Use of BlueField SmartNICs in Offloading One-Sided Communication Primitives

Benjamin Michalowicz[1], Kaushik Kandadi Suresh[1], Hari Subramoni[1], Dhabaleswar K. Panda[1], Steve Poole[2]
[1]The Ohio State University, [2]Los Alamos National Laboratory
{michalowicz.2, kandadisuresh.1, subramoni.1, panda.2}@osu.edu, swpoole@lanl.gov

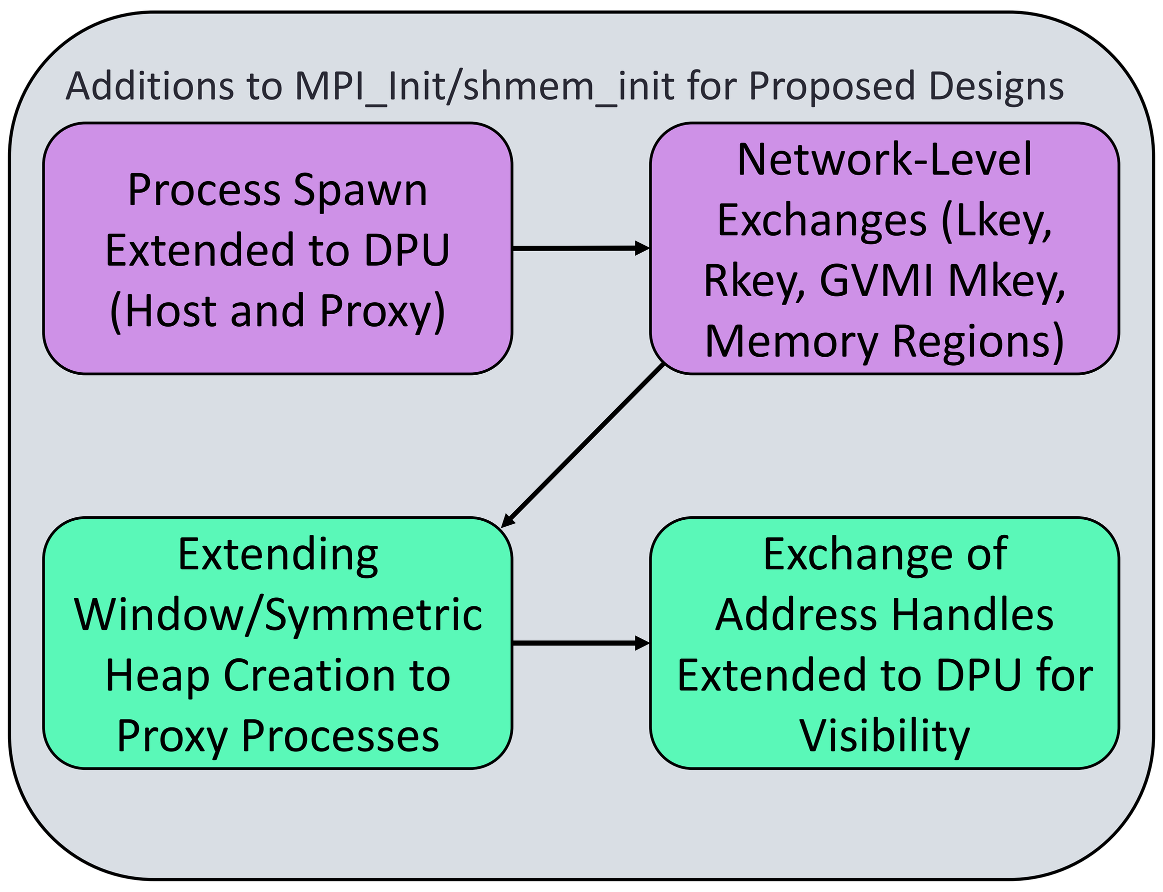
MOTIVATION

- Two-Sided Communication has been successfully offloaded to SmartNICs such as NVIDIA's BlueField-2 and BlueField-3 (BF-2/3)
- One-Sided Communication (1SC) is inherently nonblocking, which can leverage SmartNICs to gain more overlap between communication and compute.
- 1SC is used across multiple programming libraries/implementations (MPI, PGAS/OPENSHMEM, etc.)
  - We focus on MPI and OpenSHMEM

