Parallel & Concurrent Programming:
Collective Communication

Emery Berger
CMPSCI 691W
Spring 2006
Outline

- Last time:
  - Distributed parallel programming via message-passing
  - MPI – library approach
- Today:
  - Moving beyond point-to-point: collective communication

some slides from van de Geijn et al.
Collective Communication

- Instead of single target, collective communication operates on entire communicator
  - e.g., MPI_COMM_WORLD

- Advantages over point-to-point:
  - Higher-level
  - Typically far more efficient
    - e.g., $O(n) \Rightarrow O(\log n)$
    - Pipelining
Simple Analytical Model

- To send $n$ bytes:
  - $\alpha = \text{startup latency}$
  - $\beta = \text{per-byte cost}$
  - $\Rightarrow \alpha + n\beta$

- Naive broadcast ($p$ processors):
  - $\alpha(p-1) + (p-1)\beta$

- Minimum spanning tree:
  - $\alpha(\log p + p - 1) + 2(p-1)/p \times \beta$
  - Note: does not model contention
    - In practice, can avoid it
MPI Collective Ops

- MPI has rich set of collective communication operations (& duals):
  - Synchronization
    - Barrier
  - Communication
    - Broadcast
    - Scatter, Gather
    - Reduce, Scan
  - Communicator management
Barrier Synchronization

- **MPI_Barrier:**
  progress continues past barrier only after all processes have arrived

```c
// perform computation
...
MPI_Barrier (MPI_COMM_WORLD);
// all done here

// perform computation (faster)
MPI_Barrier (MPI_COMM_WORLD);
// all done here
```
From root: Broadcast

- MPI_Bcast: broadcast single datum across all processors
From root: Scatter

- **MPI_Scatter**: spread data across all processors
- **variant – MPI_Scatterv**: scatters buffer in parts (specified in array)
Scatter

Before

After
To root: Gather

- **MPI_Gather:** gather data from all processors to one processor
- **variant – MPI_Gatherv**
**MPI_Alltoall:**
send data from all processors to all processors

![Diagram of MPI_Alltoall]

- **Processors**: A0, A1, A2, A3, B0, B1, B2, B3, C0, C1, C2, C3, D0, D1, D2, D3
- **Alltoall** operation
MPI_Allgather:
send data from all processors to one vector across all processors
**Reductions: Reduce**

- **MPI_Reduce**: collect data from all processors, apply operator to each, put on one processor.

![Diagram showing MPI_Reduce operation]

```
A0 + B0 + C0 + D0
A1 + B1 + C1 + D1
A2 + B2 + C2 + D2
A3 + B3 + C3 + D3
```

Reduce (+)
**Reductions: Scan**

- **MPI_Scan:** apply partial reductions
- Operation applied to increasingly-long prefixes
2D Reduce-scatter
2D Reduce-scatter

reduce-scatter in columns
2D Reduce-scatter

reduce-scatter in rows
Summary

- From root task to all others
  - MPI_Bcast
    - (buf, cnt, type, root, comm)
  - MPI_Scatter
    - (sendbuf, cnt, type, recvbuf, recvcnt, type, root, comm)

- From all tasks to root
  - MPI_Gather
    - (sendbuf, cnt, type, recvbuf, recvcnt, type, root, comm)

- From all to all
  - MPI_Alltoall
    - (sendbuf, type, recvbuf, recvcnt, type, comm)
  - MPI_Allgather
    - (sendbuf, type, recvbuf, recvcnt, type, comm)

- Reductions
  - MPI_Reduce
    - (sendbuf, recvbuf, cnt, type, op, root, comm)
  - MPI_Scan
    - (sendbuf, recvbuf, cnt, type, op, comm)
Communicators

- Can duplicate:
  - MPI_Comm_dup (old, &new)

- Can then **split** communicators:
  - MPI_Comm_split (old, color, key, &new)

- Same color ⇒ same communicator
- Key determines rank in new

<table>
<thead>
<tr>
<th>color</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
The End

Next time:
- OpenMP