# DiOMP-Offloading: Portable OpenMP Offloading For Distributed Heterogeneous Systems

Baodi Shan[1], Mauricio Araya-Polo[2], Barbara Chapman[1]
[1]Stony Brook University, [2]TotalEnergies EP Research & Technology US

Stony Brook University · iACS Institute for Advanced Computational Science · TotalEnergies · SC25 St. Louis, MO hpc ignites.

## Abstract

As heterogeneous supercomputing becomes mainstream, traditional hybrid models such as MPI+OpenMP increasingly struggle to coordinate and manage GPU memory while maintaining portable performance.

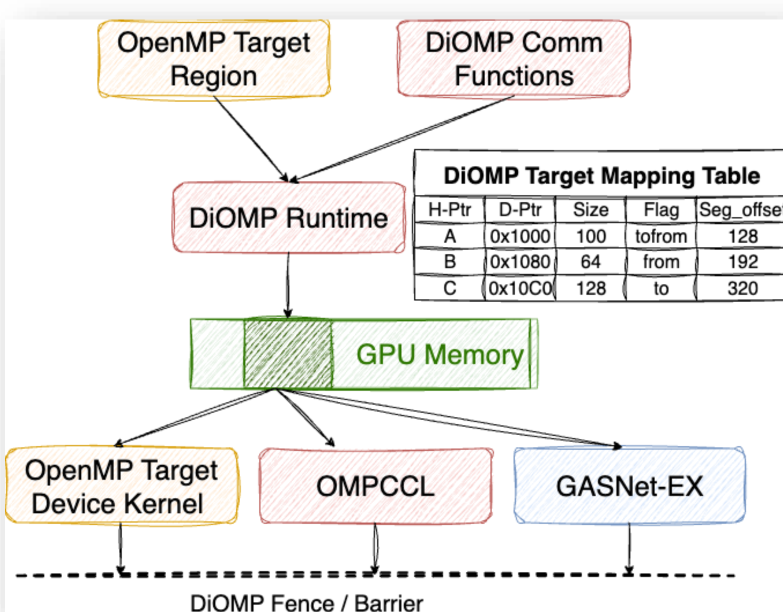
This work introduces **DiOMP-Offloading**, a framework that unifies OpenMP target offloading with a PGAS model. Built atop LLVM/OpenMP and using GASNet-EX as the communication layer, DiOMP-Offloading centrally manages global memory regions, providing a globally addressable space for remote put/get operations. It integrates OMPCCL, a portable device-side collective layer that enables the use of vendor collective backends by reconciling allocation life-cycles and address translation. Instead of relying on separate MPI+OpenMP, DiOMP-Offloading improves scalability and programmability by abstracting away replicated device-memory and communication management logic. Demonstrations on large-scale platforms show that DiOMP-Offloading delivers better performance in micro-benchmarks and applications under a single PGAS + OpenMP offloading model. These results indicate that DiOMP-Offloading can contribute to a more portable, scalable, and efficient path forward for heterogeneous computing.

## Implementation of DiOMP-Offloading

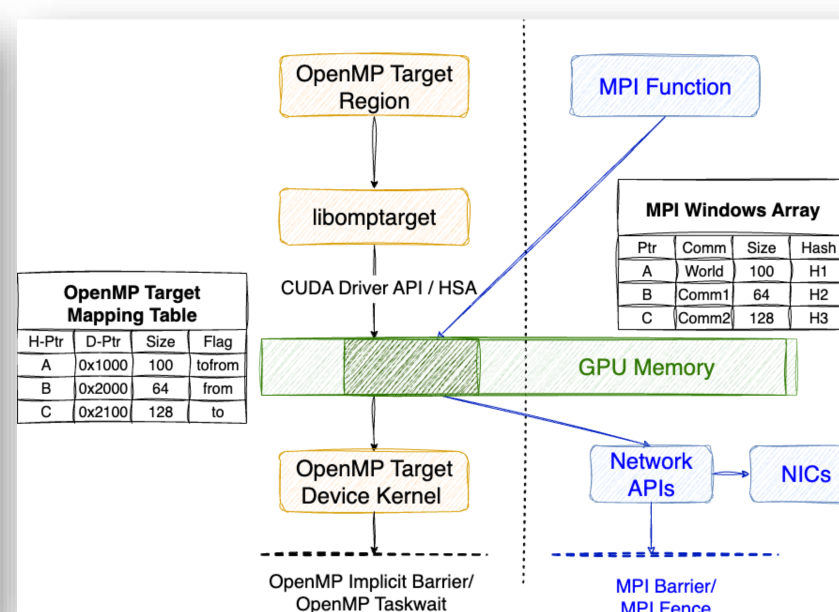
Below we compare **DiOMP-Offloading** (left) with **OpenMP+MPI workflow** (right). In the traditional model, OpenMP and MPI manage GPU memory **independently**, each with its own metadata, registration, and synchronization, which causes redundancy and inconsistency. DiOMP-Offloading extends LLVM/OpenMP's libomptarget, **intercepts device allocations**, and redirects them into a **global PGAS segment** managed by GASNet-EX. All memory regions become globally visible and jointly managed by OpenMP offloading, point-to-point transfers, and OMPCCL collectives. This unified runtime ensures consistent lifecycles, eliminates duplicated metadata, and tightly couples memory, communication, and computation.

