Asynchronism in MPI Intra-Node Communications

Hyun-Wook Jin    Pu-Rum Seo
System Software Laboratory
Dept. of Computer Science and Engineering
Konkuk University
jinh@konkuk.ac.kr
Contents

• Background and motivation

• Asynchronism in MPI intra-node communications
  – Asynchronous nonblocking data copy
  – Asynchronous blocking progress engine

• Concluding remark
BACKGROUND & MOTIVATION
I/O Models

• **Synchronous blocking I/O**
  – Application blocks until the I/O system call is complete

• **Synchronous nonblocking I/O**
  – I/O command may not be satisfied immediately, requiring that the application makes calls to await completion

<table>
<thead>
<tr>
<th></th>
<th>Blocking</th>
<th>Nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong></td>
<td>Read/Write</td>
<td>Read/Write (O_NONBLOCK)</td>
</tr>
<tr>
<td></td>
<td>I/O Multiplexing (Select/Poll)</td>
<td>Asynchronous I/O (AIO)</td>
</tr>
</tbody>
</table>

* Source: IBM Developer
I/O Models

• **Asynchronous blocking I/O**
  – Application interrogates the readiness of multiple descriptors by using select/poll before I/O calls

• **Asynchronous nonblocking I/O**
  – Application can perform other processing while the background I/O operation completes

<table>
<thead>
<tr>
<th></th>
<th>Blocking</th>
<th>Nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Read/Write</td>
<td>Read/Write (O_NONBLOCK)</td>
</tr>
<tr>
<td></td>
<td>I/O Multiplexing (Select/Poll)</td>
<td>Asynchronous I/O (AIO)</td>
</tr>
</tbody>
</table>

* Source: IBM Developer
Data Copies in MPI

• **Shared memory channel**
  – Moves messages from source to destination via a shared memory region
  – Small messages based on eager protocol

• **Memory mapping channel**
  – Directly moves messages from source to destination without intermediate copies by means of a kernel level support
  – Large messages based on rendezvous protocol
  – CMA, LiMIC2, XPMEM, ...
CPU-based Data Movement

• Data copy operations in intra-node communications are performed by CPU
  – CPU resources are wasted for communication
  – CPU-based copying hinders overlapping of computation and communication

<table>
<thead>
<tr>
<th></th>
<th>Blocking</th>
<th>Nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronous</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU-based Data Movement</td>
<td></td>
</tr>
<tr>
<td><strong>Asynchronous</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Goal #1

• Introducing asynchronism to data copy
  – Overlapping between computation and communication
  – Copy engine (CE)-based data movement
    • Asynchronous nonblocking data copy
Event Processing in MPI

• **One-to-one mapping between processes and CPU cores**
  – In HPC systems, the runtime solely dedicates a CPU core to each parallel process
  – Parallel programming libraries are optimized on the assumption that a parallel process occupies an entire CPU core

• **MPI progress engine**
  – Performs busy-waiting to check the completion of outstanding communications

![Diagram showing MPI communication process with MPI_Send and MPI_Recv calls, busy-waiting, and message passing.]
Busy-Waiting-based Event Processing

- The longer the busy-waiting time, the higher the energy consumption
  - Nonuniformity of network latency
  - Asynchronous semantics in APIs
  - Load imbalance

<table>
<thead>
<tr>
<th></th>
<th>Blocking</th>
<th>Nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Busy-Waiting-based Event Processing</td>
<td></td>
</tr>
<tr>
<td>Asynchronous</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our Goal #2

- Introducing asynchronism to progress engine
  - Energy efficiency
  - Blocking-based event processing
    - Asynchronous blocking progress engine
Asynchronism in MPI Intra-Node Communications

ASYNCHRONOUS NONBLOCKING DATA COPY
Copy Engines

• A special-purpose processor that can independently access memory and copy data
  – Does not cause cache pollution compared to the CPU-based memory copy
  – Examples
    • Intel Xeon (I/O Acceleration Technology)
    • AMD EPYC

• We can offload copy operations performed by CPU onto the copy engine
  – Can save CPU resources
  – Can improve overlapping of computation and communication
Related Work

• Exploiting I/OAT
  – [IPDPS07, Cluster07, ICPP09]
  – Additional process/thread that takes full charge of managing the copy engine for intra-node data movements and monopolizes a CPU core

