HAND: A Hybrid Approach to Accelerate Non-contiguous Data Movement using MPI Datatypes on GPU Clusters

Rong Shi Xiaoyi Lu Sreeram Potluri Khaled Hamidouche Jie Zhang

Dhabaleswar K. Panda

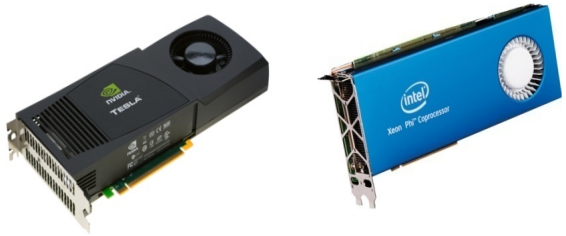
Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University

Outline

- Introduction
- Motivation & Problem Statement
- Proposed Design for HAND
- Performance Evaluation
- Conclusion and Future Work

Drivers of Heterogeneous HPC Cluster

Multi-core Processors



Accelerators / Coprocessors
high compute density, high performance/watt
>1 TFlop DP on a chip

- Multi-core processors are ubiquitous
- Accelerators/Coprocessors are becoming common in high-end systems
- Pushing the envelope for heterogeneous computing

Titan (2)

Stampede (7)

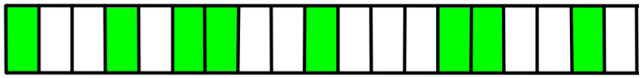
Oakley (OSC)

Blue Waters (NCSA)

MPI Derived Datatypes

MPI_Type_**vector**(int count, int blen, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)
cnt=3, blen=2, stride=3

MPI_Type_**indexed**(int count, int *arr_of_blen, int *arr_of_disps, oldtype, *newtype)
blen={1,1,2,1,2,1}



MPI_Type_create_**indexed_block**(int count, int blen, int *arr_of_disps, oldtype, *newtype)
cnt=5, blen=2, disps={0,5,9,13,18}

MPI_Type_create_**subarray**(int ndims, int arr_of_sz[], int arr_of_subsz[], int arr_of_starts[], int order, oldtype, *newtype)

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

Applications using Non-contiguous datatypes

Application	Testname	Datatype	Access Pattern
Fluid Dynamics	NAS_MG_y	vectors	3D face exchange in y direction
Geophysical Science	SPECFEM3D_oc SPECFEM3D_cm	indexed struct on indexed	unstructured exchange of data for diverse earth layers
Atmospheric Science	WRF_y_sa	struct on subarray	struct of 2D/3D/4D face exchanges in y direction
Quantum Chromodynamics	MILC_su3_zd	nested vectors	4D face exchange in z direction
Metereological Science	COSMO	3D subarray	halo-update
Stencil Code	Jacobi	2D subarray	5 point 2D Stencil

- Data Movement Strategy

CUDA memory copy APIs: poor performance (e.g. sparse data distribution)

Hand-coded CUDA Kernels: low programming efficiency and tuning

Solution: MPI runtime that hides complexity and takes advanced features

Comparison with State of the Art

Design	Datatypes	Enhancements	1	2	3	This Paper
Direct Transfer	all	active scheduling	N	N	N	Y
Targeted Kernels	vector	2D thread block	N	Y	N	Y
	subarray	1/2/3/4D targeted	Y	N	N	Y
	indexed_block	new targeted	N	N	N	Y
	the above three	adaptive tuning	N	N	N	Y
Transformation	others		N	Y	N	Y
		automatic switch	N	N	N	Y
		adaptive tuning	N	N	N	Y
Host Bypass	others		N	N	Y	Y
		automatic kernel selection	N	N	N	Y
		adaptive tuning	N	N	N	Y
Hybrid	all	automatic scheme selection	N	N	N	Y

1. H. Wang, S. Potluri, M. Luo, A. Singh, X. Ouyang, S. Sur, and D. Panda, Optimized non-contiguous MPI Datatype Communication for GPU Clusters, Cluster'11

2. H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda, GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation, TPDS'13

3. J. Jenkins, J. Dinan, P. Balaji, T. Peterka, N. F. Samatova, and R. Thakur, Processing MPI Derived Datatypes on Noncontiguous GPU-Resident Data, TPDS'13

Outline

- Introduction
- Motivation & Problem Statement
- Proposed Design for HAND
- Performance Evaluation
- Conclusion and Future Work

Motivation

- Current limitation
  1. Targeted kernels ignore the 1D/2D/4D subarray and indexed_block
  2. Transformation scheme incurs kernel launch overhead for irregular distribution
  3. Host Bypass incurs unnecessary overhead for dense/regular distribution

- Goal

  Design and implement a hybrid MPI datatype processing framework to optimize the packing/unpacking transparently and provides good programming efficiency

