Lecture 3: Collective communication

Jack Poulson and Rob Schreiber

CME 194
Stanford University

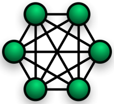
April 8, 2013

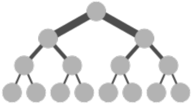
Outline

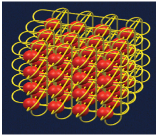
Basic communication cost model

Recall $\alpha + \beta n$ model:

- Can only send one message at a time
- Cost depends on *latency*, α, message length, *n*, and *bandwidth*, $1/\beta$

Will assume network is fully-connected, 

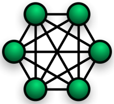
but, in practice, topology is usually *fat-tree* 

or *multi-dimensional torus* 

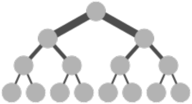
Basic communication cost model

Recall $\alpha + \beta n$ model:

- Can only send one message at a time
- Cost depends on *latency*, α, message length, n, and *bandwidth*, $1/\beta$

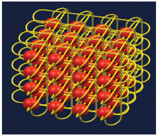


Will assume network is fully-connected,



but, in practice, topology is usually *fat-tree*

or *multi-dimensional torus*

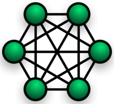


Basic communication cost model

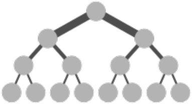
Recall $\alpha + \beta n$ model:

- Can only send one message at a time
- Cost depends on *latency*, α, message length, *n*, and *bandwidth*, $1/\beta$

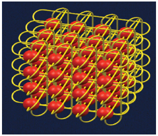
Will assume network is fully-connected,



but, in practice, topology is usually *fat-tree*



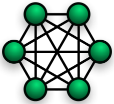
or *multi-dimensional torus*

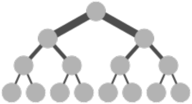


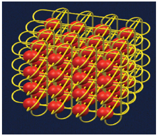
Basic communication cost model

Recall $\alpha + \beta n$ model:

- Can only send one message at a time
- Cost depends on *latency*, α, message length, *n*, and *bandwidth*, $1/\beta$

Will assume network is fully-connected, 

but, in practice, topology is usually *fat-tree* 

or *multi-dimensional torus* 

Binomial tree broadcast (short messages)

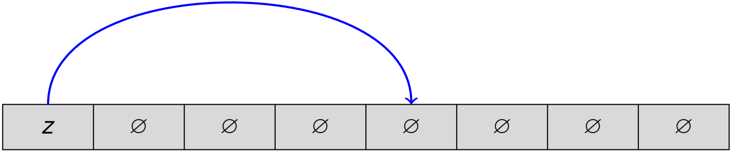
z	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

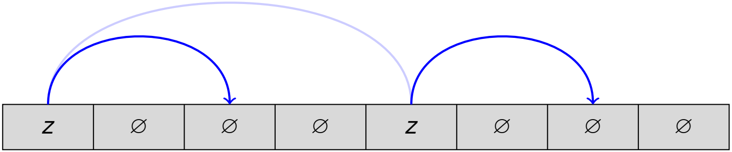


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n)\log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

