

# Introduction to Distributed Memory Models

## Introduction

1. Consider computations that can't fit in the memory of a single computer or would take hundreds of years to finish using a single computer
  - Harness the collective power of many computers
  - Develop an abstract model to reason about computing across many computers
    - Network model
    - Distributed memory model
    - Message-passing model
    - Communicating Sequential Processes model
2. Message-passing model
  - Many computers collectively carrying out a computation that communicate by passing messages to each other

## Simulating the Brain

1. Japanese researchers wrote a program to simulate 1% of the human brain
  - ~2 billion neurons and 10 trillion synapses
  - Synapses serve as a communication network, requiring 24 bytes of storage each
2. Suppose you have a computer with 16 GiB of RAM. How many are required to store the entire human brain?
  - $24 * 1e13 * 1e2 = 24e15$  total memory
  - $24e15 / 16 * 2^{34} = 1400000$  (1.4 million)
  - Sequoia (IBM supercomputer at LLNL) has ~19000 compute nodes in 2014

## A Basic Model of Distributed Memory

1. Machine model: Collection of nodes connected by a network
  - Each node has a processor connected to a private memory
  - Source must put a message on the network to communicate
    - In shared memory, we read and write shared variables
2. Rules
  - Fully connected: Always a path between two nodes
  - Bidirectional links: Link can carry a message in both directions at the same time
  - A node can perform up to one send and one receive at a time
  - Cost to send/receive  $n$  words: Time to send  $n$  words is  $a + B * n$ 
    - Cost to send a message is linear according to message size
    - $T_{msg}(n) = a + B * n$  where  $a$  is the latency [time] and  $B$  is inverse bandwidth [time/word]
  - $K$ -way congestion reduces bandwidth  $\rightarrow a + B * n * k$ 
    - Congestion is when messages are trying to use the same link at the same time
    - Cost is the same as if the beta term is serialized over the link

## Pipelined Message Delivery

1. Consider a linear (1-D) network with  $P$  nodes:
  - Message prep:  $a$  [time]
  - Link time:  $t$  [time]
  - Number of words:  $n$
2. How long does it take to send a message with  $n$  words, one word at a time?
  - $n = 1$ :  $a + t(P-1)$
  - $n = 2$ :  $a + t(P-1) + t$
  - $n = 3$ :  $a + t(P-1) + 2t$
  - $n = a + t(P-2) + tn$  ( $\alpha = a + t(P-2)$ )

## Getting a Feel for the Alpha Beta Model

1.  $T_{\text{msg}}(n) = a + B * n$ 
  - $\tau = \text{compute} [\text{time/op}]$
  - In practice,  $\tau \ll B \ll a$  ( $1e-12, 1e-9, 1e-6$ )
2. Which are true?
  - Computation < communication, so avoid communication (true)
  - It's faster to send a few large messages than many small messages (true)
  - None of the above

## Applying the Rules

1. Suppose you have a linear network with 8 nodes
  - Node 0 wants to send a message to node 2 at the same time that node 6 wants to send a message to node 3
2. How much time does it take for these messages to transmit?
  - The paths don't overlap, so the total time is  $a + B * n$

## Scenario 2 Quiz

1. Suppose you have a linear network with 8 nodes
  - Node 1 wants to send a message to node 6 at the same time that node 7 wants to send a message to node 4
2. How much time does it take for these messages to transmit?
  - $a + b * n$
  - I think there might be an error here; node 4 would be required to receive data from two messages at the same time, which violates the third rule

## Scenario 3 Quiz

1. Suppose you have a linear network with 8 nodes
  - Node 1 wants to send a message to node 6 at the same time that node 4 wants to send a message to node 7
2. How much time does it take for these messages to transmit?
  - $a + b * n * 2$  because the messages are traveling in the same direction

## Scenario 4 Quiz

1. Suppose you have a mesh network with 9 nodes arranged in a 3x3 grid
  - Node 0 wants to send a message to node 8 at the same time that node 4 wants to send a message to node 6
2. How much time does it take for these messages to transmit?
  - $a + b * n$
  - Assuming optimal pathing such that messages do not intersect

