Lecture: Distributed Memory Machines and Programming
-- MPI programming exercise

CSCE 569 Parallel Computing

Department of Computer Science and Engineering
Yonghong Yan
yanyh@cse.sc.edu
http://cse.sc.edu/~yanyh
Machines and MPI Examples

• Machines in Swearingen 1D39 and 3D22
  – https://passlab.github.io/CSCE569/resources/HardwareSoftware.html

• MPI Examples:
  – https://passlab.github.io/CSCE569/resources/mpi_examples/
    – wget https://passlab.github.io/CSCE569/resources/mpi_examples/mpihello.c

• mpicc mpihello.c -o mpihello
• mpirun -np 2 ./mpihello
• TODO #1: Row-wise data distribution
• TODO #2: Jacobi computation
  a) Update begin and end of the loop index variable
  b) Boundary (ghost region) exchange
  c) Reduction for error
• TODO #3: Data collection, opposite of TODO #1
TODO #1: Row-wise data distribution

- numprocs = 4 (4 MPI processes) and n = 100
  - n is divisible by numprocs
  - How u should be distributed into subarray and computed by each MPI process

- Processes 0 and numprocs-1 each has only one neighbors and each other process has two neighbors (top and bottom)
- The same for u and f
  - To make programming easier in TODO #2
TODO #1: Row-wise data distribution

- Process 0 has initial array and data for the full u and f
- 0 uses MPI_Send to send subarray of u and f to each other process
  - Calculate num_rows to send for each other process
    - If other is 1 to numprocs-2: n/numprocs + 2
    - If other is numprocs-1: n/numprocs + 1
  - Calculate pointer of the subarray data region for each other process
    - other is 1 to numprocs-1: u + (other*n/numprocs -1)*m
- Other processes use MPI_Recv to receive u and f subarray
  - Calculate num_rows to recv from process 0
    - If myrank is 1 to numprocs-2: n/numprocs + 2
    - If myrank is numprocs-1: n/numprocs + 1
  - Allocate memory to store subarray data received from 0
- Make sure the tag for Send/Recv pair are the same and correct.
TODO #3: Row-wise data collection

- Process will have final result for the full u
  - No need to collect f
- 0 uses MPI_Recv to recv subarray of u from each other process
  - Calculate num_rows to recv for each other process
    - If other is 1 to numprocs-2: n/numprocs + 2
    - If other is numprocs-1: n/numprocs + 1
  - Calculate pointer for storing the subarray data recvd from each other process
    - other is 1 to numprocs-1: u + (other*n/numprocs -1)*m
- Other processes use MPI_Send to send u subarray
  - Calculate num_rows to send to process 0
    - If myrank is 1 to numprocs-2: n/numprocs + 2
    - If myrank is numprocs-1: n/numprocs + 1
  - Deallocate memory for the subarray
- Make sure the tag for Send/Recv pair are the same and correct.
TODO #2: Jacobi computation

- **TODO #2:**
  a) Update begin and end of the loop index variable
  b) Boundary (ghost region) exchange
  c) Reduction for error

a) Row-wise distribution
   - i is 1 to num_rows - 1
**TODO #2: Jacobi computation**

**b) Boundary exchange using MPI_Send/Recv**
- 0: MPI_Send row `num_rows-2` to proc 1, MPI_Recv row `num_rows-1` from proc 1
- 1: MPI_Recv row 0 from 0 (myrank-1), MPI_Send row 1 to 0 (myrank-1)
- 1. MPI_Send row `num_rows-2` to myrank+1, MPI_Recv row `num_rows-1` from myrank+1
- ...
- `num_procs-1`: MPI_Recv row 0 from myrank-1, MPI_Send row 1 to myrank -1

- Make sure the tag for Send/Recv pair are the same and correct.
c) Reduction for error
   - Local_error computed by each process
   - Sum up local_error to have error and then broadcast to all processes

   ```
   MPI_Allreduce(&local_error, &error, 1, MPI_FLOAT, MPI_SUM, COMM_WORLD);
   ```
b) Boundary exchange optimization

- Currently solution serializes message passing for exchange
  - 0, 1, 2, 3, ...
- Using MPI_Isend/Irecv to have parallelized exchange
  - MPI_Wait after firing Isend/irecv, before computation
- Overlap comm and computation
  - MPI_Wait after the computation loop, but before the MPI_Allreduce for error