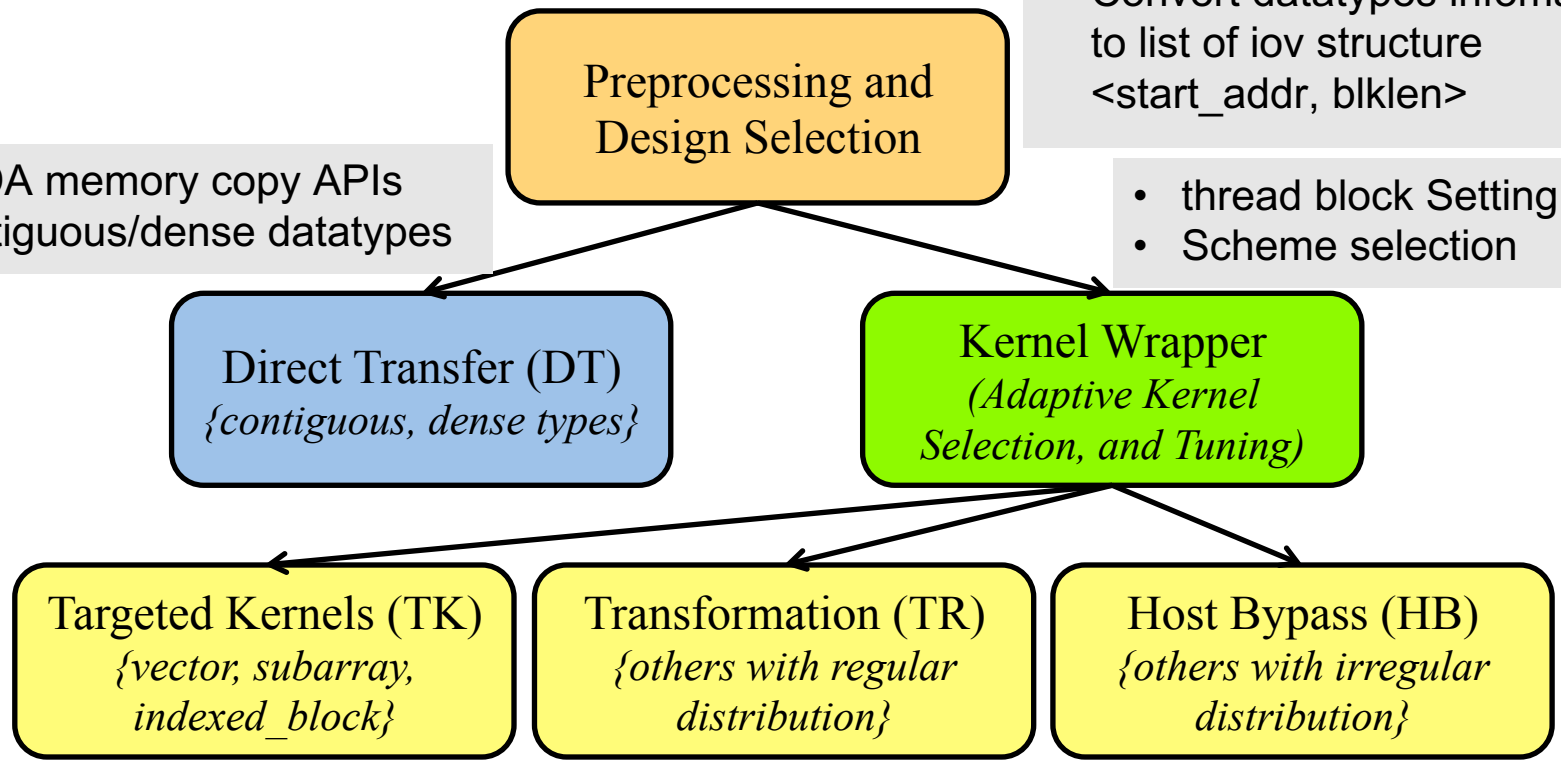
Problem Statement

- Can we optimize the existing schemes?
  e.g. targeted kernels
- Can our design integrate the existing schemes into a framework and transparently select the optimal one?
- Can the framework improve the performance of micro benchmarks and applications?

Outline

- Introduction
- Motivation & Problem Statement
- Proposed Design for HAND
- Performance Evaluation
- Conclusion and Future Work

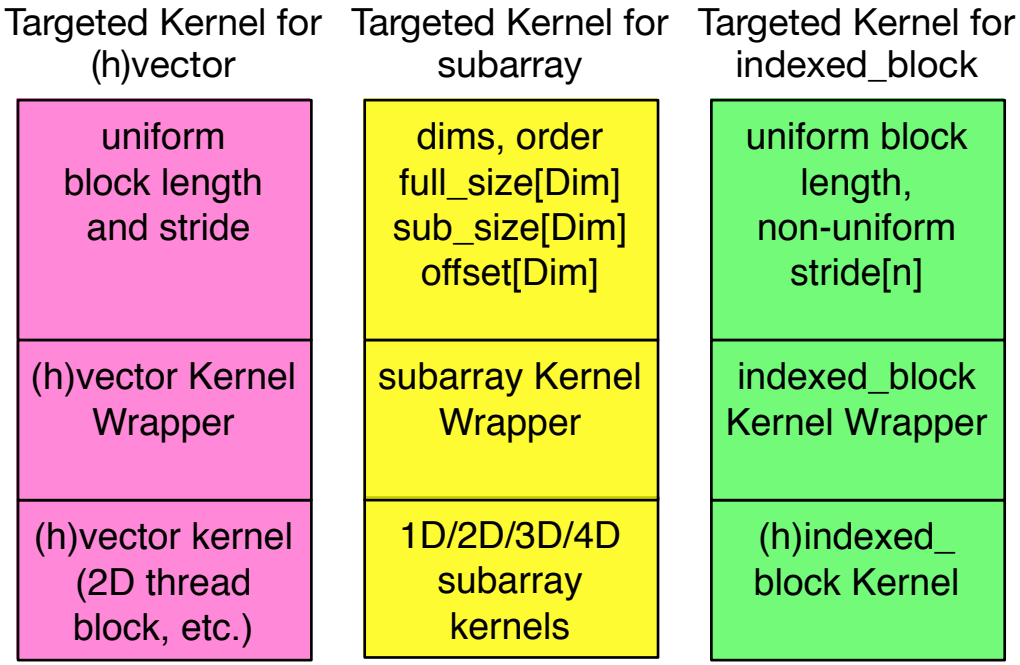
Overview of HAND Framework



- Convert datatypes infomation to list of iov structure <start_addr, blklen>
- CUDA memory copy APIs
- Contiguous/dense datatypes
- thread block Setting
- Scheme selection
- Convert to list of vector arrays
- Use targeted vector kernels
- Offload iov structure and memory copy to GPU kernels

- **Within MVAPICH2 Library (MPI communication routines)**

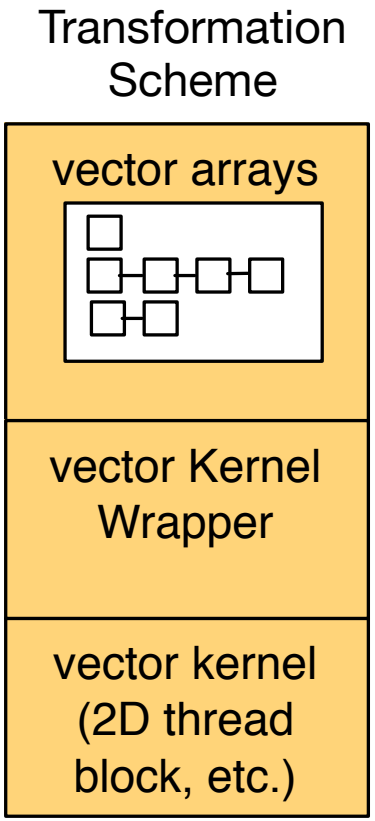
Optimization of Targeted Kernels



- (h)vector: Adaptive 2D thread block based on shapes of datatypes
- subarray: extended 1D/2D/3D/4D support with MPI_C/FORTRAN_ORDER
- (h)indexed_block: simplified Host Bypass generic kernel

Optimization of Transformation Scheme

- Irregular datatypes (e.g. indexed)
  - iov stucture → list of vector arrays
  - Use targeted vector kernel for each vector array
- Automatic Switch between transformation and Host Bypass schemes
  - datatypes with regular distribution → TR
  - arbitrary datatypes → HB
    (threshold of number of vector arrays)
- Benefit from the optimized targeted kernel (vector)



Design of Host Bypass Scheme

- Preprocessing Routine
  - iov structure (startup overhead (100/200 us))
    packiov: pinned memory, no additional copy
    copyiov: general device memory, higher device access speed

