FUJITSU
shaping tomorrow with you

Reducing manipulation overhead of remote data structure by controlling remote memory access order

Yuichiro Ajima, Takafumi Nose, Kazushige Saga, Naoyuki Shida, Shinji Sumimoto

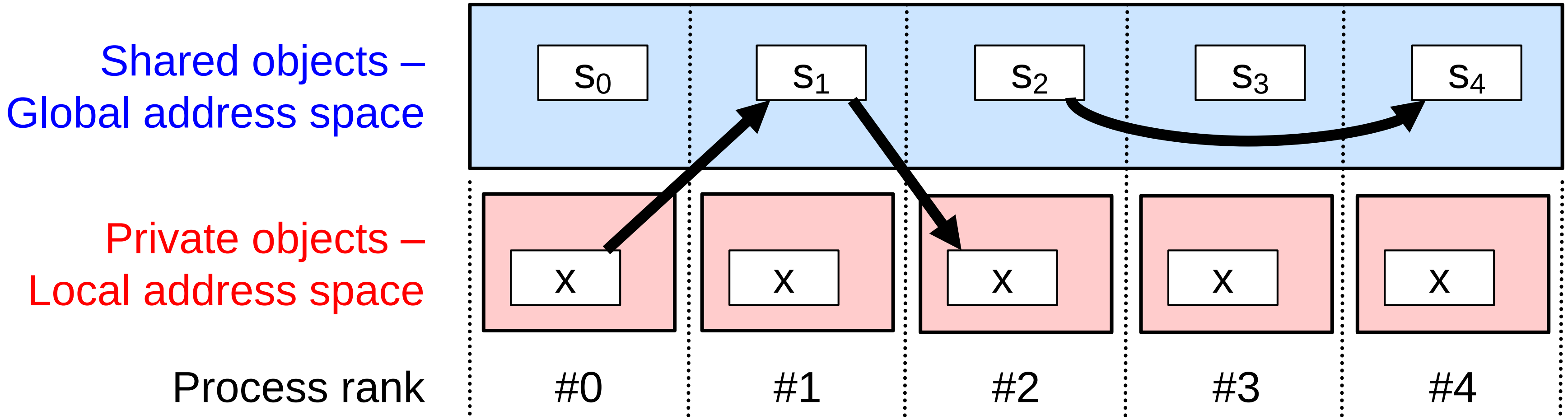
Fujitsu Limited / JST-CREST

Acknowledgement

The development of the ACP library is a part of the Advanced Communication for Exa (ACE) project, which is a research theme in the CREST research area ‘Development of System Software Technologies for post-Peta Scale High Performance Computing,’ sponsored by JST (Japan Science and Technology Agency).

PGAS Programming Model

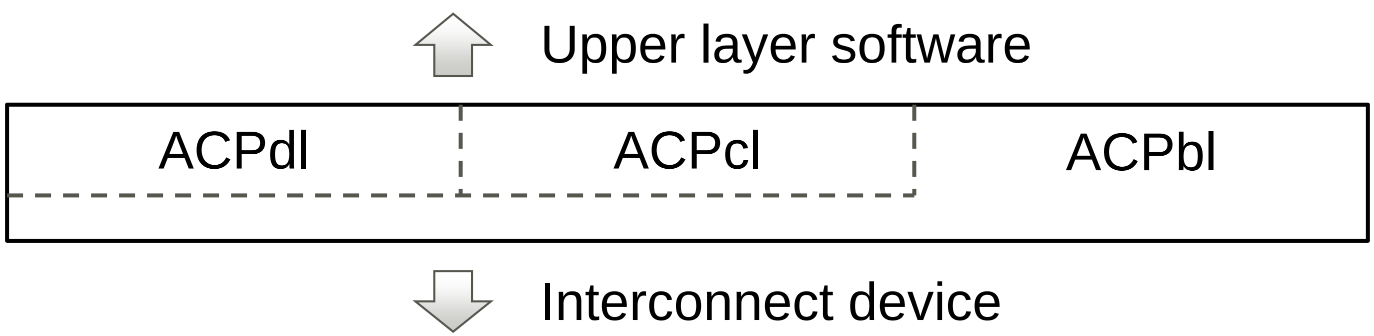
- Partitioned Global Address Space
  - A programming model for distributed memory machines
  - Global address space: shared objects are handled by a global address
  - Partitioned: each shared object is owned by a process
- Productive syntax of PGAS languages
  - Inter-process data transfer is written as an assignment statement
  - Private-to-shared, shared-to-private, and shared-to-shared



Advanced Communication Primitives (ACP)

FUJITSU

- A communication library Including a PGAS layer as a basis
  - Designed for memory-efficient programming
  - Each communication primitive consumes memory explicitly
  - Programmers can control the amount of memory consumption
- Interface categories and the software structure
  - ACPdl – data library: data structure interfaces for irregular data
  - ACPcl – communication library: message passing interfaces
  - ACPbl – basic layer: hardware abstraction with the PGAS model
    - Four implementations: UDP, InfiniBand, Tofu, Tofu2