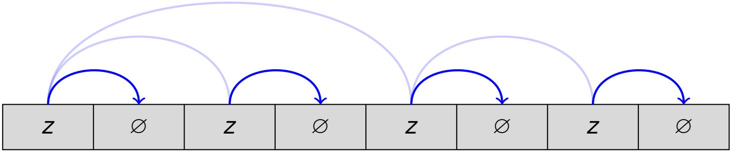


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

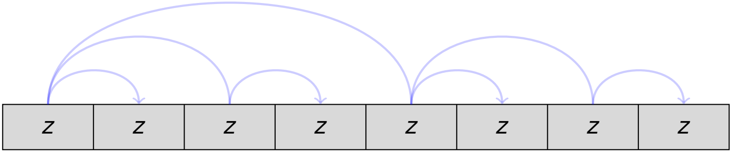


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

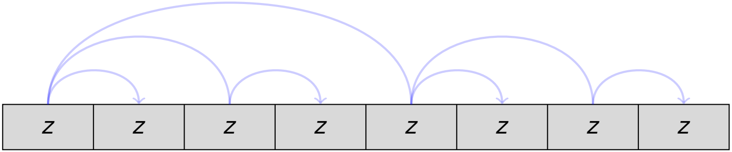


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

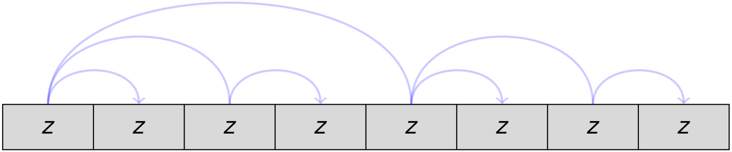


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

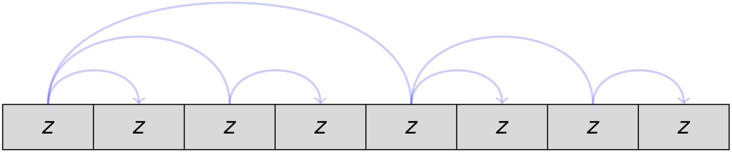


There are $\log_2 p$ stages

Total cost is $(\alpha + \beta n) \log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

Binomial tree broadcast (short messages)

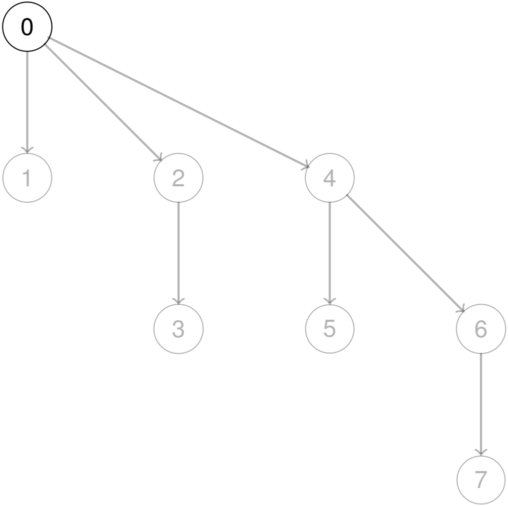


There are $\log_2 p$ stages

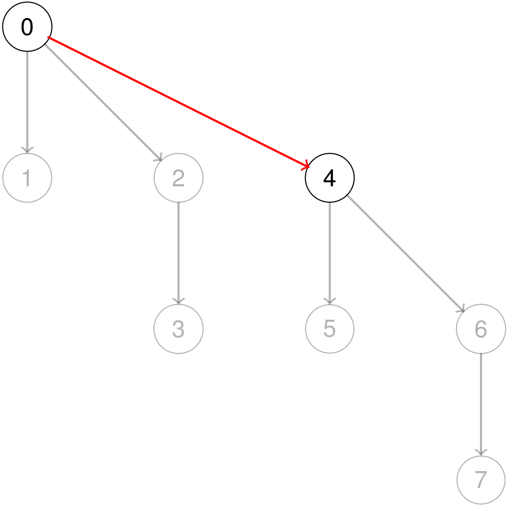
Total cost is $(\alpha + \beta n)\log_2 p$

$\beta n \log_2 p$ should be avoided when n is large...