RESEARCH CHALLENGES

- Designing a library-agnostic framework for offloading 1SC (Put and Get operations)
- Account for different approaches w/ MPI and OpenSHMEM and how the execute 1SC
- Use of low-level, advanced network primitives for efficient, scalable designs on both BF-2 and BF-3 DPUs
  - Emphasis on network and use of memory subsystem and less on advanced systems

HIGH-LEVEL CHANGES IN HPC LIBRARIES



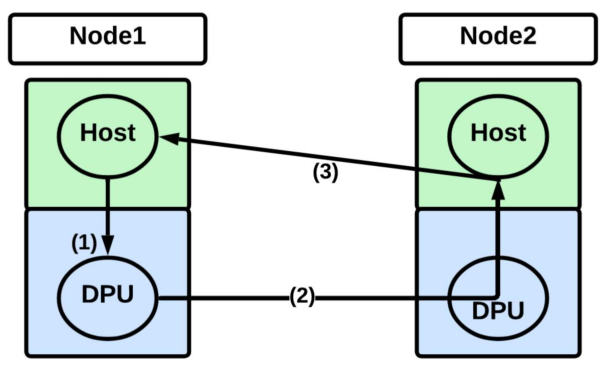
SUMMARY OF CONTRIBUTIONS

- Design of a standard/library-agnostic framework for offloading 1SC
- Scalable design at both the benchmark and the application level (BSPMM kernel)
- Design of different synchronization types to account for, e.g., flushing of MPI windows or completing nonblocking 1SC in OpenSHMEM
- Demonstration of efficiency up to 512 Processes with 4 BF-3 DPUs and AMD-Epyc CPUs (128-core)
- Demonstration of scalability up to 256 Processes with 8 BF-2 DPUs and Intel Broadwell CPUs (32-core)
- Results: Up to 24x speedup compared to a blocking kernel (System 1) and up to 99x speedup compared to a non-blocking kernel (System 2)

DESIGNS AND OMB BENCHMARK PERFORMANCE (VIA BF-2)

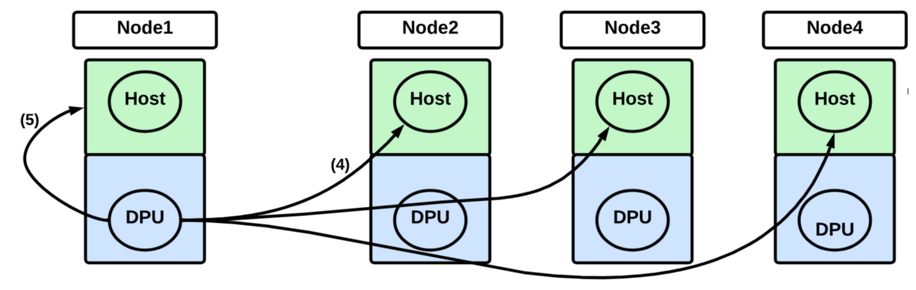
DESIGN: EMPHASIS ON "GET" AND "QUIET/FLUSH", API "INTEGRATION"

Design for Nonblocking "Get" – MPI_Get and shmem_get_nbi



- Step 1: Host1 issues non-blocking get, sends metadata to the DPU, increments counter (per-process, per-window)
- Step 2: DPU utilizes GVMI firmware in BF-2/3 to perform RDMA operations
- Step 3: "Return Data" (in Flush/synchronization)

Design for synchronization – MPI_Win_flush_all/shmem_quiet



- Step 4: Per-process, per-window counter is used on host side, DPU-proxy counter is used on worker-side to issue fetch-add operations
- Step 5: Each proxy process "returns" to their matched host process

