EFFICIENT BREADTH FIRST SEARCH ON MULTI-GPU SYSTEMS USING GPU-CENTRIC OPENSHMEM

Sreeram Potluri, *Anshuman Goswami – NVIDIA*

Manjunath Gorentla Venkata, Neena Imam - ORNL

NVIDIA

SCOPE OF THE WORK

Reliance on CPU for communication limits scaling on GPU clusters

NVSHMEM provides GPU kernel-side OpenSHMEM API

NVLink is a high-bandwidth memory-like fabric between GPUs

Effectiveness of NVSHMEM over NVLink with Breadth First Search

CPU-initiated MPI vs GPU-initiated SHMEM in a particular implementation of BFS

AGENDA

GPU Programming Models: MPI and SHMEM

NVSHMEM Overview

DGX and NVLink

MPI+CUDA implementation of BFS

SHMEM implementation of BFS

Performance Evaluation

Conclusion and Future Work

GPU CLUSTER PROGRAMMING

Offload model
- Compute on GPU
- Communication from CPU

Synchronization at boundaries

Overheads on GPU Clusters
- Offload latencies
- Synchronization overheads

Limits strong scaling

More CPU means more power

```
void 2dstencil (u, v, …)
{
  for (timestep = 0; …) {
    interior_compute_kernel <<<…>>> (…)
    pack_kernel <<<…>>> (…)
    cudaStreamSynchronize(…)
    MPI_Irecv(…)
    MPI_Isend(…)
    MPI_Waitall(…)
    unpack_kernel <<<…>>> (…)
    boundary_compute_kernel <<<…>>> (…)
    …
  }
}
```

GPU-CENTRIC COMMUNICATION

GPU capabilities

- Warp scheduling to hide latencies to global memory
- Implicit coalescing of loads/stores to achieve efficiency

CUDA helps to program to these

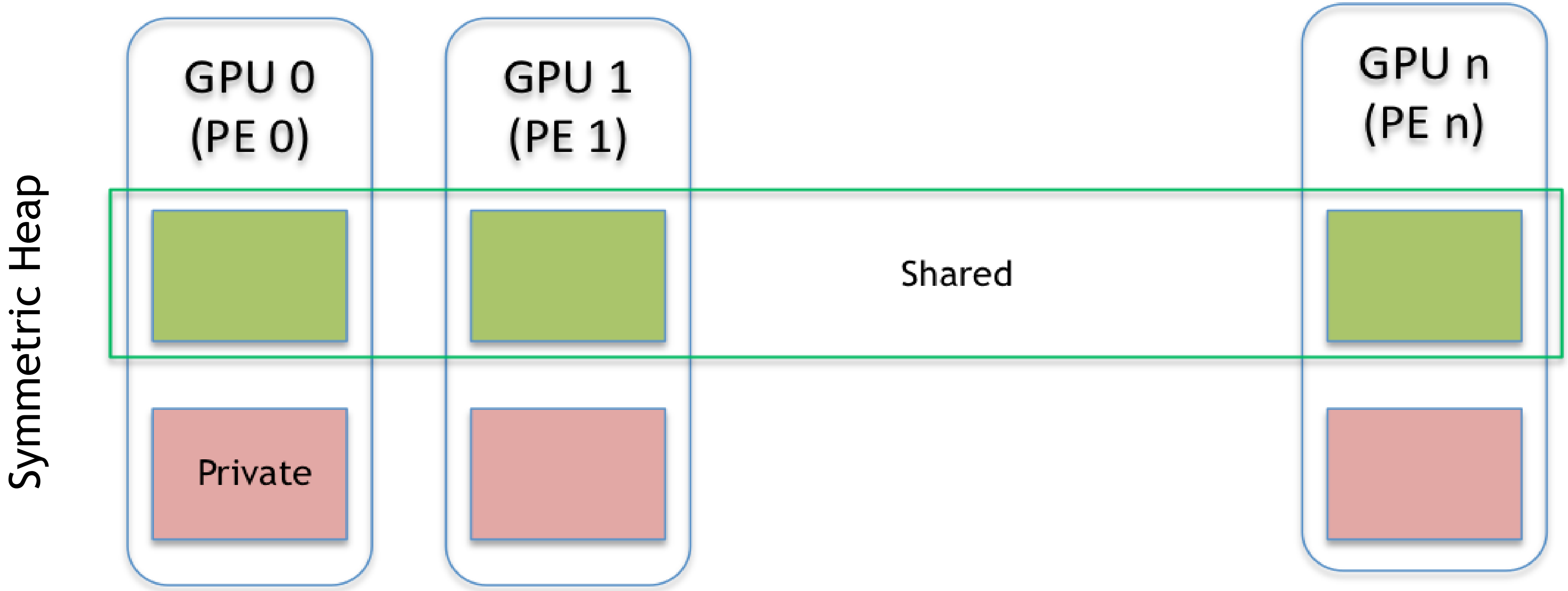
Should also benefit when accessing data over the network

