Greedy Trees for MPI Reductions

Julien Langou
University of Colorado Denver

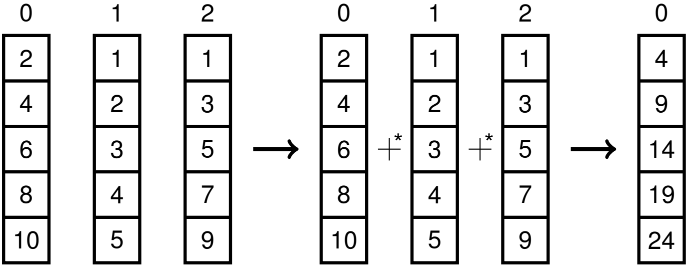
MARCH 28, 2013

Table of Contents

Table of Contents

What is a reduction?

- We will consider a set of p processors with distributed memory and each processor has a message of size m.
- A reduction combines the messages entry-wise, and returns the value on one specified processor.
- Example: $p = 3$, $m = 5$



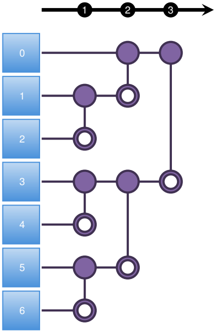
* Can be any associative operation.

Communication Model

- Unidirectional system - At any given time a processor is allowed to send a message to another processor or receive a message from another processor, but not both.
- Communication time between two processors is given by the linear model, $\alpha + \beta m$, where α is the latency (start up time) and β is the inverse bandwidth.
- The time for the computation is given γm.
- Cannot overlap communication and computation.
- In practice we have the relationship, $\alpha >> \beta > \gamma$.
- The message m can be split into q segments of size s_i.
- Uniform segmentation: $s_i = s$ for all i.

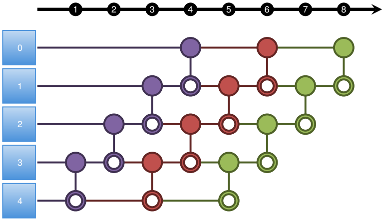
Table of Contents

Binomial Tree



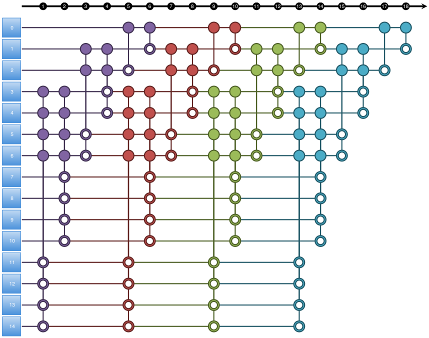
- Best for small messages, $\alpha >> \beta m$.
- Minimizes the number of communications started.
- No segmentation. Only increases latency.
- $time = \lceil \log_2 p \rceil (\alpha + \beta m + \gamma m)$

Pipeline Tree



- Best for long messages, $\alpha << \beta m$.
- Poor startup, but optimal overhead for trailing segments.
- *time* $=$ $((p-1)+2(q-1))(\alpha+\beta s+\gamma s)$

Binary Tree



- At each step, two different processors send to the same receiving processor.
- An iteration is therefore twice as long as compared to the other trees.
- Good for medium sized messages, $\alpha \approx \beta m$.
- $time = (2(\lceil \log_2 p + 1 \rceil - 1) + 4(q - 1))(\alpha + \beta s + \gamma s)$

Time Complexity for Binomial, Pipeline, and Binary

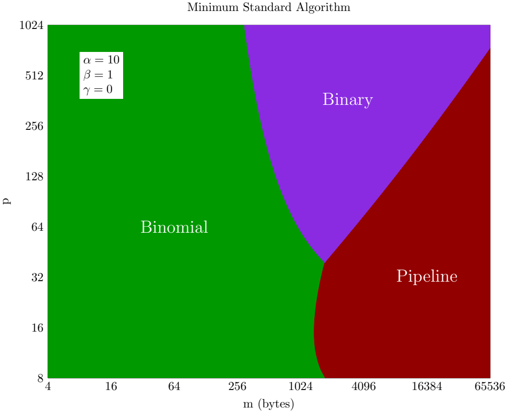
Binomial	Time	$\lceil log_2 p \rceil (\alpha + \beta m + \gamma m)$
Pipeline	Time	$(p-1)(\alpha + \beta s + \gamma s) + 2(q-1)(\alpha + \beta s + \gamma s)$
	s_{opt}	$\left(\dfrac{2m\alpha}{(p-3)(\beta+\gamma)} \right)^{1/2}$
	T_{opt}	$\left[((p-3)\alpha)^{1/2} + (2m(\beta+\gamma))^{1/2} \right]^2$
Binary	Time	$2\left(\lceil log_2(p+1) \rceil - 1\right)(\alpha + \beta s + \gamma s) + 4(q-1)(\alpha + \beta s + \gamma s)$
	s_{opt}	$\left(\dfrac{2m\alpha}{(N-3)(\beta+\gamma)} \right)^{1/2}$
	T_{opt}	$2\left[((N-3)\alpha)^{1/2} + (2m(\beta+\gamma))^{1/2} \right]^2$

$N = \lceil \log_2(p+1) \rceil$, s_{opt} is the optimal equi-segment size, and T_{opt} is the time for the algorithm at s_{opt}. Formulae are valid for $p > 3$.

