ACPdl: Data-Structure and Global Memory Allocator Library over a Thin PGAS-Layer

Yuichiro Ajima, Takafumi Nose, Kazushige Saga, Naoyuki Shida, and Shinji Sumimoto
Fujitsu Limited / JST CREST

November 15th, 2015, ESPM2 2015 Workshop
Index

- Introduction
- Advanced Communication Primitives
- ACP Data Library
- Evaluation, Discussion and Summary
Index

- Introduction
- Advanced Communication Primitives
- ACP Data Library
- Evaluation, Discussion and Summary
Increasing Core Count per System

- The maximum core count per system will exceed 10 million by around 2020
- Many-core and heterogeneous processors will extend the trend
- The growth rate of median core count is slowing down
- Commodity clusters will keep using multi-core processors
Memory Technology Trends

- The diverging trends of memory technologies
  - High-throughput 3D stacked memories for “Exaflops” systems
e.g. HMC, HBM
  - Large-capacity vertical 3D memories for “Exabyte” systems
e.g. 3D NAND, 3D Xpoint
  - All-in-one machines will have a deeper memory hierarchy

- A memory issue with “Exaflops” systems
  - Each core will have a small memory capacity
  - Peer-to-peer communication contexts will consume large amount of memory
Remote Direct Memory Access (RDMA) is a type of protocol

- Accesses data directly by specifying its memory address
  - Get protocol transfers data from remote memory to local memory
  - Put protocol transfers data from local memory to remote memory

Pros

- Connection-less, as far as the reliable delivery is guaranteed
- Supported by almost all modern HPC interconnect devices

Cons

- Too primitive interfaces and the lack of device standards
**PGAS Programming Model**

- **Partitioned**: a local reference refers a local object
- **Global Address Space**: a global reference refers a remote object
  - Referring a global reference implicitly designates an RDMA transfer
  - PGAS languages are easy to write communication
Index

- Introduction
- Advanced Communication Primitives
- ACP Data Library
- Evaluation, Discussion and Summary
Advanced Communication Primitives (ACP)

- ACP is a new communication library to build PGAS middleware
  - Such as domain-specific languages and parallel computation frameworks
  - ACP is a library for developers, not for ordinary application programmers

- ACP does not include parallelism and parallel execution models
  - The upper-layer software may provide them

- Interfaces are designed for the memory-efficient programming
  - Each primitive consumes memory explicitly
  - Programmers can maintain the amount of memory consumption
Software Structure of ACP

- **ACP basic layer (ACPbl)**
  - A thin PGAS layer to abstract features of interconnect devices

- **ACP data library (ACPdl)**
  - Data structure library for processing irregular data

- **ACP communication library (ACPcl)**
  - Message passing library for processing regular data
  - Not included in this presentation

| ACPbl | ACPdl | ACPcl |
ACP Basic Layer

- Key features
  - 64-bit global address
  - Starter memory
  - Remote-to-remote data transfer

- Design philosophy
  - Features should be implementable in hardwired logic without micro-controllers
  - An implementation over existing devices may create a communication thread to emulate the features
ACPbl: 64-bit Global Address

- A 64-bit global address (GA) is assigned to registered memory
  - 64-bit value is easy to be handled by overlying software

- Memory registration and address translation interface
  - `acp_register_memory()` returns a translation key
  - `acp_query_ga()` translates a local address to a GA using a translation key
  - `acp_query_address()` translates a GA to a local address
ACPbl: Starter Memory

- A pre-registered memory region for each process
  - The size is specified by environment variables or command line options

- A ready-to-use shared resource
  - Allocated and zero-cleared in \texttt{acp_init()}
  - \texttt{acp_query_starter_ga()} returns GA of starter memory of an arbitrary process

- The intended usage is to place shared static variables
  - cf. data and bss segments in program memory
**ACPbl: Remote-to-Remote Data Transfer**

- `acp_copy()` performs a remote-to-remote data transfer
  - Copy protocol is a superset of Get and Put protocols
    - Equivalent Get and Put sequence may be redundant
  - Non-blocking function with a memory fence designation