Direct accesses to remote memory simplifies programming

Achieving efficiency while making it easier to program

Continuous fine-grained accesses smooths traffic over the network

GPU GLOBAL ADDRESS SPACE



COMMUNICATION FROM CUDA KERNELS

Long running CUDA kernels

Communication within parallelism

```
void 2dstencil (u, v, …)
{
    stencil_kernel <<<…>>> (…)
}
```

```
__global__ void 2dstencil (u, v, sync, …)
{
   for(timestep = 0; …) {
      if (i+1 > nx) {
         v[i+1] = shmem_float_g (v[1], rightpe);
      }
      if (i-1 < 1) {
         v[i-1] = shmem_float_g (v[nx], leftpe);
      }

      u[i] = (u[i] + (v[i+1] + v[i-1] . . .

      if (i < 2) {
         shmem_int_p (sync + i, 1, peers[i]);
         shmem_quiet();
         shmem_wait_until (sync + i, EQ, 1);
      }
      //intra-kernel sync
      …
   }
}
```

NVSHMEM

Implementation of OpenSHMEM Specification (1.3)

Symmetric heap on GPU memory

Implements multi-threading support as being discussed in PR #43

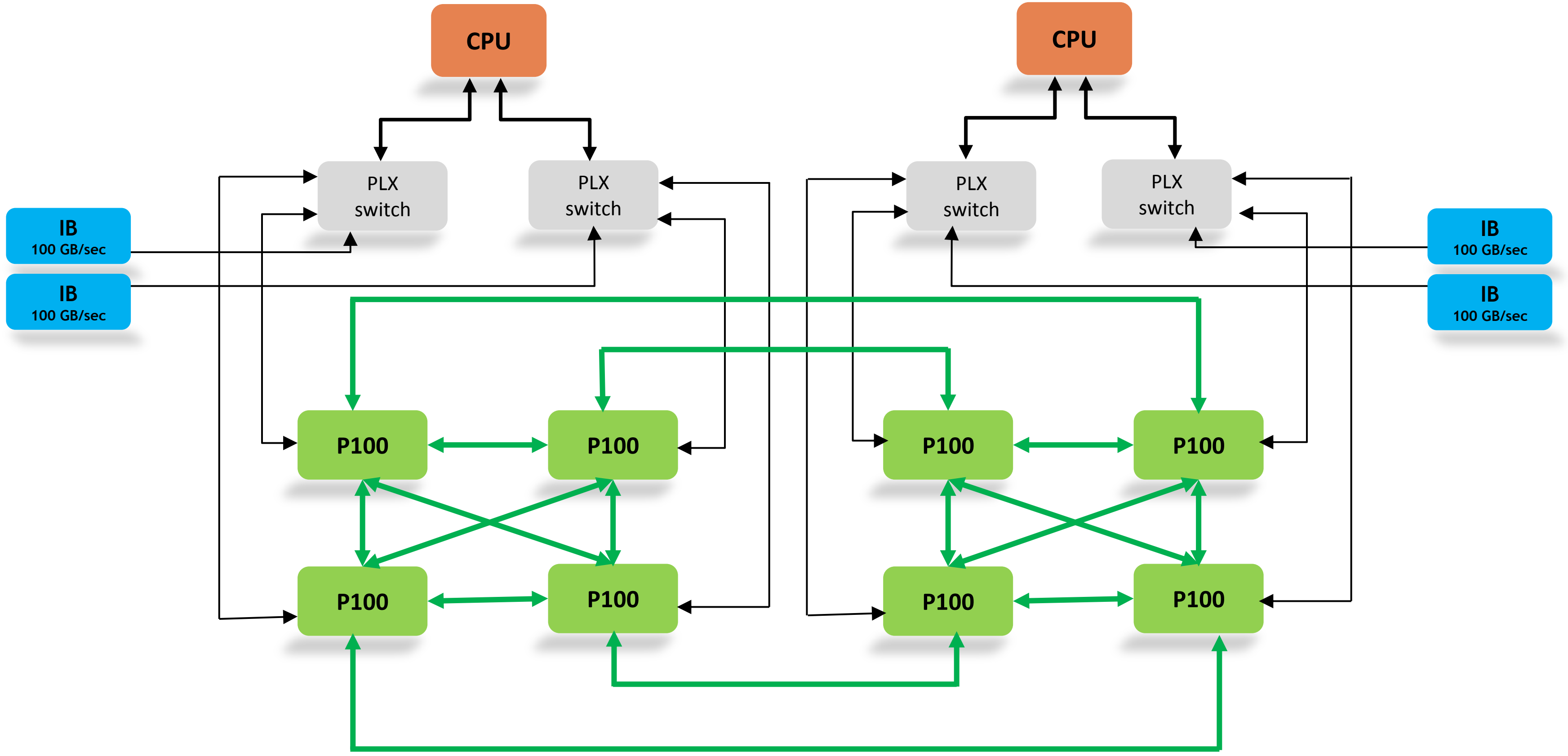
HOST ONLY	HOST/GPU
Library setup, exit and query Memory management	shmem_ptr Remote memory access Atomic memory operations Collectives Point-to-point synchronization Memory ordering Locking routines