Lower Bounds for each term in communication time

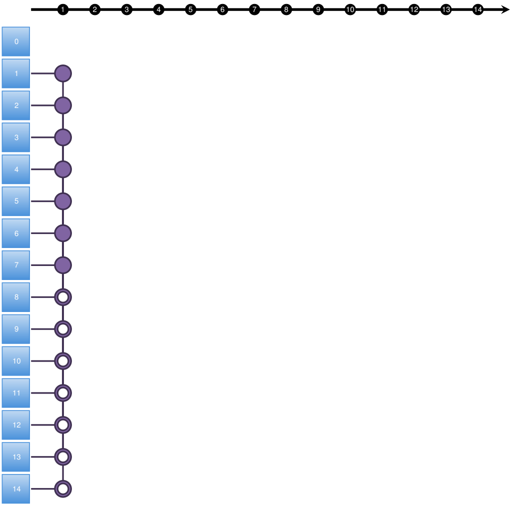
	Latency	Bandwidth	Computation
Reduce	$\lceil log_2 p \rceil \alpha$	$2m\beta$	$\frac{p-1}{p} m\gamma$
Binomial	$\lceil \log_2 p \rceil \alpha$	$\lceil \log_2 p \rceil m\beta$	$\lceil \log_2 p \rceil m\gamma$
Pipeline	$(p-1)\alpha$	$(p-3+2m)\beta$	$(p+2m-3)\gamma$
Binary	$2(N-1)\alpha$	$2(N-3+2m)\beta$	$2(N-3+2m)\gamma$

*$N = \lceil \log_2(p+1) \rceil$



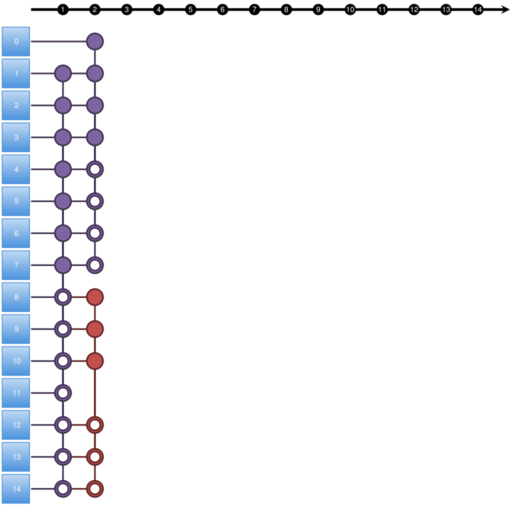
Regions where binomial or pipeline or binary is better in term of the number of processors (p) and the message size (m). For each algorithm, each p and each m, the optimal segment size is used. The machine parameters are $\alpha = 10$, $\beta = 1$, $\gamma = 0$.

Greedy Tree - Unidirectional, No Computation



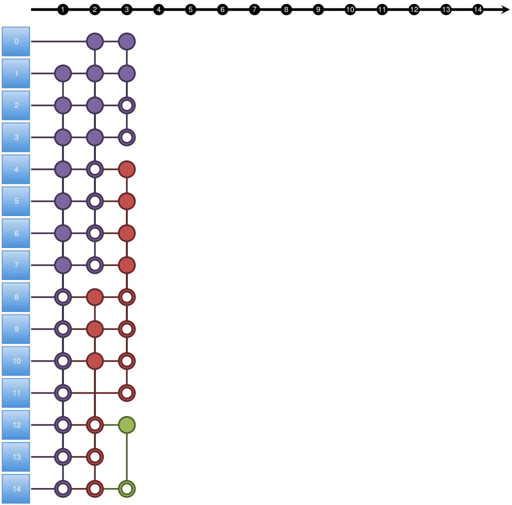
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



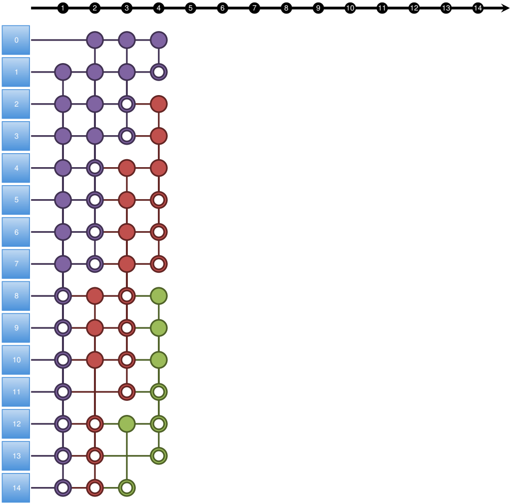
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



