Efficient Multi-GPU Communication with NVSHMEM in Chapel

Sosuke Hosokawa, Kenjiro Taura

The University of Tokyo

Limitation in inter-GPU Communication in Current Chapel

- In current Chapel, a GPU cannot access directly to the memory on another GPU.

```
on Locales[0] do on here.gpus[0] {
  // On GPU A
  var arrOnGpuA: [0..N] real;

  on Locales[1] do on here.gpus[0] {
    // On GPU B
    var arrOnGpuB: [0..N] real;

    // forall converted to a GPU kernel
    forall i in 0..N {
      arrOnGpuA[i] = i * i;
    }
  }
}
```

In current Chapel, this is NOT allowed operation:

- **A GPU kernel can only access the memory on the device it runs on.**

Limitation in inter-GPU Communication in Current Chapel

```
on Locales[0] do on here.gpus[0] {
  // On GPU A
  var arrOnGpuA: [0..N] real;

  on Locales[1] do on here.gpus[0] {
    // On GPU B
    var arrOnGpuB: [0..N] real;

    // Copy from arrOnGpuA to arrOnGpuB
    arrOnGpuB = arrOnGpuA;

    // forall converted to a GPU kernel
    forall i in 0..N do
      arrOnGpuB[i] = i;

    arrOnGpuA = arrOnGpuB;
  }
}
```

The only way to access the memory of another GPU in Chapel is **array-wise copy**.
This operation triggers a data transfer handled by **DMA** (intra-node) or **GASNet via CPU** (inter-node, without GPUDirect RDMA)

Disadvantages of this approach:
- High latency for small data transfer
- Poor inter-node bandwidth due to the lack of GPUDirect support.

NVSHMEM: PGAS-style communication for GPUs

NVSHMEM is a library for efficient PGAS-style inter-GPU communication.

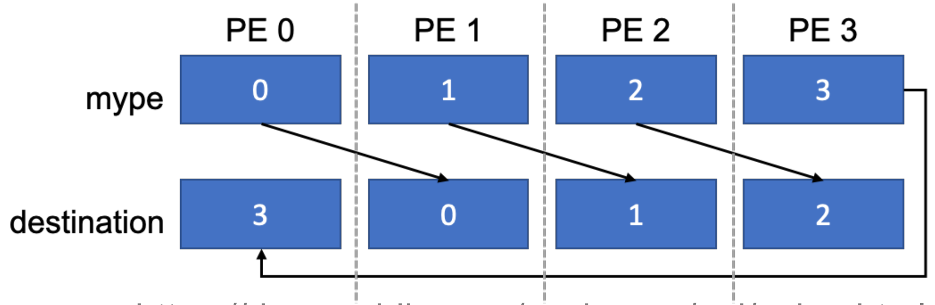
Major features of NVSHMEM are:

- **Device-initiated** remote memory accesses (RMA).
- Reduces RMA overhead by **leveraging some sophisticated technologies**, such as GPUDirect RDMA.
- SHMEM-like **symmetric memory style PGAS** API.

```
__global__ void simple_shitf(int *mem) {
    int mype = nvshmem_my_pe();
    int npes = nvshmem_n_pes();
    int nextpe = (mype + 1) % npes;

    // Memory access to another GPU
    nvshmem_int_p(mem, mype, nextpe);
}
```

Example kernel using NVSHMEM



https://docs.nvidia.com/nvshmem/api/using.html

`nvshmem_int_p` is inter-GPU memory put operation inside kernel.

Idea: Integrate NVSHMEM into Chapel

- In this work, we are trying to build **more efficient inter-GPU communication library for Chapel** than Chapel's runtime on top of NVSHMEM.
- There are a lot of challenges both in its implementation and designing its API.

```
coforall loc in Locales do on loc {
    var pe = nvshmem_my_pe();
    var next = (pe + 1) % n_gpus;

    // Shift data to the next GPU
    forall i in 0..<1 {
        // Inside a GPU kernel
        nvshmem_int_p(sym_ptr, pe, next);
    }
}
```

What if we can use NVSHMEM with Chapel…?

Challenges in implementing NVSHMEM integration

There are some challenges in implementing NVSHMEM as a Chapel library binding.

The main challenges are:

- Build pipeline of FFI CUDA library for Chapel
- Runtime initialization of NVSHMEM

CUDA device function FFI for Chapel

- Currently, Chapel does not support CUDA‘s device function FFI.
- But, of course, NVSHMEM’s device-side API is implemented as device function of CUDA.
- We overcome this issue by modifying CUDA code build pipeline inside Chapel compiler.
- Specifically, added **GPU code linking step** into compile pipeline.

```
__device__
int cuda_dev_func(int arg) {
   ...
}
```

Device-callable function implemented in CUDA.

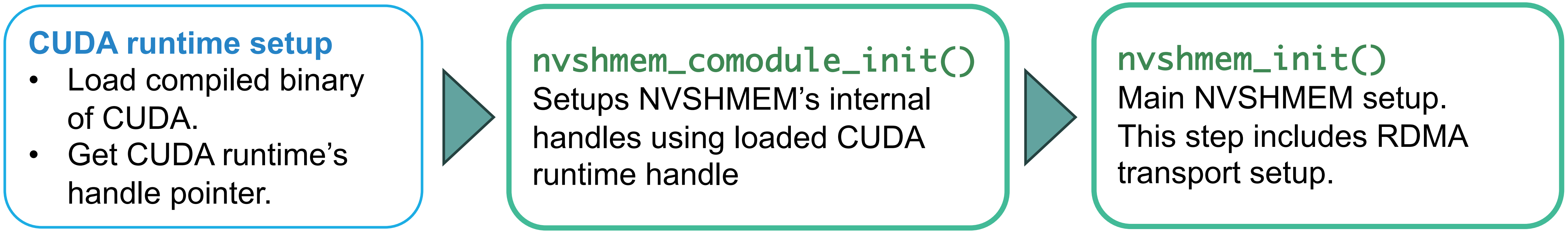
$\Rightarrow$

```
extern proc
dev_func(arg: c_int): c_int
```

extern function in Chapel.