• No support for collective communication
  – Only for point-to-point communication or one-sided communication
Asynchronous Nonblocking Data Copy

• We aim at exploiting copy engines for intra-node MPI blocking collective communications
  – MPI_Bcast
  – MPI_Gather

• Asynchronous nonblocking data copy
  – CE-based approach
  – CE-CPU Hybrid approach
  – Enhancement of CPU-based approach
Synchronous Blocking Semantics

• **Traditional collective interfaces**
  – Do not return its control to user application until the collective communication is completed
  – Progress engine performs busy waiting to poll the completion or data copying to move data
  – No overlapping between computation and communication

• **Our collective interfaces**
  – Return asynchronously though the collective communication is not completed
  – Application can perform computation while the collective communication is in progress (by the copy engine)
  – Reserve synchronous blocking semantics by utilizing the memory protection mechanism (segmentation fault)
Core-to-Channel Mapping

- There can be multiple copy engines in the same node, and each copy engine provides several channels
  - Our experimental system
    - Two copy engines, each of which provides eight channels
    - The copy engine processes requests in channels in a round-robin fashion

- Core-to-channel mapping
  - In a round-robin manner for each NUMA node
Copy Engine (CE)-based Approach

- **Step 1**
  - Communication buffers are locked, and their descriptors (physical addresses of page frames and length) are sent to leaf processes

- **Step 2**
  - Leaf processes insert requests to channel

- **Step 3**
  - Copy engines move messages
CE-CPU Hybrid Approach

• **Hybrid approach**
  – Uses CPU to move data when lowering the overhead is more important than overlapping
  – Segmentation fault handler switches the copy device from copy engine to CPU

• **Virtual queues**
  • We can neither preempt nor cancel the DMA request already submitted to a channel
  • Provide a mechanism that switches from the CE mode to the CPU mode in the middle of data movements
    – A DMA request for a collective communication is fragmented into several requests, each of which include vectors for only $n$ pages
    – A callback function invoked whenever a fragmented request is completed moves fragmented requests in virtual queues to channels
Enhancement of CPU-based approach

• **Existing design**
  – Both memory mapping and copy operations are done on the receiver side

• **New design**
  – Segregates memory mapping and copy operations
  – The root process performs memory mapping, and the leaf processes perform data copy
Performance Measurements

• **Experiment system**
  – NUMA-based multi-core system
    • Two Intel Xeon 3.10 GHz 10-core Haswell processors
    • DDR4 128 GB memory
    • Crystal Beach DMA v3.2 copy engine
  – Linux kernel version 5.3.7
  – Intel QuickData Technology Driver 5.00

• **Comparisons**
  – Default approach (MVAPICH2 version 2.3.7)
  – Enhanced CPU-based approach (MVAPICH2 version 2.3)
  – CE-CPU hybrid approach (MVAPICH2 version 2.3)
OSU Micro-Benchmark

- **MPI_Bcast**
  - Enhanced CPU-based approach outperforms the existing CPU-based approach and reduces the latency of MPI_Bcast up to 67%
OSU Micro-Benchmark

- **MPI_Gather**
  - Enhanced CPU-based approach reduces the latency of MPI_Gather up to 85%
Overlapping with Computation

- **MPI_Bcast**
  - The enhanced CPU-based approach and the CE-CPU hybrid approach could reduce the overall execution time up to 45% and 58%, respectively
  - 20-process case with 4, 8, and 16 MB messages
Overlapping with Computation

• MPI_Gather
  – The enhanced CPU-based approach and the CE-CPU hybrid approach could reduce the execution time up to 63% and 65%, respectively
  – 20-process case with 4, 8, and 16 MB messages
Asynchronism in MPI Intra-Node Communications

ASYNCHRONOUS BLOCKING PROGRESS ENGINE
CPU Power Management States

• **Dynamic Voltage and Frequency Scaling (DVFS)**
  – Provides different levels of voltage and frequency for operating processors
  – P-states (ACPI)
    • P0: Maximum power and frequency
    • Pn: Less than P(n–1) voltage and frequency scaled

• **Core-Idling**
  – Turns off hardware components of idle cores
  – C-states (ACPI)
    • C0: Active
    • C1: Halt
    • C2: Stop-clock
    • C3: Sleep
Related Work

