Week 4 – Collective communication, topologies

- Main topics this week
  - Collective communication, topologies
  - Project match making
- Reading
  - Pacheco, Chapters 4 and 5

Lecture 7

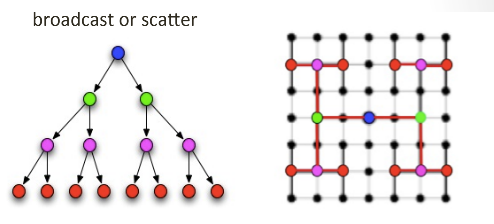
Collective Communication and block partitioning

MPI collective communication operations

- MPI_BCAST
- MPI_GATHER
- MPI_SCATTER
- MPI_REDUCE

Arguments similar to those of MPI_SEND and MPI_RECV

Implement by embedding virtual binary tree into actual topology



Other operations similarly based on trees

- Broadcast and scatter go from root to leaves
- Gather and reduce go from leaves to roots, reduce operates on data while gather just collects it
- Other ops like MPI_ALLGATHER: all to all, communication goes in one direction then other
- Next week's lab assignment: butterfly allreduce
  - Topology is butterfly
  - Communication goes in both directions

All that said

- You probably won't beat MPI_ALLGATHER which uses a better communication algorithm than plain old MPI_BCAST

But you can try!

Consider an algorithm that requires collective communication

$\alpha = x^T y$

$x = (x_1, x_2, \ldots, x_n)$

$y = (y_1, y_2, \ldots, y_n)$

We have p processors, how to implement?

first, assume p divides n

how do we divide the work?

how do we accumulate the result?

does replicating data make sense?

Data Decomposition Options

- Interleaved (cyclic)
  - Easy to determine "owner" of each index
- Block
  - Balances loads
  - More complicated to determine owner if *n* not a multiple of *p*

Block Decomposition Options

- Want to balance workload when *n* not a multiple of *p*
- Each process gets either $\lceil n/p \rceil$ or $\lfloor n/p \rfloor$ elements
- Seek simple expressions
  - Find low, high indices given an owner
  - Find owner given an index

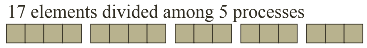
Method #1

- Let $r = n \bmod p$
- If $r = 0$, all blocks have same size
- Else
  - First r blocks have size $\lceil n/p \rceil$
  - Remaining p-r blocks have size $\lfloor n/p \rfloor$

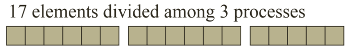
Examples

17 elements divided among 7 processes

17 elements divided among 5 processes



17 elements divided among 3 processes



Method #1 Calculations

- Indexing starts with 0
- First element controlled by process i

  $$i\lfloor n/p \rfloor + \min(i,r)$$
- Last element controlled by process i

  $$(i+1)\lfloor n/p \rfloor + \min(i+1,r) - 1$$
- Process controlling element j

  $$\min(\lfloor j/(\lfloor n/p \rfloor + 1) \rfloor, \lfloor (j-r)/\lfloor n/p \rfloor \rfloor)$$

Method #2

- Scatters larger blocks among processes
- First element controlled by process *i*

  $\lfloor in/p \rfloor$
- Last element controlled by process *i*

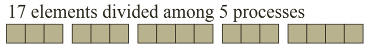
  $\lfloor (i+1)n/p \rfloor - 1$
- Process controlling element *j*

  $\lfloor p(j+1)-1)/n \rfloor$

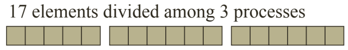
Examples

17 elements divided among 7 processes

17 elements divided among 5 processes



17 elements divided among 3 processes



Comparing Methods

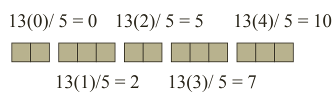
Our choice

Operations	Method 1	Method 2
Low index	4	2
High index	6	4
Owner	7	4

Assuming no operations for "floor" function

Pop Quiz

- Illustrate how block decomposition method #2 would divide 13 elements among 5 processes.



Block Decomposition Macros

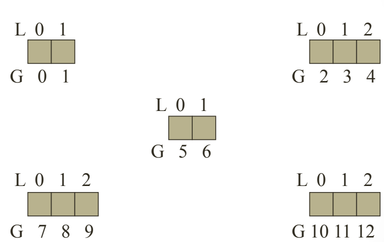
```
#define BLOCK_LOW(id,p,n)   ((i)*(n)/(p))

#define BLOCK_HIGH(id,p,n) \
        (BLOCK_LOW((id)+1,p,n)-1)

#define BLOCK_SIZE(id,p,n) \
        (BLOCK_LOW((id)+1)-BLOCK_LOW(id))

#define BLOCK_OWNER(index,p,n) \
        (((p)*(index)+1)-1)/(n))
```


Local vs. Global Indices



Looping over Elements

- Sequential program

```
for (i = 0; i < n; i++) {
   …
}
```

- Parallel program

Index *i* on this process…

```
size = BLOCK_SIZE (id,p,n);
for (i = 0; i < size; i++) {
   gi = i + BLOCK_LOW(id,p,n);
}
```

…takes place of sequential program's index *i*

When is a parallel implementation of our algorithm worth it?

- Cost of sending k byte message: $T = t_s + k\, t_c$
- Cost of a floating-point operation: ω
- On Frost (very roughly),
- $T = (3 \times 10^{-6}) + k\,(6 \times 10^{-10})$ sec
- $\omega = 4 \times 10^{-14}$ sec
  23 Tflops = 23×10^{12} floating-point ops / second
- $T/\omega = 7 \times 10^{7}$

Let's take it up a notch

- Matrix-vector multiply b = A*x (dense nxn A)
- One algorithm does dot products of rows of A with x
- Can we use what we did with dot products alone?
  Or do we need to reconsider?
- What's the best way to divide up this work?
- Consider costs of
  - Arithmetic
  - Communication

Ways to partition A

- Block row
- Block column
- Checkerboard

- Vectors are partitioned accordingly

How does the picture change for matrix-matrix multiply?

- C = A*B, both A and B nxn and dense (so C is, too)

- Need to partition all three matrices this time.

What about matrix transpose?

- It's all about communication—no arithmetic!