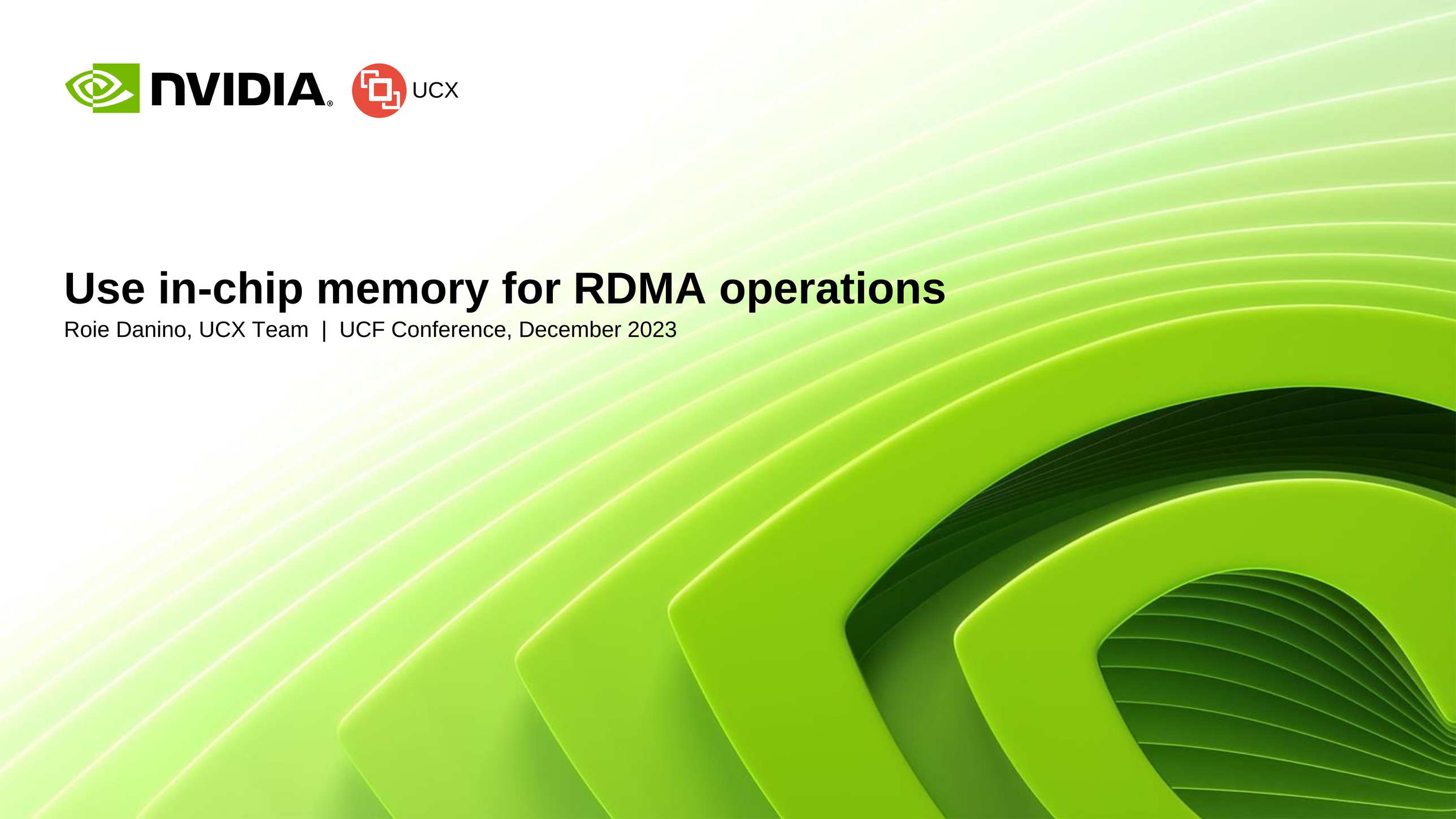


# Use in-chip memory for RDMA operations

Roie Danino, UCX Team | UCF Conference, December 2023

# UCX Library

## Unified Communication X

Abstracts communication transports

Selects best available route(s) between endpoints

TCP, RDMA, Shared Memory, GPU

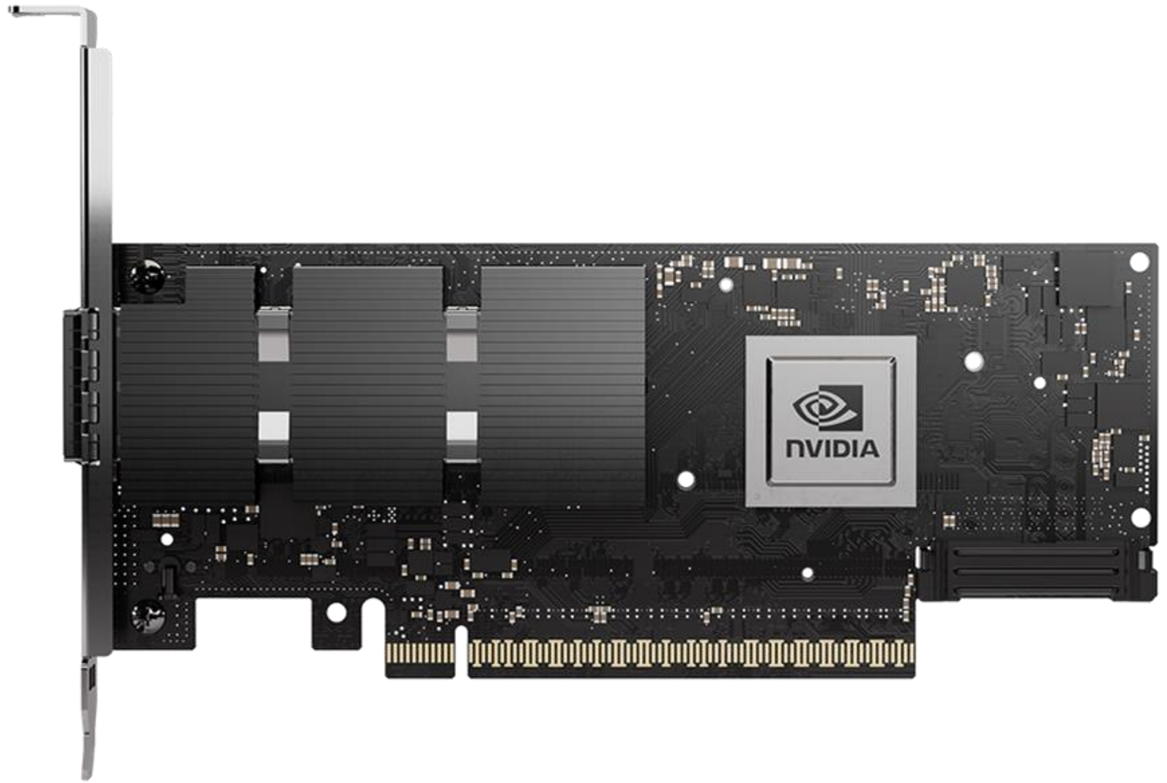
Zero-copy GPU memory transfers over RDMA

RDMA requires network support (IB or RoCE)