## Collective Operations - Part 1

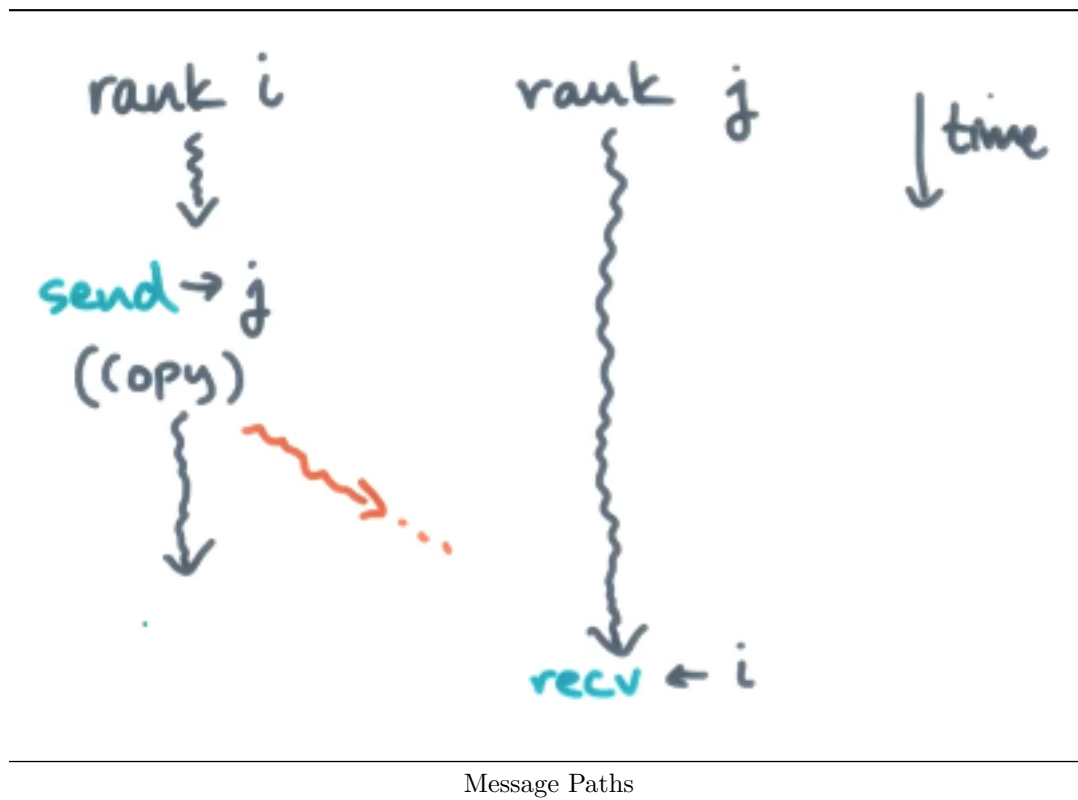
1. Tree-based reduce
  - Odd ranks send value to even ranks
  - Continue sending odd of the remaining ranks to the evens until only the zeroth node remains
    - It will contain the total sum
  - You can tell which nodes should be sending by their least significant bit
    - If LSB is 1, it should send. Otherwise, it should receive.

## Point to Point

1. Sequential pseudocode
  - Single-program, multiple data (SPMD)
  - Running copy = process
    - Rank is ID of running process (unique)
    - P is the number of running processes
  - Asynchronous send
    - `handle <- sendAsync(buf[1:n] -> dest)`
    - Return does NOT imply `buf[:]` has been sent. Do not modify “buf”
  - Asynchronous receive
    - `handle <- recvAsync(buf[1:n] -> src)`
    - Return does NOT imply `buf[:]` has been received
  - Blocking wait
    - `wait(handle1, ...)`
    - `wait(*)` waits for all pending sends and receives

## Point to Point Completion Semantics

1. Return implies corresponding “buf” is available for reuse
  - $\Rightarrow$  delivered, for `recvAsync()`
  - $\Rightarrow$  not much, for `sendAsync()`
2. Two-sided messaging: Every send must have a matching receive



