Lecture 14

More collective communication
Advanced sorting

Announcements

- Assignment #5 due in section on Friday 5/21
- Discuss some results on Valkyrie
Ring – bandwidth (8 nodes, 20 iter)

MPI Ring Test (8 nodes, 20 iterations)

- Message Bandwidth
- Message Length (bytes)

Ring – with 1 copy

MPI Ring Test (8 nodes, 20 iterations) w/ 1 Copied Buffer

- Message Bandwidth
- Message Length (bytes)
Ring –w/ and w/o copy (8 nodes, 20 iter)

Ring –variations in running time
The collective operations

- We’ve looked at two collective communication operations so far
  - Reduce
  - Broadcast
- We’ll next look at some other important ones
  - All to all
  - Scatter
- We won’t discuss Allgather & Allreduce
- See the reference material for the details, esp Thakur and Gropp paper

The technology

- “Beowulf” clusters with Myrinet, switched Fast Ethernet, or gigabit ethernet
  - All nodes are equidistant
  - Single-ported, bidirectional links
- Communication time is $\alpha + \beta n$ in the absence of contention
  - Determined by bandwidth $\beta^{-1}$ for long messages
  - Dominated by latency $\alpha$ for short messages
- We may use different algorithms for different message size regimes
Inside MPI-CH

- Uses a “tree-like” algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
- The block size may be configured at installation time
- If there is hardware support, then it is given responsibility to carry out the broadcast

All to all

- Also called total exchange or personalized communication
- Each processor sends a different chunk of data to each of the others
- Implements transpose
How do we implement this?

- Simplest algorithm:
- Using point-to-point communication …
- … each processor sends data to all others

for i = 0 to p-1
    irecv(src = i, …)
    isend(dest = i, …)
end for

Waitall()

Performance

- Each processor sends P-1 messages, each of size n/P (Let n = N^2)
- Message passing time = (P-1)(α + β (n/P))
  = α(P-1) + β n (P-1)/P
- For short messages, this may be reasonable
- But for long messages, we’ll flood the network with O(P^2) messages
Another approach: the ring algorithm

- Let’s modify our ring passing algorithm to route the data in P steps
- Each processor passes a unique set of data to its neighbors downstream along the ring
- Data received from the upstream neighbor is passed downstream.
- But before sending the data on its way, the processor picks off the top chunk of data, sending only the remainder

The ring algorithm in action
The ring algorithm in action
The ring algorithm in action

Performance

- In the first step, all send \( P-1 \) chunks of data
- Then \( P-2, P-3 \), down to 1 chunk
- Each chunk has size \( \frac{n}{P^2} \)
- The asymptotic running time:

\[
\sum_{i=1}^{P-1} \left( \alpha + \beta \frac{n}{P^2} (P-i) \right) = \frac{P-1}{P} n \beta
\]

- Same running time as the linear broadcast, but only \( O(P) \) messages sent at any one time
Another approach: the hypercube algorithm

• Consider the case of $P=2^d$
• Recall that we may split a $d$-cube into two $(d-1)$ cubes
• Let’s split the cube into a left and right half
• Each processor in the left half swaps half its data with the corresponding neighbor in the right half
• Now each processor has all the data that its neighbor intended to send to the processor’s half of the cube
• Repeat along the other directions

Flow of information

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Flow of information

A B C D ↔ A B C D

10  11

00  01

A B C D ↔ A B C D
Relative costs

- Hypercube algorithm
  \[ \alpha \lg P + \frac{(P-1)}{P} N^2 \beta \]

- The Ring algorithm
  \[ \alpha(P-1) + \frac{(P-1)}{P} N^2 \beta \]

Gather (and Scatter)
Scatter

• Simple linear algorithm
  – Using point-to-point communication …
  – … each processor sends a chunk of data to all others
  – For long messages, this is reasonable. Why?
    \[(p - 1)\alpha + \frac{p - 1}{p} n\beta\]

• For short messages we use a minimal spanning tree algorithm

• See the Thakur article

Advanced sorting
In search of a faster sort algorithm

• We’ve talked about compare exchange sorts
• Long running times: data moves incrementally to its final destination
• What if we could compute the processor owner for each key
• We can then communicate in one step
• But how do we know which processor is the owner?
• Depends on the distribution of keys

A first pass

• Assume that the keys are distributed uniformly over 0 to $K_{\text{max}}-1$
• Assign each key to processor $P \times \text{key}/(K_{\text{max}}-1)$
• The assignment of keys to processors is based only on the knowledge of $K_{\text{max}}$
• But if they keys are distributed non-uniformly, then this approach will result in an imbalance
• In the worst case, all the keys could go to one processor
Bucket sorting