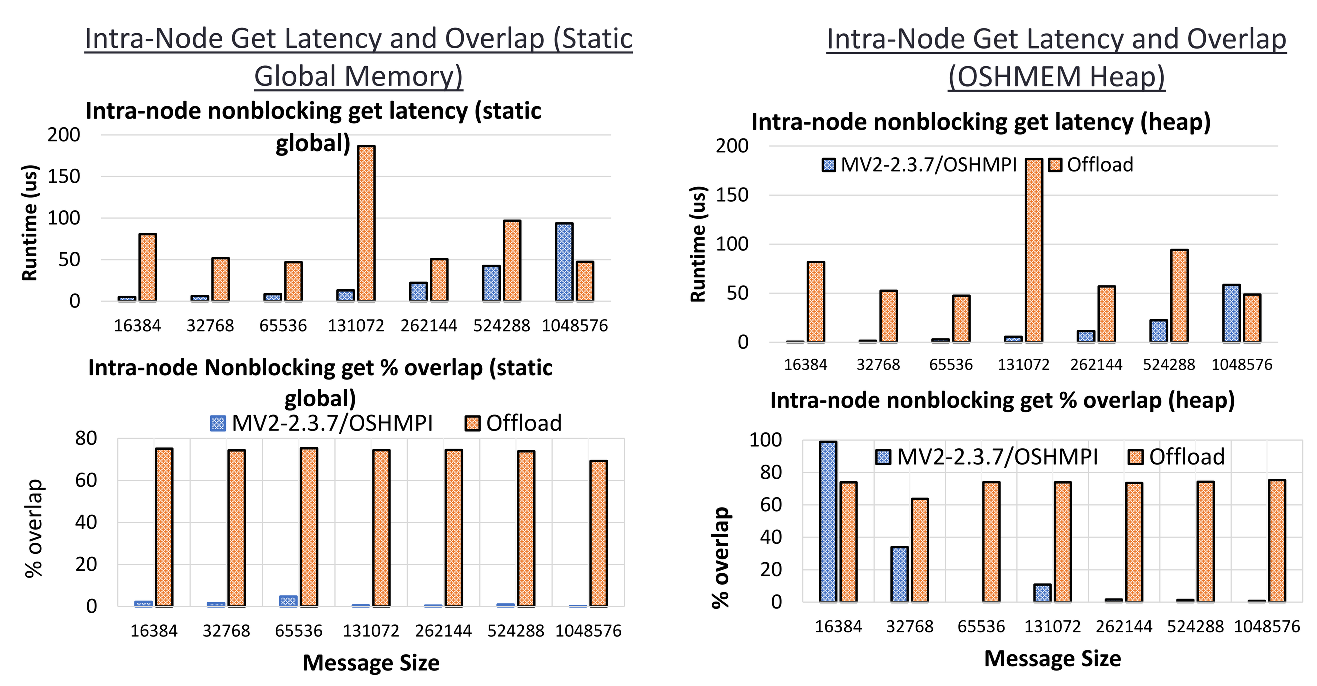
API Integration into MPI and OpenSHMEM Libraries

```
MPI_Win_allocate(..., win, &win_buffer){
    window = win_init(win, win_buffer);
    win_populate(window, win_buffer);
    proxy_exchange_win(&window, win_buffer);
    return window;
}
MPI_Put(addr1, count1, datatype1, target_rank,
    target_disp, target_count, target_datatype,
    window){
    // Other metadata setup ...
    addr2 = buf_of (window) + disp
    bytes = count*get_size(datatype);
    return Offload_put(addr1, addr2, target, bytes);
}
MPI_Get(addr1, count1, datatype1, target_rank,
    target_disp, target_count, target_datatype,
    window){
    // Other metadata setup ...
    addr2 = buf_of (window) + disp
    bytes = count*get_size(datatype);
    return Offload_get(addr1, addr2, target, bytes);
}
MPI_win_flush_all(window){
    return Offload_flush(window);
}
```