Extensions for GPUs

- CUDA stream-based SHMEM communication
- Thread cooperative SHMEM communication

Expected availability: Q4 2017 (single OS Multi-GPU nodes: NVLink, PCIe, QPI)

DGX-1 ARCHITECTURE



Inter-GPU Copies - Single Process

Unified Virtual Addressing available since CUDA 4.0 (2011)

cudaSetDevice(0);

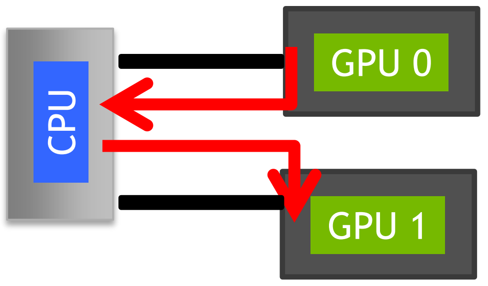
cudaMalloc(&buf0, size)

cudaSetDevice(1);

cudaMalloc(&buf1, size)

----

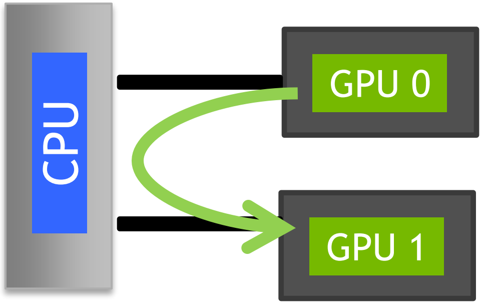
cudaMemcpy (buf0, buf1, size, cudaMemcpyDefault)



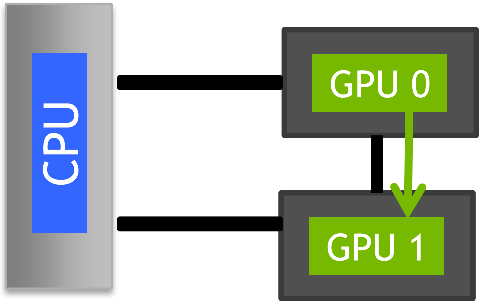
GPUDirect P2P: by-passing CPU memory

cudaDeviceCanAccessPeer(int *access, int device, int peerdevice);
cudaEnablePeerAccess(int peerDevice, unsigned int flags)

```
cudaSetDevice(0);
cudaMalloc(&buf0, size);
cudaCanAccessPeer (&access, 0, 1);
assert(access == 1);
cudaEnablePeerAccess (1, 0);
cudaSetDevice(1);
cudaMalloc(&buf1, size);
----------
----------
cudaSetDevice (0);
cudaMemcpy (buf0, buf1, size, cudaMemcpyDefault);
```