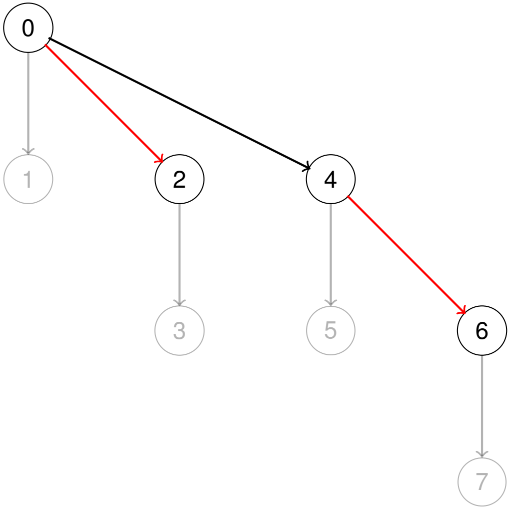
Binomial tree broadcast (short messages)



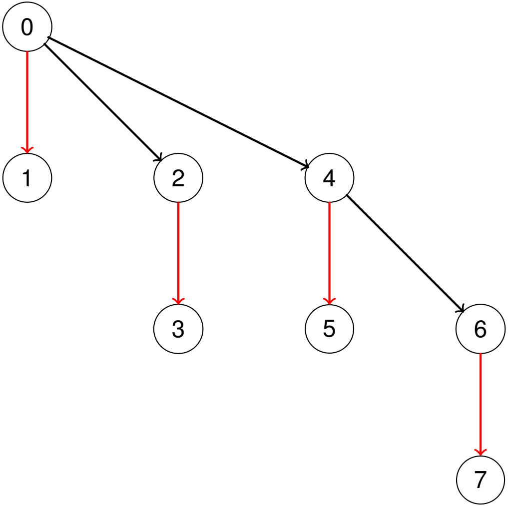
Binomial tree broadcast (short messages)



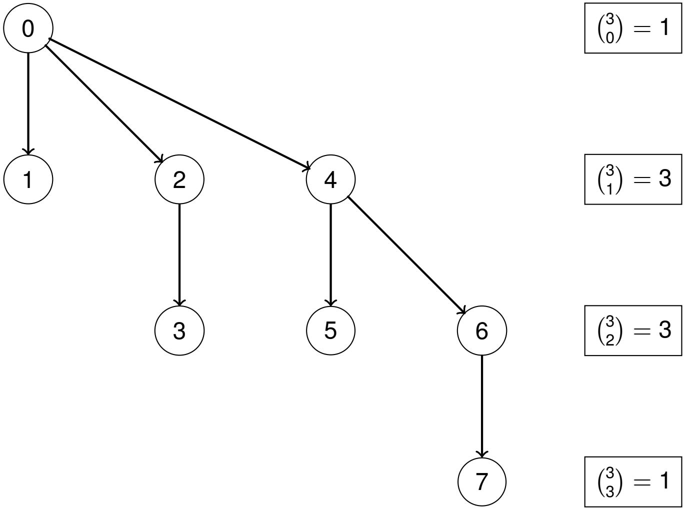
Binomial tree broadcast (short messages)



Binomial tree broadcast (short messages)



Binomial tree broadcast (short messages)



Fox/van-de-Geijn algorithm (long messages)

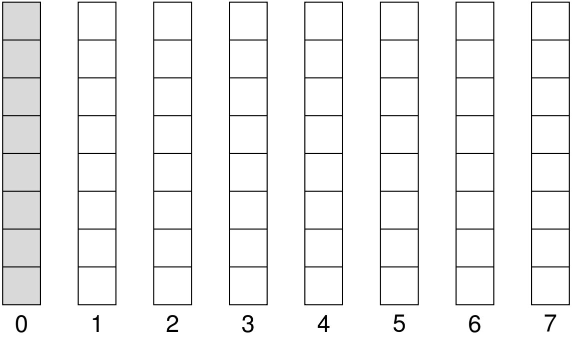
Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast

Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

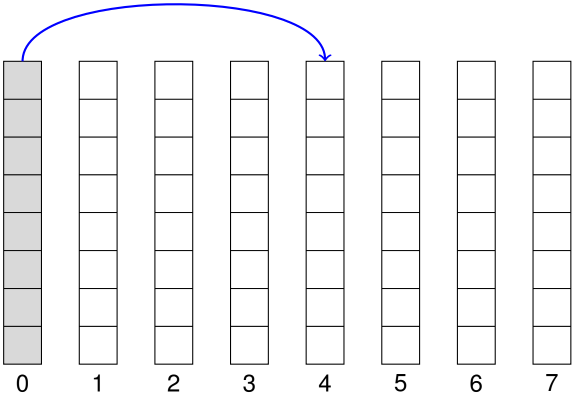
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