Index

Global Memory Management

- Static global memory – statically allocated for each process
  - Global address of any process is available after the initialization
- Dynamic global memory – locally registered
  - Globally accessible, but the address translation must be done locally

```
#include <stdlib.h>
#include <acp.h>

int main(int argc, char** argv)
{
    acp_init(&argc, &argv);

    acp_ga_t my_bss = acp_query_starter_ga(acp_rank());

    void* buf = malloc(65536);
    acp_atkey_t key = acp_register_memory(buf, size, 0);
    acp_ga_t my_heap = acp_query_ga(key, buf);

    /* ... */
```

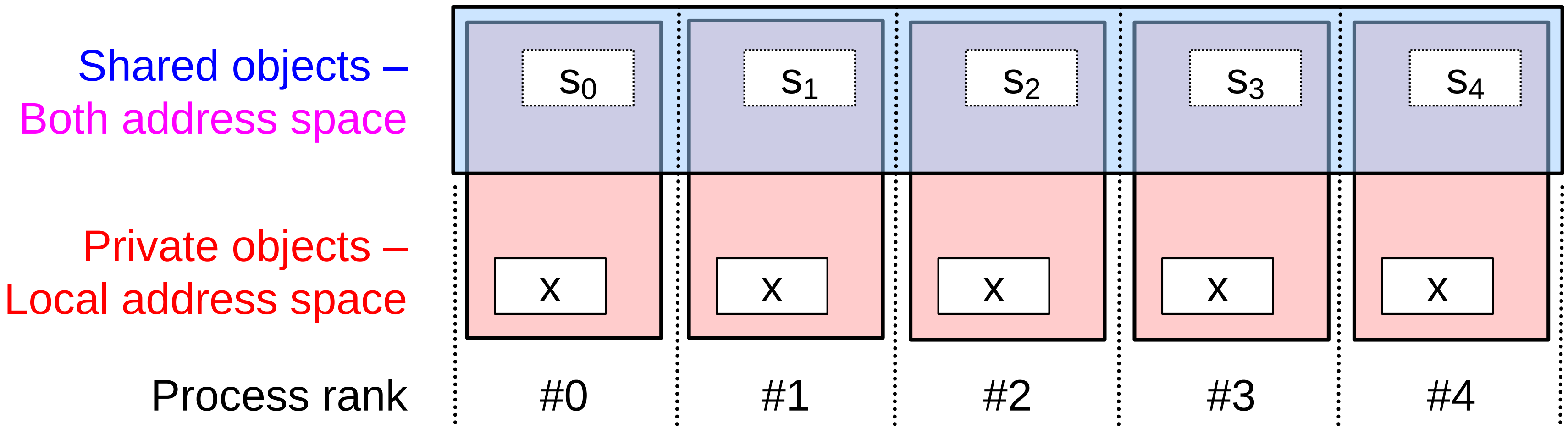
Static global memory

Dynamic global memory

Global Address Model

FUJITSU

- Each shared object has both global and local addresses



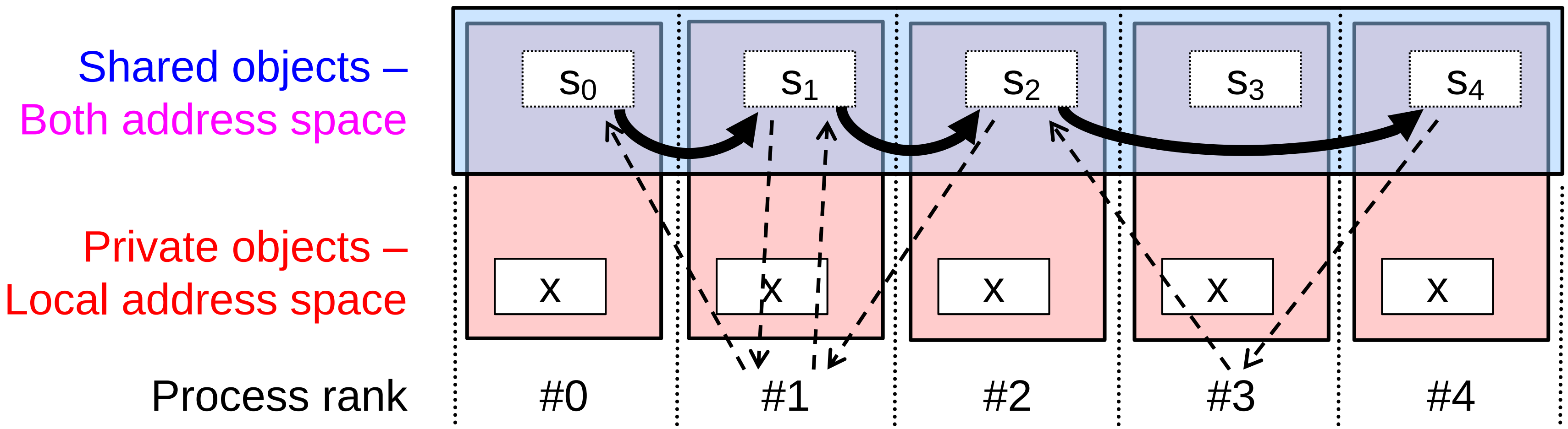
- Only the local process can obtain the local address corresponding to a global address

```
/* ... */
acp_ga_t my_bss = acp_query_starter_ga(acp_rank());
void* pointer = acp_query_address(my_bss + 64);
/* ... */
```