**Advantages:**

- **Unified runtime**: one control plane for memory & communication
- **Integrated collectives**: OMPCCL inside OpenMP target regions
- **Scalable & portable**: efficient across diverse GPU platforms




DiOMP-Offloading Workflow




MPI + OpenMP Offloading Workflow

## Programmability Comparison

### "Get" Data from Remote Node on DiOMP-Offloading, MPI One-sided, MPI Two-sided, and OpenSHMEM

DiOMP Offloading: Get or Memcpy + Fence

```
// 1. Directly issue a non-blocking get operation
ompx_get( … );
// or memcpy
omp_target_memcpy_async( … );

// 2. Synchronize to ensure the operation completes
ompx_fence();
```

MPI Two-Sided: Request + Pragma + Isend/Rect + Wait

```
if (my_rank == sender_rank) {
    // Asynchronous send returns a request handle
    MPI_Request req;

    #pragma omp target use_device_ptr(dst_ptr)
    MPI_Isend( … , &req);

    MPI_Wait(&req, MPI_STATUS_IGNORE);
} else if (my_rank == receiver_rank) {
    MPI_Request req;

    #pragma omp target use_device_ptr(dst_ptr)
    MPI_Irecv( … , &req);

    MPI_Wait(&req, MPI_STATUS_IGNORE);
}
```

MPI One-Sided: Win Create + Fence + Pragma + Get + Fence

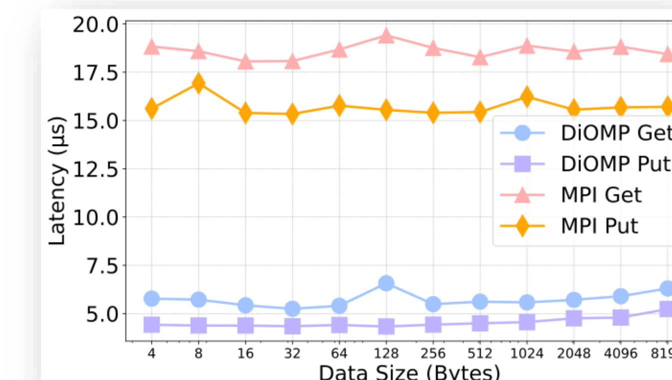
```
// 1. Setup Window: A window must be created.
MPI_Win_create( … , &win);

// 2. Communication & Synchronization: Requires Fence/Lock.
MPI_Win_fence(0, win);

// 3. Ensure MPI can access the device pointer.
#pragma omp target use_device_ptr(dst_ptr)
MPI_Get( … , win);

MPI_Win_fence(0, win); // End epoch

// 4. Cleanup
MPI_Win_free(&win);
```

OpenSHMEM: Malloc + Update(From) + Get + Sync + Update(To)

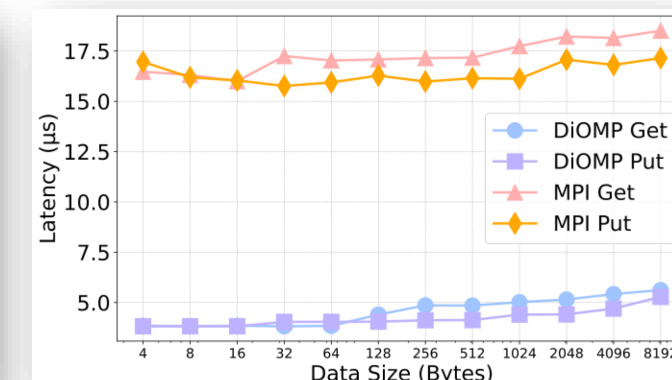
```
// 1. Allocate symmetric memory using a special function
src_ptr = shmem_malloc(size);

// 2. Copy data from device to host
#pragma omp target data update(from: … )

// 3. Communication: Performed between host memories
shmem_get( … );

// 4. Synchronize SHMEM operation
shmem_quiet();

// 5. Copy data from host back to device
#pragma omp target data update(to: … )
```