- Kernel Wrapper
  - 2D Thread block Mapping (generic to other schemes)
    1st tid block dim (tidx) to blocklen, 2nd dim (tidy) to count,
    successive memory access and better data locality
  - Non-power-of-two counts
    fast bit operations, saved threads → dim of blocklen

- Packing/Unpacking Generic kernels
  - Prefix-sum (scan): sequential (O(N)) v.s. parallel (O(logN))
  - 2D thread-level Memory Copy: each tid → one/more datatype primitives

Design of Host Bypass Scheme

- Packing kernels

Algorithm 1: GPU Packing Kernel with Parallel Scan

```
Input: dstbuf, iov_structure, count
Output: desbuf
__shared__size_t prefixsum[1024];
i ← threadIdx.x;
j ← threadIdx.y;
offset ← 1;
// exclusive parallel prefix-sum calculation
prefixsum[i * blockDim.y + j] ←
iov_structure[j].iov_len;
for int k = (blockDim.y) >> 1; k > 0; k >>= 1 do
    // build sum in place up the tree
    offset <<= 1
end
for int k = 1; k < (blockDim.y); k <<= 1 do
    offset >>= 1
    __syncthreads();
    // traverse down tree and build scan
end
__syncthreads();
if j < count then
    iter ← iov_structure[j].srclen/blockDim.x;
    for k = 0; k <= iter; k++ do
        if i < iov_structure[j].srclen then
            dstbuf[prefixsum[i*blockDim.y+j]+i]
            ← *((iov_structure + j).iov_base + i);
        end
        i ← i + blockDim.x;
    end
end
```

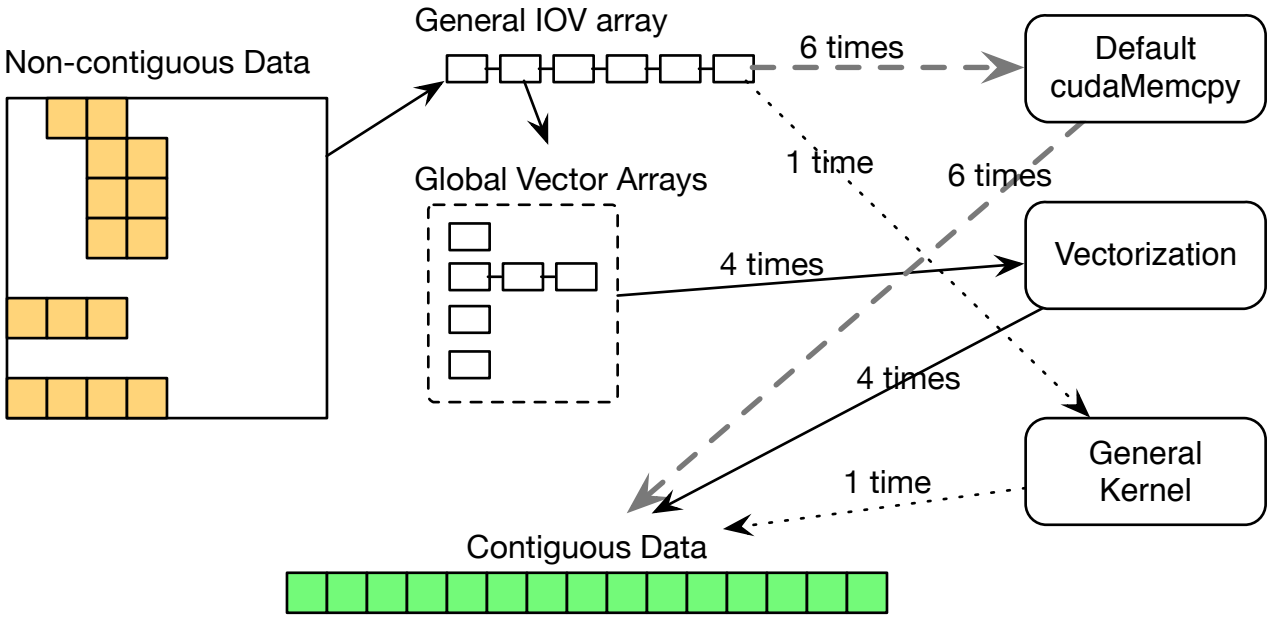
Algorithm 2: GPU Packing Kernel with Sequential Scan

```
Input: dstbuf, iov_structure, count
Output: desbuf
i ← threadIdx.x;
j ← threadIdx.y;
// Sequential prefixsum calculation
if j < count then
    // initialize block_length,
    // source and destination buffer displacements
    for k = 0; k < j; k++ do
        // calculate prefixsum for each subblock
    end
    iter ← iov_structure[j].srclen/blockDim.x;
    for ; k <= iter; k++ do
        if i < iov_structure[j].srclen then
            dstbuf[prefixsum + i]
            ← *((iov_structure + j).iov_base + i);
        end
        i ← i + blockDim.x;
    end
end
```

$$T\_par = P * log(count) + iter + P\_overhead$$

$$T\_seq = S * count + iter + S\_overhead$$

Case Study of DT/TR/HB Schemes



indexed datatype (cnt=6, blen={2,2,2,2,3,4}, disps={1,10,18,26,40,56})
DT: 6 calls to cudaMemcpy
TR: 4 calls to vector kernel
HB: 1 calls to generic kernel

Outline

- Introduction
- Motivation & Problem Statement
- Proposed Design for HAND
- Performance Evaluation
- Conclusion and Future Work

Experimental Setup

- Experiment Environment

Cluster Specs	A	Oakley
CPU Processor	Intel Xeon E5630	Intel Xeon X5650
CPU Clock	2.53GHz	2.66GHz
Node Type	two 4-core sockets	two 6-core sockets
CPU Memory	12 GB	46 GB
GPU Processor	NVIDIA Tesla C2050	NVIDIA Tesla M2070
GPUs per Node	1	2
GPU Memory	3 GB	6 GB
Compilers	gcc 4.4.7	gcc 4.4.7
MPI Library	MVAPICH2 2.0b	MVAPICH2 2.0b
Interconnect	Mellanox IB QDR	Mellanox IB QDR

- MPI Library: MVAPICH2

High Performance open-source MPI Library for InfiniBand, 10Gig/iWARP, and RDMA over Converged Enhanced Ethernet (RoCE)