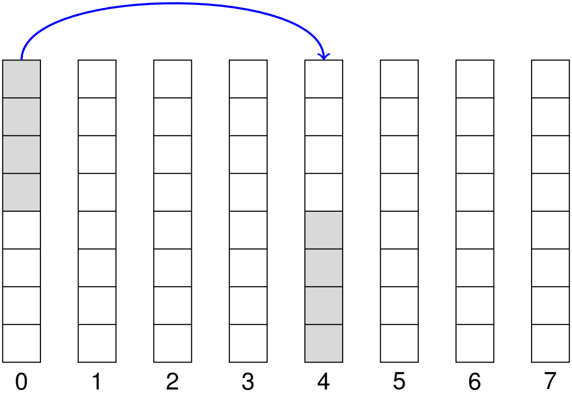
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

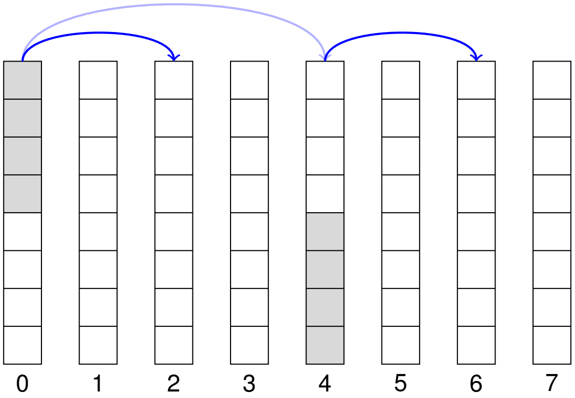
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

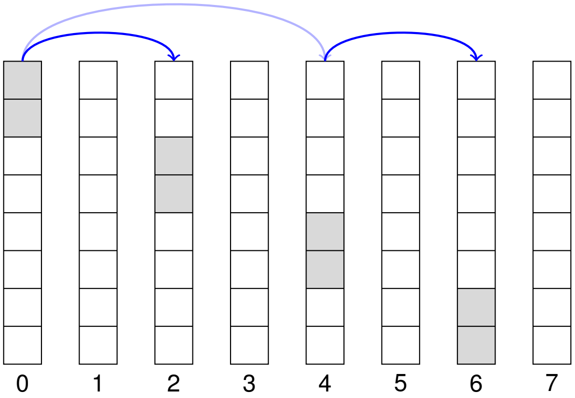
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

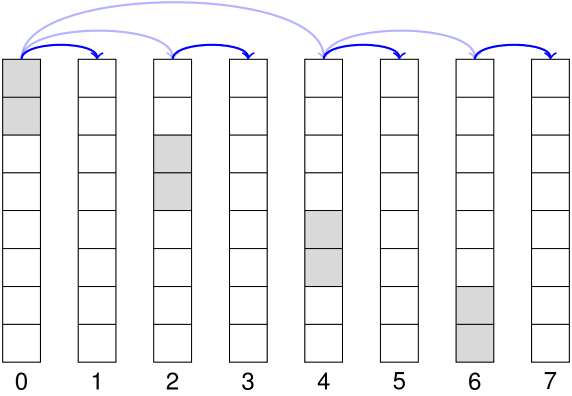
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

