Lecture 30: Considerations When Using Collective Operations

William Gropp

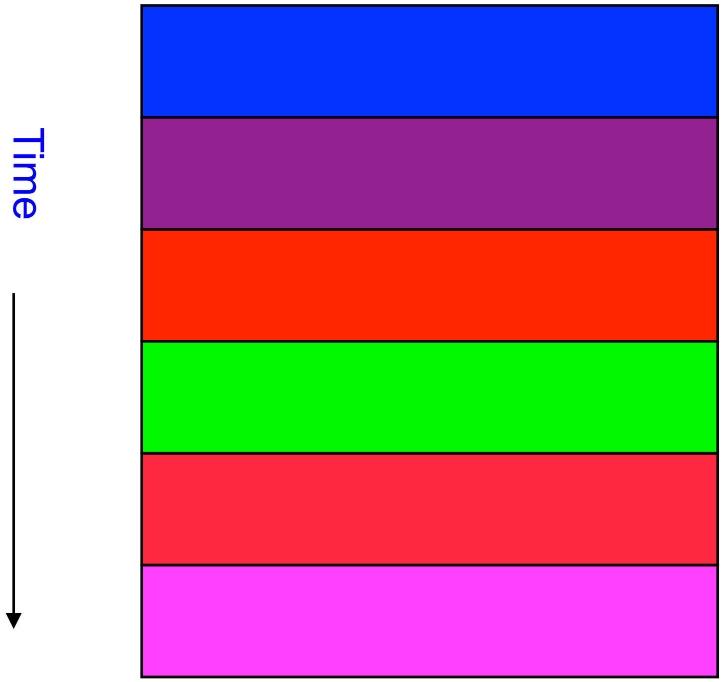
www.cs.illinois.edu/~wgropp

When *not* to use Collective Operations

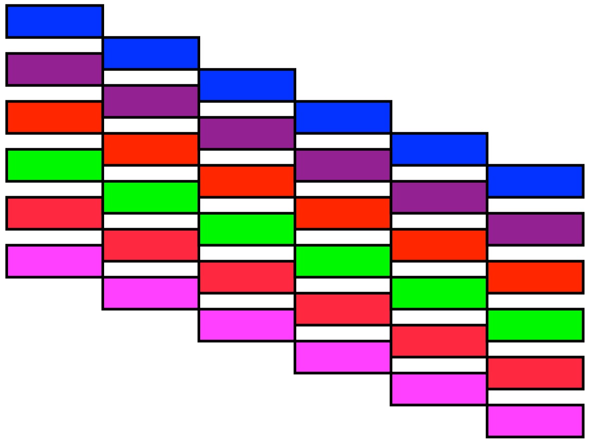
- Sequences of collective communication can be pipelined for better efficiency
- Example: Process 0 reads data from a file and broadcasts it to all other processes.
  - ```
    do i=1,m
       if (rank .eq. 0) read *, a
       call mpi_bcast(a, n, MPI_INTEGER, 0, comm, ierr)
    enddo
    ```
- Question: How long will this take on p processes?
  - Assume a broadcast takes (s log p + r n) time, and m=p
    - Yes, not (log p) * (s + rn); the best algorithm is *not* a distribution tree
    ```

Broadcast of n Items p Times

- If each takes (s logp + r n) and p = m; then the entire loop takes
  - s * p log p + p r n
- But there is a way to accomplish this in s p + p r n time!
  - Log p times as fast if n is small

Pipeline the Messages

- Process 0 reads data from a file and sends it to the next process. Others forward the data.
  - ```
    do i=1,m
      if (rank .eq. 0) then
        read *, a
        call mpi_send(a, n, MPI_INTEGER, 1, 0, comm, ierr)
      else
        call mpi_recv(a, n, MPI_INTEGER, rank-1, 0, &
                          comm, status,  ierr)
        call mpi_send(a, n, MPI_INTEGER, next, 0, comm,&
                           ierr)
      endif
    enddo
    ```
- next = rank+1 unless rank + 1 == size, in which case use MPI_PROC_NULL
    ```

Concurrency Between Steps

- Broadcast:



- Pipeline



Each broadcast takes less time than pipeline version, but total time is longer

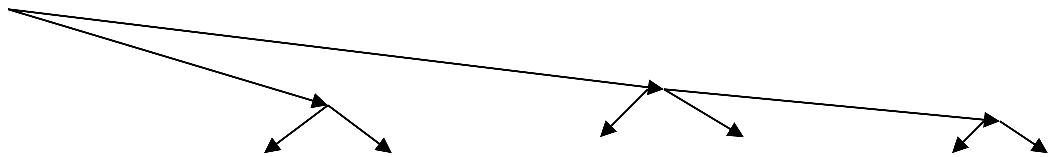
Another example of deferring synchronization

Notes on Pipelining Example

- When reading and distributing data from a file, use `MPI_File_read_all` instead
  - ♦ Even more optimizations possible
    - Multiple disk reads
    - Pipeline the individual reads
    - Block transfers
- This algorithm is sometimes called "digital orrery"
  - ♦ Circulate particles in n-body problem
  - ♦ Even better performance if pipeline never stops
- "Elegance" of collective routines can lead to fine-grain synchronization
  - ♦ And hence a performance penalty