Used by more than 2,200 organizations (HPC Centers, Industry and Universities) in 73 countries

http://mvapich.cse.ohio-state.edu/

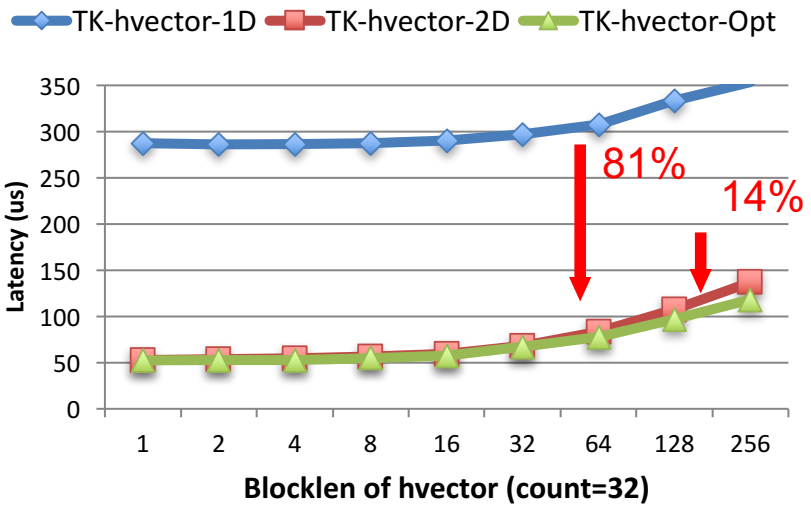
Performance of hvector and 4D subarray

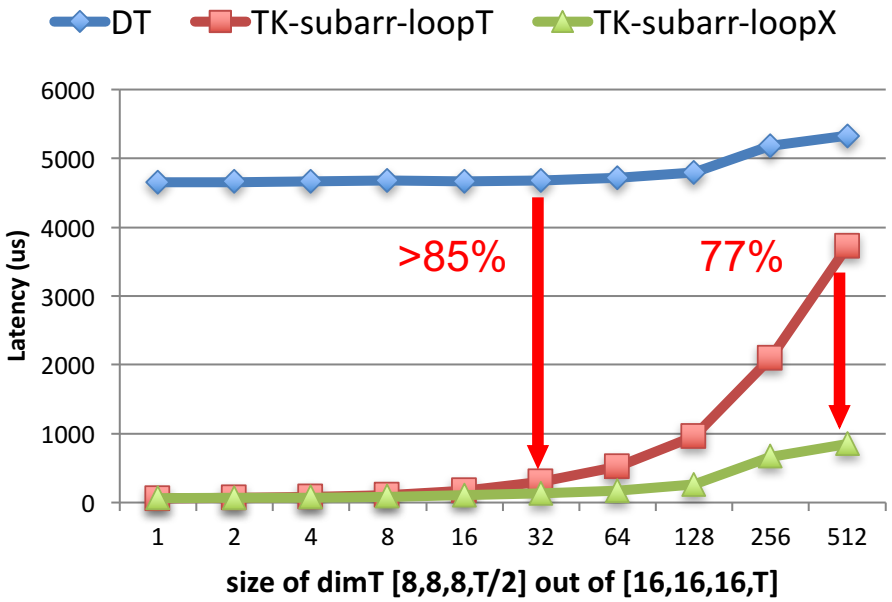
Ping-Pong latency using 2 GPU nodes on Cluster A

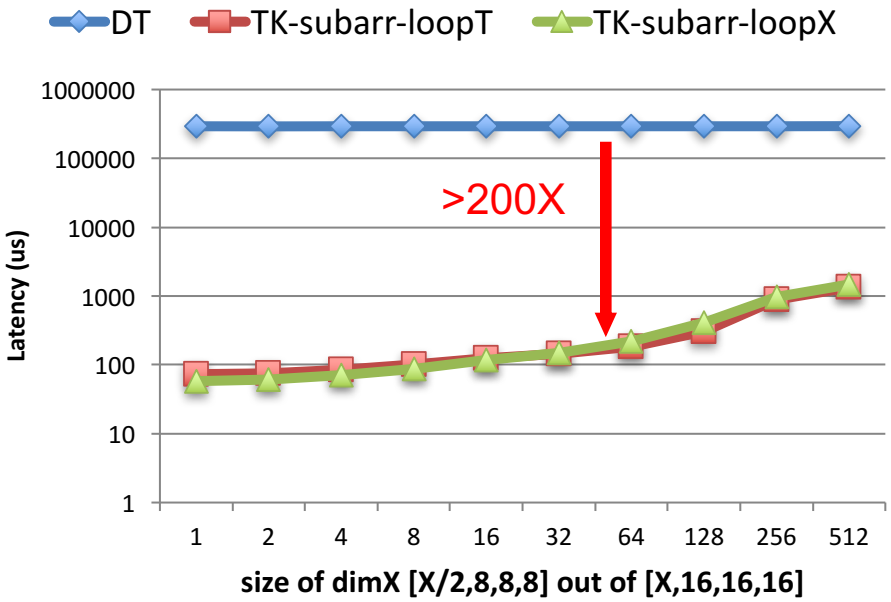
DT/TK: direct transfer / targeted kernel