- Assuming that the keys are evenly distributed over the range ….
- Divide the range of keys into equal subranges and associate a bucket with each range
- Examine each key, and place in the appropriate bucket \( O(N) \)
- Sort the buckets \( O(N \lg (N/m)) \)
  - If the keys are evenly distributed, then each bucket has \( N/m \) elements
- Merge the buckets \( O(N) \)
- Total running time is \( O(N \log(N/m)) \)

Parallel algorithm

- A processor can have keys over the full range of possible key values
- Each of \( P \) processors maintains \( P \) local buckets
  - Assigns each key to a local bucket
  - Routes each local bucket to the correct owner (each local bucket has about \( N/P^2 \) elements)
  - Sorts all incoming data into a single bucket
Running time

- Local bucket assignment: $N/P$
- Route each local bucket to the correct owner
  All to all: what does this cost?
- Local sorting: $N/P \log(N/P)$

Worst case behavior

- We’ve assumed that the keys are distributed uniformly over the range
- If the keys are integers in the range $[0, Q-1]$ we can assign each processor $k$ to a subrange $[k*Q/P, (k+1)*Q/P-1]$
- E.g. for $Q=2^{30}$, $P=64$, each processor gets $2^{24} = 16$ M elements
- But what if all the keys are in the range $[0, 2^{24}-1]$?
- We’ll next look at an algorithm called sample sort, which is designed to remedy the problem
The idea behind sample sort

- Uses a heuristic to determine the key range for each processor, such that each processor will get about the same number of keys
- Sample the keys to determine a set of P-1 splitters that partition the key space into P disjoint regions
- Each region is assigned to processor, and is treated as a bucket
- Once each processor knows the splitters, it can distribute its keys to the other buckets accordingly
- Each processor sorts the keys sent it

Sample sort

- We’ll look at a few variations
- For details see:
  http://www.umiacs.umd.edu/research/EXPAR/papers/spaa96/spaa96.html
Splitter selection

• Each processor chooses $s < N/P$ keys at random and sorts them into a list of candidate splitters
• Candidate splitters are collected by one processor and then sorted
• The sorted list is sampled at uniform positions $0, s, 2s, \ldots (P-1)s$ to generate the splitter list
• The splitter list is distributed to the other processors

Limitations

• Tradeoff: as $s$ increases…
  – the distribution of the final sorted keys over the processors becomes more even
  – the cost of determining the splitters increases
• For some inputs, communication patterns can be highly irregular with some pairs of processors communicating more heavily than others
• This imbalance degrades communication performance
Enhancement: random sample sorting

- Mix up the keys as a preprocessing step
- Each processor randomly assigns each of its $n/p$ elements to one of $p$ buckets
- We treat this communication like a transpose
- Each processor sorts its assigned values locally
- A distinguished processor selects the splitters and broadcasts to the others
- Each processor collects its local keys into $p$ buckets and routes $p-1$ of these to the other processors (another transpose)
- Each processor merges the incoming keys (use radix sort)

Performance bounds

- $T(n,p) = T_{\text{comp}}(n,p) + T_{\text{comm}}(n,p)$
- With “high probability” $1-n^{-\epsilon}$
- No processors exchange $> c_2 n/p^2$ keys when
  for any $c_2 \geq 3.10$, $p^2 \leq n/(3 \ln n)$
- No processor will obtain more than $\alpha n/p$ keys
  for any $\alpha \geq 1.77$, $p^2 \leq n/(3 \ln n)$
A word on probabilities

- For $\alpha \geq 1.77$, $p^2 \leq n/(3 \ln n)$ …
- Each bucket will have no more than $L \leq \alpha n/p$ keys with probability $n^{-\epsilon}$

- $\epsilon = (1-1/\alpha)^2 s/2$  
- For $\alpha = 3$, $s = 32$, $n = 100,000$, $n^{-\epsilon} \approx 5 \times 10^{-5}$  
- For $\alpha = 3$, $s = 64$, $n = 1,000,000$, $n^{-\epsilon} \approx 3 \times 10^{-13}$

Radix sort

- We need a good sorting algorithm to do the local sorts  
- With integers keys, radix sort is the candidate of choice  
- We sort the keys in passes, choosing an r-bit block at a time
A simple example

- Following an example in the NIST
  Dictionary of Algorithms and Data Structures
  http://www.nist.gov/dads/
- Uses buckets to sort the keys in passes
- Running time is $O(cn)$, $c$ depends on size of the keys and the number of buckets
- Need a stable sort: output preserves order of inputs that have the same value

Radix sort in action

- Consider the input keys
  34, 12, 42, 32, 44, 41, 34, 11, 32, and 23
- Use 4 buckets
- Sort on each digit in succession, least significant to most significant
Radix sort in action

- Consider the input keys
  34, 12, 42, 32, 44, 41, 34, 11, 32, and 23
- Use 4 buckets
- Sort on each digit in succession, least significant to most significant
- After pass 1
  41 11 12 42 32 32 23 34 44 34

- After pass 2
  11 12 23 32 32 34 34 41 42 44