Lightweight MPI Communicators with Applications to Perfectly Balanced Quicksort

Michael Axtmann, Peter Sanders, Armin Wiebigke

IPDPS · May 22, 2018
Overview

- Communicators and communication
- Disadvantages of communicator construction
- Solutions for MPI
- RBC communicators
- Case study on sorting
Communicators in MPI

MPI_COMM_WORLD

0  1  2
3  4  5

Subcommunicator
0 → 3; 1 → 4; 2 → 1

Blocking point-to-point

Send

Receive

0
1
2

Nonblocking point-to-point

ISend  Compute and Test

0
1
2

IRreceive and Test

Blocking collective

Scan

Scan

0
1
2

Nonblocking collective

IScan  Compute and Test

0
1
2

IScan  Test
**MPI Examples**

**Communication over rows and columns**

0 1 2
3 4 5
6 7 8

**Divide and conquer**

0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7

**Usage of communicators**

- Divide tasks into fine-grained subproblems
- Elegant algorithms and comfortable programming

Communicators make life easier at no cost!?
Current Implementations
OpenMPI and MPICH

PE group
- Mapping from PE ID to process ID required
- Explicit representation as table

Context ID
- Separates communication between communicators
- part of each message
- Unique for all PEs of the PE group
- Blocking Allgather-operation on context ID mask
Current Implementations
OpenMPI and MPICH

PE group
- Mapping from PE ID to process ID required
- Explicit representation as table

Communicator creation takes time linear to the communicator size

Context ID
- Separates communication between communicators
- Part of each message
- Unique for all PEs of the PE group
- Blocking Allgather-operation on context ID mask
Current Implementations
OpenMPI and MPICH

PE group
- Mapping from PE ID to process ID required
- Explicit representation as table

Communicator creation takes time linear to the communicator size

Context ID
- Separates communication between communicators
- part of each message
- Unique for all PEs of the PE group
- Blocking Allgather-operation on context ID mask

Communicator creation is a blocking collective operation
Blocking Communicator Creation

“...nonblocking collective operations can mitigate possible synchronizing effects...”

“...enabling communication-computation overlap...”

“...perform collective operations on overlapping communicators, which would lead to deadlocks with blocking operations.”

– MPI Standard

A collective operation is invoked by all PEs of a communicator

**BUT**: Communicator creation breaks nonblocking idea
Communicator Construction

Splitting a communicator into two communicators of half the size

Communicator construction time is linear to PE group size
Communicator Construction

Splitting $2^{15}$ PEs into two communicators of size $2^{14}$

Collective operation on $2^{14}$ cores

Communicator construction is expensive compared to collectives

SuperMUC – 32 768 cores – IBM MPI
Communicator Construction

Splitting a communicator into overlapping communicators of size four

PE group invokes
MPI_Comm_create_group

Alternating

Cascading

Blocking communicator creation causes delays

Comm Size (Cores)

Running time [ms]

SuperMUC – Intel MPI
Proposals for MPI

PE group
- Sparse representations
- E.g. MPI_Group_range_incl

Context ID
- User-defined tag
- Calculate by MPI: Concatenation of counters

Subcommunicator
first = 1, last = 4, stride = 2

Subcommunicator
Context ID 0
Subcommunicator
Context ID 1

MPI_COMM_WORLD
{0}  {1}  {2}  {3}
{0,0}  {0,1}  {2,0}  {2,1}  {2,2}
Our RBC library

Range-based communicator in $O(1)$ time

- Local construction
  - Select MPI or RBC operations
- Local splitting:
  \[
  \text{Split\_RBC\_Comm}(\text{Comm}\&, \text{Comm}\&, \\
  \text{int}\ first, \text{int}\ last, \text{int}\ stride)
  \]
- Only adjust range

<table>
<thead>
<tr>
<th>Blocking Ops</th>
<th>Nonblocking Ops</th>
<th>Classes</th>
<th>Local Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbc::Bcast</td>
<td>rbc::Ibcast</td>
<td>rbc::Request</td>
<td>rbc::Create_RBC_Comm</td>
</tr>
<tr>
<td>rbc::Reduce</td>
<td>rbc::Ireduce</td>
<td>rbc::Comm</td>
<td>rbc::Split_RBC_Comm</td>
</tr>
<tr>
<td>rbc::Allreduce</td>
<td>rbc::Iallreduce</td>
<td>rbc::Comm_rank</td>
<td></td>
</tr>
<tr>
<td>rbc::Scan</td>
<td>rbc::Iscan</td>
<td></td>
<td>rbc::Comm_size</td>
</tr>
<tr>
<td>rbc::Gather</td>
<td>rbc::Igather</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Gatherv</td>
<td>rbc::Igatherv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Barrier</td>
<td>rbc::Ibarrier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Send</td>
<td>rbc::Isend</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Recv</td>
<td>rbc::Irecv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Probe</td>
<td>rbc::Iprobe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Wait</td>
<td>rbc::Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rbc::Waitall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initial MPI communicator

\[
\begin{array}{cccc}
0 & 1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{array}
\]

first = 1, last = 7, stride = 3
Our RBC library
Implementation Details

(Non)blocking point-to-point communication
- Maps rank to rank of MPI communicator
- Call MPI counterpart