Runtime Initiatialization of NVSHMEM

- NVSHMEM requires extra step for initialization when it is called from outside of CUDA.
- We modified Chapel's GPU runtime initialization step to use NVSHMEM.



CUDA runtime setup
- Load compiled binary of CUDA.
- Get CUDA runtime's handle pointer.

`nvshmem_comodule_init()`
Setups NVSHMEM's internal handles using loaded CUDA runtime handle

`nvshmem_init()`
Main NVSHMEM setup. This step includes RDMA transport setup.

Chapel runtime's GPU initialization step with NVSHMEM

🟦 Steps from normal GPU initialization

🟩 Steps added for NVSHMEM initialization

API of NVSHMEM bindings

- In our current implementation of NVSHMEM bindings for Chapel, we only exposes **low-level NVSHMEM API**.
- However, this API is **not user-friendly for Chapel programmers** as it directly follows **SHMEM's symmetric-memory style PGAS**.
- We plan to improve this in the future, but for now, we **leave it as future work**.

```
on Locales[0] do on here.gpus[0] {
  // On GPU A
  var arrOnGpuA: [0..N] real;

  on Locales[1] do on here.gpus[0] {
    // On GPU B
    var arrOnGpuB: [0..N] real;

    // forall converted to a GPU kernel
    forall i in 0..N {
      arrOnGpuA[i] = i* i; // Inter-GPU access
    }}}
```

A **GPU-ready, Chapel-style PGAS** model like this is ideal,
but it is challenging to implement on top of a symmetric-memory-based PGAS.

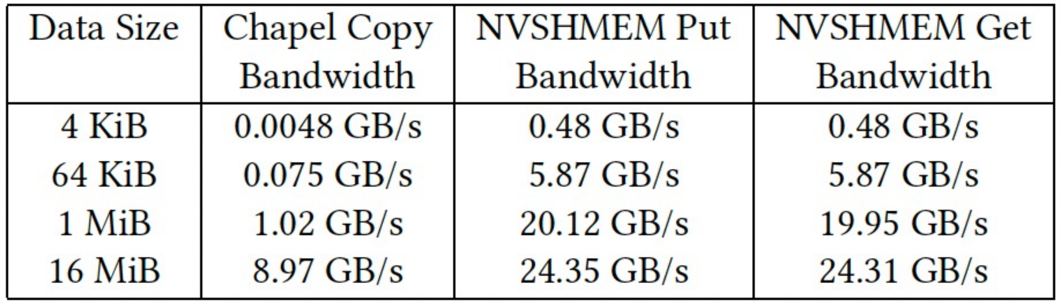
Evaluation: Environment

- We evaluated our NVSHMEM bindings on Utokyo’s miyabi-g supercomputer.

Miyabi-g node spec:

- NVIDIA Grace CPU (72 cores/CPU, memory 72 GB)
- NVIDIA Hopper H100 GPU x1
- GPU memory: 96 GB
- Interconnect: Infiniband NDR (200 Gbps)

Evaluation: Bandwidth



Data Size	Chapel Copy Bandwidth	NVSHMEM Put Bandwidth	NVSHMEM Get Bandwidth
4 KiB	0.0048 GB/s	0.48 GB/s	0.48 GB/s
64 KiB	0.075 GB/s	5.87 GB/s	5.87 GB/s
1 MiB	1.02 GB/s	20.12 GB/s	19.95 GB/s
16 MiB	8.97 GB/s	24.35 GB/s	24.31 GB/s

```
forall i in 0..<1 {
  for iteration in 0..<numIters {
    nvshmem_int64_put(
      dstPtr, srcPtr, dataSize, dstPe);
  }}

forall i in 0..<1 {
  for iteration in 0..<numIters {
    nvshmem_int64_get(
      dstPtr, srcPtr, dataSize, dstPe);
  }}
```

- The memory access bandwidth of our runtime is better than original Chapel runtime copy
  - Both in small and large data size.
  - In small data size: ~ 100x
  - In large data size: ~ 3x
- Achieve theoretical bandwidth (25 GB/s = 200 Gbps) in large communication size.

Evaluation: latency

Results:

NVSHMEM on Chapel:	6.744 us
NVSHMEM on CUDA:	6.653 us
CPU OpenSHMEM:	2.140 us

- NVSHMEM on Chapel (ours) result is as good as NVSHMEM on CUDA.
- CPU OpenSHMEM ping-pong is faster.

```
cobegin {
  on Locales[0] {
    var me = nvshmem_my_pe(); var peer = 1 - me;
    on here.gpus[0] {
      forall i in 0..<1 {
        for i in 0..<numIters {
          nvshmem_int_atomic_inc(sym_ptr, peer);
          nvshmem_int_wait_until(sym_ptr, CMP_EQ, i+1);
  }}}}


  on Locales[1] {
    var me = nvshmem_my_pe(); var peer = 1 - me;
    on here.gpus[0] {
      forall i in 0..<1 {
        for i in 0..<numIters {
          nvshmem_int_wait_until(sym_ptr, CMP_EQ, i+1);
          nvshmem_int_atomic_inc(sym_ptr, peer);
}}}}}
```

Conclusion

- In this work, we explored the integration of NVSHMEM with Chapel.
- Our implementation achieves good performance in microbenchmarks, but its API still needs improvement.
- We plan to further enhance our implementation to make Chapel's multi-GPU support more efficient and user-friendly.