## Send and Receive in Action

1. `RANK = 1`
  - `text[1:5] = '' // empty`

- `sendAsync('hello' -> 2)`
  - `waitAll()`
  - `recvAsync(text[:] <- 2)`
  - `waitAll()`
2. `RANK = 2`
- `text[1:5] = '' // empty`
  - `sendAsync('world' -> 1)`
  - `waitAll()`
  - `recvAsync(text[:] <- 1)`
  - `waitAll()`
3. What is the value of `text[:]` when the processes complete?
- hello on rank 1, world on 2
  - world on rank 1, hello on 2
  - Either is possible
  - Neither is possible
  - Processes don't complete (true, this causes deadlock)

## Send and Receive Revisited

1. The initial values of six nodes are as follows:
- Rank 0: `x = 6`
  - Rank 1: `x = 6`
  - Rank 2: `x = 7`
  - Rank 3: `x = 3`
  - Rank 4: `x = 8`
  - Rank 5: `x = 4`
2. What is the final state of the `x` values after the following pseudocode completes?

```
for i <- 0 to P-1 do
  sendAsync(x -> (RANK+1) % P)
  recvAsync(y -> (RANK+P-1) % P)
  waitAll()
  swap(x,y)
```

\* initial: 6, 6, 7, 3, 8, 4  
 \* i = 0: 4, 6, 6, 7, 3, 8  
 \* i = 1: 8, 4, 6, 6, 7, 3  
 \* i = 2: 3, 8, 4, 6, 6, 7  
 \* i = 3: 7, 3, 8, 4, 6, 6  
 \* i = 4: 6, 7, 3, 8, 4, 6  
 \* i = 5: 6, 6, 7, 3, 8, 4

## All to One Reduce Pseudocode

1. Assume  $P = 2^k$

```
let s = local value
bitmask <- 1
while bitmask < P do
  partner <- rank ^ bitmask
  if rank & bitmask then
    sendAsync(s -> partner)
    waitAll()
    break
  else
    recvAsync(t <- partner)
```

```

        waitAll()
        s <- s + t
    bitmask <- (bitmask << 1)
if rank = 0
    print(s)

```

## All to One Reduce Pseudocode Quiz

1. Fix the pseudocode to work if P is not a power of 2
  - Only senders drop out
  - Senders have 1 at the bitmask position

```

let s = local value
bitmask <- 1
while bitmask < P do
    partner <- rank ^ bitmask
    if rank & bitmask then
        sendAsync(s -> partner)
        waitAll()
        break
    elseif partner < P
        recvAsync(t <- partner)
        waitAll()
        s <- s + t
    bitmask <- (bitmask << 1)
if rank = 0
    print(s)