http://openucx.org

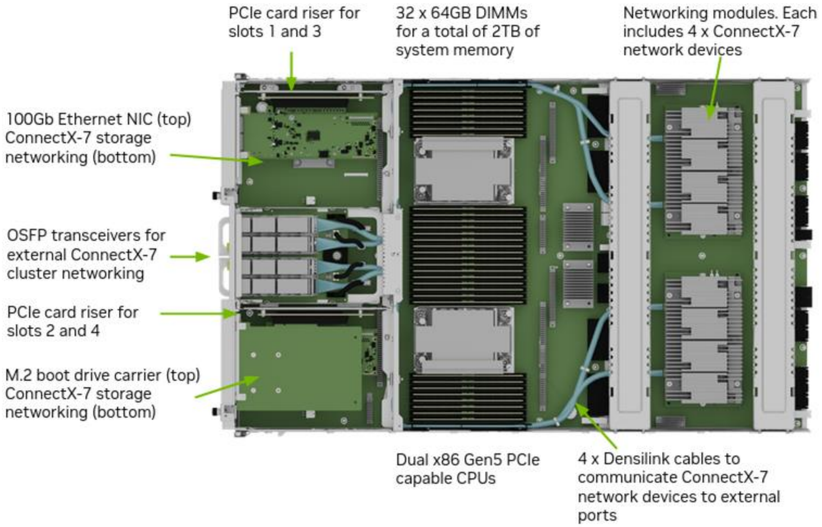
# InfiniBand MEMIC - Introduction

- Connect-X and BlueField devices contain a fast memory chip called MEMIC – Memory Mapped to InterConnect
- The MEMIC can be used for RDMA operations, as well as mapped to a process on the CPU
- Accessible over the network or the PCIe