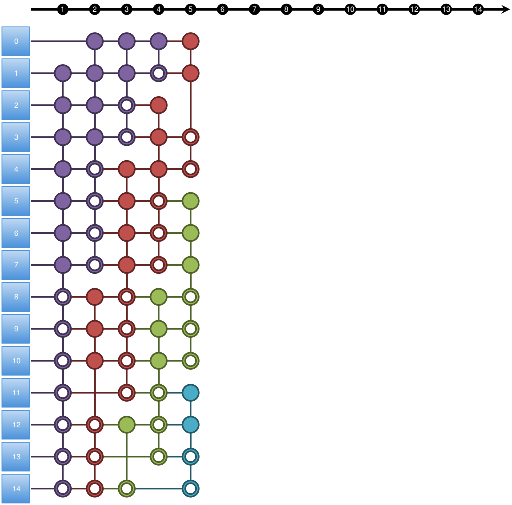
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



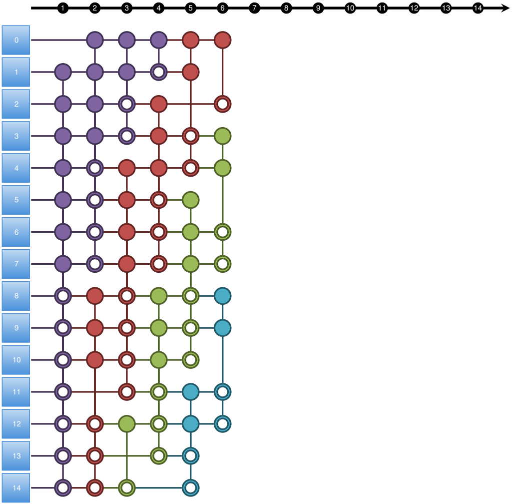
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



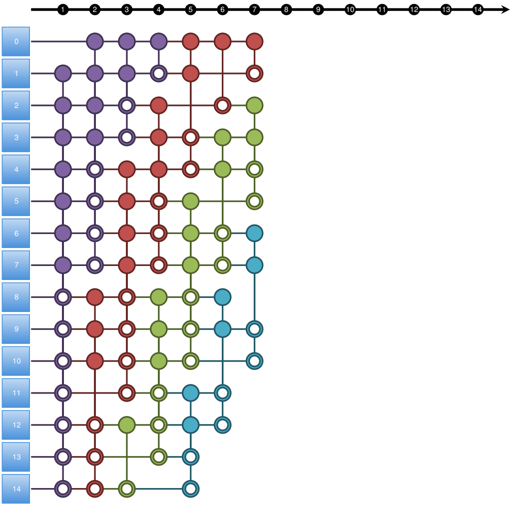
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



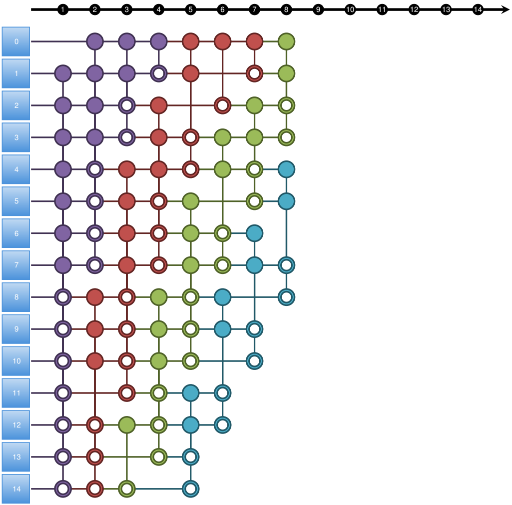
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



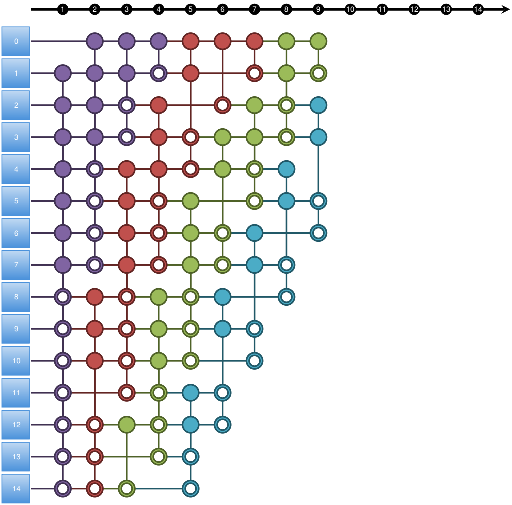
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