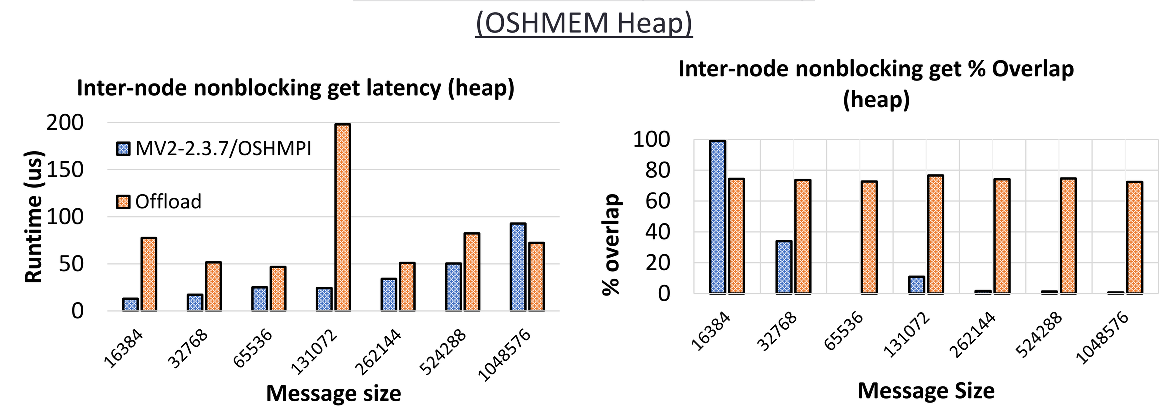
```
shmem_init(){
    initiate_symm_heap();
    proxy_exchange_symm_heap();
}
shmem_malloc(size){
    buffer = allocate();
    barrier(); /* All procs allocate */
    proxy_exchange_update(buffer);
    return buffer;
}
shmem_TYPE_put_nbi(TYPE *src, const TYPE *dst, int
    count, int target){
    bytes = count * sizeof(TYPE);
    Offload_put(dst, src, target, bytes);
}
shmem_TYPE_get_nbi(TYPE *src, const TYPE *dst, int
    count, int target){
    bytes = count*sizeof(TYPE);
    Offload_get(dst, src, target, bytes);
}
shmem_quiet(){
    Offload_flush(NULL);
}
```

- Proxy_exchange in shmem_malloc() there to "show" DPUs are aware of the usage of space in the symmetric heap.

Intra-Node OSU OpenSHMEM Benchmarks



Inter-Node OpenSHMEM Benchmarks



- Overhead largely comes from effort to offload small messages. Larger msgs take advantage of overlap (goal of offloading, not as much for improved transfer time) – DPU-aware designs more sensitive to cache compared to pure-host, leads to spike at 128KB on BF-2
- Intra-node comparison: Designs still go over the network, and while shared memory is fast, minimal overlap occurs after 64KB messages – host-based progression will get impacted in more dense execution.

BLOCK SPARSE MM KERNEL: BF-2 AND BF-3 PERFORMANCE ("BABY" VARIANT OF NWCHEM)

Blocking and Non-blocking variants via OpenSHMEM

- "Get-Compute-Update" Pattern
- Mesh: X, Y parameters, but Y has another "dimension" inside (X rows, X blocks per row, Y cells per block) → Calculate buffer as X*X*Y * (sizeof(double))
- Blocking variant:

```
while (work_unit!=max_unit_count) {
        blocking_get(); dgemm(); update();
}
```

- Nonblocking variant:

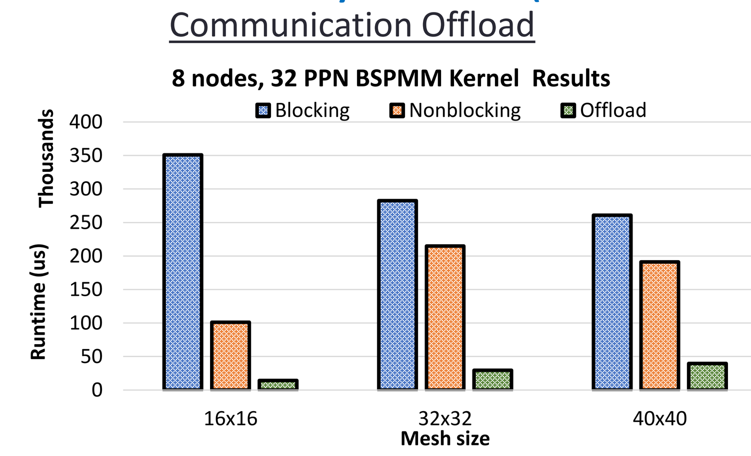
```
blocking_get(cur_bufs);
while(work_unit!=max_unit_count) {
        nb_get(next_bufs);
        dgemm(cur_bufs);
        sync(next_bufs);
        update();
        cur_bufs = next_bufs;
}
```