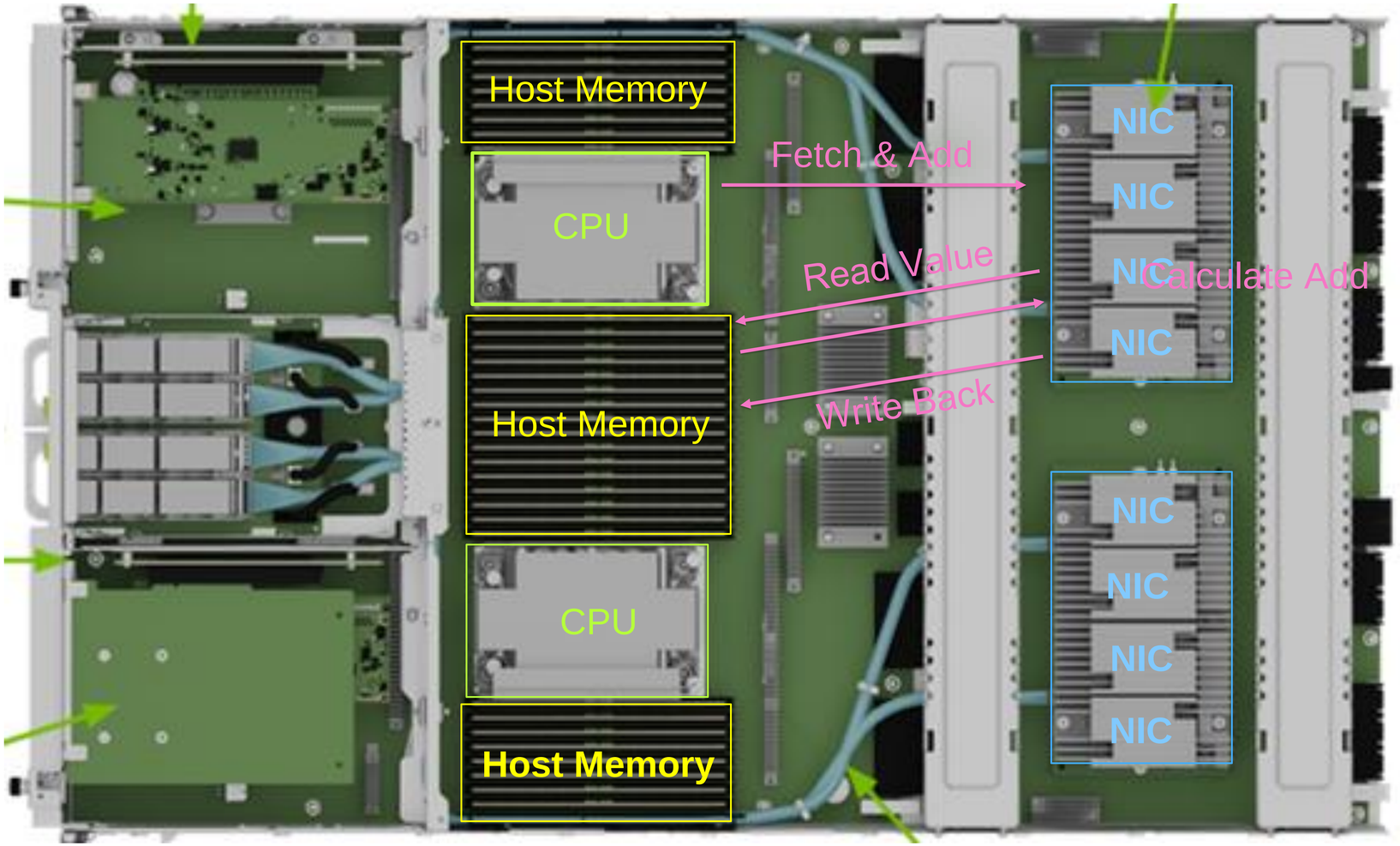


# InfiniBand MEMIC Motivation

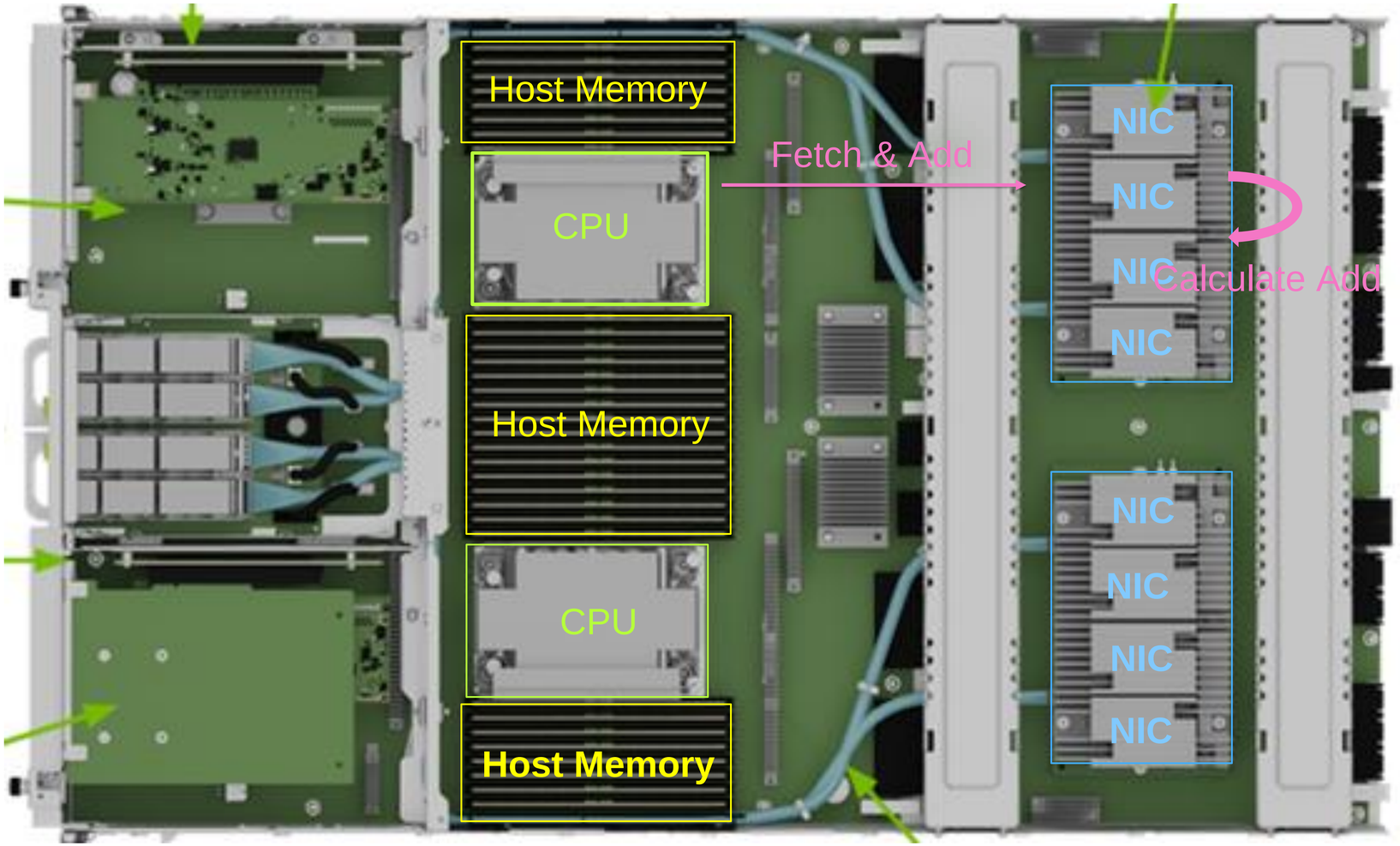
- Accessing host memory requires a round-trip of >350 nsec over the PCIe bus when performing atomic operations
- Using local on-NIC memory avoids that round-trip
- Reduce the latency of RDMA read and fetching atomic operations