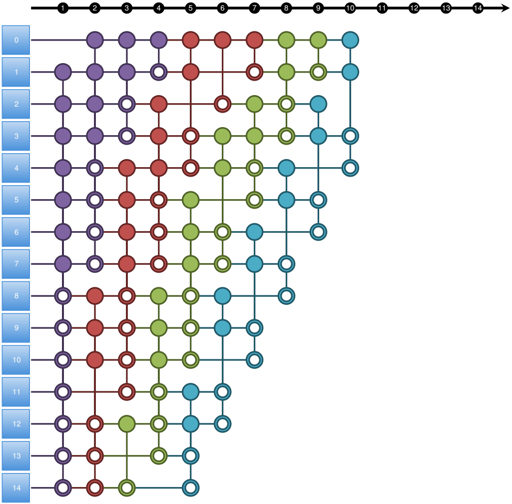
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



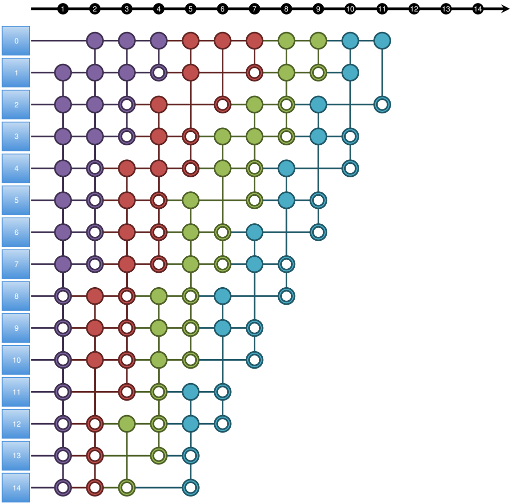
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



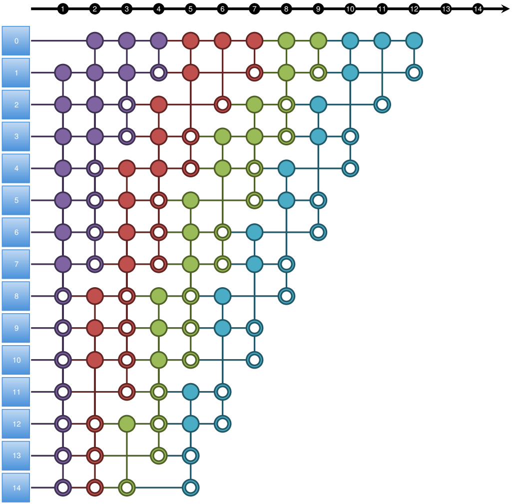
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



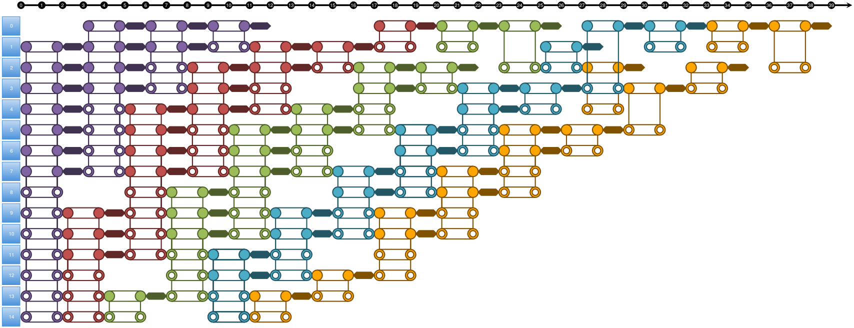
- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional, No Computation



- Optimal for uniform segmentation.
- Motivated by greedy QR factorization scheme. [Cosnard and Robert '86]
- Different tree for each segment.
- We assume the operation is commutative (and associative as well, of course).

Greedy Tree - Unidirectional with Computation



15 processors, 5 equi-segments, $t_{comm} = 2$ and $t_{comp} = 1$.
Filled in circles represent receiving processors, open circles represent sending processors, and hexagons represent computation.

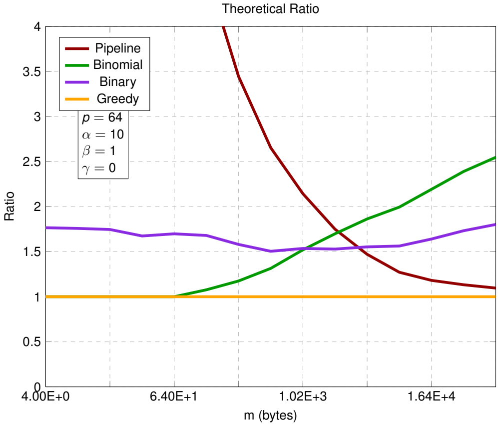
Theoretical Results

Theorem

In an unidirectional system assuming that segments are reduced in order the time complexity of the greedy algorithm is no worse than any reduction algorithm.