Thinking about Broadcast

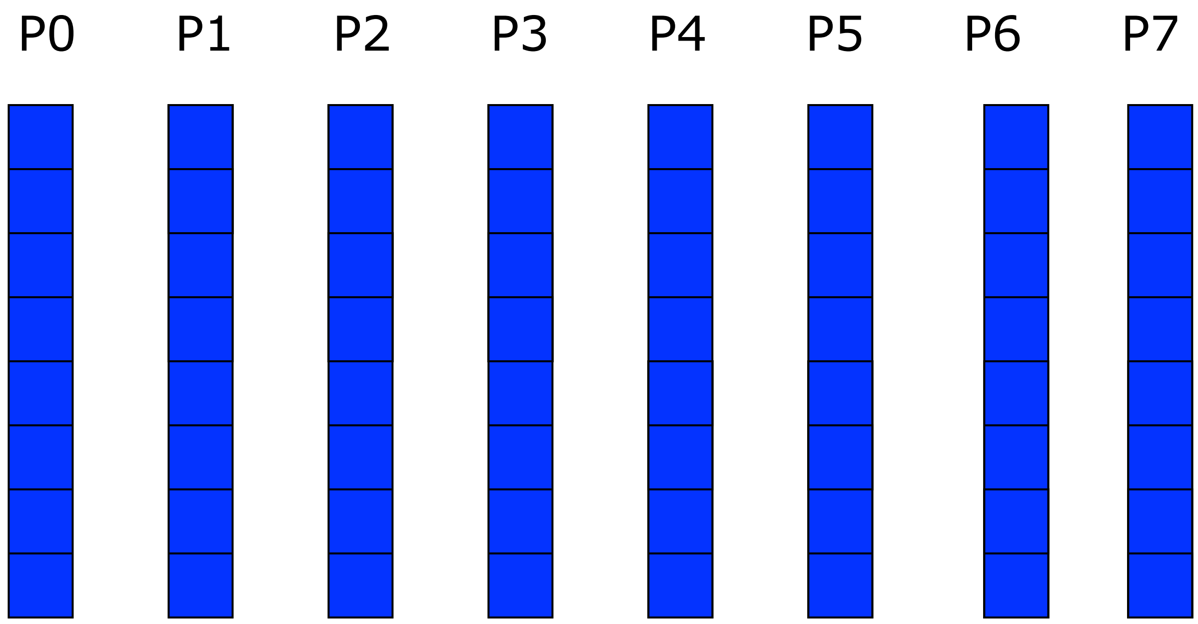
- MPI_Bcast(buf, 100000, MPI_DOUBLE, …);
- Use a tree-based distribution:



- Use a *pipeline*: send the message in b byte pieces. This allows each subtree to begin communication after b bytes sent
- Improves total performance:
  - Root process takes same time (asymptotically)
  - Other processes wait less
    - Time to reach leaf is *b log p + (n-b)*, rather than *n log p*
- Special hardware and other algorithms can be used …

Make Full Use of the Network

- Implement MPI_Bcast(buf,n,…) as
  MPI_Scatter(buf, n/p,…, buf+rank*n/p,…)
  MPI_Allgather(buf+rank*n/p, n/p,…,buf,…)



Optimal Algorithm Costs

- Optimal cost is O(n) (O(p) terms don't involve n) since scatter moves n data, and allgather also moves only n per process; these can use pipelining to move data as well
  - Scatter by recursive bisection uses log p steps to move n(p-1)/p data
  - Scatter by direct send uses p-1 steps to move n(p-1)/p data
  - Recursive doubling allgather uses log p steps to move
    - N/p + 2n/p + 4n/p + … (p/2)/p = n(p-1)/p
  - Bucket brigade allgather moves
    - N/p (p-1) times or (p-1)n/p
- See, e.g., van de Geijn for more details

Implementation Variations

- Implementations of collective operations vary in goals and quality
  - ♦ Short messages (minimize separate communication steps)
  - ♦ Long messages (pipelining, network topology)
- MPI's general datatype rules make some algorithms more difficult to implement
  - ♦ Datatypes can be different on different processes; only the type signature must match

Using Datatypes in Collective Operations

- Datatypes allow noncontiguous data to be moved (or computed with)
- As for all MPI communications, only the *type signature* (basic, language defined types) must match
  - Layout in memory can **differ** on each process

Example of Datatypes in Collective Operations

- Distribute a matrix from one process to four
  - Process 0 gets A(0:n/2,0:n/2), Process 1 gets A(n/2+1:n,0:n/2), Process 2 gets A(0:n/2,n/2+1:n), Process 3 gets A(n/2+1:n,n/2+1:n)
- Scatter (One to all, different data to each)
  - Data at source is not contiguous (n/2 numbers, separated by n/2 numbers)
  - Use vector type to represent submatrix