1D/2D/Opt: thread block dim

LoopT/LoopX: [X, Y, Z, T] subarray, loop on innermost T dim / outermost X dim

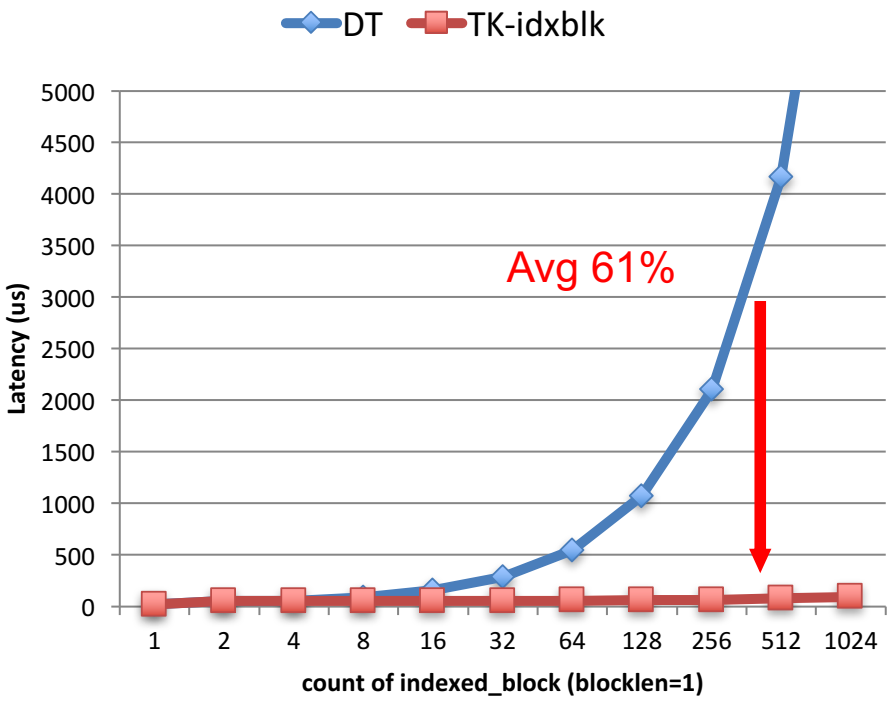


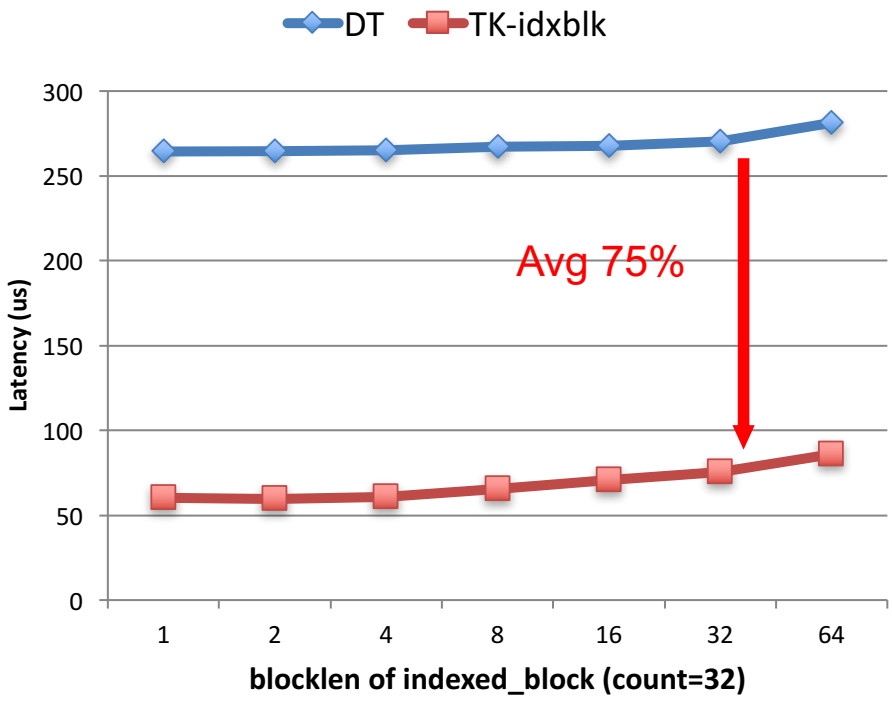




Performance of indexed_block

DT/TK: direct transfer / targeted kernel



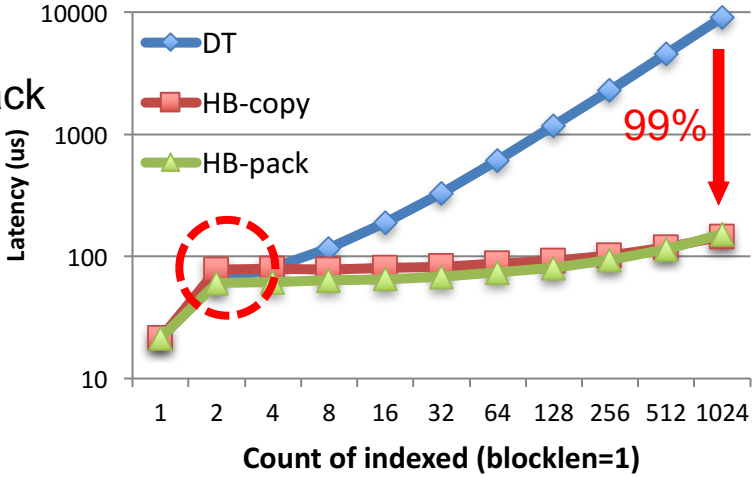


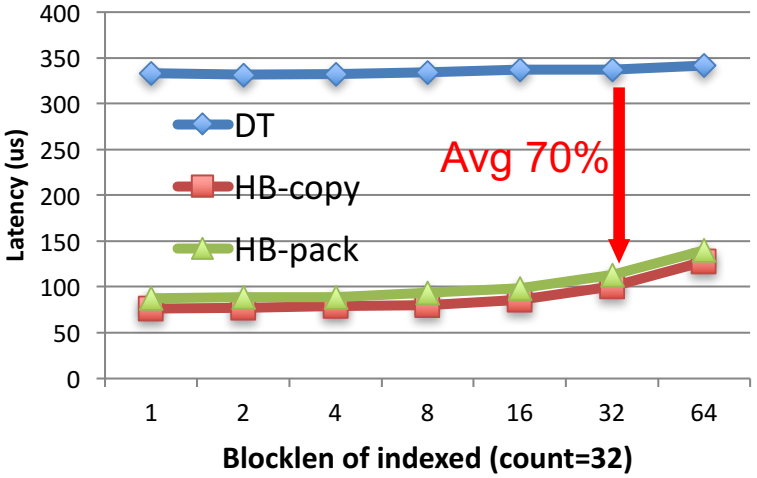
Performance of Host Bypass Scheme

DT/HB: direct transfer / Host Bypass Scheme on indexed datatype

IOV Passing

HB-Copy/HB-Pack

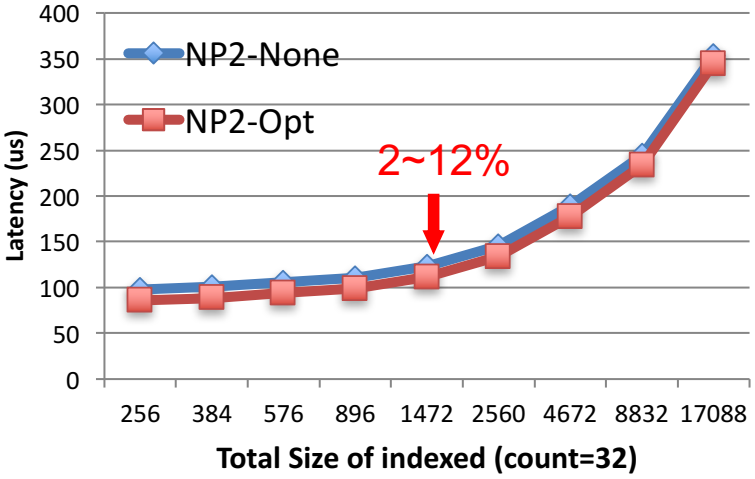


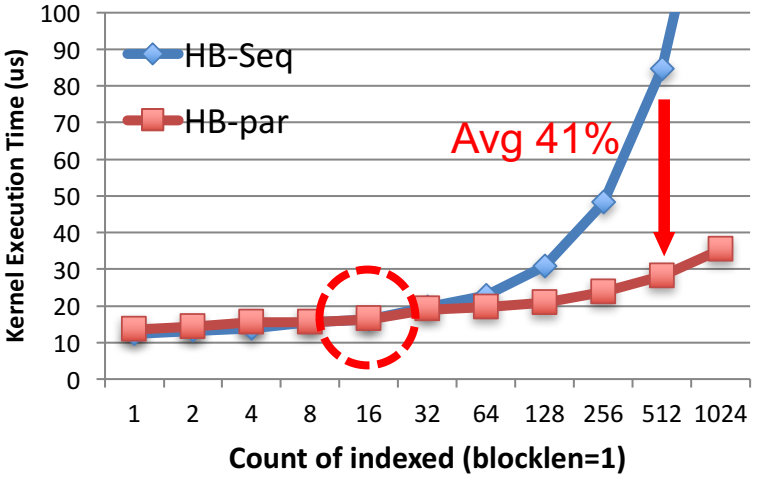


NP2-None/NP2-Opt

HB-Seq (scan)