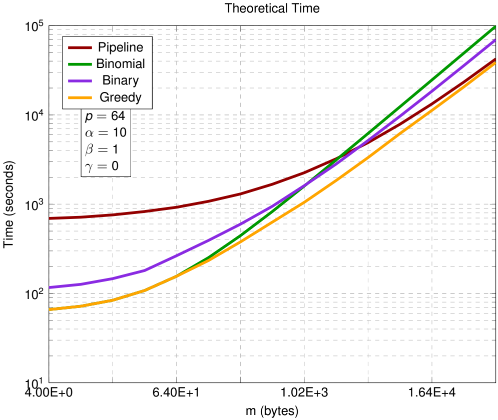
- Segments do not have to be equal size.
- Reducing segments in order allows for less work space (buffer size is $2 \times$ segment size), also this will work for non-commutative operations.
- Each algorithm is tuned for optimal uniform segmentation.
- For given parameters, $p, m, \alpha, \beta, \gamma$, which algorithm is the best?

Theoretical Results (ratio)



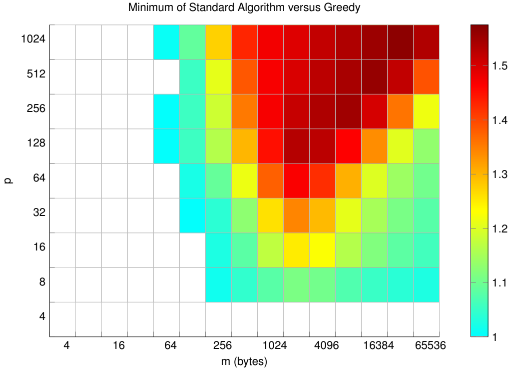
- Each algorithm is tuned for optimal uniform segmentation.
- 50% speed up for “medium sized” messages.

Theoretical Results (time)



- Each algorithm is tuned for optimal uniform segmentation.
- 50% speed up for “medium sized” messages.

Theoretical Results (ratio versus best of “standard algorithms”)



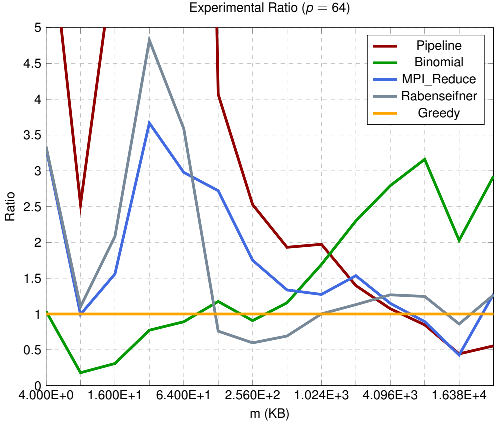
- Each algorithm is tuned for optimal uniform segmentation.
- 50% speed up for “medium sized” messages.

C code for the unidirectional greedy reduction algorithm.

```c
#include "reduce.h"
int global_s;
int Reduce_greedy( void *sendbuf_notype, void *recvbuf_notype, int m,
        MPI_Datatype mpi_datatype, MPI_Op mpi_op, int root, MPI_Comm mpi_comm){

        int pool_size;
        int my_rank;
        int j, i, q, z, s2, s3,qstart, qend, snext;
        int s=global_s;
        MPI_Status status;
        int *tempbuf, *tempbuf2;
        int *sendbuf, *recvbuf;
        int *hist;

        sendbuf = (int *) sendbuf_notype;
        recvbuf = (int *) recvbuf_notype;

        if (root != 0 ) MPI_Abort( mpi_comm, 512);

        MPI_Comm_size(mpi_comm, &pool_size); MPI_Comm_rank(mpi_comm, &my_rank);

        if(my_rank != 0 ) tempbuf = (int*)malloc(s*sizeof(int));
        tempbuf2 = (int*)malloc(s*sizeof(int));

        if ( my_rank == 0)
                for(i=0;i<m;i++) recvbuf[i] = sendbuf[i];
        else
                for(i=0;i<s;i++) tempbuf[i] = sendbuf[i];

        q = m/s;
        if( (m % s) != 0 ){ s2 = m % s; q++;}
        else s2 = s;

        hist = (int *)malloc(q*sizeof(int));
        for(i = 0; i < q; i++) hist[i] = pool_size;
        qstart = 0;
        qend = 0;
        while( hist[q-1] > 1 ){
                if (qstart != q-1 && hist[qstart+1]-hist[qstart] > 1){ qstart++;
                }
                for(i = qstart; i >= qend; i--){
                        z = (i == 0) ?  hist[0]/2 : (hist[i]-hist[i-1])/2;
                        s3 = ( i == q-1 ) ? s2 : s;
                        if( my_rank < hist[i] && my_rank >= hist[i] - z ){
                                MPI_Send(tempbuf, s3, mpi_datatype, my_rank-z, 512, mpi_comm );
                                if( i < q-1 ){
                                        snext = (i == q-2) ? s2 : s;
                                        for(j=0;j<snext;j++) tempbuf[j] = sendbuf[(i+1)*s + j];
                                }
                        }
                        if( my_rank < hist[i] - z && my_rank >= hist[i] - 2*z ){
                                MPI_Recv( tempbuf2, s3, mpi_datatype, my_rank+z, 512, mpi_comm, &status );
                                if ( my_rank == 0 ){
                                        for( j = 0;j < s3; j++ ){ recvbuf[i*s + j] += tempbuf2[j];
                                        }
                                }
                                else
                                        for( j = 0; j < s3; j++) tempbuf[j] += tempbuf2[j];
                        }

                        hist[i] -= z;
                        if (hist[i] == 1){
                                 hist[i] = 0; qend++;
                        }
                }
        }
        free( hist ); if(my_rank != 0) free( tempbuf ); free( tempbuf2 );
        return 0;
}
```