PCIe P2P
(copy/LD/ST)

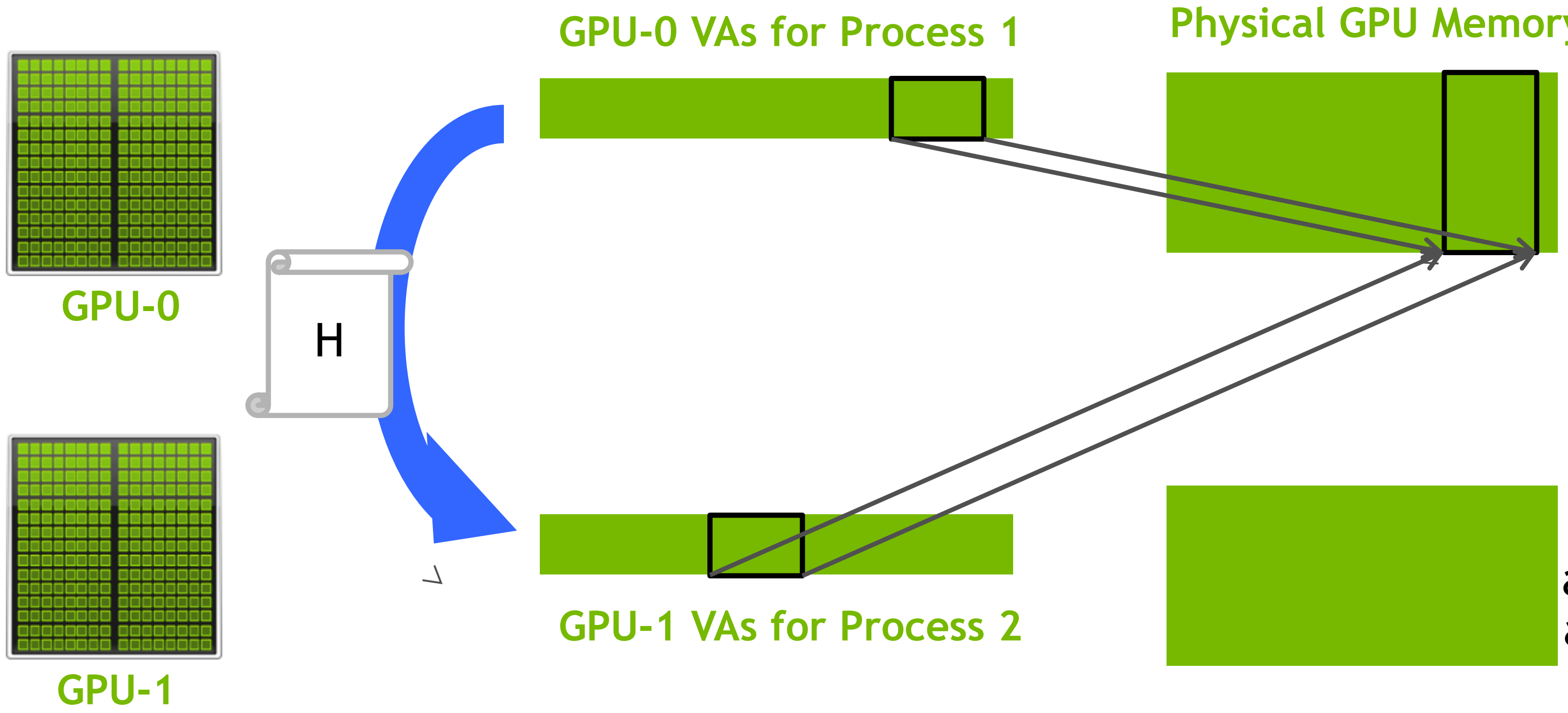


NVLink
(copy/LD/ST
/atomics)

Can also access (LD/ST) the peer GPU memory from inside a CUDA kernel

cudaIpcGetMemHandle(), cudaIpcOpenMemHandle() & cudaDeviceEnablePeerAccess()

