A simple, pipelined all-gather algorithm for large irregular problems

Jesper Larsson Träff, Andreas Ripke, Christian Siebert, Pavan Balaji, Rajeev Thakur, William Gropp
NEC Labs Europe, ANL, UIUC
An irregular all-gather data-exchange operation

- A set of processes, 0, ..., p-1
- Each has a block of data $B_i$ of (possibly) different size

All processes have all blocks $B_0, B_1, ..., B_{(p-1)}$

```
MPI_Allgatherv(sendbuf,...,recvbuf,...,counts,...);
```

counts[i] ≈ |$B_i$|, all processes know the size of all blocks!
MPI_Allgatherv used in numerical libraries eg. PETSc

- Other irregular collectives, eg. MPI_Alltoallw

Irregular collectives algorithmically (much!) more difficult - challenging - than regular collectives:

1. Different amounts of data between processes (in different rounds)
   - Load imbalance

2. Partial information for the processes (MPI_Alltoallw, MPI_Gatherv, ...)
   - Difficult/expensive to compute schedule
   - Optimality is (NP)-hard!
Related work:

• Bruck et al. ’97: simultaneous binomial tree algorithm:

• Benson et al. ’03: MPI implementation of allgather algorithms on switched networks [non SMP-aware]

• Thakur et al. ’04: mpich2 implementations of Bruck and other [non SMP-aware]

• Mamidala et al. ’06: SMP implementations

• Träff’06: Graceful degradation from Bruck to linear ring [SMP-aware]

MPI_Allgather

MPI_Allgatherv

• Balaji et al ’07: optimization in the context of PETSc
An algorithm for large, regular all-gather problems

Linear ring: p-1 rounds

- Process i receives block B(i-1-k) and sends block B(i-k) in round k, k=0, ..., p-1
Analysis:

Size per process $m_i = |B_i| = m'$ (for regular problem)

Total problem size $m = \sum m_i$

- $p-1$ (regular) communication rounds
- Each round takes time $O(mi)$, total $O((p-1)mi) = O(m-m')$

Assumptions:
- Processes can send and receive simultaneously
- Cost of sending/receiving data of size $m'$ is $O(m')$
- Homogeneous communication along ring

Optimal: no idle time, no superfluous data!

NOTE: For small $m'$ algorithm with log $p$ rounds preferable!
Linear ring: p-1 rounds

- Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, ..., p-1
Linear ring: p-1 rounds

- Process i receives block $B(i-1-k)$ from $i-1$ and sends block $B(i-k)$ to $i+1$ in round $k$, $k=0, ..., p-1$
Linear ring: p-1 rounds

- Process i receives block $B(i-1-k)$ from i-1 and sends block $B(i-k)$ to i+1 in round $k$, $k = 0, \ldots, p-1$
Linear ring: p-1 rounds
- Process $i$ receives block $B(i-1-k)$ from $i-1$ and sends block $B(i-k)$ to $i+1$ in round $k$, $k=0, \ldots, p-1$

BUT:
- Time is $O((p-1)\max |B_i|)$ for last process to finish
Observation 1: linear ring works for clustered systems also

Linear ring: p-1 rounds

• Process i receives block B(i-1-k) from i-1 and sends block B(i-k) to i+1 in round k, k=0, ..., p-1

**MODIFICATION**: Use virtual ranking, one process per node sends, one process per node receives
Analysis:

• p-1 rounds
• Inter-node connections busy in all rounds: one process per node sends, one process per node receives
• Each node sends and receives (p-1) blocks

IMPROVEMENT:

a node never receives a block that it already has (replace by intra-node all-gather).
Observation 2:
linear ring on cluster solves irregular problem over nodes

Irregular problem can be solved by simulating clustered algorithm
Handle each block $B_i$ as node of $\text{ceil}(B_i/B)$ regular blocks of some size $B$. 

\[ B(i-1) \quad B \quad B(i+1) \]

process $i-1$ process $i$ process $i+1$
The algorithm for large, irregular all-gather problems

The blocked/pipelined ring algorithm
The blocked ring algorithm:

1. For each process $i$, cut data $B_i$ into $b_i = \max(1, \lceil \frac{m_i}{B} \rceil)$ blocks $B_j'$ of some chosen size (at most) $B$
1. For each process $i$, cut data $B_i$ into $b_i = \max(1, \text{ceil}(m_i/B))$ blocks $B'_j$ of some chosen size (at most) $B$ - each process has at least one block.