HB-Par (scan)





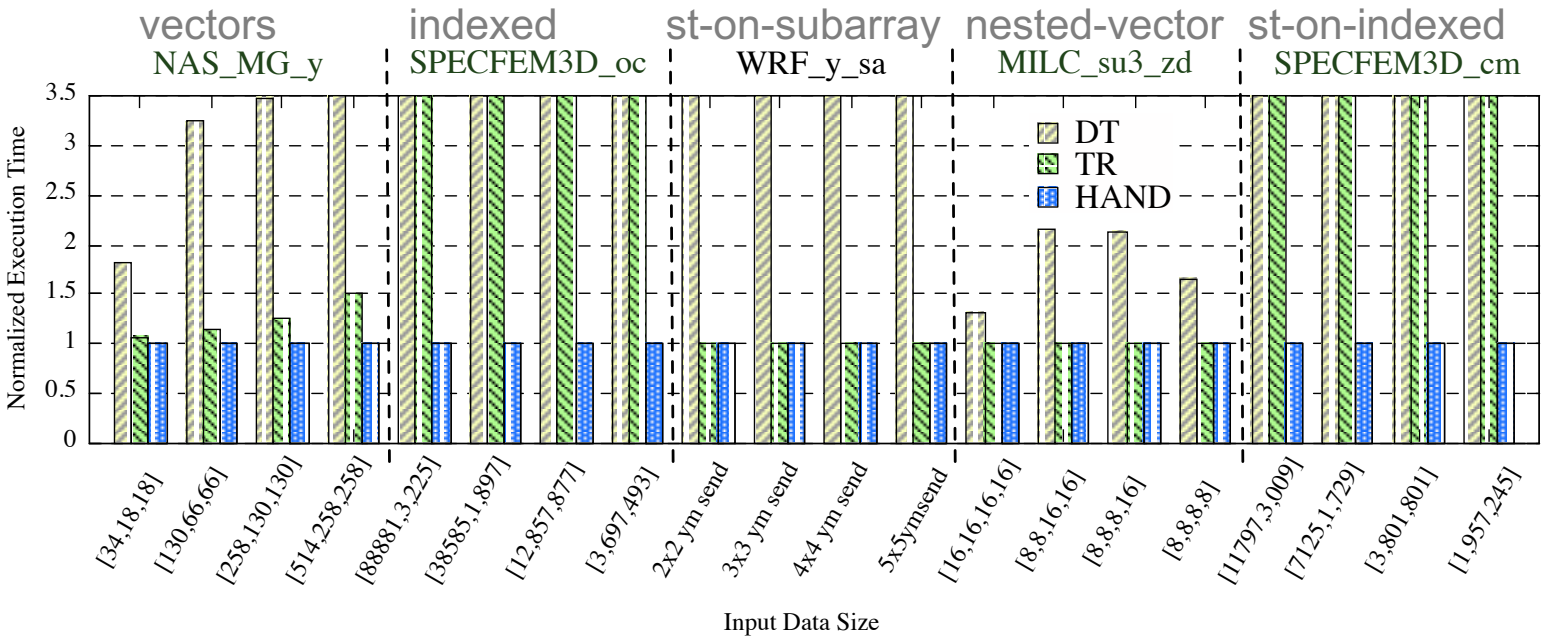
Performance of DDTBench

DDTBench: micro-application that usage of datatypes in real applications

Modification: allocating data on device memory

DT/TR/HAND: Direct Transfer / Transformation Scheme / HAND framework

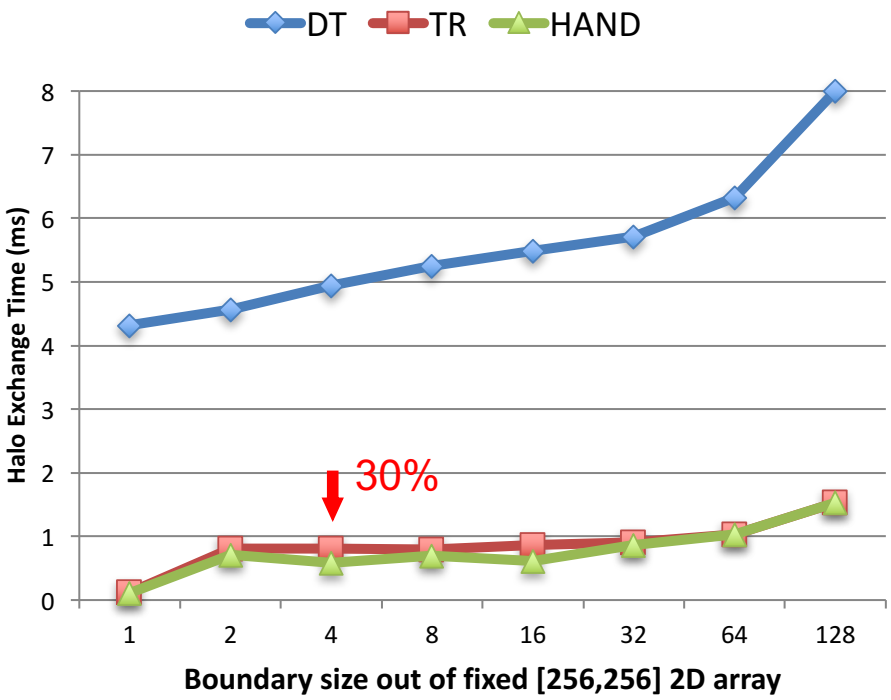
HAND gains an average of 66%, 99%, 97%, 43% and 97% compared to DT, 18%, 0, 99%, 0 and 97% compared to TR

