Lecture 17

Advanced Collective Communication
Announcements

• TA Evaluation
• Professor and Course Evaluation
• MPI? CUDA?
Today’s lecture

• Short topic on MPI
• CG implementation
• Spare matrices
• More collective communication patterns
• Collective communication polyalgorithms
Correctness and fairness

1. Iteration 1: 1 → 2&0   0 → 1 (0 → 2)   2 → 0&1
2. 1 begins iteration 2: 1 → 2
3. 0 → 2 (but for iteration 1)
4. Problem: irecv in P2 receiving data from P1 in iteration 2 while it expects data from P0 in iteration 1

For i = 1 to n
MPI_Request req1, req2;
MPI_Status status;
MPI_Irecv(buff, len, CHAR, ANY_NODE, TYPE, WORLD,&req1);
MPI_Irecv(buff2,len, CHAR, ANY_NODE, TYPE, WORLD,&req2);
MPI_Send(buff, len, CHAR, nextnode, TYPE, WORLD);
MPI_Send(buff, len, CHAR, prevnode, TYPE, WORLD);
MPI_Wait(&req1, &status);
MPI_Wait(&req2, &status);
End for
\[ x_0 = 0, \ r_0 = b, \ d_0 = r_0 \]

\textbf{for} \ k = 1, 2, \ldots \ \\
\[ \alpha_k = \frac{(r^T_{k-1}r_{k-1})}{(d^T_{k-1}Ad_{k-1})} \]  \hspace{2em} \text{// Step length} \\
\[ x_k = x_{k-1} + \alpha_k d_{k-1} \]  \hspace{2em} \text{// Approximate solution} \\
\[ r_k = r_{k-1} - \alpha_k Ad_{k-1} \]  \hspace{2em} \text{// Residual} \\
\[ \beta_k = \frac{(r^T_k r_k)}{(r^T_{k-1}r_{k-1})} \]  \hspace{2em} \text{// Improvement} \\
\[ d_k = r_k + \beta_k d_{k-1} \]  \hspace{2em} \text{// Search Direction} \\
\textbf{end for}
Implementing CG

Matrix-Vector Multiplication: an important kernel used in sparse linear algebra

\[ y[i] += A[i,j] \times x[j] \]

\[ x_k = x_{k-1} + \alpha_k \, d_{k-1} \quad \text{// Approximate solution} \]

\[ r_k = r_{k-1} - \alpha_k \, A d_{k-1} \quad \text{// Residual} \]
Sparse Matrices

- A matrix where knowledge about the location of the non-zeroes is useful
- Consider Jacobi’s method with a 5-point stencil

\[
u'[i,j] = (u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i, j+1] - h^2f[i,j]) / 4
\]
Web connectivity Matrix: 1M x 1M

1M x 1M submatrix of the web connectivity graph, constructed from an archive at the Stanford WebBase

3 nonzeros/row
Sparse Matrix Vector Multiplication

- Assume $\mathbf{x}[]$ fits in memory of 1 processor
  \[ y[i] += A[i,j] \times x[j] \]
- Many storage formats for sparse matrices, common one (on one core) is Compressed Sparse Row (CSR)
Advanced Collective Communication
Collective communication

• Diverse applications
  ▶ Fast Fourier Transform
  ▶ Sorting

• Collective operations are called by all processes within a communicator

• Basic collectives seen so far
  ▶ Broadcast: distribute data from a designated root process to all the others
  ▶ Reduce: combine data from all processes returning the result to the root process

• Other Useful collectives
  ▶ Scatter/gather
  ▶ All to all
  ▶ Allgather
Underlying assumptions

• Fast interconnect structure
  - 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
Inside MPI-CH

• Tree like algorithm to broadcast the message to blocks of processes, and a linear algorithm to broadcast the message within each block
• Block size may be configured at installation time
• If there is hardware support, then it is given responsibility to carry out the broadcast
• Polyalgorithms apply different algorithms to different cases, i.e. long vs. short messages, different machine configurations
• We’ll use hypercube algorithms to simplify the special cases when $P=2^k$, $k$ an integer
Details of the algorithms