# InfiniBand MEMIC – Atomic Fetch & ADD

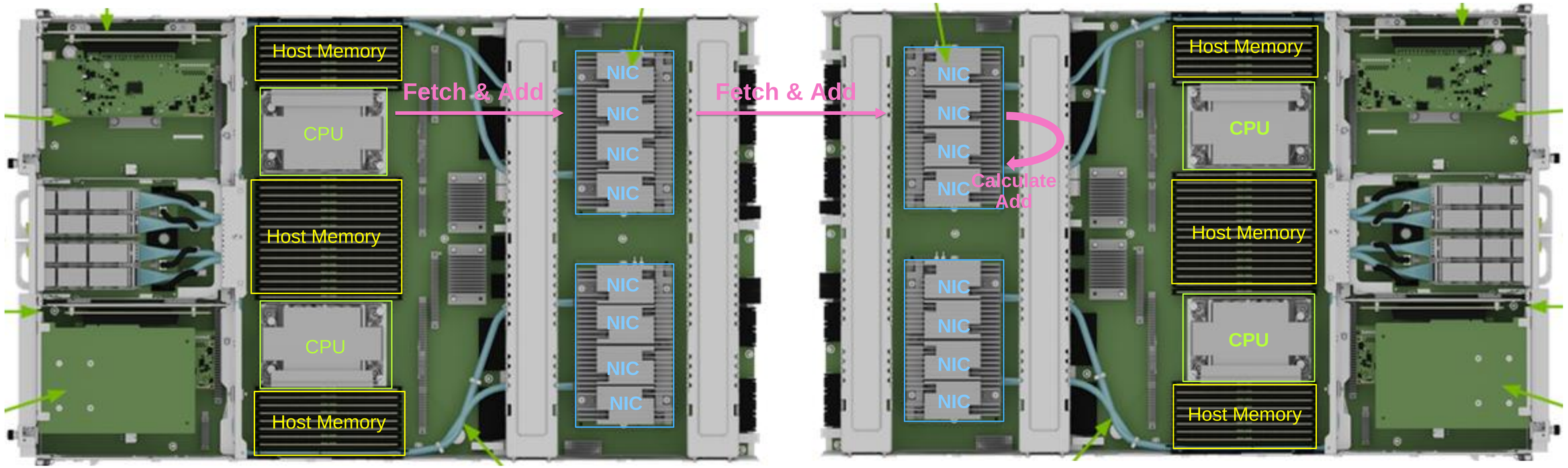


# InfiniBand MEMIC – Atomic Fetch & ADD



# InfiniBand MEMIC – Atomic Fetch & ADD

## MEMIC – Over the network



# InfiniBand MEMIC Usage / API

## UCX

- A new memory type was introduced: UCS_MEMORY_TYPE_RDMA
- MEMIC usage somewhat resembles GPU memory usage, since in both cases it's a region in the process address space that's accessible to the transport but not to the CPU.
- It can be allocated just like any other memory type using ucp_mem_map passing UCS_MEMORY_TYPE_RDMA as a memory type parameter.

```
ucp_mem_map_params_t mem_map_params;
ucp_mem_h mem_h;
ucs_status_t status;

mem_map_params.field_mask =
UCP_MEM_MAP_PARAM_FIELD_MEMORY_TYPE;

mem_map_params.memory_type = UCS_MEMORY_TYPE_RDMA;

status = ucp_mem_map(context, &mem_map_params, &mem_h);
```