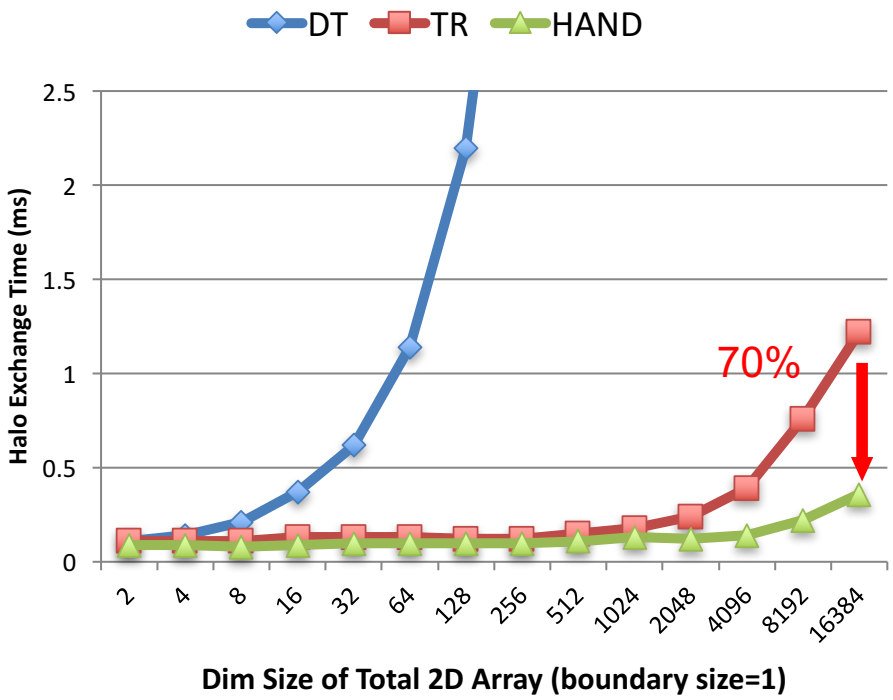


T. Schneider, R. Gerstenberger, and T. Hoefler, Micro-Applications for Communication Data Access Patterns and MPI Datatypes, EuroMPI'12

Performance of Stencil2D (2D subarray)

Stencil2D communication kernel on 4 GPUs using MPI_Isend/Irecv

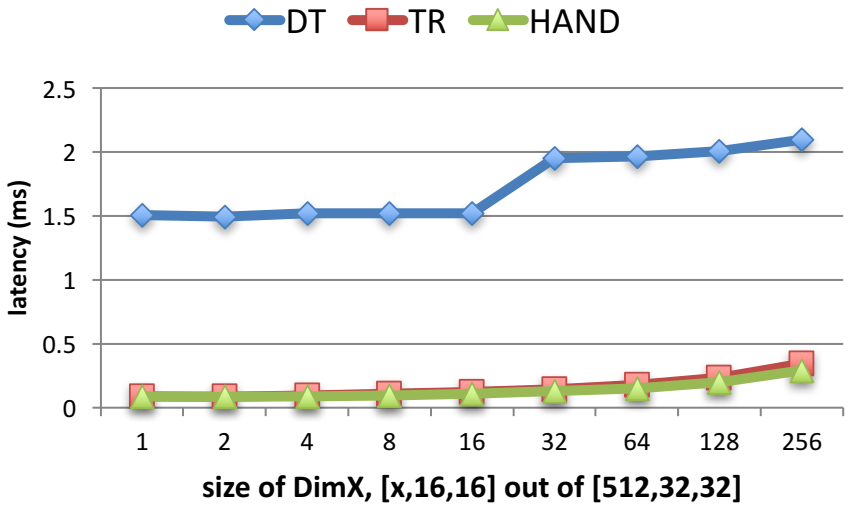


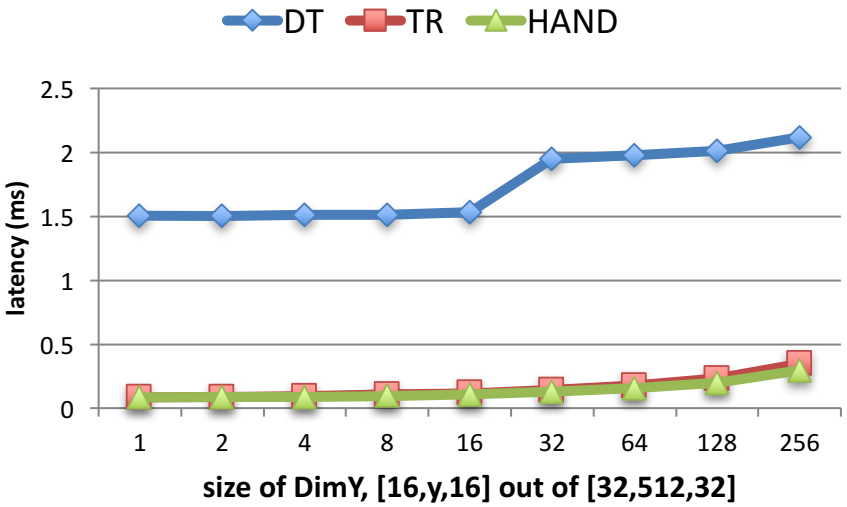


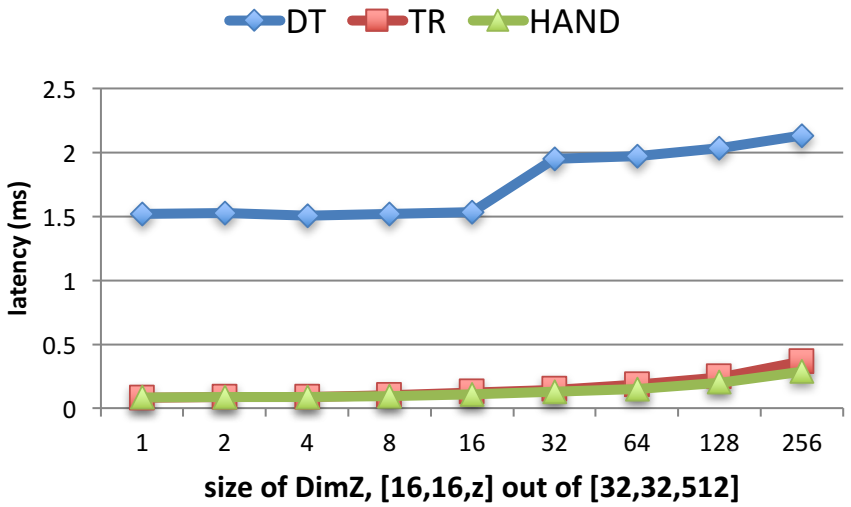
Performance of Stencil3D (3D subarray)

Stencil3D communication kernel on 2 GPUs with various X, Y, Z dimensions using MPI_Isend/Irecv

HAND gains up to 15%, 15% and 22% compared to TR, above 86% compared to DT







M. Bianco and U. Varetto, A Generic Library for Stencil Computations, CoRR, 2012

Performance of N-Body Application

N-Body particles simulation implemented with MPI+GPGPU programming model

Steps of one iteration:

1) each GPU node gets a subset of particles and uses the GPU kernels to calculate the forces for local particles;
2) All GPU nodes exchange data with others by MPI Allgather communication;
3) each GPU node updates positions and velocities of local particles.

