Runtime Algorithm Selection of Collective Communication with RMA-based Monitoring Mechanism

Takeshi Nanri (Kyushu Univ. and JST CREST, Japan)
Background

- Difficulties in static optimization of parallel codes because:
  - Larger number of nodes
  - More complexed network topologies
  - Load imbalances
  - Congestions among jobs
  - etc.

- Needs for efficient method of runtime optimization
Cycle of Runtime Optimization

- **Monitor**: Gather information about current status
- **Analyze**: Decide how to adjust the system
- **Apply**: Change the system according to the decision
Motivation of this work

• Examine efficiency of using RMA (Remote Memory Access) interface of MPI in the Monitoring Phase of runtime optimization

• Why RMA?
  • Asynchronous
  • Non-blocking

  Enable low-overhead monitoring on parallel systems

• Target in this work:
  Runtime algorithm selection of collective communication
Algorithms of Collective Communications

- Various candidates for each function:

- Different characteristics:
  - Number of steps
  - Possibility of pipelining
  - Robustness against load imbalances etc.

- No champion algorithm that is fastest in any situations
Traditional, Static Algorithm Selection

- Switch algorithm according to static thresholds
  - Message sizes and number of processes

- Cannot adapt to the different situations at runtime:
  - topological location, load balance, network traffic, etc.
STAR-MPI (A. Faraj, et al., 2006)

- A framework for runtime selection of collective communication algorithms

- Learning phase:
  - For each invocation, examine one candidate
  - All candidates are examined -> Choose the fastest

- Probing phase:
  - Monitor chosen algorithm
  - Detect change? -> Re-select

```c
... for (i=0; i < 10000; ++i) {
    ...
    Alltoall();
    ...
}
```

Learning

```
1st : Bruck
2nd : Bruck
...
11th : Ring
12th : Ring
...
100th : Ring
101st : Ring
...
```

Select

Probing
In this work

- Apply RMA-based monitoring to the Probing Phase of STAR-MPI
  - Instead of using Allreduce

- Use "Persistent Collective"-like interface
  - Instead of specifying "Call Site ID" argument in STAR-MPI
    - "Call Site ID": Extra argument to represent position of collective call in the program
      - Choose best algorithm for each invocation
Persistent Collective

- Currently discussed in the "Persistent WG" of MPI Forum
- ex)

```c
MPI_Allgatherv_init( ..., &request1);
MPI_Allgatherv_init( ..., &request2);

for (...) {
    ...
    MPI_Start(request1);
    ...
    MPI_Wait(request1);
    ...
    MPI_Start(request2);
    ...
    MPI_Wait(request2);
}
```

- Requests can represent the position of invocation in a program
Overview of Runtime Algorithm Selection

Init
- Prepare collective and create a request

Start
- Start clock
- Start collective(req)

Wait
- Complete collective(req)
- Stop clock
- if (Learning Phase)
  - record time for the algorithm
  - if (all algorithms are examined?)
    - choose the fastest
    - go to probing phase
- else
  - Monitor
  - Analyze
  - Apply

Probing Phase

Code snippet:
```c
MPI_Allgather_init( ..., &request1);
MPI_Allgather_init( ..., &request2);
for (...) {
    ...
    MPI_Start(request1);
    ...
    MPI_Wait(request1);
    ...
    MPI_Start(request2);
    ...
    MPI_Wait(request2);
}
```
Probing Phase with Allreduce (STAR-MPI)

- **Monitor**
  - Record time
  - if (N-th monitor)
    - $\text{AVE} = \frac{\text{total(Recorded Times)}}{N}$
    - **MPI_Allreduce (AVE)**

  Call MPI_Allreduce every \( N \) times of Probing Phase

- **Analyze + Apply**
  - if (AVE changed)
    - go back to Learning Phase
Probing Phase with RMA-based Monitoring

- **Monitor (all ranks)**
  - Record time
  - if (N-th monitor)
    - if (Change is determined)
      - Notify to Master