# InfiniBand MEMIC Usage / API

## Open SHMEM

- Use the SHMEM_HINT_DEVICE_NIC_MEM hint for allocating device memory.
- If MEMIC is not available, host memory is allocated instead.

```
int main()
{
    int *buffer;

    shmem_init();

    buffer =
shmem_malloc_with_hints(sizeof(*buffer),

SHMEM_HINT_DEVICE_NIC_MEM);

    shmem_int_atomic_set(atomic_variable, 0,
my_id);

    shmem_free(buffer);

    shmem_finalize();

    return 0;
}
```

# InfiniBand MEMIC – Implementation

## Device memory allocation flow in UCX

1. Device memory is allocated using ibv_alloc_dm()
2. An address range is reserved using mmap, ensuring it is inaccessible from CPU:

```
address = mmap(NULL, dm_attr.length, PROT_NONE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
```

3. The MEMIC region is registered using ibv_dm_reg_mr(). The address is 0, because device memory is zero-based:

```
ibv_reg_dm_mr(md->pd, dm, 0, length, access_flags | IBV_ACCESS_ZERO_BASED)
```

# InfiniBand MEMIC – KSM Mapping

Problem:

- Device Memory is Zero Based – meaning its RDMA address is 0.
- Using 0 as the base address for all memory segments allocated on the IB device is undesirable

Solution:

1. Use *mmap* to reserve an address range that will be dedicated for each device memory allocation
2. Map 0 to the reserved address using KSM mechanism of the NIC

KSM enables indirect memory mapping in the NIC, for example:
mapping an existing remote key to a different custom RDMA address.

| Key | Value |
|---|---|
| 0xfff2340 | 0x100 |
| 0xffab320 | 0 |
| … | |
| | |
| | |

# InfiniBand MEMIC – Benchmarks

## P2P (ucx_perftest) Atomics Benchmarks

### ucx_perftest

- a single sender and a single receiver (one direction)

### Benchmarks

- ucp_fadd – fetch & add – atomic add and returns the old value.
- ucp_add – atomic add – a posted operation.

Command Line Example:

```
$ ./ucx_perftest -t ucp_fadd -c 0 -m host,rdma -s 8 -O16 <other host>
```

| Command Line Parameter | Description |
|---|---|
| -t | ucx_perftest test name: ucp_fadd, ucp_add |
| -c | CPU affinity |
| -m | Memory type: <sender, receiver> |
| -s | Message size (bytes): either 4 or 8 for atomics. |
| -O | Window size - number of uncompleted outstanding sends |

# InfiniBand MEMIC - Results

## P2P (ucx_perftest)

Units

Latency - µsec (Lower is better)

Message rate - Million ops / sec (Higher is better)

| | Window Size | 1 | | | 16 | | |
|---|---|---|---|---|---|---|---|
| | Atomic Allocation | Host | MEMIC | Diff | Host | MEMIC | Diff |
| Fetch & Add | Latency | 1.653 | 1.426 | -14% | 0.42 | 0.123 | -71% |
| | Message Rate | 604999 | 701437 | 16% | 2381344 | 8131869 | 241% |
| Atomic Add | Latency | 0.627 | 0.299 | -52% | 0.588 | 0.3 | -49% |
| | Message Rate | 1595301 | 3341689 | 109% | 1699894 | 3328350 | 96% |

Hardware

2 nodes, each with:

2x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz (15 cores each)