- Scatter/gather
- All to all
- Allgather
- Revisiting broadcast
Scatter/Gather family

\[ \begin{align*} &P_0 & &P_1 & &P_{p-1} & &\text{Gather} & &\text{Scatter} & &\text{Root} \end{align*} \]
Scatter

• Simple linear algorithm
  ✷ Root processor sends a chunk of data to all others
  ✷ Reasonable for long messages

\[(p - 1)\alpha + \frac{p - 1}{p} n\beta\]

• Similar approach taken for Reduce and Gather
• For short messages, we need to reduce the complexity of the latency (\(\alpha\)) term
Minimum spanning tree algorithm

- Recursive hypercube-like algorithm with \( \lfloor \log P \rfloor \) steps
  - Root sends half its data to process \((\text{root} + p/2) \mod p\)
  - Each receiver acts as a root for corresponding half of the processes
  - MST: organize communication along edges of a minimum-spanning tree covering the nodes
- Requires \( O(n/2) \) temp buffer space on intermediate nodes
- Running time:
  \[
  \lceil \lg P \rceil \alpha + \frac{p - 1}{p} n\beta
  \]
Details of the algorithms

• Scatter/gather
• All to all
• Allgather
• Revisiting broadcast
All to all

- Also called *total exchange* or *personalized communication*: a transpose
- Each process sends a different chunk of data to each of the other processes
- Used in sorting and the Fast Fourier Transform
Exchange algorithm

- \( n \) elements / processor (\( n \) total elements)
- \( p - 1 \) step algorithm
  - Each processor exchanges \( n/p \) elements with each of the others
  - In step \( i \), process \( k \) exchanges with processes \( k \pm i \)

for \( i = 1 \) to \( p-1 \)
  
  src = (rank – i + p) mod p  
  dest = (rank + i ) mod p  

sendrecv( from src to dest )
end for

- Good algorithm for long messages
- Running time:

\[
(p - 1)\alpha + (p - 1)\frac{n}{p} \beta \approx n\beta
\]
Recursive doubling for short messages

• In each of \([\log p]\) phases all nodes exchange \(\frac{1}{2}\) their accumulated data with the others

• Only \(P/2\) messages are sent at any one time

\[
D = 1
\]

\textbf{while} (D < p)

\hspace{1em} Exchange & accumulate data with rank \(\otimes D\)

\hspace{1em} Left shift D by 1

\textbf{end while}

• Optimal running time for short messages

\[
[\lg P] \alpha + nP \beta \approx [\lg P] \alpha
\]
Flow of information
Flow of information
Flow of information
Summarizing all to all

• Short messages \([\lg P]_\alpha\)

• Long messages \(\frac{P-1}{P}n\beta\)
“Vector” variants

- Generalize all-to-all, gather, etc.
- Processes supply varying length datum
- Vector all-to-all

`MPI_Alltoallyv (void *sendbuf, int sendcounts[], int sDispl[], MPI_Datatype sendtype, void* recvbuf, int recvcounts[], int rDispl[], MPI_Datatype recvtype, MPI_Comm comm)`
Details of the algorithms

• Scatter/gather
• All to all
• Allgather

• Revisiting broadcast
Revisiting Broadcast

- \( P \) may not be a power of 2
- We use a binomial tree algorithm
- We’ll use the hypercube algorithm to illustrate the special case of \( P=2^k \)
- Hypercube algorithm is efficient for short messages
- We use a different algorithm for long messages
Strategy for long messages

• Based van de Geijn’s strategy

• Scatter the data
  ♦ Divide the data to be broadcast into pieces, and fill the machine with the pieces

• Do an Allgather
  ♦ Now that everyone has a part of the entire result, collect on all processors

• Faster than MST algorithm for long messages

\[ 2 \frac{p-1}{p} n\beta \ll \lfloor \lg p \rfloor n\beta \]
Algorithm for long messages

The scatter step

\[ P_0 \rightarrow P_1 \rightarrow P_{p-1} \rightarrow \text{Root} \]
Algorithm for long messages

AllGather step

\[ P_0 \quad P_1 \quad P_{p-1} \]