PSI Framework: A Dynamic Optimization of MPI

Hyacinthe NZIGOU M., Vivien Oddou and Feng Long GU

PETASCALE SYSTEM INTERCONNECT PROJECT

2008年 2月 19日
Presentation Flow

- **Background**
  - Collective Communications
  - Algorithms
- **Motivation and Goal**
- **Current System Limitation**
- **PSI Framework Flowchart**
- **Experimental Results**
  - MPI Allgather
  - MPI Alltoall
- **Conclusions and Future work**
# Background: Collective Communications

<table>
<thead>
<tr>
<th>Coll. Comm.</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast</td>
<td>Sequential, Chain, Binary, Binomial …</td>
</tr>
<tr>
<td>Scatter</td>
<td>Sequential, Chain, Binary …</td>
</tr>
<tr>
<td>Gather</td>
<td>Sequential, Chain, Binary …</td>
</tr>
<tr>
<td>Reduce</td>
<td>Gather followed by operation, Chain, Binary, Binomial and Rabenseifner …</td>
</tr>
<tr>
<td>Allreduce</td>
<td>Reduce followed by broadcast, Allgether by operation, Chain, Binary, Binomial and Rab…</td>
</tr>
<tr>
<td>Allgather</td>
<td>Gather + broadcast, Circular, Recursive db …</td>
</tr>
<tr>
<td>Barrier</td>
<td>Extended ring, Distributed binomial and Tournament …</td>
</tr>
<tr>
<td>Alltoall</td>
<td>Simple spread, Ring, Bruck, Pair-wise …</td>
</tr>
</tbody>
</table>
Alltoall algorithms (32 Procs)

- S. Spread
- Ring
- Bruck
- Recurs. Db
- Original

Message Size (Bytes):

- 1
- 2
- 4
- 8
- 16
- 32
- 64
- 128
- 256
- 512
- 1024
- 2048
- 4096
- 8192

Time [microseconds]
Background: Algorithm performances

Alltoall algorithms (32 Procs)

- S. Spread
- Ring
- Bruck
- Pair light ba.
- Original

Time [microseconds]
Message Size (Bytes)
Motivation and Objective

- Find techniques that dynamically select the best performing algorithm for a given situation
  - Network topology and parameters (Latency, Bandwidth etc.)
  - Message size
  - Load imbalance

- Develop collective communication routines that can adapt System/Application configuration.
Current system limitation

- In most static MPI implementations such as MPICH, Fujitsu MPI etc. adaptability is based on
  - Message size
  - Number of processors involved in the communication

- Difficult to achieve high performance of such MPI implementations for random behavior of the systems.
System Design

User Program

PSI Framework

MPI Library

MPI_Send
MPI_recv

Parallel machine
User application program

**Code.c**

```c
#include “mpi.h”
...
MPI_Alltoall(... args ...);
...
```

**Psi.c**

```c
#include “Psi_mpi.h”
...
PSI_MPI_Alltoall(... args...);
...
```

Diagram:
- 0 Byte
- 4 Kb
- 2 Mb
Start Program, Initializations

Alltoall(... args ...); Allgather(... args ...)

Learn: \( t_1 \leftarrow \text{Algo1} \times 5 ; t_2 \leftarrow \text{Algo2} \times 5; \ldots \)

Compare: \( t_{\min} = \min(t_1, t_2, t_3 \ldots) \)

\( t_i = t_{\min} \) ?

Select Alg \( i \)

Probing: Alltoall (... args ...) \( \leftarrow \) Algo \( i \)
Example of Learning and Probing

user program:

```c
main()
{
  ..
  init();
  ..

  for (i=0; i < 150; ++i)
  {
      ..
      MPI_Alltoall();
      ..
  }
}
```

0 : MPI_Alltoall() - bruck avg 2s
1 : MPI_Alltoall() - bruck
2 : MPI_Alltoall() - bruck
3 : MPI_Alltoall() - bruck
4 : MPI_Alltoall() - ring avg 1s
5 : MPI_Alltoall() - ring
6 : MPI_Alltoall() - ring
7 : MPI_Alltoall() - ring
8 : MPI_Alltoall() - s spread avg 0.5s
9 : MPI_Alltoall() - s spread
10 : MPI_Alltoall() - s spread
11 : MPI_Alltoall() - s spread
12 : MPI_Alltoall() - pairwise
13 : MPI_Alltoall() - pairwise avg 1s
14 : MPI_Alltoall() - pairwise
15 : MPI_Alltoall() - pairwise
16 : MPI_Alltoall() - s spread
17 : MPI_Alltoall() - s spread
18 : MPI_Alltoall() - s spread
19 : MPI_Alltoall() - s spread
20 : MPI_Alltoall() - s spread

........................
149 : MPI_Alltoall() - s spread
## Experimental system Configuration

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Riken PC Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>128</td>
</tr>
<tr>
<td>CPU</td>
<td>Inter Xeon 3.06GHz (2 CPU/node)</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB/node</td>
</tr>
<tr>
<td>O.S</td>
<td>RedHat Linux Enterprise</td>
</tr>
<tr>
<td>Compiler</td>
<td>Fujitsu C compiler 5.0</td>
</tr>
<tr>
<td>Interconnect</td>
<td>InfiniBand</td>
</tr>
<tr>
<td>MPI software</td>
<td>Fujitsu MPI</td>
</tr>
</tbody>
</table>
The framework always chooses the best performing algorithm that is in this case Ring algorithm.
Experimental Results (Load Balance)

Alltoall 32x1 Procs

Original vs. Framework

Message Sizes (Bytes)

Time (ms)

Alltoall 32x1 Procs

Message Sizes (Bytes)

Time (ms)
Experimental Results (Load imbalance)

Performance Evaluation

Message Sizes (Bytes)

Ratios

Rank order
Conclusions and Future work

- The experimental results of our framework show encouraging benefits
  - It can adapt to different platform systems
  - It always select the most suitable algorithm for different charge of the load
- However, for a given situation the slow algorithms in the learning phase degrade the overall performance
- Coding of the dynamic algorithm selection based on performance models
  - Algorithms grouping implementation
Conclusions and Future work (cont~)

Start Program, Initializations

Alltoall ( ... args ...)

\[ t_1 \leftarrow \text{Pring} = (P-1)(L + 2g(m)) \]

\[ t_2 \leftarrow \text{Algo 2} ; \ldots \]

\[ t_i \leftarrow \text{Algo i} \]

\[ t_i \leftarrow \min (t_1, t_2, t_3, \ldots) \]

\[ t_j \leftarrow t_i + 10\% \times t_i \]

\[ t_i \leq t_j ? \]

no

yes

Select Algo i

Learn: \[ t_i \leftarrow \text{Algo i} \times 5; \ t_j \leftarrow \text{Algo j} \times 5; \ldots \]
ノードの負荷アンバランスの影響

ご静聴ありがとうございました。

PETASCALE SYSTEM INTERCONNECT PROJECT