2x Connect-X 6 device

Software

UCX – v1.16.0

# Contended Atomics Benchmark

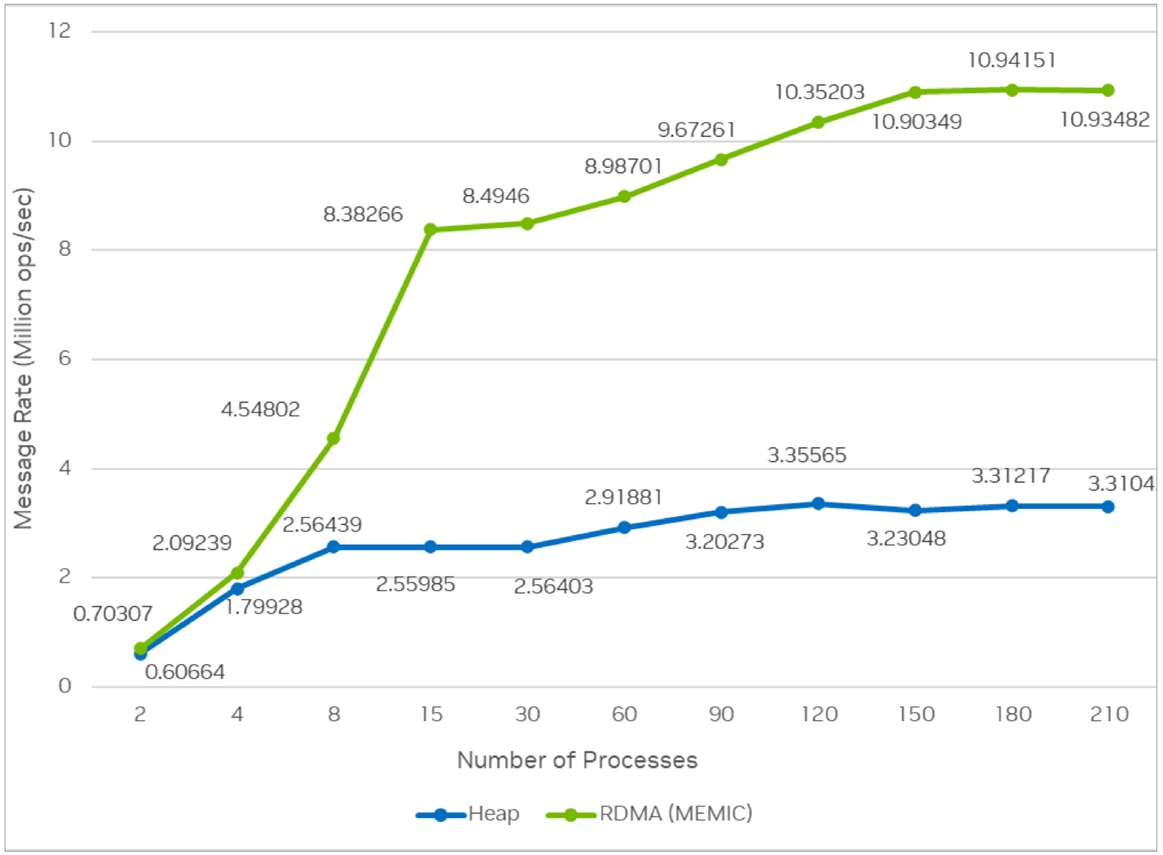
## OpenSHMEM Fetch & Add

1. Root process allocates an atomic variable in either:
   1. symmetric heap
   2. global data segment
   3. MEMIC
2. Its value is set to 0
3. Barrier
4. For each iteration:
   - Fetch and add 1 to the shared atomic variable
5. Barrier
6. Collect results from all processes