Layout of Matrix in Memory

N = 8 example

Process 0

0	8	16	24
1	9	17	25
2	10	18	26
3	11	19	27

Process 2

32	40	48	56
33	41	49	57
34	42	50	58
35	43	51	59

Process 1

4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

Process 3

36	44	52	60
37	45	53	61
38	46	54	62
39	47	55	63

Matrix Datatype

- MPI_Type_vector(n/2 per block,
  n/2 blocks,
  dist from beginning of one block to next = n,
  MPI_DOUBLE_PRECISION,&subarray_type)
- Can use this to send
  - ♦ Do j=0,1
    Do i=0,1
    call MPI_Send(a(1+i*n/2, 1+j*n/2), 1,
    subarray_type, …)
  - ♦ Note sending **ONE** type contain multiple basic elements
  - ♦ Pass the (address of the) first element to be sent to MPI_Send
  - ♦ This looks like an MPI_Scatter, but with interleaved data

Scatter with Datatypes

- Scatter is like
  - Do i=0,p-1
    call mpi_send(a(1+i*extent(datatype)),….)
    - "1+" is from 1-origin indexing in Fortran
  - Extent is the distance from the beginning of the first to the end of the last data element
  - For our subarray_type, it is
    ((n/2-1)n+n/2) * extent(double)
  - "extent(double)" is simply the number of bytes in DOUBLE PRECISION item (often 8)
    - In Fortran, you can use MPI_Type_size(MPI_DOUBLE_PRECISION, extent, ierr)
    - Or MPI_SIZEOF(a) (with the MPI or MPI_F08 module)
    - Or storage size(1.0do)/8 (in Fortran 2008)

  to get this value

If Only We Could Change the Extent of subarray_type…

- To make the communication work with Scatterv, set Extent of each datatype to n/2
  - ♦ Size of contiguous block all are built from
- Use Scatterv (independent multiples of extent)
- Location (beginning location) of blocks
  - ♦ Process 0: 0 * 4 (doubles)
  - ♦ Process 1: 1 * 4 (doubles)
  - ♦ Process 2: 8 * 4 (doubles)
  - ♦ Process 3: 9 * 4 (doubles)
- How can we change the extent of a datatype?

Changing the Extent

- MPI allows you to change the extent of a datatype with MPI_Type_create_resized
- In our case (in C),
- MPI_Type_create_resized(
    subarray_type, 0,
    (n/2)*sizeof(double), &newtype)
  - Sets the lower bound to zero (almost always the right thing) and the extent to n/2 doubles.

Scattering A Matrix

- sdisplace(1) = 0
  sdisplace(2) = 1
  sdisplace(3) = n
  sdisplace(4) = n + 1
  scounts(1,2,3,4)=1
  call MPI_Scatterv(a, scounts, sdispls, newtype, &
      alocal, n*n/4, MPI_DOUBLE_PRECISION, &
      0, comm, ierr)
  - Note that process 0 sends 1 item of newtype but all processes receive $n^2/4$ double precision elements
- Test yourself: Work this out and convince yourself that it is correct

Dense Matrix Vector Multiply

- Let the matrix be partitioned across processes by columns, and the vector by corresponding rows.
  - If process i has columns M:N of the matrix, it also has elements M:N of the vectors
  - Simple partition (process 0 has the first block of columns, process 1 the second block, etc.)
  - process i has columns col(i):col(i+1)-1
- Problem: Compute the matrix-vector product with the distributed data structures
  - Send/receive requires intermediate buffers
  - Collective solution

Using MPI_Reduce_scatter

- Each process needs to accumulate a contribution from every process to its part of the result vector

```
do i=1,p
    recvcounts(i) = col(i+1)-col(i)
enddo
do j=1,n
   sum = 0
   do k=1, recvcounts(myrank)
        sum = sum + mv(j,k) * v(k)
   enddo
    localmv(j) = sum
enddo
call MPI_Reduce_scatter(localmv, my_vec, recvcounts, &
                 MPI_DOUBLE_PRECISION, MPI_SUM, comm, ierr)
```

Matrix times vector

Meaning of Reduce Scatter

- Reduce_scatter
  - Scatters contributions from all processes to all others
  - Combines (reduces) incoming contributions into a single buffer
  - MPI_Reduce_scatter_block like MPI_Reduce_scatter, but with the same size block on all processes
- Reduce_scatter also be used for distributed in-memory checkpoint with error correction
  - See SCR https://computation.llnl.gov/project/scr/
  - Providing Efficient I/O Redundancy in MPI Environments, Gropp, Ross, Miller, EuroPVM/MPI 2004, http://link.springer.com/chapter/10.1007/978-3-540-30218-6_17

Some Performance Issues

- MPI Collectives must handle the general case
- Implementations usually optimize for each collective operation separately
    - ♦ Assumption is make each individual collective as fast as possible, not the overall application
    - ♦ A Study of Process Arrival Patterns for MPI Collective Operations, Faraj, Patarasuk, Yuan, IJ Parallel Programming, 36:6 2008 http://link.springer.com/article/10.1007%2Fs10766-008-0070-9
- Implementations sensitive to progress (availability of CPU to advance communication)
    - ♦ Particularly important for nonblocking collectives
    - ♦ Nonblocking doesn't ensure concurrent execution