Experimental Setup

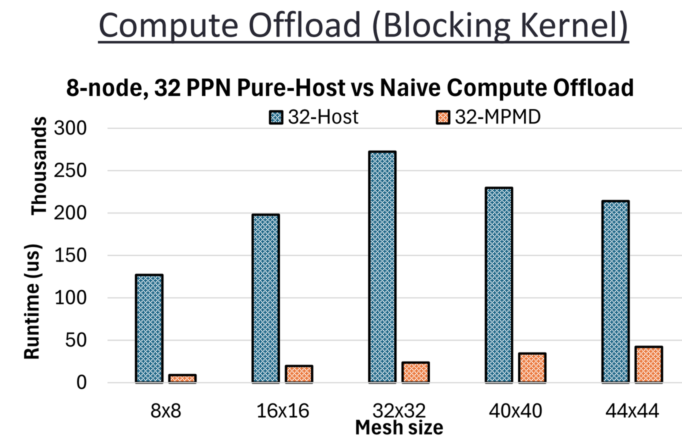
- Experimental Systems:
  - System 1: Intel Skylake (20 cores x 2 sockets) w/ BF-2s and HDR100 IB
  - System 2: AMD EPYC (64 cores x 2 sockets) w/ BF-3s and HDR200 IB
- Libraries
  - MVAPICH2-2.3.7 and OSHMPI Framework
  - Offshoot of MVAPICH with 1SC designs and OSHMPI Framework
  - OSHMPI – Standard-Compliant Framework to have OpenSHMEM be emulated by MPI primitives
    - Symmetric Heap → One MPI Window
    - shmem_put → MPI_Put + immediate MPI_Win_flush_all()
    - Etc.

Performance on System 1 (Comm and Compute Offload)

Communication Offload



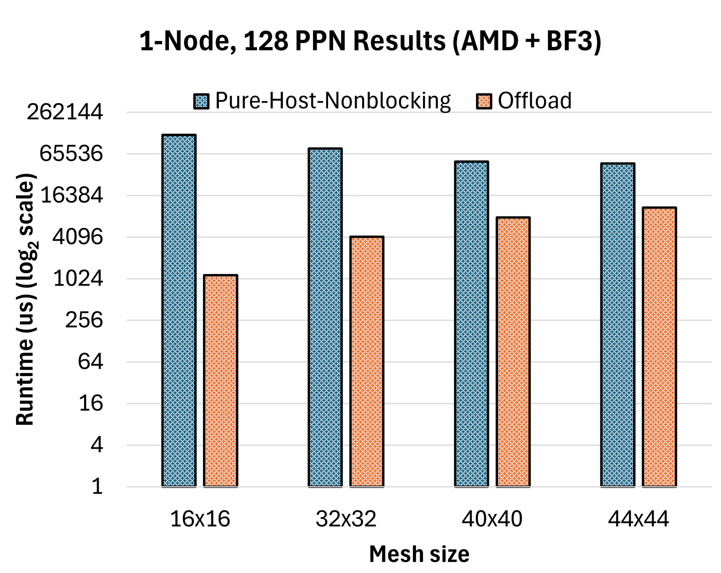
Compute Offload (Blocking Kernel)



- Get-Compute-Update lends itself nicely to "naïve" compute offload (utilizing BF-2 cores)
- Dense communication → lack of progress resources available on host = perfect use for DPUs (Smaller scale and PPN results in some benefits, but not as much)
- Up to 91% improvement with BF-2's for both compute and communication offload