(Non)blocking collective operations
- Calls point-to-point operations of RBC
- One globally reserved tag
- Nonblocking details
  - Optional user-defined tag
  - Round-based schedule

```
rbc::Ibcast(void *buff, int cnt, MPI_Datatype datatype, int root,
            rbc::Comm comm, rbc::Request *request, int tag = RBC_IBCAST_TAG)
```

Interferences if...
- nonblocking collective operations with same tag and more than one shared PE
- nonblocking point-to-point from same PE with same tag
- or MPI_ANY_SOURCE is used

No interferences if...
- blocking point-to-point and no MPI_ANY_SOURCE
- nonblocking point-to-point from same PE over different communicators
- or nonblocking collective operations not overlapping in more than one PE
- or different tags and no MPI_ANY_SOURCE
RBC vs. MPI

Splitting a communicator into two communicators of half the size

RBC splitting comes with almost no cost

SuperMUC
**RBC vs. MPI**

Splitting a communicator with $2^{15}$ PEs into two communicators of size $2^{14}$ and performing a broadcast operation on both communicators

RBC splitting comes with almost no cost

SuperMUC – 32 768 cores – IBM MPI
Cost of Communicators

Splitting a communicator into overlapping communicators of size four

PE group invokes
MPI_Comm_create_group

Cascading

Alternating

Time

Comm Size (Cores)

Running time [ms]

MPI – Alternating
MPI – Cascade
RBC – Cascade
RBC – Alternating

Cascades do not have an effect on RBC

SuperMUC – Intel MPI
Quicksort on Distributed Systems

Single ported message passing
- Sending of \( l \) machine words: \( \alpha + \beta l \)
- Analyze critical path

Small inputs
- Minimal latency
- \( O(\alpha \log^2 p + \beta \frac{n}{p} \log p + \frac{n}{p} \log n) \)

Hypercube Quicksort
- Static communication pattern
- Precomputable communicators
- Bad for skewed inputs
- Only works for \( p = 2^k \)
Janus Sort

- \( O(\alpha \log^2 p + \beta \frac{n}{p} \log p + \frac{n}{p} \log n) \)
- Arbitrary \( p \)
- Calculate communicators on the fly
- Perfectly balanced data

\[ O(\alpha \log^2 p + \beta \frac{n}{p} \log p + \frac{n}{p} \log n) \]

Arbitrary \( p \)
Calculate communicators on the fly
Perfectly balanced data

Roman God Janus Twofaced

www.wikipedia.org

Image taken by Fubar Obfusco
Janus Sort

PE $i$  
Janus PE
PE $i + 2$
PE $i + 3$
PE $i + 4$

**Janus PE**

**execution**
Janus Sort

PE \( i \)  Janus PE  PE \( i + 2 \)  PE \( i + 3 \)  PE \( i + 4 \)

pivot selection  pivot selection

execution
Janus Sort

```
--- group j ---
PE i    Janus PE    PE i + 2    PE i + 3    PE i + 4

| pivot selection | partitioning     | pivot selection | partitioning     |
```

--- group j + 1 ---
Janus Sort

- group $j$
  - PE $i$
  - Janus PE
  - PE $i+2$
  - PE $i+3$
  - PE $i+4$

- pivot selection
- partitioning
- data exchange

- group $j+1$
Janus Sort

- **group $j$**
  - PE $i$
  - Janus PE
  - PE $i + 2$
  - PE $i + 3$
  - PE $i + 4$

  - pivot selection
  - partitioning
  - data exchange
  - comm creation

- **group $j + 1$**
  - PE $i$
  - Janus PE
  - PE $i + 2$
  - PE $i + 3$
  - PE $i + 4$

  - pivot selection
  - partitioning
  - data exchange
  - comm creation

- **base case**
  - Janus PE
  - PE $i + 2$
  - Janus PE

- **group $k$**
  - PE $i + 2$

- **group $k + 1$**
  - PE $i + 4$
Janus Sort

Pivot selection: $O(\alpha \log p)$

Regular PE

Janus PE

Left group

Right group
Janus Sort

Pivot selection: $O(\alpha \log p)$
Partitioning: $O(\log \frac{n}{p})$

Regular PE

Left group
Binary search

Janus PE

Right group
Binary search

Comm creation

Group $j$

PE $i$
Janus PE

PE $i + 2$

PE $i + 3$

PE $i + 4$

Group $j + 1$

Base case

Group $k$

Group $k + 1$

Data exchange

Pivot selection

Partitioning

Binary search
Janus Sort

Pivot selection: $O(\alpha \log p)$
Partitioning: $O(\log \frac{n}{p})$
Data exchange: $O(\alpha + \beta \frac{n}{p})$
Janus Sort

Pivot selection: $O(\alpha \log p)$
Partitioning: $O(\log \frac{n}{p})$
Data exchange: $O(\alpha + \beta \frac{n}{p})$
Comm creation: $O(1)$

Regular PE

Left group
Creation of even subcommunicator

New Janus PE
Create comm

Right group
Creation of odd subcommunicator
Janus Sort
Experimental Results

Sorting of double values
32 768 cores

Weak scaling on the SuperMUC
16 384 double values per core

IBM MPI
Conclusion

- Communicator creation is expensive
- Blocking communicator creation breaks idea of nonblocking communication
- Extension for MPI
- Range-based communicators
- Case study on sorting
- Code published at https://github.com/MichaelAxtmann/RBC