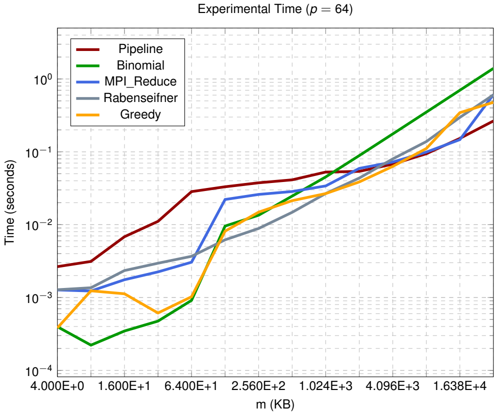
- `global_s` is the size of a segment and needs to be initialized (if possible "tuned") in advance.
- The implementation is restricted to `root` being `0`, `MPI_Datatype` being `int`, and `MPI_Op` being `+`. These restrictions are not a consequence of the algorithm and can be removed.

Experimental Results (ratio)



- Implemented with OpenMPI v1.4.3 point-to-point functions MPI_Send and MPI_Recv.

Experimental Results (time)



- Implemented with OpenMPI v1.4.3 point-to-point functions MPI_Send and MPI_Recv.

Table of Contents

Nonuniform Segmentation

(Note: our algorithm is optimal no matter the segmentation, uniform or not.)
Why does segmentation have to be uniform?

- Experiment: Fix the message size to $m = 10$ and check all possible segmentations.

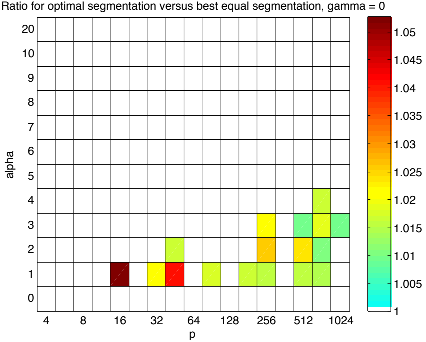
- Results for greedy
  - 61 of the 986 total trials where optimized by a nonuniform segmentation.
  - The maximum improvement of nonuniform versus uniform segmentation was 7.3%.
  - Of the 61 trials optimized by a nonuniform segmentation the average improvement was 2%.

- For pipeline, all trials where optimized by uniform segmentation.

Nonuniform Segmentation

Sample Segmentations:

Parameters	Percent	Best Uniform	Optimal
$p = 12, \alpha = 0, \beta = 1, \gamma = 1$	7.3%	(1,1,1,1,1,1,1,1,1,1)	(2,2,1,1,1,1,1,1)
$p = 48, \alpha = 1, \beta = 1, \gamma = 1$	2.6%	(2,2,2,2,2)	(3,2,2,2,1) (2,2,1,2,2,1)
$p = 256, \alpha = 5, \beta = 1, \gamma = 1$	3.0%	(4,4,2)	(5,3,2)



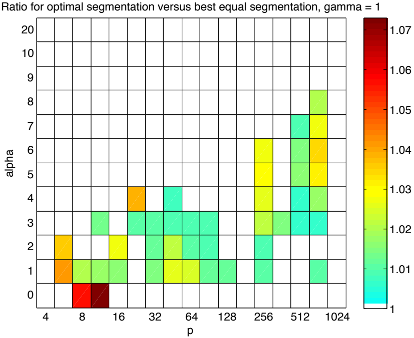
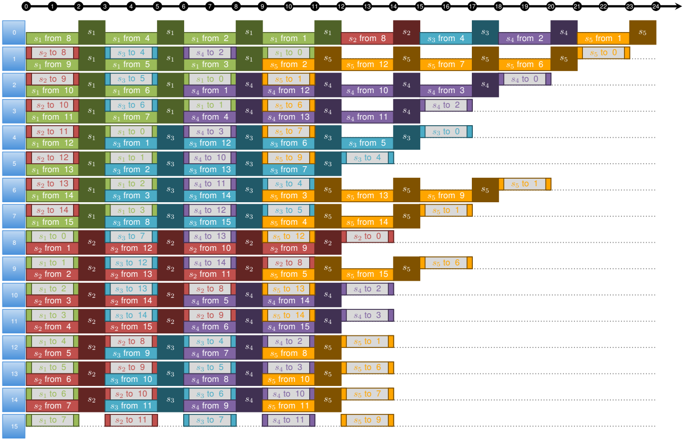


Table of Contents

Bidirectional

- Adapt the unidirectional greedy algorithm to the bidirectional context.
- Reduction and broadcast are essentially the same in bidirectional context. (All processors are performing computation.)
- Optimal broadcast algorithm [Traff and Ripke '08, Bar-Noy and Kipnis '94] scheduled in reverse has time $(\lceil \log_2 p \rceil + q - 1)(t_{comm} + t_{comp})$.
- Bidirectional greedy has same time complexity (conjecture).
- Rabenseifner ['04] provides algorithm for optimal computation.