  - The return value will be NULL if the global address is out of the process

Data Transfer Model

- Only shared-to-shared data transfer is provided
  - The source and the destination can be an arbitrary global address
  - An arbitrary process can be the initiator of a data transfer
- Assumed protocol
  1. The initiator sends a request to the source
  2. The source transfers data to the destination
  3. The destination sends a notification to the initiator



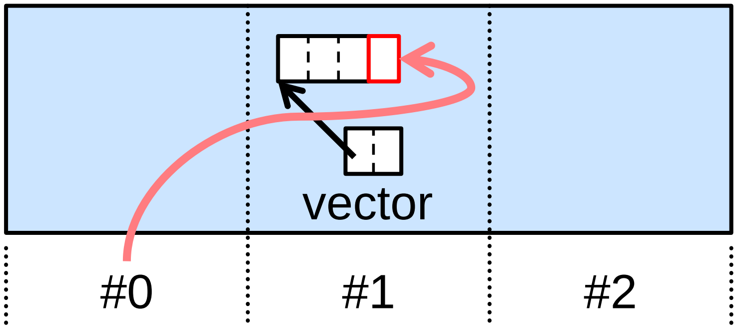
Index

- Introduction
- ACP basic layer
- ACP data library
- Issue and proposal
- Evaluation results
- Future work and summary

Categories of ACPdl Interfaces

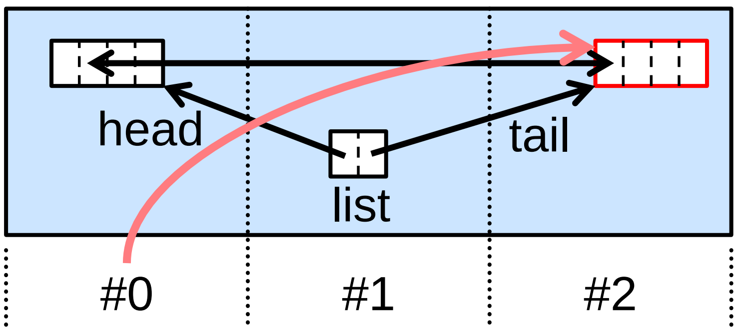
FUJITSU

- Memory allocator
  - acp_ga_t **acp_malloc** (*size_t* size, *int* rank);
    - Allocate a block of global memory at the specified process
  - *void* **acp_free** (acp_ga_t ga);
    - Deallocate a block of global memory
- Remote data structure types
  - acp_vector_t: dynamic array
  - acp_deque_t: double-ended queue



acp_push_back_vector (vector, element)

- Distributed data structure types
  - acp_list_t: doubly-linked list
  - acp_set_t: unordered dictionary
  - acp_map_t: unordered associative array

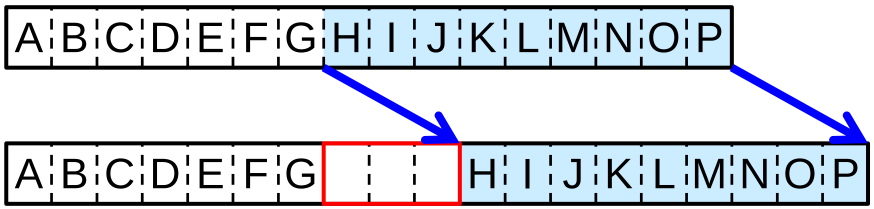


acp_push_back_list (list, element, 2)

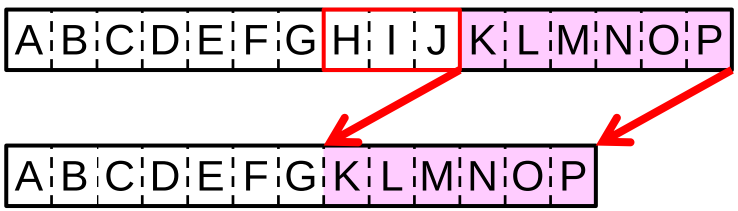
Manipulation of Remote Data

FUJITSU

- Inserting or erasing data at an arbitrary offset has high-cost
  - The data after the offset must be moved backward before the insertion