---

**Copy**

```
P1  P2  ...  Pn
```

**Get and Put**

```
P1  P2  ...  Pn
```
ACPbl: Four Implementations

- ACPbl/UDP
  - A reference implementation with software-implemented reliable delivery

- ACPbl/InfiniBand (ACPbl/IB)
  - Using RDMA over reliable connection
  - Implemented distributed address translation tables and translation cache

- ACPbl/Tofu and ACPbl/Tofu2
  - Using connection-less RDMA with remote-side memory fence

- All implementations are thread safe
ACP Data Library

- ACPbl is too low level, even for developers
- ACPdl provides data structure interfaces for irregular data
- ACPdl includes the global memory allocator
  - To create and delete remote data asynchronously
- Interfaces are derived from familiar libraries
  - Memory allocator from C standard library: `acp_malloc()`, `acp_free()`
  - Data structure types from C++ STL: `vector`, `list`, `map`, etc.
- Algorithms have been revised from the ground up
Index

- Introduction
- Advanced Communication Primitives
- ACP Data Library
- Evaluation, Discussion and Summary
Initial Memory Management Algorithm

- This algorithm is similar to that in the Kernighan and Ritchie’s book
- Free blocks are linked and sorted by address
- The allocation algorithm has O(1) computational complexity of typical cases
**O(n) Complexity of the Deallocation**

- To avoid fragmentation, adjacent free blocks should be merged.
- Nearest free blocks for a block can be found from the free list.
- Searching a list has $O(n)$ computational complexity.
  - This may be an issue for memory efficient programming.
**O(n) Complexity of the Deallocation**

- To avoid fragmentation, adjacent free blocks should be merged.
- Nearest free blocks for a block can be found from the free list.
- Searching a list has $O(n)$ computational complexity.
  - This may be an issue for memory efficient programming.
Improved Memory Management Algorithm

- `acp_free()` and `acp_malloc()` cooperatively merge free blocks
- Merge in `acp_free()` is done without searching the free list
  - Each block indicates a linked-free/unlinked-free/allocated flag for merging
- The deallocation algorithm has been improved to O(1) complexity
Merging Free Blocks in `acp_free()`

- The freed block will be …
  - Merged if the next block is not allocated or the previous block is unlinked-free
  - Unlinked-free if the previous block is linked-free and the next block is allocated
- Otherwise, the freed block is linked into the head of the free list
Merging Free Blocks in `acp_free()`

- The freed block will be …
  - Merged if the next block is not allocated or the previous block is unlinked-free
  - Unlinked-free if the previous block is linked-free and the next block is allocated
- Otherwise, the freed block is linked into the head of the free list
Merging Free Blocks in `acp_malloc()`

- When a free block is tested for allocation, the next block is checked
  - If the next block is unlinked-free, the next block will be merged into the block
  - If the next block is linked-free, the block is removed from the list and merged into the next block, then the search for allocation is restarted from the head
When a free block is tested for allocation, the next block is checked:

- If the next block is unlinked-free, the next block will be merged into the block.
- If the next block is linked-free, the block is removed from the list and merged into the next block, then the search for allocation is restarted from the head.
Merging Free Blocks in `acp_malloc()`

- When a free block is tested for allocation, the next block is checked:
  - If the next block is unlinked-free, the next block will be merged into the block.
  - If the next block is linked-free, the block is removed from the list and merged into the next block, then the search for allocation is restarted from the head.
Vector Data Structure

- A resizable contiguous memory
- `acp_create_vector()` creates a vector on the specified process
- `acp_push_back_vector()` appends data to a vector

```
v = acp_create_vector( ... , 2);
acp_push_back_vector(v, ...);
```
List Data Structure

- A bi-directional linked list
- `acp_create_list()` creates an empty list on the specified process
- `acp_push_back_vector()` appends an element on the specified process to a list

```
l = acp_create_list(2);
acp_push_back_list(l, ..., 1);
acp_push_back_list(l, ..., n);
```
Map Data Structure

- A distributed associative array on the specified group of processes
- `acp_create_map()` creates a map on the specified process
- `acp_insert_map()` puts a key-value pair to a map
- `acp_find_map()` gets the value of a key

```c
m = acp_create_map(n, &ranks, ..., 2);  // Create map
acp_insert_map(m, ...);  // Insert key-value pair
```

```
P1  P2  ...  Pn

1  2  ...  n