- **Analyze (at Master rank)**
  - if (Num. of notify exceeds limit)
    - Notify to all

- **Apply (all ranks)**
  - check notify from Master
  - if (Notify arrived)
    - go back to Learning Phase
Notify to Master with RMA

- A window is prepared and "lock-all"ed in Init function
  - Passive target
    - if (rank == Master)
      - MPI_Win_create(counter, ..., win)
    - else
      - MPI_Win_create(NULL, ..., win)
    - MPI_Win_lock_all(0, *win)

- Remote atomic operation to increment a counter in Master only when notification is required
  - if (N-th monitor)
    - if (Change is determined)
      - MPI_Fetch_and_op(..., ..., MPI_INT, Master, ..., MPI_SUM, win)
    - MPI_Flush(Master, win)
Notify from Master to All with Send + Probe

- Master sends notification with MPI_Isend
  - if ((N+1)-th monitor)
    - if ((rank == Master) && (counter > threshold))
      - FLAG = 1
      - for i = 0 to procs - 1
        MPI_Isend(FLAG, rank + i)

- Others check arrival of FLAG at (N+2)-th monitor
  - Depends on (N+2)-th collective to make sure that MPI_Isend(FLAG)s by Master have been completed already
  - if ((N+2)-th monitor)
    - if (rank != Master)
      - MPI_Iprobe(Master, &arrived)
      - if (arrived)
        MPI_Recv(FLAG)
    - if (FLAG) Go back to Learning Phase
Asynchronous Notification: RMA vs Send+Probe

- Notification with RMA (atomic, passive mode)
  - Latency may be higher than Send + Probe
  - Receiver does not have to perform any MPI function

Suitable for gathering notifications to Master (as far as the frequency of notification is low enough)

- Notification with Send+Probe
  - Receiver needs to call MPI_Iprobe for every possible senders
  - Latency of Send/Recv is lower for short messages than MPI_Put

Suitable for propagating notifications from Master (since there is only one possible sender per rank)
Experiments

• Examine overhead of monitoring
  • RMA vs Allreduce vs No Monitor

• Study effects of runtime optimization

• Experimental platform: PC Cluster (Fujitsu CX400)
  • Intel Xeon E5-2680 x 2, 128GB, RedHat 6.1
  • up to 512 nodes / 1476, one process / node
  • InfiniBand FDR, Mellanox MT4099
  • MVAPICH2-2.2rc1 + GCC 4.4.6

• Benchmark program: OSU Benchmarks 5.1
  • Modified "osu_iallgather.c":
    • Use "persistent collective"-like interface
    • Fixed amount of dummy computation
Average time of Comm + Dummy-Comp

- Alg1 ~ 3: each algorithm
- No Monitor:
- Allreduce 5, 20: perform allreduce every 5 or 20 times of monitoring
- RMA 5, 20: check changes every 5 or 20 times of monitoring

These are measured in stable situation. With dummy notification every 200 times in RMA5 and 20.

RMA-based Monitoring shows lower overheads than Allreduce-based
Ratio over "No Monitor"

Gap between Allreduce-based and RMA-based glows according to the number of nodes
Effect of Runtime Optimization

- Scenario:
  Change load-balance of computation "before" collective communication at 250th, 400th, 550th and 700th iteration of "osu_iallgather.c"

- Check if the framework can detect the change and re-select the best algorithm.
Results:
Sometimes, it worked well

At 700th step, best algorithm changed from Alg3 to Alg1.

After re-entering "Learning Phase", RMA5 and 20 could re-select the best one.
Sometimes, not.

Wrong detection of performance change caused worse performance than "No Monitor".
Conclusion

• Examined RMA-based monitoring in the framework of runtime algorithm selection of collectives.

• Confirmed reduction of overhead.

• Future works:
  • Refinement of runtime algorithm selection
    - Modify policies to avoid miss detection
  • Other collectives
  • Other runtime optimizations
  • Common framework for runtime optimization