  - The latter part of the remaining data must be moved forward



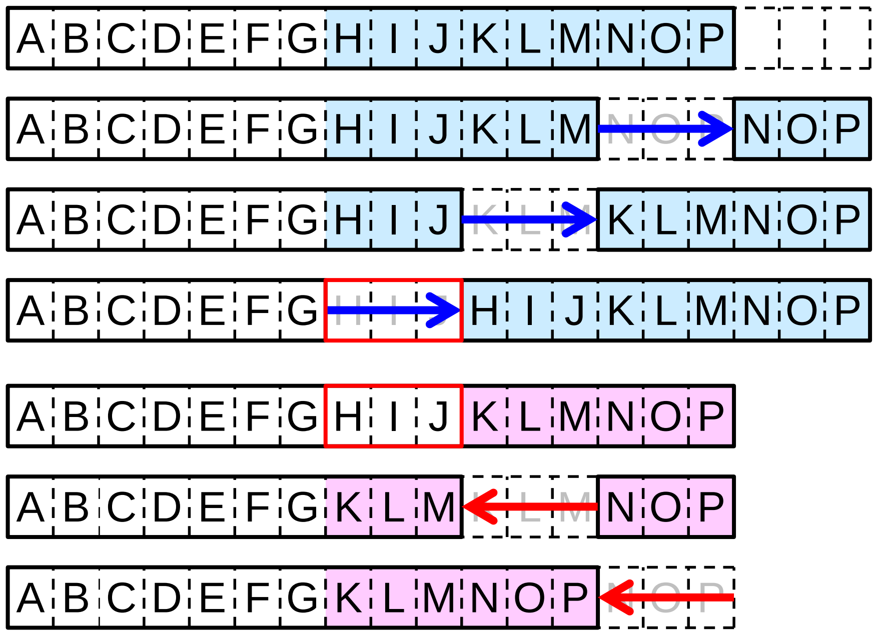
  - The source and destination of the data movement are overlapped
  - However, using a temporary buffer consumes additional memory

    cf. the memmove function of the standard C library

- Inserting or erasing data at a particular position has low-cost
  - At the end of vector
  - At the start or end of deque

In-Place Data Movement

- ACPdl uses the in-place algorithm for memory efficiency
  - Divide data into chunks and copy them sequentially
  - Insert
  - Erase



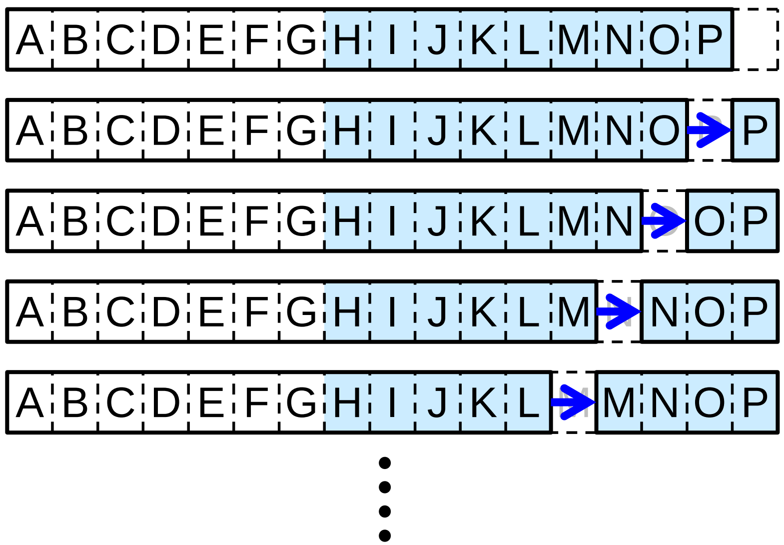
  - No temporal buffer is required
  - Minimum data movement
- Disadvantage
  - Increase in number of data transfers

Index

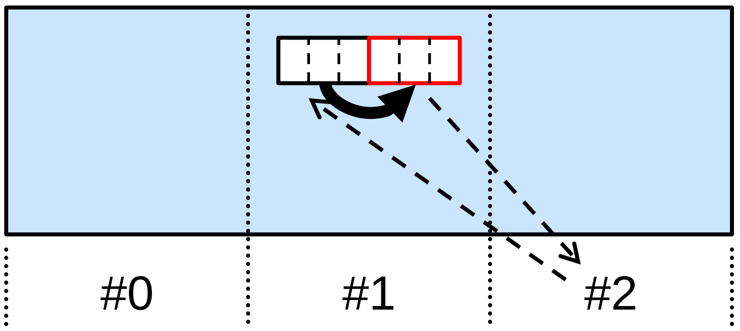
- Introduction
- ACP basic layer
- ACP data library
- Issue and proposal
- Evaluation results
- Future work and summary

Issues of the In-Place Data Movement

- Smaller element size increases the number of data transfers



- Issue: protocol overhead for each copy
  - The interaction between the initiator and the source incurs overhead