```

Copyright 2015 FUJITSU LIMITED
Index

- Introduction
- Advanced Communication Primitives
- ACP Data Library
- Evaluation, Discussion and Summary
Evaluation

- **acp_copy()**
  - Remote-to-local, local-to-remote and remote-to-remote data transfer latencies

- **acp_malloc() and acp_free()**
  - Consecutive allocations of random, maximum 32KB, size of memory
  - And then consecutive out-of-order deallocations

- **acp_insert_map() and acp_find_map()**
  - Consecutive insertions of key-value pairs
  - And then consecutive searches of random key, 50% chance to hit
# Evaluation Environment

<table>
<thead>
<tr>
<th></th>
<th>ACPbl/UDP</th>
<th>ACPbl/IB</th>
<th>ACPbl/Tofu</th>
<th>ACPbl/Tofu2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node</strong></td>
<td>Fujitsu PRIMERGY RX200 S5</td>
<td>Fujitsu PRIMERGY RX200 S7</td>
<td>Fujitsu PRIMEHPC FX10</td>
<td>Fujitsu PRIMEHPC FX100</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Xeon E5520 4 cores 2.27 GHz</td>
<td>Xeon E5-2609 4 cores 2.4 GHz</td>
<td>SPARC64™ IXfx 16 cores 1.848 GHz</td>
<td>SPARC64™ XIfx 32+2 cores 1.975 GHz</td>
</tr>
<tr>
<td></td>
<td>2 sockets/node</td>
<td>2 sockets/node</td>
<td>1 socket/node</td>
<td>1 socket/node</td>
</tr>
<tr>
<td><strong>Memory/node</strong></td>
<td>DDR3 SDRAM 48GB 51.2 GB/s</td>
<td>DDR3L SDRAM 8GB 25.6 GB/s</td>
<td>DDR3 SDRAM 64GB 85 GB/s</td>
<td>HMC 32GB 480 GB/s</td>
</tr>
<tr>
<td><strong>Network/link</strong></td>
<td>Gigabit Ethernet 0.125 GB/s</td>
<td>InfiniBand QDR 4.0 GB/s</td>
<td>Tofu interconnect 5.0 GB/s</td>
<td>Tofu interconnect 2 12.5 GB/s</td>
</tr>
</tbody>
</table>
Results: `acp_copy()` remote-to-local

- IB, Tofu and Tofu2 are two orders of magnitude faster than UDP.
Results: \texttt{acp\_copy()} local-to-remote

- Results show the nearly same performance as remote-to-local data transfer.
Results: `acp_copy()` remote-to-remote

- Results show the almost same throughput as remote-to-local and local-to-remote.
- Latencies for small data deteriorated due to control communication.
Results: \texttt{acp\_malloc()} and \texttt{acp\_free()}

- The improved algorithm reduced the latency of \texttt{acp\_free()}
- Tofu and Tofu2 are only one order of magnitude faster than UDP
- IB is much worse
Results: \texttt{acp\_insert\_map()}, \texttt{acp\_find\_map()}

- Similar but two or three times longer results than \texttt{acp\_malloc}
Discussion

- It seems that the dependency control affected the results
  - For `acp_copy`, UDP was two orders of magnitude slower than the others
    Dependencies are controlled in the communication thread
  - For the ACPdl functions, UDP was only one order of magnitude slower
    Dependencies are controlled in the main thread
    Overhead between main and communication threads may deteriorate message rate

- It seems that translation cache misses affected ACPbl/IB’s results
  - Evaluations of the ACPdl functions accessed remote memory randomly

- These issues will be further investigated in our future work
Summary

- ACP is a new communication library to build PGAS middleware
  - Interfaces are designed for the memory-efficient programming
- ACP basic layer is a thin PGAS layer
  - 64-bit global address, Starter memory, Remote-to-remote data transfer
- ACP data library provides data structure interfaces
  - An improved algorithm for global memory allocator is implemented
- Four versions of the basic layer and data library are evaluated
  - It seems that communication thread overhead and translation cache miss deteriorate the performance of the data library