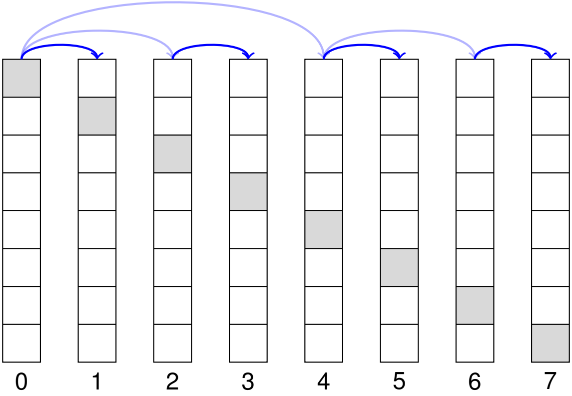
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

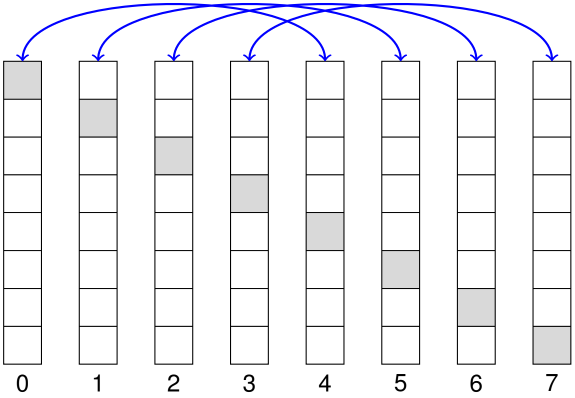
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

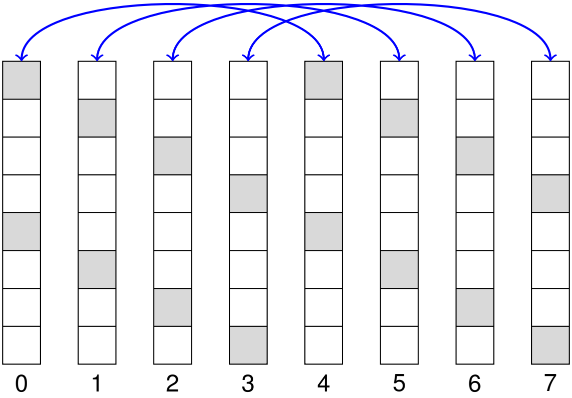
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

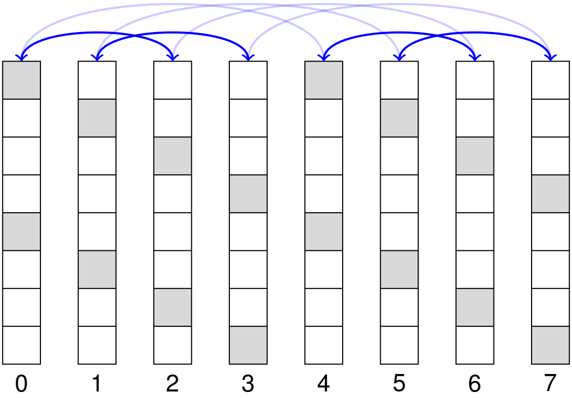
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

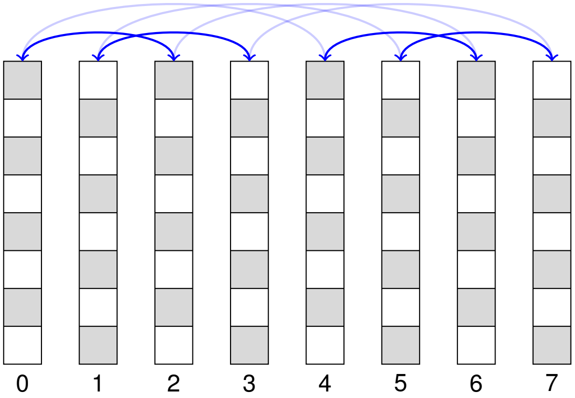
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