Copy Function and Ordering

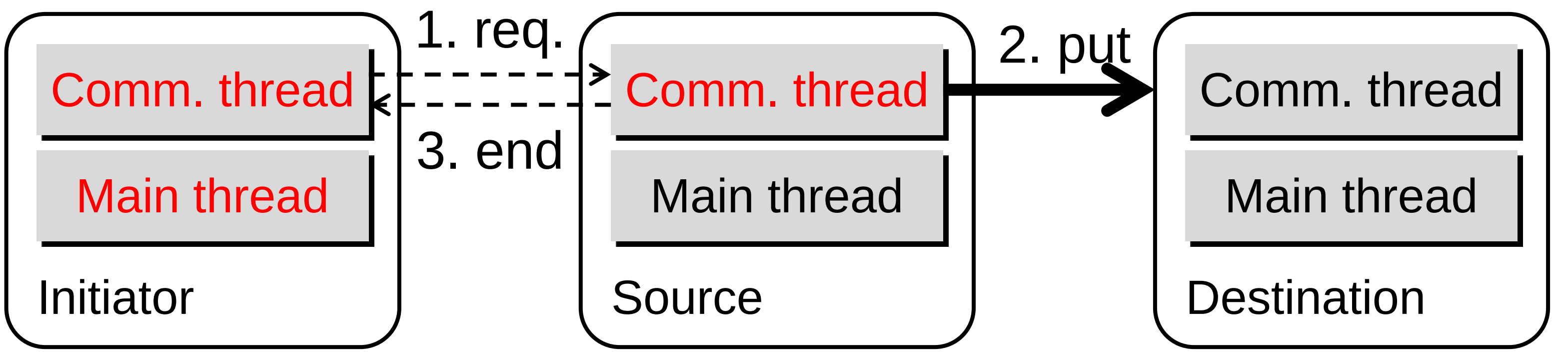
- Synopsis of the copy function

```
acp_handle_t acp_copy(acp_ga_t dst, acp_ga_t dst,
                              size_t size, acp_handle_t order);
```

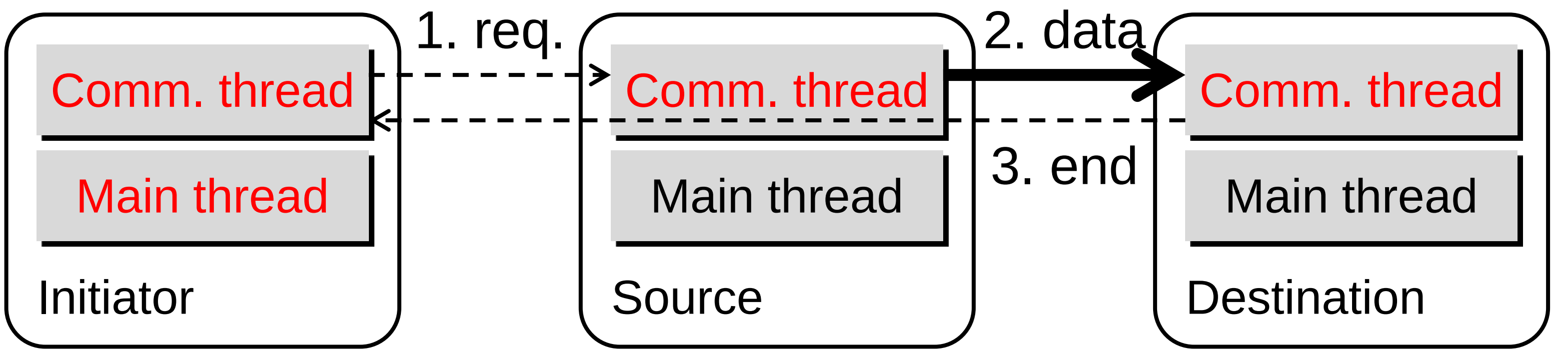
- Non-blocking
  - It returns an ordering handle value before starting the protocol
  - Completion can be done using the **acp_complete** function
- Strongly ordered data transfer
  - The start of the protocol is delayed by specifying an ordering handle until the completion of the specified data transfer
  - The defined macro ACP_HANDLE_ALL can be used instead of the most recently returned handle

Implementation with and without RDMA

- The protocol is processed by a communication thread
- Typical protocol implementation with RDMA