Total number of blocks $b = \sum b_i$
The blocked ring algorithm:

\[ bi = \text{ceil}(mi/B) \]

2. Run linear pipe on blocks, in round \( k \) process \( i \) receives block \( B'(si-1-k) \) from \( i-1 \) and sends block \( B'(si+bi-1-k) \) to \( i+1 \)

\[ si: \text{first block of process } i, \quad si = \sum_{j<i} bi \]
2. Run linear pipe on blocks, in round \( k \) process \( i \) receives block \( B'(si-1-k) \) from \( i-1 \) and sends block \( B'(si+bi-1-k) \) to \( i+1 \)

\( si: \) first block of process \( i \),

\( si = \sum_{j<i} bi \)

\( bi = \text{ceil}(mi/B) \)
The blocked ring algorithm:

\[ s_i: \text{first block of process } i, \quad s_i = \sum_{j < i} b_i \]

\[ b_i = \text{ceil}(m_i / B) \]

2. Run linear pipe on blocks, in round k process i receives block \( B'(s_i-1-k) \) from i-1 and sends block \( B'(s_i+b_i-1-k) \) to i+1
The blocked ring algorithm:

\[ bi = \text{ceil}(mi/B) \]

\[
\begin{align*}
& \text{B'0} & \text{B'1} & \text{B'2} \\
& \text{B'3} & \text{B'4} & \text{B'5} & \text{B'6} & \text{B'9} \\
& & & & & \\
\end{align*}
\]

\[ si: \text{first block of process } i, \quad si = \sum_{j<i} bi \]

2. Run linear pipe on blocks, in round \( k \) process \( i \) receives block \( \text{B'(si-1-k)} \) from \( i-1 \) and sends block \( \text{B'(si+bi-1-k)} \) to \( i+1 \)
The blocked ring algorithm:

2. Run linear pipe on blocks, in round \( k \) process \( i \) receives block \( B'(s_i - 1 - k) \) from \( i-1 \) and sends block \( B'(s_i + b_i - 1 - k) \) to \( i+1 \)

\[ si: \text{first block of process } i, \quad si = \sum_{j<i} b_i \]

\[ bi = \text{ceil}(m_i / B) \]
The blocked ring algorithm:

\[ bi = \text{ceil}(mi/B) \]

\[ \text{si: first block of process } i, \quad \text{si} = \sum_{j<i} bi \]

2. Run linear pipe on blocks, in round \( k \) process \( i \) receives block \( B'(si-1-k) \) from \( i-1 \) and sends block \( B'(si+bi-1-k) \) to \( i+1 \)
The blocked ring algorithm:

At most one block per process may not be full (may be empty)

MODIFICATION:

- Empty blocks are neither sent nor received
- Only actual data of partial blocks is sent/received
- No process receives a block it already has
Analysis:

- \( b-1 \) (almost) regular communication rounds
- Each round takes time \( O(B) \), total time \( O((b-1)B) = O((p+\sum \text{floor}(Bi/B))B) \approx O(m) \)

MUCH BETTER than \( O((p-1)\text{max}(mi)) \) of linear ring without blocking
General principle:

1. Regular collective operation solves similar irregular problem on clustered system

2. By simulation, algorithm for regular problem on clustered system can be converted to algorithm for irregular problem. **ASSUMPTION**: communication capability of node and processor similar, eg. one ported

3. Irregular operation (on processes) remains irregular problem on clustered system

- Blocked ring algorithm also works for MPI_Allgatherv on clustered system
Choosing the block size $B$: 

1. **All $m_i>0$:** choose $B=\min(m_i)$ - smallest data of some process (not too small, threshold).