```

## Vector Reductions

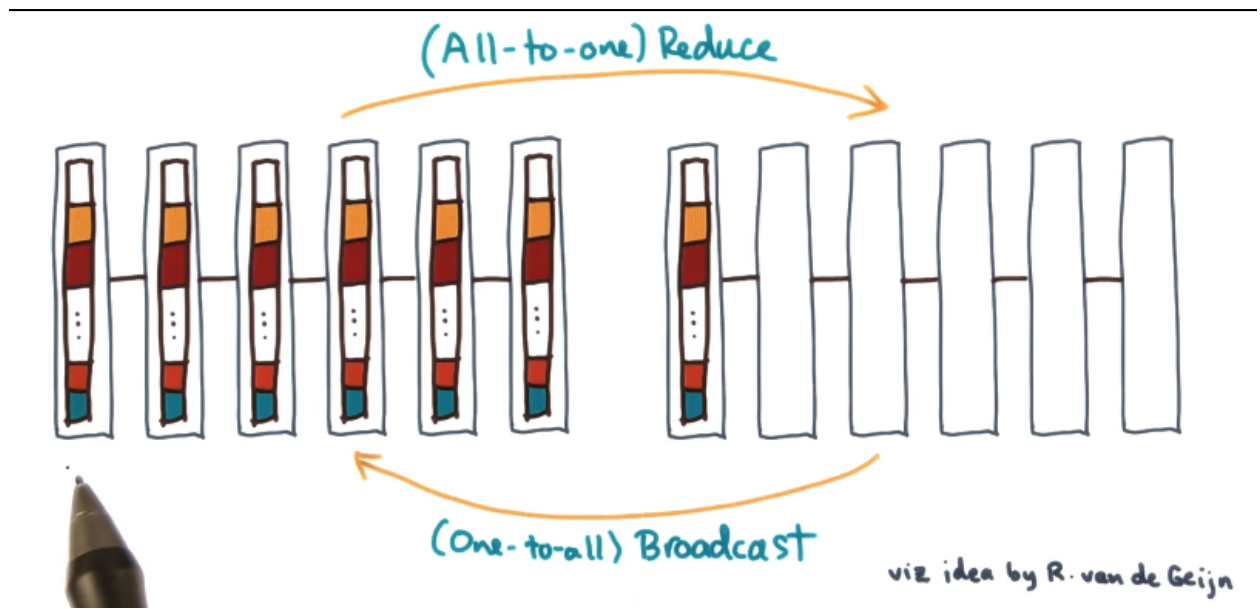
1. Vector reductions means applying the operation element-wise to a vector
  - Instead of sending a scalar, send a vector (sendAsync(s[:] -> partner)

## Vector Reductions Quiz

1. What is the time to do a vector reduction?
  - $a + Bn$
  - $a * \log P + Bn$
  - $a + B * n * \log P$
  - $(a + Bn) * \log P$  (true)

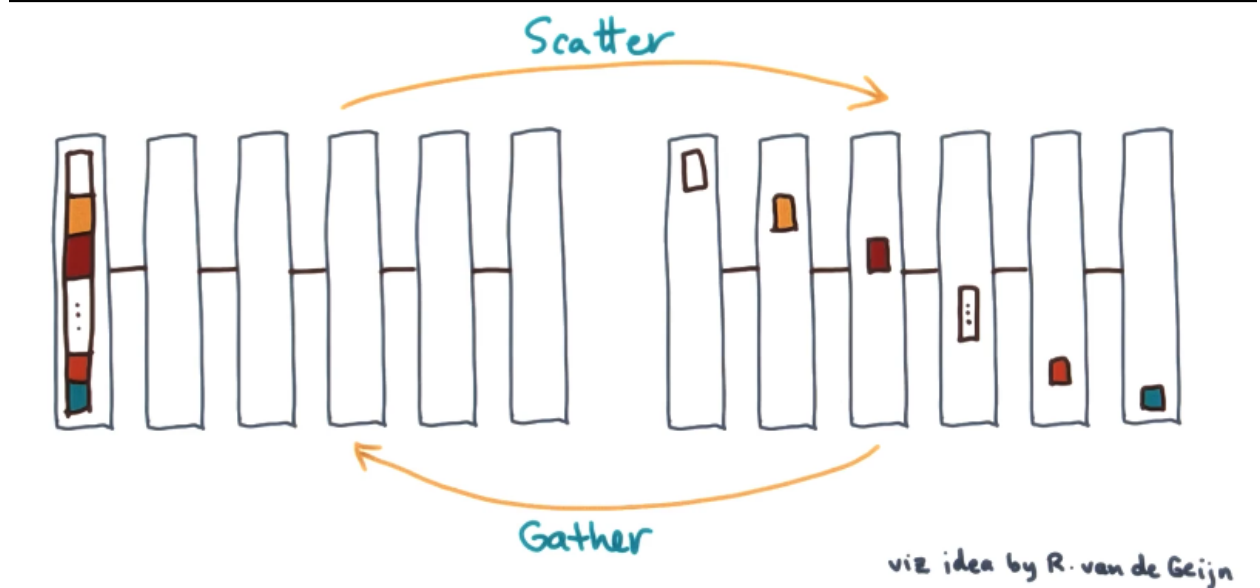
## More Collectives

1. Corollary to a reduce is a one-to-all broadcast
  - One processor has all the data initially and wants to send a copy to all other processors
  - Reduce and broadcast are duals



