
\text{if } (\text{rank}==i) \{
\text{Recv}(y,\text{rank}-1);
S = x + y;
\text{if } (\text{rank} < P-1) \ Send(S,\text{rank}+1);
\}
\]
Performance model

- Number of additions per node = 1
- Number of messages sent per node = 1 (except the last node)
- Number of messages received per node = 1 (except the first node)
- Let the startup time be $\alpha$ time units, and $\beta$ be $1/(\text{peak bandwidth})$
- Running time = $(N-1) \times (\alpha + \beta \times 1) + (N-1)$
Time constrained scaling

• Consider a summation algorithm
• We add up N numbers on P processors
• \( N \gg P \)
• Determine the largest problem that can be solved in time \( T = 10^4 \) time units on 512 processors
Performance model

- Local additions: $N/P - 1$
- Reduction: $(1+\alpha)(\log_2 P - 1)$
- $T(N,P) = N/P + \alpha \log_2 P$
- Determine the largest problem that can be solved in time $T=10^4$ time units on $P=512$ processors, $\alpha = 10$ time units, and addition costs one unit of time
- Consider $T(512,N) \leq 10^4$
  \[ \Rightarrow (N/512) + \alpha \log(512) \leq 10^4 \]
  \[ \Rightarrow (N/512) + 90 \leq 10^4 \]
  \[ \Rightarrow N \leq 5\times10^6 \text{ (approximately)} \]
Revisiting communication costs

• Plots from Valkyrie
• Plots from Blue Horizon
Ring – bandwidth (Fall 2003)

MPI Ring Test (8 nodes, 20 iterations)
Ring – variations in running time (F03)
Ring - Copy costs

![Graph showing Bandwidth vs Message Size]

*BW base max*
*BW 1 copy max*
*BW 2 copy max*
*BW 2-send max*
Correctness issues

- When there are multiple outstanding iRecvs, MPI doesn’t say how incoming messages are matched…
- Or even if the process is fair

```c
MPI_Request req1, req2;
MPI_Status status;
MPI_Irecv(buff, len, CHAR, nextnode, TYPE, WORLD,&req1);
MPI_Irecv(buff2,len, CHAR, prevnode, TYPE, WORLD,&req2);
MPI_Send(buff, len, CHAR, nextnode, TYPE, WORLD);
MPI_Wait(&req1, &status);
MPI_Wait(&req2, &status);
MPI_Send(buff2, len, CHAR, prevnode, TYPE, WORLD);
```
Datastar communication characteristics

![Graph showing data communication characteristics.]

- $T(4\ B) = 5.7\ \mu s$
- $\alpha = 5.4\ \mu s$
- $1611\ \text{MB/sec}$

<table>
<thead>
<tr>
<th>Data Star</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{1/2}$</td>
<td></td>
</tr>
<tr>
<td>22528</td>
<td>790.08</td>
</tr>
<tr>
<td>22656</td>
<td>792.64</td>
</tr>
<tr>
<td>22784</td>
<td>795.81</td>
</tr>
<tr>
<td>22912</td>
<td>798.60</td>
</tr>
<tr>
<td>23040</td>
<td>807.12</td>
</tr>
<tr>
<td>23168</td>
<td>806.20</td>
</tr>
<tr>
<td>23296</td>
<td>810.40</td>
</tr>
</tbody>
</table>
Short message, increments of 4 bytes
1K to 70K
Debugging and coding example

• Print out a matrix distributed over multiple processors
• Collect the values on a single processor for output
**Design**

- Create row and column communicators
- Collect data in each row to the root
- Collect the rows onto a column global root

<table>
<thead>
<tr>
<th></th>
<th>p(0,0)</th>
<th>p(0,1)</th>
<th>p(0,2)</th>
<th>p(1,0)</th>
<th>p(1,1)</th>
<th>p(1,2)</th>
<th>p(2,0)</th>
<th>p(2,1)</th>
<th>p(2,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(0)</td>
<td>green</td>
<td>red</td>
<td>red</td>
<td>darkred</td>
<td>darkred</td>
<td>darkred</td>
<td>green</td>
<td>green</td>
<td>green</td>
</tr>
<tr>
<td>p(1)</td>
<td>red</td>
<td>green</td>
<td>green</td>
<td>red</td>
<td>red</td>
<td>red</td>
<td>darkred</td>
<td>darkred</td>
<td>darkred</td>
</tr>
<tr>
<td>p(2)</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
<td>green</td>
</tr>
</tbody>
</table>
Gather

int MPI_Gather ( void *sendbuf, int sendcnt,
MPI_Datatype sendtype,
void *recvbuf, int recvcount,
MPI_Datatype recvtype,
int root, MPI_Comm comm )

Root

P_0

P_{p-1}

Gather

Scatter

Root
Code

\( N \times N = \text{size of array}, \ q = \sqrt{p}, \ nsq = (N/q)^2 \)

\text{MPI\_Comm \ row\_comm, \ col\_comm;}
\text{ROOT} = 0;

// Each row root gathers its rows
float* Local\_Matrix[N/q][N/q], rows = new float[q*nsq];
\text{MPI\_Gather ( Local\_Matrix, nsq, MPI\_FLOAT,}
\text{rows, nsq, MPI\_FLOAT, ROOT, row\_comm );}

// Gather all the data from each root
float* cols = new float[N*N];
\text{MPI\_Gather ( rows, q*nsq, MPI\_FLOAT,}
\text{cols, q*nsq, MPI\_FLOAT, ROOT, col\_comm );}
Finishing up

if (!grid.my_rankR && !grid.my_rankC) {
    OUTPUT
}

\[ 
\begin{array}{ccc}
  p(0,0) & p(0,1) & p(0,2) \\
  p(1,0) & p(1,1) & p(1,2) \\
  p(2,0) & p(2,1) & p(2,2) \\
\end{array} \]