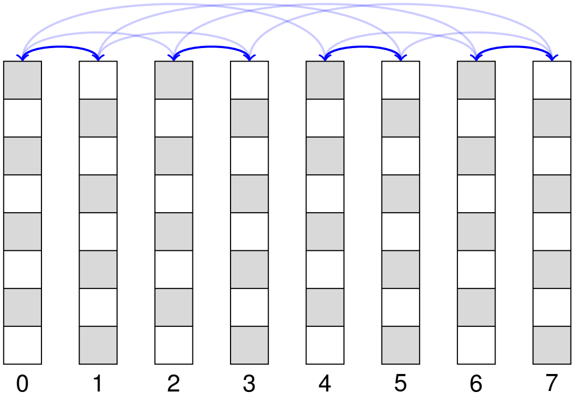
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

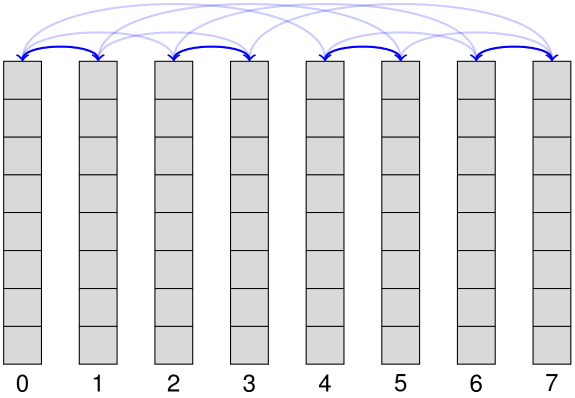
Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



Fox/van-de-Geijn algorithm (long messages)

Scatter ($\alpha \log_2 p + \beta n \frac{p-1}{p}$) then AllGather ($\alpha \log_2 p + \beta n \frac{p-1}{p}$)

Lower volume cost than $\alpha \log_2 p + \beta n \log_2 p$ binomial Broadcast



MPI_Bcast

`MPI_Bcast( void* buffer, int n, MPI_Datatype type, int root, MPI_Comm comm )`

Broadcasts *n* items of type `type` from process with rank `root` with respect to communicator `comm`.

Typically switches from binomial to Fox/vdG algorithm when *n* is sufficiently large.

`MPI_Bcast`

```
MPI_Bcast( void* buffer, int n, MPI_Datatype
type, int root, MPI_Comm comm )
```

Broadcasts *n* items of type `type` from process with rank `root` with respect to communicator `comm`.

Typically switches from binomial to Fox/vdG algorithm when *n* is sufficiently large.

```
MPI_Bcast( void* buffer, int n, MPI_Datatype
type, int root, MPI_Comm comm )
```

Broadcasts *n* items of type `type` from process with rank `root` with respect to communicator `comm`.

Typically switches from binomial to Fox/vdG algorithm when *n* is sufficiently large.

Reduce

Each process has vector of same length and we would like to give sum to single process:

$$x = \sum_{i=0}^{p-1} \hat{x}_i$$

Can run Broadcast algorithms in reverse (with summation).

Therefore, same communication costs!

Can of course replace summation with other binary operations, e.g., entrywise maximum or product

Reduce

Each process has vector of same length and we would like to give sum to single process:

$$x = \sum_{i=0}^{p-1} \hat{x}_i$$

Can run Broadcast algorithms in reverse (with summation).

Therefore, same communication costs!

Can of course replace summation with other binary operations, e.g., entrywise maximum or product

Reduce

Each process has vector of same length and we would like to give sum to single process:

$$x = \sum_{i=0}^{p-1} \hat{x}_i$$

Can run Broadcast algorithms in reverse (with summation).

Therefore, same communication costs!

Can of course replace summation with other binary operations, e.g., entrywise maximum or product

Reduce

Each process has vector of same length and we would like to give sum to single process:

$$x = \sum_{i=0}^{p-1} \hat{x}_i$$

Can run Broadcast algorithms in reverse (with summation).

Therefore, same communication costs!