- Typical protocol implementation without RDMA

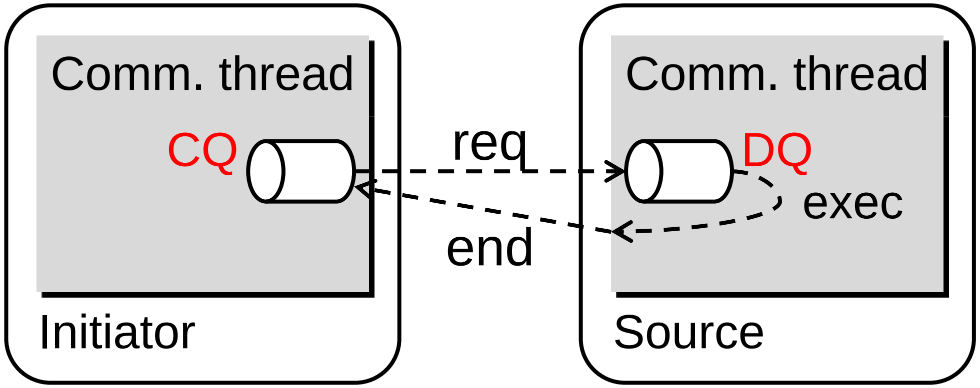


- This work studied the interaction between the initiator and the source, therefore using RDMA or not is of no consequence

Details of the Implemented Protocol

FUJITSU

- The initiator controls the execution order
  - The initiator awaits a notification of the preceding request to dequeue the next request from the command queue (CQ)
- The source executes received requests out of order
  - The source dequeues a request from the delegate queue (DQ) whenever the communication resource is available



- The request and notification round-trip is necessary even if the sources of consecutive copies are the same