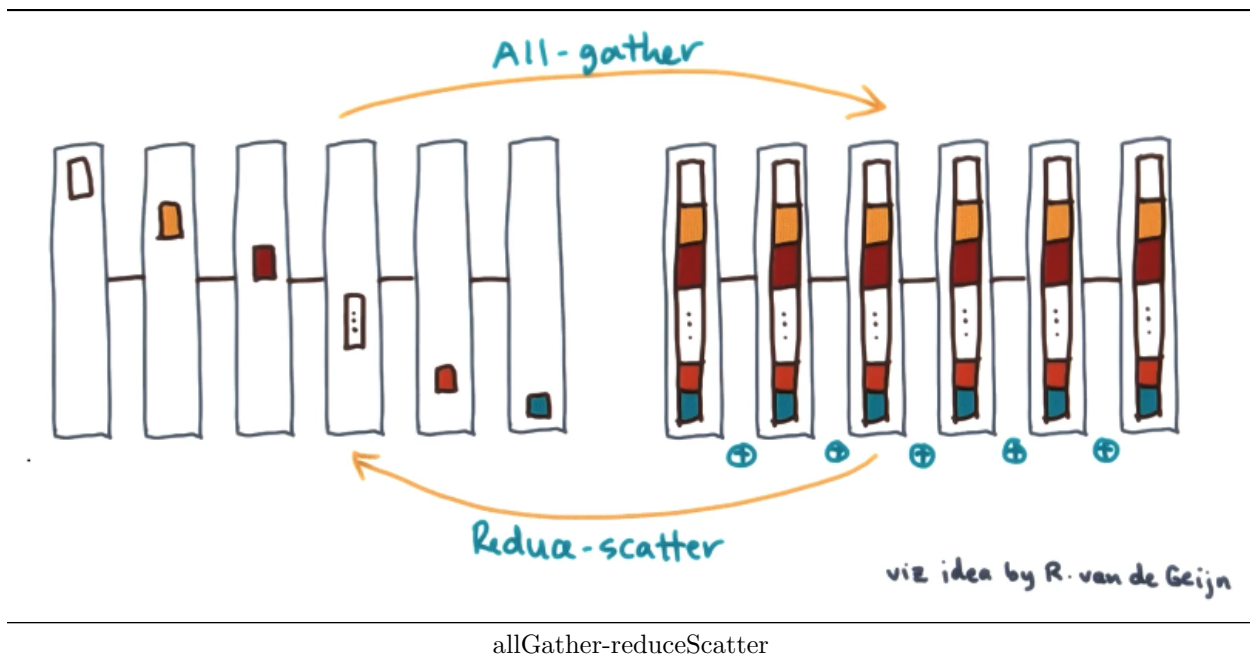
Reduce-Broadcast

2. Scatter sends a piece of its data to each of the other processors
  - The dual to a scatter is a gather



Scatter-Gather

3. All-gather: Similar to a gather, but instead of only the root having all of the data, each node contains all of the data
  - Dual is a reduce-scatter
    - All processes contain a vector of data
    - They globally reduce the vector using some sort of vector-reduce
    - Result is distributed to all processes



## A Pseudocode API for Collectives

1. Suppose every processor has a private array of size  $n$ 
  - `reduce(Alocal[1:n], root)`
    - Must be executed on all processors
  - `broadcast(Alocal[1:n], root)`
  - `gather(In[1:m], Out[1:m][1:P], root)`
    - Out is only valid on the root processor
    - $n = m * P$
  - `scatter(In[1:m][1:P], root, Out[1:m])`
  - `allGather(In[1:m], Out[1:m][1:P])`
  - `reduceScatter(In[1:m][1:P], Out[1:m])`
2. Reshaping
  - `reshape(A[1:m][1:n]) -> A[1:m * n]`
  - `reshape(A[1:m] * n) -> A[1:m][1:n]`
  - Column-major by convention