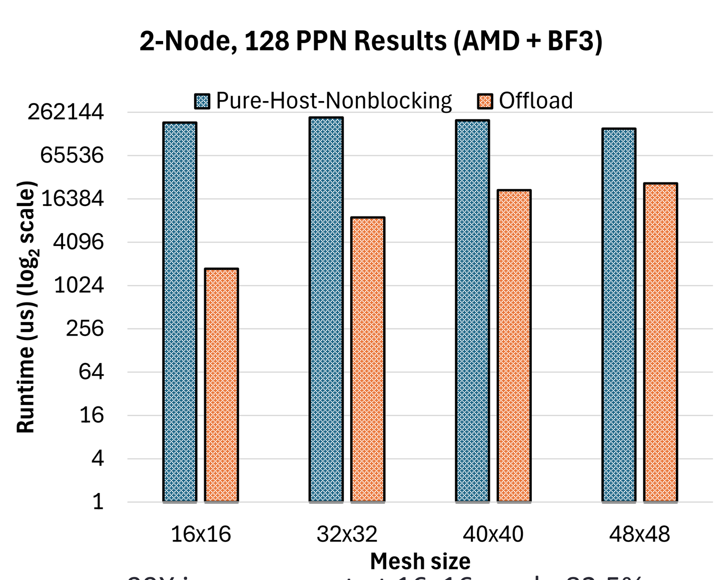
Performance on System 2 (Comm Offload)

1-node, 128-PPN Results (Nonblocking Kernel)



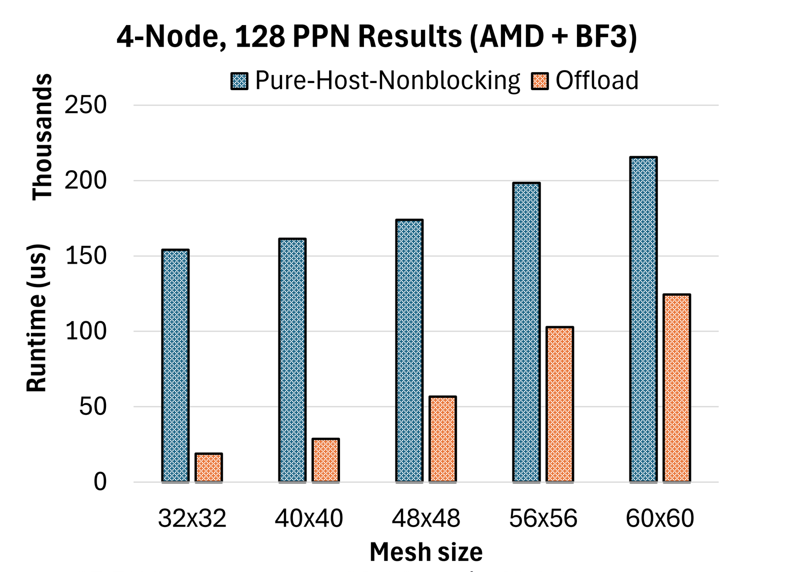
99X improvement at 16x16 mesh, 77% improvement at 44x44 mesh

2-node, 128-PPN Results (Nonblocking Kernel)



99X improvement at 16x16 mesh, 82.5% improvement at 44x44 mesh

4-node, 128-PPN Results (Nonblocking Kernel)



87.7% improvement at 32x32 mesh, 42.2% improvement at 60x60 mesh (16x16 mesh could not be run at larger scales, hence lack of log-scale y-axis)

References

1. BSPMM Kernel, R. Zambre and S. Bhattacharya, https://github.com/rzambre/bspmm
2. NVIDIA, "NVIDIA BlueField Networking Platform" https://www.nvidia.com/en-us/networking/products/data-processing-unit/
3. B. Michalowicz, K. K. Suresh, H. Subramoni, M. Abduljabbar, D. K. Panda and S. Poole, "Effective and Efficient Offloading Designs for One-Sided Communication to SmartNICs," 2024 IEEE 31st International Conference on High Performance Computing, Data, and Analytics (HiPC), Bangalore, India, 2024, pp. 23-33, doi: 10.1109/HiPC62374.2024.00012.

Acknowledgements

1. HPCA-AI Advisory Council
2. LANL SOW #19537, NSF Grants #231927 and #2007991