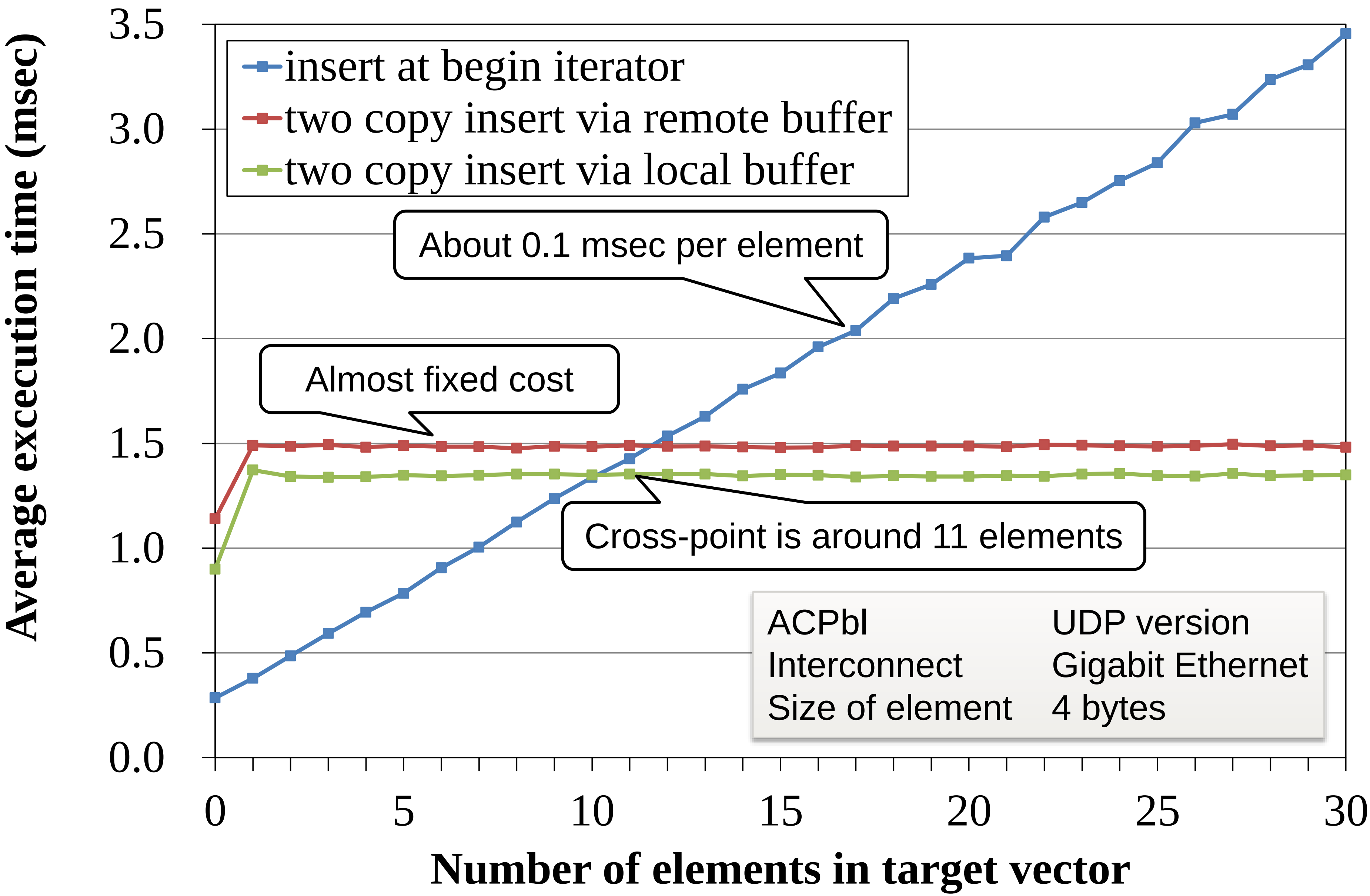
Proposal: Remote Ordering with Fence

- Concept of remote ordering
  - The initiator forwards the next request before receiving the notification
  - The source controls ordering of the forwarded request
- Fence flag
  - A simple implementation technique of the remote ordering
  - The initiator forwards the next request when it satisfies the following
    - It specifies sequential order
    - Its source process is the same as that of the previous request
    - Its destination process is the same as that of the previous request
  - The forwarded request has the fence flag
  - The source waits to dequeue a request with a fence flag from the DQ until sending all notifications of preceding requests in the DQ

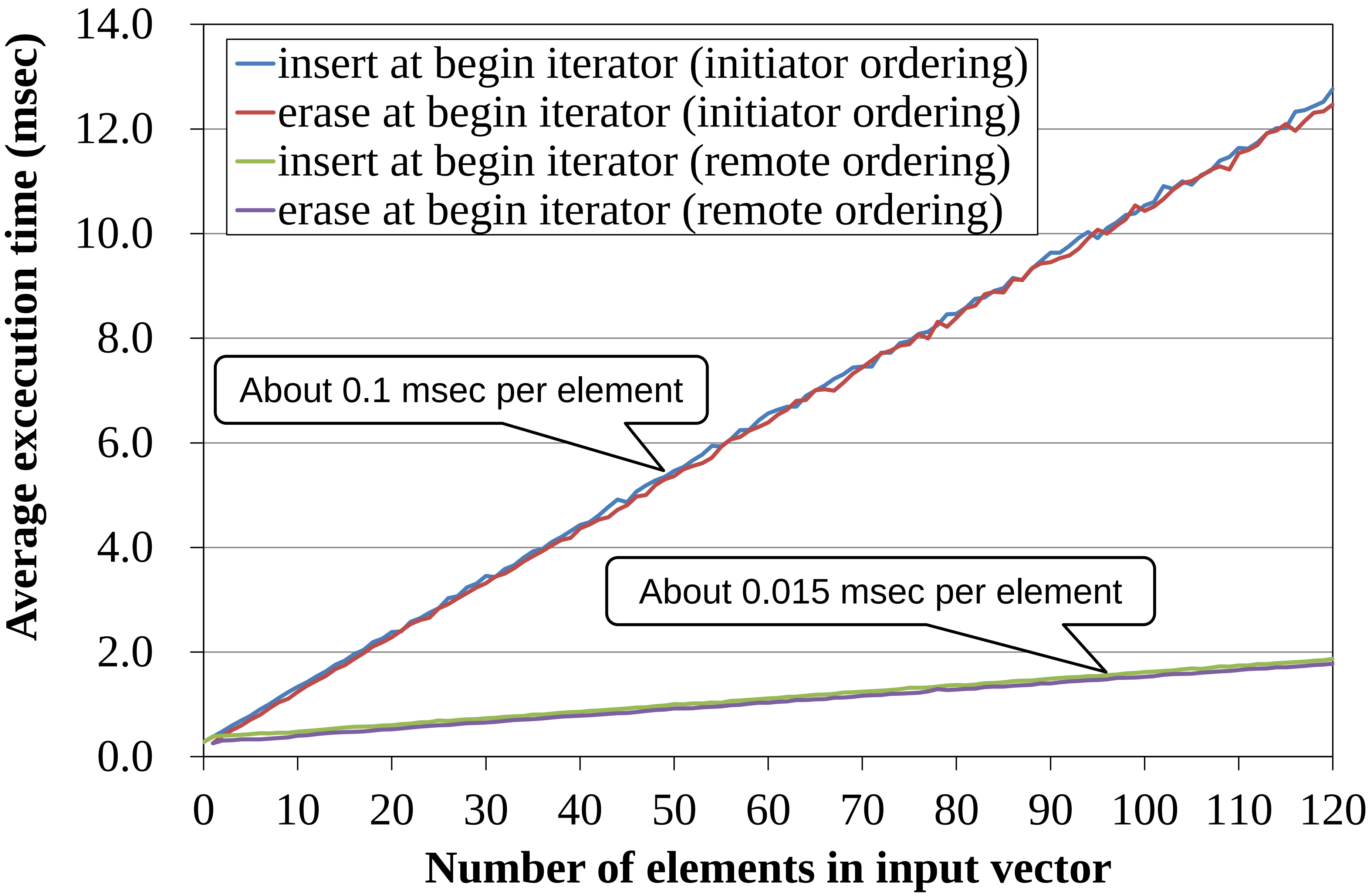
Index

- Introduction
- ACP basic layer
- ACP data library
- Issue and proposal
- Evaluation results
- Future work and summary