## All Gather - From Building Blocks

1. Implement an allGather using reduce, broadcast, scatter, gather, reshape

```
gather(In, Out, root)
broadcast(reshape(Out), root)
```

## Collective Lower Bounds

1.  $T(n) = (a + B * n) * \log(P)$ 
  - On a linear network, if a node can only send and receive one message at a time, we require at least  $\log(P)$  rounds of communication
    - Therefore,  $a * \log(P)$  is optimal
  - Each process has  $n$  words of data and must send all  $n$  words
    - $T(n) = (a + B * n) * \log(P) \geq n * (P-1) \text{ words}$
    - If all nodes send their data simultaneously, the lower bound on time is  $n * B$

- This suggests the tree-based scheme is sending too much data by a factor of  $\log(P)$
- Lower bound for all collectives:
    - $T(n) = O(a * \log(P) + B * n)$

## All Gather Quiz

- If we implement allGather using the gather/broadcast approach and gather and broadcast both achieve the lower bound, is allGather optimal?
  - Yes; a constant number of optimal primitives is still optimal

## Implement Scatter Quiz

- Consider the following pseudocode:

```
scatter(In[1:m][1:P], root, Out[1:m])
  if RANK == root then
    for i != root do
      sendAsync(In[:] [i], i)
  else
    recvAsync(Out[:], root)
waitAll()
```

- How much communication time does this algorithm need?
  - $a + B * n$
  - $a * \log(P) + B * m$
  - $(a + B * m) * \log(P)$
  - $a * P + B * m$
  - $(a + B * m) * P$  (true)

## Implementing Scatter and Gather - Part 2

- Instead of the naive implementation that scales linearly with  $P$ , we need a different approach
  - Instead, split the data in half and send it to another node
  - Continue splitting in half at each node until the data has propagated to all nodes
- What is the communication complexity?
  - Iteration  $i$ :  $n_i = n / 2^i$
  - $T(i) = a + B * n_i = a + B * n / 2^i$
  - $T(n) = \sum(T_i) \text{ from } 1 \text{ to } \log(P)$ 
    - $a * \log(P) + B * n * (P-1) / P$
    - This is the lower bound with respect to latency and bandwidth

## When to Use Tree-Based Reduce

- When is the tree-based scheme okay? ( $T(n) = a * \log(P) + B * n * \log(P)$ )
  - $B * n \ll a$  (true)
  - $a \ll B * n$
  - $\log(P) \ll P$
  - $n$  is “small” (true)
  - inverse bandwidth  $\gg$  latency

## What’s Wrong with Tree-Based Reduce?

- What causes the  $B$  term to be suboptimal in a tree-based reduce?
  - There is redundant communication; each round sends the same data



## Bucketing Algorithm for Collectives

1. Bandwidth term encourages every process sending data at each round
  - This results in  $P-1$  communication steps
  - $T(n = m * P) = (a + Bn/P)(P-1) \sim aP + Bn$ 
    - Suboptimal with respect to the alpha term
    - This is okay if  $n/P \ll a/B$

## Bandwidth Optimal Broadcast

1. Give a bandwidth-optimal algorithm for broadcast:
  - Assuming allGather uses bucketing

```
broadcast(A[1:m*P], root)
  let B[1:m][1:P] <- reshape(A)
  T[1:m] = temp array
  scatter(B[1:m][1:P], root, T[1:m])
  allGather(T[1:m], B[1:m][1:P], root)
  A <- reshape(B)
```

## All Reduce

1. All-reduce is similar to reduce, but the answer is on all processors
2. Which pair of collectives can be combined to obtain a bandwidth-optimal implementation of allReduce?
  - Scatter
  - Gather
  - reduceScatter (true)
  - allGather (true)

## Conclusion

1. How do we think about efficiency in terms of communication and computation?
2. Message-passing model thoughts:
  - Who/how many processes
  - When/how processes communicate
  - How processes are connected
3. Open research question: Is there a framework for developing efficient algorithms independent of the network?