Can of course replace summation with other binary operations, e.g., entrywise maximum or product

```
MPI_Reduce( void* sendBuf, void* recvBuf, int
n, MPI_Datatype type, MPI_Op op, int root,
MPI_Comm comm )
```

Reduces each process in communicator `comm`'s *n* items from `sendBuf` into process `root`'s `recvBuf` using binary operation implied by `op`, e.g., `MPI_SUM` or `MPI_MAX`.

Related routines are `MPI_Allreduce` (give every process the result) and `MPI_Reduce_scatter` (give every process a subset of the result)

MPI_Reduce

```
MPI_Reduce( void* sendBuf, void* recvBuf, int
n, MPI_Datatype type, MPI_Op op, int root,
MPI_Comm comm )
```

Reduces each process in communicator `comm`'s *n* items from `sendBuf` into process `root`'s `recvBuf` using binary operation implied by `op`, e.g., `MPI_SUM` or `MPI_MAX`.

Related routines are `MPI_Allreduce` (give every process the result) and `MPI_Reduce_scatter` (give every process a subset of the result)

MPI_Reduce

```
MPI_Reduce( void* sendBuf, void* recvBuf, int
n, MPI_Datatype type, MPI_Op op, int root,
MPI_Comm comm )
```

Reduces each process in communicator `comm`'s *n* items from `sendBuf` into process `root`'s `recvBuf` using binary operation implied by `op`, e.g., `MPI_SUM` or `MPI_MAX`.

Related routines are `MPI_Allreduce` (give every process the result) and `MPI_Reduce_scatter` (give every process a subset of the result)

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta) \log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta) \log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta)\log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta) \log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta)\log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta)\log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed inner-product

$$\alpha := y^H x$$

- Each process gets subset of x and y, e.g., x_i and y_i
- Then $\alpha = \sum_{i=0}^{p-1} y_i^H x_i = \sum_{i=0}^{p-1} \hat{\alpha}_i$
- Have each process form $\hat{\alpha}_i := y_i^H x_i$ then Reduce
- Work is $2\frac{n}{p} + (\alpha + \beta)\log_2 p$ with binomial Reduce

Setting $y = x$ yields parallel $\|x\|_2$.

Setting $y = \text{ones}(n, 1)$ yields parallel $\|x\|_1$.

Example: distributed $\| \cdot \|_\infty$

$$\|x\|_\infty = \max_j \chi_j$$

If we give each process subset of x, say x_i, then

$$\|x\|_\infty = \max_i \|x_i\|_\infty$$

Just like distributed $\|x\|_1$, but using max instead of $+$ for the reduction operation.

Example: distributed $\|\cdot\|_\infty$

$$\|x\|_\infty = \max_j \chi_j$$

If we give each process subset of x, say x_i, then

$$\|x\|_\infty = \max_i \|x_i\|_\infty$$

Just like distributed $\|x\|_1$, but using max instead of $+$ for the reduction operation.

Example: distributed $\|\cdot\|_\infty$

$$\|x\|_\infty = \max_j \chi_j$$

If we give each process subset of x, say x_i, then

$$\|x\|_\infty = \max_i \|x_i\|_\infty$$

Just like distributed $\|x\|_1$, but using max instead of $+$ for the reduction operation.

Homework 1: Due Friday, April 12

- Write binomial Reduce which works for power-of-two numbers of processes
- Generate random distributed vector x and compute its two-norm
- BONUS: Use `MPI_Reduce` with custom `MPI_Op` to efficiently compute sum of maximum k absolute values of entries of x

Homework 1: Due Friday, April 12

- Write binomial Reduce which works for power-of-two numbers of processes
- Generate random distributed vector x and compute its two-norm
- BONUS: Use `MPI_Reduce` with custom `MPI_Op` to efficiently compute sum of maximum k absolute values of entries of x

Homework 1: Due Friday, April 12

- Write binomial Reduce which works for power-of-two numbers of processes
- Generate random distributed vector x and compute its two-norm
- BONUS: Use `MPI_Reduce` with custom `MPI_Op` to efficiently compute sum of maximum k absolute values of entries of x