Bidirectional Greedy Tree with computation



16 processors, 5 equi-segments, $t_{comm} = 2$ and $t_{comp} = 1$.

Solid rectangles represent receiving processors and rectangles with end strips represent sending processors. The darker rectangles represent computation.

Pseudo-code for bidirectional greedy algorithm

Algorithm 2: Bidirectional Greedy Algorithm

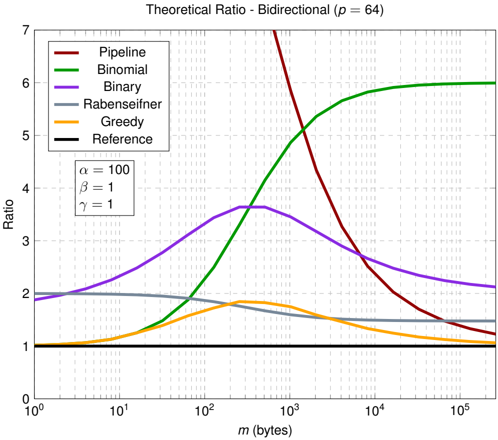
```
S = zeros(p, 1);
R = zeros(p, 1);
C = zeros(p, 2);
t = 0;
M = zeros(p, q);
while min{M(i, j) | 1 ≤ i ≤ p and 1 ≤ j ≤ q} = 0 do
    segStart = min{j | M(1, j) = 0};
    stop = 1;
    j = segStart − 1;
    while stop do
        j ← j + 1;
        COMP = {i | C(i, 1) ≤ t < C(i, 2) or t + t_comm(j) > C(i)};
        I = {i | M(i, j) = 0};
        sendProc = I \ COMP \ {i | S(i) > t};
        recvProc = I \ COMP \ {i | R(i) > t};
        freeProc = sendProc ⋂ recvProc;
        sendProc ← sendProc \ freeProc;
        recvProc ← recvProc \ freeProc;
        s = |sendProc|;
        r = |recvProc|;
        f = |freeProc|;
        if s = r then
            y = ⌊f/2⌋;
            sendProc ← sendProc ⋃ {freeProc(i) | f − y + 1 ≤ i ≤ f};
            recvProc ← recvProc ⋃ {freeProc(i) | 1 ≤ i ≤ y};
        else if s < r then
            y = r − s;
            m = min(f, y);
            x = ⌊(f − m)/2⌋;
            if m > 0 then
                sendProc ← sendProc ⋃ {freeProc(i) | (f − (m + x) + 1 ≤ i ≤ f};
            if x > 0 then
                recvProc ← recvProc ⋃ {freeProc(i) | 1 ≤ i ≤ x};
        else if r < s then
            y = s − r;
            m = min(f, y);
            x = ⌊(f − m)/2⌋;
            if m > 0 then
                recvProc ← recvProc ⋃ {freeProc(i) | 1 ≤ i ≤ m + x};
            if x > 0 then
                sendProc ← sendProc ⋃ {freeProc(i) | f − x + 1 ≤ i ≤ f};
        l = min(|sendProc|, |recvProc)
        if l = 0 then
            sendProc ← ∅;
            recvProc ← ∅;
        else
            sendProc = {sendProc(i) | 1 ≤ i ≤ l};
            recvProc = {recvProc(i) | 1 ≤ i ≤ l};
        M(i, j) = t + t_comm(j), ∀ i s.t. i ∈ sendProc;
        S(i) = t + t_comm(j), ∀ i s.t. i ∈ sendProc;
        R(i) = t + t_comm(j) + t_comp(j), ∀ i s.t. i ∈ recvProc;
        C(i, 1) = t + t_comm(j), ∀ i s.t. i ∈ recvProc;
        C(i, 2) = t + t_comm(j) + t_comp(j), ∀ i s.t. i ∈ recvProc;
        if |{i | M(i, j) = 0}| = 1 then
            M(1, j) = max({M(i, j) | 1 ≤ ilep}) + t_comp(j);
        if |{i | S(i) ≤ t}| + |{i | R(i) ≤ t}| < 2 then
            stop = 0;
        else if j ≥ q then
            stop = 0;
    t = t + 1;
```

Lower bounds for each term in communication time

	Latency	Bandwidth	Computation
Reduce	$\lceil log_2 p \rceil \alpha$	$m\beta$	$\frac{p-1}{p} m\gamma$
Binomial	$\lceil \log_2 p \rceil \alpha$	$\lceil \log_2 p \rceil m\beta$	$\lceil \log_2 p \rceil m\gamma$
Pipeline	$(p-1)\alpha$	$(p-3+m)\beta$	$(p-3+m)\gamma$
Binary	$2(N-1)\alpha$	$(N-3+2m)\beta$	$(N-3+2m)\gamma$

