Optimization of MPI_Allreduce and MPI_Reduce

Rolf Rabenseifner, Panos Adamidis
{rabenseifner, adamidis}@hlrs.de
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

What do we want to do

Allreduce

Input buffers of 4 processes

V_{rank=0} \ V_{rank=1} \ V_{rank=2} \ V_{rank=3}
with element-wise operation *

Output buffers: Same result on all processes
Basic Principles

Principle I

• Different optimizations for latency and bandwidth
• Latency optimization, e.g.,
  – sending the full input buffers to all processors
  – executing the reduction on all processors
• Bandwidth optimization:
  – splitting the input buffers
  – transferring cross-wise between processes i.e., reduce_scatter
  – reduction operation only on partial buffers
  – allgather step at the end

Principle II

• In case where the number of processors is a power-of-two, then optimization is possible by buffer halving and distance doubling
• In case where the number of processors is non-power-of-two, various algorithms are shown.

Background

• 37% of MPI time in MPI_Allreduce
• 25% of user time with non-power-of-two number of processes
  – data from automatic profiling of all customers on HLRS CRAY T3E
Rabenseifner's Algo., Nov. 1997

- Standard algorithm used in mpich1:
  - MPI_Reduce = binomial tree
  - MPI_Allreduce = binomial tree + MPI_Bcast
  - Binomial tree is inefficient
    - half of the processes gets inactive ➔ bad load balancing

- Better algorithms (butterfly-algorithms):
  - MPI_Reduce = Reduce_scatter + Gather
  - MPI_Allreduce = Reduce_scatter + Allgather
Scheme with Rabenseifner’s Algo., Nov. 1997 (1st part)

Rank 1st part: Reduce_scatter ... (with halving the buffers)

new rank

<table>
<thead>
<tr>
<th>Rank</th>
<th>newrank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A–D&lt;sub&gt;0&lt;/sub&gt;1</td>
</tr>
<tr>
<td>1</td>
<td>E–H&lt;sub&gt;0&lt;/sub&gt;1</td>
</tr>
<tr>
<td>2</td>
<td>A–D&lt;sub&gt;2&lt;/sub&gt;3</td>
</tr>
<tr>
<td>3</td>
<td>E–H&lt;sub&gt;2&lt;/sub&gt;3</td>
</tr>
<tr>
<td>4</td>
<td>A–D&lt;sub&gt;4&lt;/sub&gt;5</td>
</tr>
<tr>
<td>5</td>
<td>E–H&lt;sub&gt;4&lt;/sub&gt;5</td>
</tr>
<tr>
<td>6</td>
<td>A–D&lt;sub&gt;6&lt;/sub&gt;7</td>
</tr>
<tr>
<td>7</td>
<td>E–H&lt;sub&gt;6&lt;/sub&gt;7</td>
</tr>
<tr>
<td>8</td>
<td>A–D&lt;sub&gt;8&lt;/sub&gt;9</td>
</tr>
<tr>
<td>9</td>
<td>E–H&lt;sub&gt;8&lt;/sub&gt;9</td>
</tr>
<tr>
<td>10</td>
<td>A–H&lt;sub&gt;10&lt;/sub&gt;</td>
</tr>
<tr>
<td>11</td>
<td>A–H&lt;sub&gt;11&lt;/sub&gt;</td>
</tr>
<tr>
<td>12</td>
<td>A–H&lt;sub&gt;12&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Always computing: \{ [(0+1)+(2+3)] + [(4+5)+(6+7)] \} + \{ [(8+9)+(10)] + [(11)+(12)] \}

exchanging half of data in both directions

A–H<sub>10</sub> = send buffer at rank 10 with logically 8 seg. (ABCDEFGH)

A–D<sub>0</sub>1 = computing reduction of data segments A–D with data from ranks 0-1

Scheme with Rabenseifner’s Algo., Nov. 1997 (2nd part)

Rank 2nd part: Allgather ... (with doubling the buffers) ...

new rank

<table>
<thead>
<tr>
<th>Rank</th>
<th>newrank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>1</td>
<td>E&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>2</td>
<td>C&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>3</td>
<td>G&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>4</td>
<td>B&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>5</td>
<td>F&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>6</td>
<td>D&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
<tr>
<td>7</td>
<td>H&lt;sub&gt;0&lt;/sub&gt;12</td>
</tr>
</tbody>
</table>

always computing: { [(0+1)] + [(2+3)] + [(4+5)+(6+7)] + [(8+9)+(10)] + [(11)+(12)] }