## Evaluation

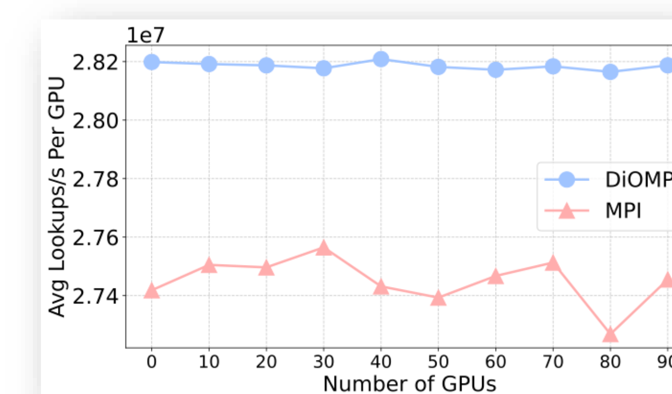
We tested DiOMP-Offloading and Cray MPICH on **NVIDIA A100** (NERSC's Perlmutter) and **AMD MI250X** (LLNL LC's Tioga) GPUs. Shown here are selected results: **P2P latency, XSBench, matrix multiplication,** and **Minimod** (a stencil code from *TotalEnergies*).



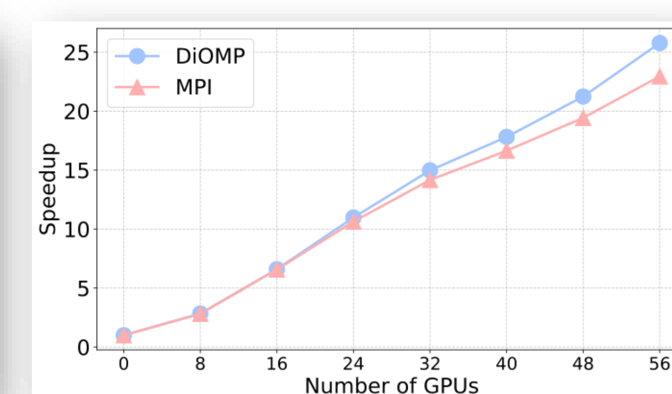

Latency on Perlmutter (Lower is better)



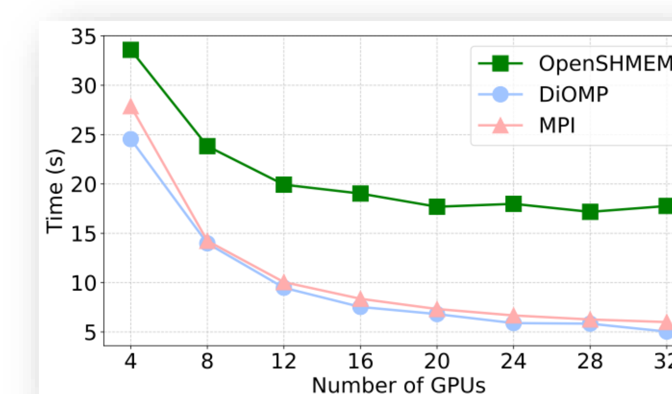

Latency on Tioga (Lower is better)



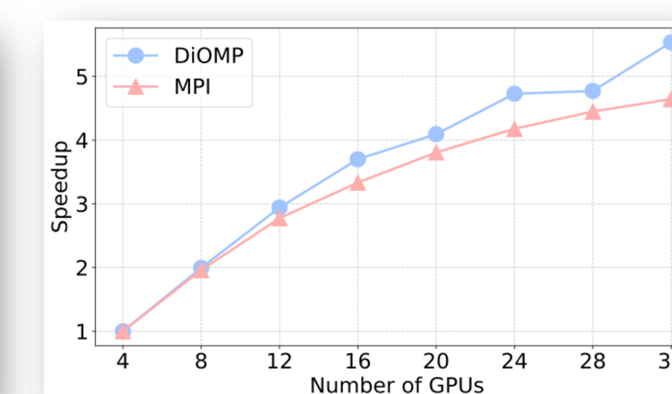

XSBench on Perlmutter (Higher is better)




Matrix Multiplication Speedup on Tioga (Higher is better)




Minimod Execution Time on Perlmutter (Lower is better)




Minimod Speedup on Tioga (Higher is better)

**Acknowledgement:** We would like to thank TotalEnergies E&P Research and Technologies US for their support of this work. Our gratitude also extends to Alice Koniges from the University of Hawaii for providing access to the NERSC Perlmutter system. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.