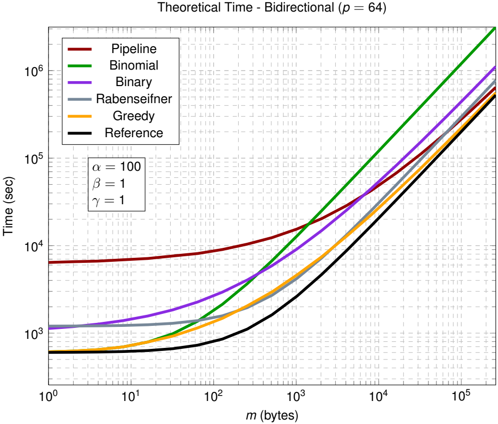
$^*N = \lceil \log_2(p+1) \rceil$

Theoretical Results (ratio)



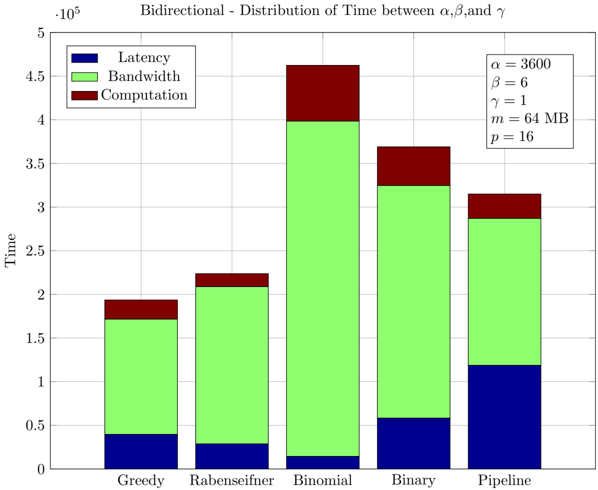
Reference Line: $\lceil \log_2 p \rceil \alpha + m\beta + \frac{p-1}{p}\gamma$

Theoretical Results (time)

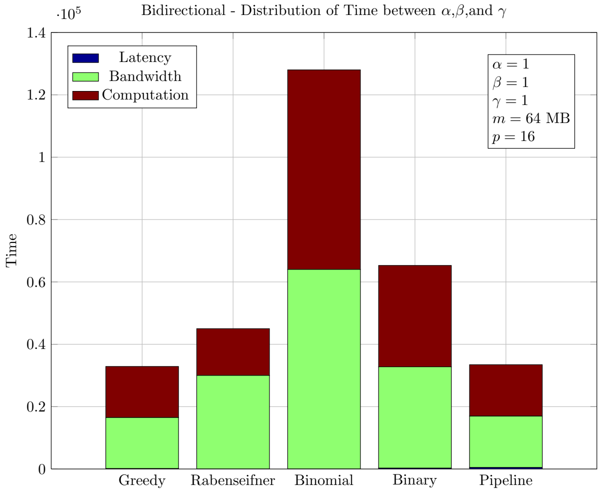


Reference Line: $\lceil \log_2 p \rceil \alpha + m\beta + \frac{p-1}{p}\gamma$

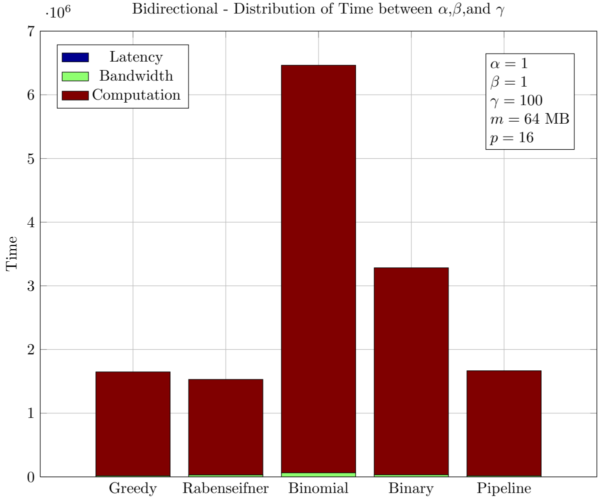
Distribution of time between α, β, γ



Distribution of time between α, β, γ



Distribution of time between α, β, γ



Conclusion

Unidirectional:

- Compared the greedy reduction with three standard algorithms.
- Greedy was the best theoretically.
- Most improvement is for medium sized messages (1Kb - 1Mb).
- Nonuniform segmentation is considered.
- Greedy is optimized by nonuniform segmentation for some machine parameters.

Bidirectional:

- Adapt unidirectional greedy algorithm for a bidirectional system
- Same time complexity of optimal broadcast algorithm.
- Implementation coming soon...

Questions?