MPI_Allreduce & MPI_Reduce Optim.  Rolf Rabenseifner
Slide 7 / 20  Höchstleistungsrechenzentrum Stuttgart

© Rolf Rabenseifner: Collective Reduction Operation on Cray X1 and Other Platforms.
New Binary Blocks Halving+Doubling, July 2003 (1st part)

Rank 1st part: Reduce scatter ...
newrank

binary decomposition

Rank 2nd part: Allgather ...
newrank

blocks: binary decomposition

Always computing: \[ \{(0+1)+(2+3)\} + \{(4+5)+(6+7)\} + \{(8+9)+(10+11)\} + \{(12)\} \]

swapping necessary results

Rule: Each process in a lower block starts with two messages from the next higher block. The messages contain both halves.
**Compared Protocols**

- **Vendor** (MPI_Allreduce and MPI_Reduce of the used MPI library)
- **Binomial tree** + Bcast (i.e., without latency optimization)
- **Recursive doubling** with full buffers (i.e., with latency optimization)
- **Reduce_scatter + Allgather (or Gather)**
  - **Pairwise & Ring**
    - input buffer is divided into (#proc.) pieces of same size
    - optimal load balance but high latency
    - $O(2x \#\text{processes}) + O(2x \text{vector size})$
  - **Halving & Doubling**
    - $O(2x \lfloor \log(\#\text{processes}) \rfloor) + O(4x \text{vector size})$
  - **Binary Blocks Based Halving & Doubling**
    - normally better than halving & doubling
    - except for special #processes, e.g. 17, 33, 65,....

**Comparison: Fastest Protocol on T3E 900/512**

Benchmarks on T3E 900/512, sum of doubles, bandwidth := buffersize / wallclock time
Best algorithm more than x7 faster than vendor’s

Vendor's algorithm best for \(2^n\) processes & 64 – 1k byte

New algorithms 2-10x faster

Reduce: One of the 4 algorithms is always the best

Reduce(sum,db) = ratio = best bandwidth of 4 new algo.s / vendor’s bandwidth
**MAXLOC**: Vendor’s MPI\_DOUBLE\_INT is extreme slow

![Graph showing performance comparison between vendor's MPI\_DOUBLE\_INT and optimized binomial-tree algorithm.](chart)

**MAXLOC**: Comparing with optimized binomial-tree

![Graph showing performance comparison between vendor's MPI\_DOUBLE\_INT and optimized binomial-tree algorithm.](chart)
MPI_Allreduce on IBM SP at SDSC

- 3–5 x faster for larger buffers and more processes

Real Benefit

- depends on real usage-pattern
  - # calls
  - # bytes (bandwidth vs. latency)
  - # processes (power-of-two?)
  - operation (MPI_SUM, MPI_MAX, MPI_MAXLOC, user-defined, …)
  - Allreduce / Reduce

- Cray X1
  - code is developed on T3E
  - code is tested on CRAY Opteron clusters, Linux clusters, IA32/IA64, IBM
  - code has a problem on Cray X1, need more time for debugging
  - first result on X1:
    - MAXLOC problem with Cray-MPI: same as on T3E
Acknowledgments

- Thanks for helpful discussions
  - Rajeev Thakur (Argonne)
  - Jesper Larsson Träff (NEC)

- Thanks for help with benchmarking
  - Gerhard Wellein (Uni. Erlangen)
  - Thomas Ludwig, Ana Kovatcheva (Uni. Heidelberg)
  - Rajeev Thakur (Argonne)
  - Monika Wierse, Andy Mason (Cray)
  - Patrick H. Worley (ORNL)
  - Terry Hewitt, Mike Pettipher, Adrian Tate (Uni. Manchester)

Conclusion & Future Work

- Latency & Bandwidth optimization of MPI_Allreduce and MPI_Reduce is
  - possible
  - important
  - the ’97 algorithm is now part of mpich

- Future work:
  Integrated algorithm under construction
  - smooth optimization for any vector size
  - nearly optimal for any # processes
  - again significantly better bandwidth for non-power-of-two