Extend the definition of particle structure to simple particle and complex particle, and define struct and indexed (force_x/y/z and charge)

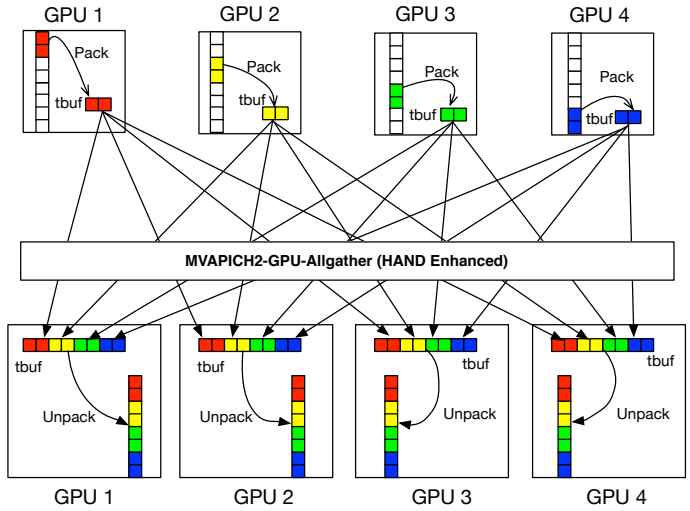
```
typedef struct {
double position_x, position_y, position_z ;
double velocity_x, velocity_y, velocity_z;
double force_x, force_y, force_z;
double mass;
float properties[20]; //complex struct only
int charge;
} Particle;
```

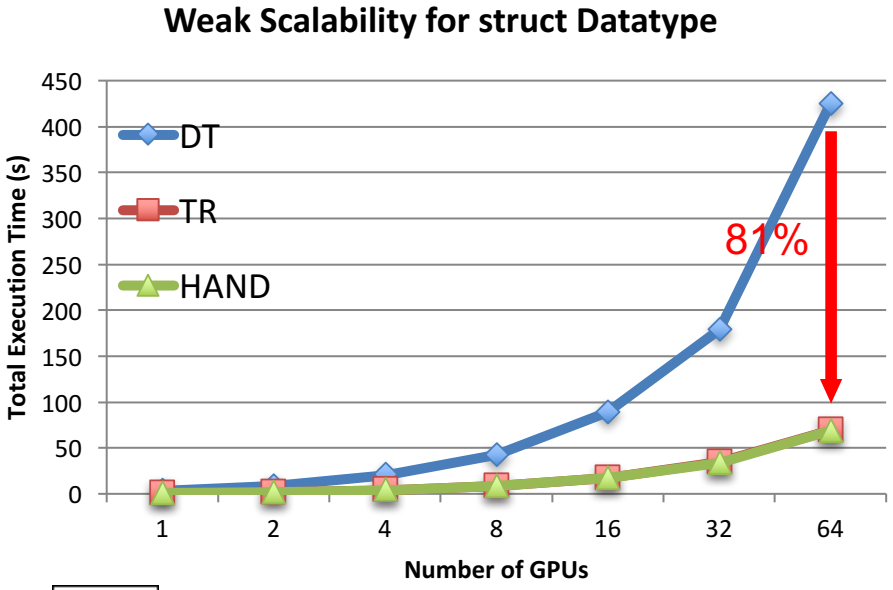
BWSIP, http://www.shodor.org/petascale/materials/hybrid/code

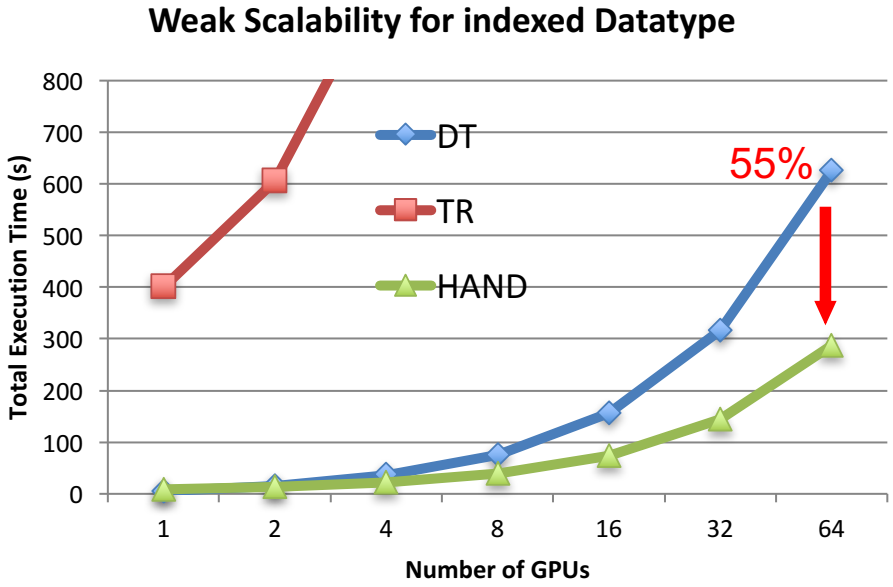
Performance of N-Body Application

N-body simulation with 200 iterations
Fixed 128k particles / GPU node

Left: struct datatype
Right: indexed datatype







Outline

- Introduction
- Motivation & Problem Statement
- Proposed Design for HAND
- Performance Evaluation
- Conclusion and Future Work

Conclusion

- Propose a novel HAND framework to efficiently pack/unpack non-contiguous datatypes on GPU clusters
- Seamlessly integrates all existing schemes into HAND framework and transparently select the optimal scheme based on the shape of datatypes
- Exhibits sustained performance scalability across different platforms with various GPU configurations
- Support for the new proposed designs is available in MVAPICH2-GDR 2.0 release

Future Work

- Can the HAND framework take advantage of the new MPI runtime features?

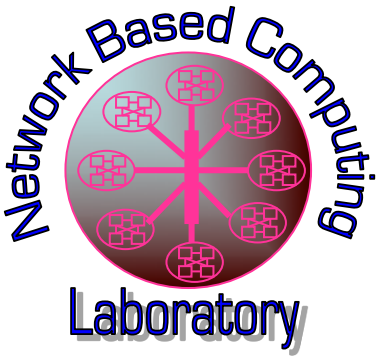
  e.g. GPUDirect RDMA

- How to incorporate the design with new GPU architecture?

  e.g. NVIDIA Kepler GPUs, AMD GPUs with OpenCL programming model

Thank You!

{shir, luxi, potluri, hamidouc, zhanjie, panda} @cse.ohio-state.edu



Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page

http://mvapich.cse.ohio-state.edu/