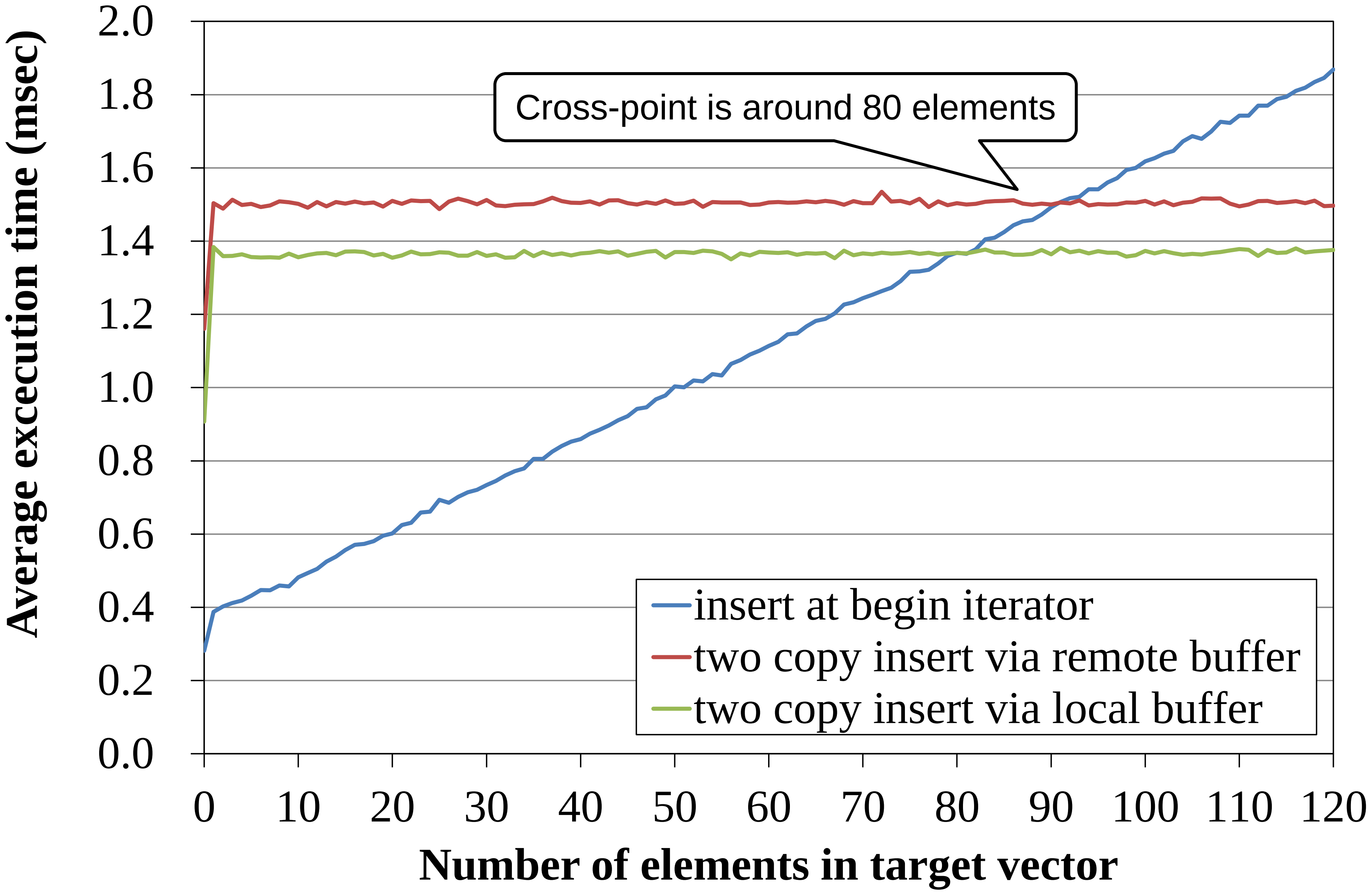
Results of Original Insertion



Results of Remote Ordering



Results of Improved Insertion



Index

- Introduction
- ACP basic layer
- ACP data library
- Issue and proposal
- Evaluation results
- Future work and summary

Future Work

- For the ACP data library
  - Further investigation of memory movement algorithms
    - Semi in-place hybrid algorithms
    - Adaptive algorithms
  - Providing a simple memory movement function
- For the ACP basic layer
  - Further investigation of remote-to-remote data transfer protocols

Summary

- The ACP is a communication library including a PGAS layer
  - Designed for memory-efficient programming
- The ACP includes remote data structure types
  - Memory-efficient in-place algorithms are implemented for manipulation
- The ordering control of the ACP basic layer caused overheads
- A new concept called remote ordering is proposed
  - A simple implementation technique with fence flag was also proposed
- The evaluation results showed the reduction of overheads from about 0.1 to about 0.015 milli-seconds per element

shaping tomorrow with you