• Decision policies
  – EAM [SC’15]
    • Estimates the duration of MPI and communication phases based on temporal execution patterns
    • Interrupt-based core-idling
  – COUNTDOWN [ToC 2021]
    • Intercepts MPI calls and uses a time-out strategy for DVFS
    • Countdown Slack [TPDS 2020]
  – EAR/EARL [Cluster 2020]
    • Detects iterative regions and maintains application signatures by intercepting MPI calls
    • Decides the CPU frequency based on an energy model

• No support for core-idling on intra-node communication channels
Asynchronous Blocking Progress Engine

• We aim to provide a framework that efficiently supports core-idling over multiple MPI communication channels
  – Intra-node communication channels
    • Shared memory
    • Memory mapping

• Asynchronous Blocking Progress Engine
  – Framework for energy-efficient MPI
  – Asynchronous blocking intra-node communication
  – Integration with blocking inter-node communication
Framework for Energy Efficient MPI

• Interfaces
  – APIs
    • MPI_Energy_handler_reg()
      – int *(enter_function) (MPI_Energy_Info*)
    • MPI_Energy_handler_dereg()
  – Application can change the policy at runtime
  – Hooking of MPI calls
    • MPI_Init() and MPI_Finalize()
    • No application-level modification is required
Framework for Energy Efficient MPI

- **Internal runtime information**
  - Ranks
  - Communication channel
  - Message size
  - Number of busy-waiting iterations
  - Current busy-waiting time
  - Last busy-waiting time
  - ...
Signal-based Blocking Communication

- **CPU dependent implementation**
  - Assembly instructions (e.g., `mwait`)

- **CPU independent implementation**
  - Timers: only for coarse-grained controls
  - Semaphores: deadlock-prone
  - Signals: lossy
    - Easy to support callback functions
    - Flexible enough to support the inter-node communication channel
    - Able to leverage existing decision policies used in DVFS and core-idling approaches
Signaling Points

- **Shared memory channel**
  - When a shared buffer becomes available
  - When a new message is arrived
Signaling Points

- **Memory mapping channel**
  - When a control message of rendezvous protocol arrives

![Diagram of signaling points]

- **MPI_Send** (Blocked)
- **MPI_Recv** (Blocked)
- **MPI_Send**
- **MPI_Recv** (Blocked)
- **MPI_Send**
Integration with Blocking Inter-Node Communication

- **Blocking inter-node communication**
  - MV2_USE_BLOCKING
- **epoll-based integration**
  - File descriptors
    - Signal for intra-node communication
    - Completion channel for inter-node communication
Performance Measurements

• **Experiment system**
  – Two ARM-based multi-core systems
    • Ampere eMAG 8180 (ARMv8) 3 GHz 32-core processor
    • DDR4 250 GB memory
  – NVIDIA ConnectX-5 InfiniBand adapter
  – Linux kernel version 5.4.0-156-generic
  – Wattman HPM-100A power meter

• **Comparisons**
  – Default blocking mode
    • MVAPICH2 version 2.3.7 with MV2_USE_BLOCKING
  – Our energy efficient framework
    • MVAPICH2 version 2.3.1
OSU Micro-Benchmark

- **MPI_Alltoall**
  - Execution time: 43.4% reduction (1 MB)
  - Energy consumption: 41.8% saving (1 MB)
OSU Micro-Benchmarks

- **MPI_Allreduce**
  - Execution time: 28.1% reduction (128 KB)
  - Energy consumption: 28.9% saving (128 KB)
NAS Parallel Benchmarks

- **Class C**
  - Execution time: 16.5% reduction (CG)
  - Energy consumption: 14.9% saving (CG)
CONCLUDING REMARK
Conclusions

• **Asynchronous nonblocking data copy**
  – A scheme to exploit multiple copy engines and CPUs for intra-node MPI collective communications

• **Asynchronous blocking progress engine**
  – A framework for better supports for energy-aware decision policies over multiple MPI communication channels
Future Work

• **Asynchronous nonblocking data copy**
  – Other collective calls
    • Blocking and nonblocking collective communications
  – Integration with inter-node communication
  – Measurement with real applications

• **Asynchronous blocking progress engine**
  – Various policies
  – Measurement with real applications