2. **Fixed block size**

3. **Some $m_i=0$:** number of rounds is $m/B+(p-z)/2$, $z$ number of $m_i=0$, assuming partial blocks half full. Time per round in linear model is $a+\beta B$, best block size $\approx \sqrt{2am/\beta(p+z-2)}$

1. All processes busy in all rounds, for regular problems algorithm identical to linear ring

2. Simple solution - can lead to load imbalance for some distributions

3. Optimizes pipelining effect, for extreme problems with $m_0=m$, $m_i=0$ similar to pipelined broadcast. Linear model not accurate enough!
Experimental results

Comparison of blocked ring algorithm to standard ring:

- Performance
- Load balance
- Effect of block size B

Target systems

- NEC SX-8 - up to 30 nodes used
- Linux clusters with IB and Gig. Ethernet - 16 nodes, 24 nodes
- IBM Blue Gene/P - up to 4096 processes
- SiCortex 5832 - 5784 processes
Distributions

1. Regular
2. Broadcast
3. Spike
4. Half
5. Linear
6. Geometric

2, 3: same total amount of data
\( m = c, \) \( c \) is Base Size

1, 4, 5, 6: same total amount of data
\( m = pc, \) \( c \) is Base Size

Comparable running times
SX-8, 30x1 processes

MPI_Allgatherv (Bcast)

- Pipelined ring
- Standard ring

Time (microseconds)

Base Data Size (Bytes)

Fixed B = 1MByte

> factor 10

only ~15% slower than MPI_Bcast
SX-8, 30x8 processes

MPI_Allgatherv (Bcast)

Fixed $B = 1$MByte
SX-8, 30x1 processes

MPI_Allgatherv (Spike)

Fixed $B = 1$MByte
SX-8, 30x8 processes

MPI_Allgatherv (Spike)

Fixed B = 1MByte
SX-8, 30x1 processes

MPI_Allgatherv (Regular)

Fixed \( B = 1 \text{MByte} \)
SX-8, 30x1 processes

MPI_Allgatherv (Half)

Fixed $B = 1M$Byte

Time (microseconds)

Base Data Size (Bytes)

September 8-10, 2008 EuroPVM/MPI 2008, Dublin © NEC Laboratories Europe
SX-8, 30x1 processes

MPI_Allgatherv (Decreasing)

Fixed $B = 1$MByte
SX-8, 30x1 processes

MPI_Allgatherv (Geometric curve)

Fixed $B = 1$MByte

September 8-10, 2008

EuroPVM/MPI 2008, Dublin

© NEC Laboratories Europe
Linux cluster, 96 processes

Linear ring

Blocked ring

Yellow: idle time
Blue: communication time
Linux cluster, 96 processes

Fixed $B = 32$KByte
Linux cluster, varying number of processes

Fixed $B = 32$KByte
Blue Gene/P, 4096 processes

Fixed $B = 64\text{KByte}$
SiCortex, 5784 processes

Fixed $B = 1$MByte
Linux cluster, 16x2 procs

MPI_Allgatherv (Decreasing)

Time (Seconds)

Base Data Size (Bytes)

32k
64k
128k
512k
1024k
non-pipelined

Factor 2

Too small B

Factor 2

Too small B

Factor 2

Factor 2

Factor 2

Factor 2

Factor 2

Factor 2

Factor 2
Linux cluster, 16x2 procs

MPI_Allgatherv (Geometric curve)

Time (Seconds)

Base Data Size (Bytes)

Little effect of block size B
Linux cluster, 16x2 procs

MPI_Allgatherv (Regular)

```
Time (Seconds)
```

```
Base Data Size (Bytes)
```

- 32k
- 64k
- 128k
- 512k
- 1024k
- non-pipelined

September 8-10, 2008
EuroPVM/MPI 2008, Dublin
© NEC Laboratories Europe
Linux cluster, 16x2 procs

MPI_Allgatherv (Bcast)

Time (Seconds)

Base Data Size (Bytes)

Little effect of block size B

p

non-pipelined
Linux cluster, 16x2 procs

MPI_Allgatherv (Spike)

Time (Seconds)

Base Data Size (Bytes)

Too large B
Little/some effect of block size B

p

32k
64k
128k
512k
1024k
non-pipelined
Summary

• Simple, blocked linear ring algorithm for MPI_Allgatherv
  • NEW? Observation not found in literature
• Large performance gains for large problems on different systems
• Good limit behavior: identical to linear ring for regular problems, similar to pipelined broadcast for extreme distributions
• Tuning of block size: dependent on data distribution, linear model inadequate, experimental work needed
• There are relationships between regular and irregular collectives (on processes and nodes) that can (sometimes) be exploited for design of new algorithms