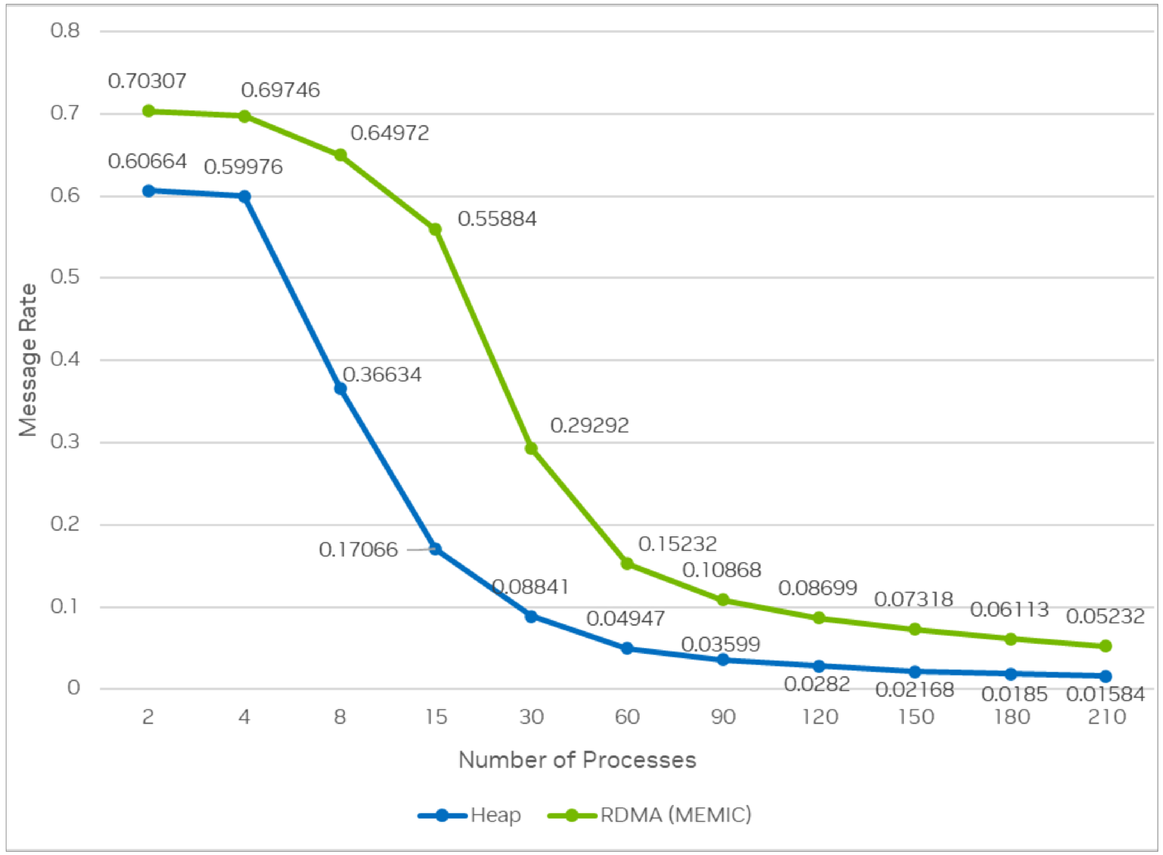
```
void benchmark_fadd(int my_id, int npes, int* atomic_variable, unsigned long iterations)
{
    double begin, end;
    int i;
    static double rate = 0, sum_rate = 0, min_rate = 0, max_rate = 0;

    shmem_int_atomic_set(atomic_variable, 0, my_id);
    shmem_barrier_all();

    if (my_id != 0) {
        int value = 1;
        int old_value;

        begin = get_wall_time();
        for (i = 0; i < iterations; i++) {
            old_value = shmem_int_fadd(atomic_variable, value, 0);
        }
        end = get_wall_time();

        rate = ((double)iterations) / (end - begin);
    }

    shmem_barrier_all();

    shmem_double_sum_to_all(&sum_rate, &rate, 1, 0, 0, npes, pwrk, psync);
    shmem_double_max_to_all(&max_rate, &rate, 1, 0, 0, npes, pwrk, psync);

    /* Small hack to exclude root process from minimum calculation */
    if (my_id == 0) {
        rate = DBL_MAX;
    }
    shmem_double_min_to_all(&min_rate, &rate, 1, 0, 0, npes, pwrk, psync);
    print_operation_rate(my_id, "shmem_int_fadd", sum_rate/1e6, min_rate/1e6,
                         max_rate/1e6, npes);
}
```

# InfiniBand MEMIC - Results

## SHMEM Fetch & Add Benchmark – 15 Nodes




Total Message Rate – 15 Nodes
(Million ops/sec)




Message Rate Per Process – 15 Nodes
(Million ops/sec)

Hardware:15 Nodes, with
2x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz (15 cores each)
2x Connect-X 6 device
Software: UCX – v1.16.0, OpenMPI – v5.0.0

# InfiniBand MEMIC – Conclusions

1. Using NIC memory for atomics improves performance compared to host memory
2. It is noticeable that from a certain number of processes per NIC the improvement is less significant for each added process.
3. Using more NICs preserves per-process performance when using NIC memory, as opposed to host memory.


NVIDIA
Thanks