Memory patterns and communication performances for MPI libraries.

George Bosilca
bosilca@cs.utk.edu
Introduction

» Why?
  » Not everything is contiguous
  » Improving performances of moving scattered data
  » Toward a zero memory copy approach
  » One sided operations
  » Matching the behavior of the transport layer

» How?
  » Distributing knowledge between several layers
  » Modifying the behavior of several layers

» What?
  » Derived datatype engine
  » Smart transport layer
Layers in MPI libraries

- **Message management**
  - Datatype Engine
  - Myrinet
  - TCP
  - ShMem

- **Hardware Level**
- **MPI Level**
MPI_Datatype

» Convenient set of functions describing memory patterns
» Ranging from simple patterns
  » Contiguous
  » To slightly more complex
    » Indexed
    » To extremely complex
      » Structures
      » Upper bound, lower bound, extent, true extent, alignment
MPI_Datatype

» Complex ?

\[ \text{true lb} \quad \text{true ub} \]

\[ \text{true extent} \]
Low level communication device

» Basic and not smart
» Limited role: just moving contiguous bytes around
The heterogeneous case ...

- Heterogeneous = at least one data representation is different between the 2 architectures
  - Hardware design differences: double internal representation, long double
  - Compile time options: forcing integers as 8 bytes
- Worst possible solution
  - Converting to a intermediary format (XDR)
  - Both sides have to do a conversion
- Receiver side vs. Sender side
- Detecting the node properties (signature)
- Using MPI_PACKED communications
  - On pack add in the beginning of the buffer the id of the architecture
  - On unpack use the id to correctly handle conversions.
How to reach our goals?

» Toward a zero memory copy approach

» One sided operations
  » The data description should be compact
  » All information describing the datatype should be included
  » Special management for user defined boundaries

» Matching the behavior of the transport layer
  » Use pinned-down memory
  » Specific length for messages

» Packing/Unpacking by segments
New approach

MPI Level

Datatype Engine 1

Message management

Datatype Engine 2

Myrinet
TCP
ShMem

Hardware Level
Datatype description

» Add a internal datatype type: LOOP

flags  type  count  displacement  extent

» And then stack all the descriptions
Datatype optimizations

» Decreasing the number of memcpy/conversion functions that have to be called

» Detecting contiguous data

» Merging 2 data if the partial signature match

» Discover different patterns
Datatype example

10 * struct {
    5 * struct {
        int usefull[10];
        double useless;
    }
    5 * struct {
        int usefull[10];
        double useless;
    }
    int usefull;
    5 * struct {
        int usefull;
        float useless;
    }
}

Description
### Datatype example

```
10 * struct {
  5 * struct {
    int usefull[10];
    double useless;
  }
  5 * struct {
    int usefull[10];
    double useless;
  }
  int usefull;
  5 * struct {
    int usefull;
    float useless;
  }
}
```

### Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Datatype representation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Datatype example

10 * struct {
    5 * struct {
        int usefull[10];
        double useless;
    }
    5 * struct {
        int usefull[10];
        double useless;
    }
    int usefull;
    5 * struct {
        int usefull;
        float useless;
    }
}

Description  Datatype representation  Optimized representation

<table>
<thead>
<tr>
<th>LOOP</th>
<th>10</th>
<th>0</th>
<th>524</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>5</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>C</td>
<td>INT</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOOP</th>
<th>10</th>
<th>0</th>
<th>524</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>10</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>C</td>
<td>INT</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

| LOOP  | 5  | 0 | 48  |
| C     | INT| 10| 0   | 4   |

| LOOP  | 5  | 0 | 48  |
| C     | INT| 10| 0   | 4   |

| LOOP  | 5  | 52 | 8   |
| C     | INT| 1 | 52  | 4   |

| LOOP  | 4  | 60 | 8   |
| C     | INT| 1  | 60  | 4   |

<table>
<thead>
<tr>
<th>LOOP</th>
<th>10</th>
<th>0</th>
<th>524</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>10</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>C</td>
<td>INT</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Decoding a datatype

» New entity containing information about the datatype as well as the current position in the conversion process
» Know how to convert the data depending on the architecture of the sender and the receiver
» Behavior defined at creation time
  » What kind of data we have: contiguous or not, overlapped
  » What is the requirement for the driver: accepting any memory allocated buffers or require specific memory locations
» Respect all these constraints when converting the data
Decoding the datatype

» Avoiding recursive calls
» Using a stack based approach

Convert 40 bytes
Integration with the communication driver

- Prepare some initial data
- Driver callbacks when it need additional data
- Double buffering approach
  - The driver should always have as much data as he can send
    - TCP/IP with buffers of 128K should have at least 128K of data (in several iovecs) if possible.
- Allow the driver to prepare sending several data in same time (keeping filled the iovecs)
- Slightly different or receive as the data can be posted only after the header is matched.
Results

- Communication with self (used in some global communications)
Results

» Communication with self

Bandwidth for indexed datatypes

- new ddt optimized
- old ddt optimized
- new ddt without optimization
- old ddt
- mpich2 without optimizations
- mpich2 optimized

MB/s

Message size in Bytes

1000
100
10

1e+06
100000
10000
FT-MPI approach

- Unique communication driver
- Split the message in several packets
- Driver always accept any kind of memory
- Accepting data split in iovec’s
- Always use receiver side conversion (!)
Open MPI approach

» Communications over several supports (potentially with different behaviors)
» Stripping the message across several transport layers, not in several iovec’s
» The size of the segments is defined by an external entity, the datatype have to conform to the specified size.
» Decoding the same datatype with different algorithms as several drivers expect different kind of data.
Future

» Integrating message corruption detection at the datatype engine level
» Securing the communications by encrypting the data
» Fast as all these operations can be done in same type as the conversion and on the same segments of data
» Investigating the cache, TLB ... effect on the conversion