Using GPU memory from another process



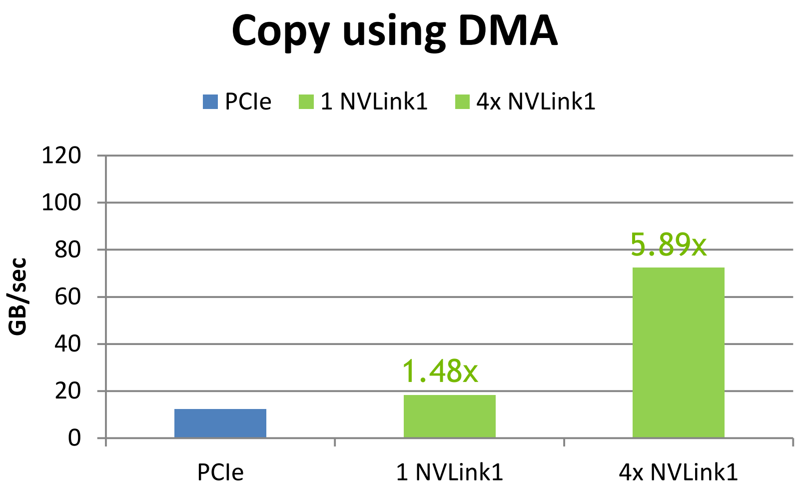
Existing allocation

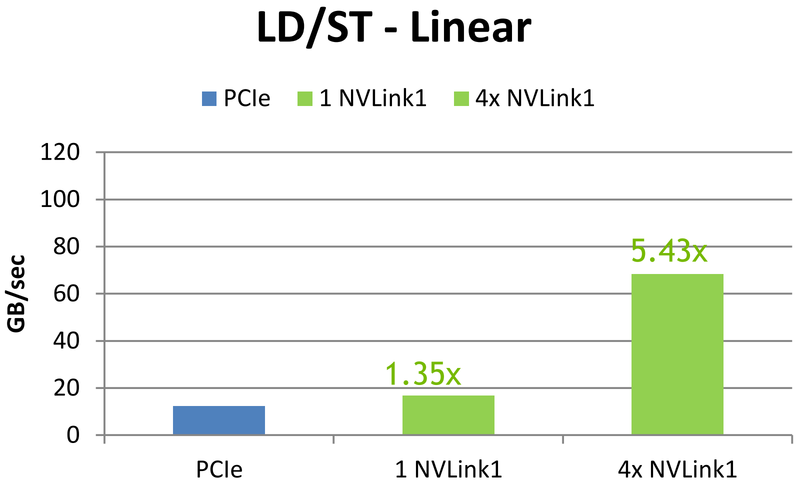
Get handle and send to other process

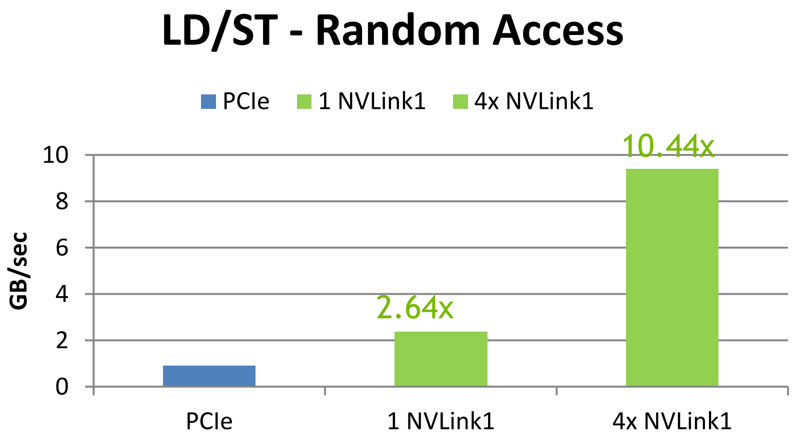
Opening handle creates mapping in receiving process

Available for access by another process AND from a second GPU use peering

NVLINK1 W/ PASCAL







- GPU memory model is relaxed and scoped
- Same memory model works within/across GPUs

BREADTH FIRST SEARCH (BFS)

Baseline: an MPI-based multi-GPU implementation of BFS by M.Bisson et. al.

Has been shown to scale to thousands of GPUs on the Tsubame supercomputer

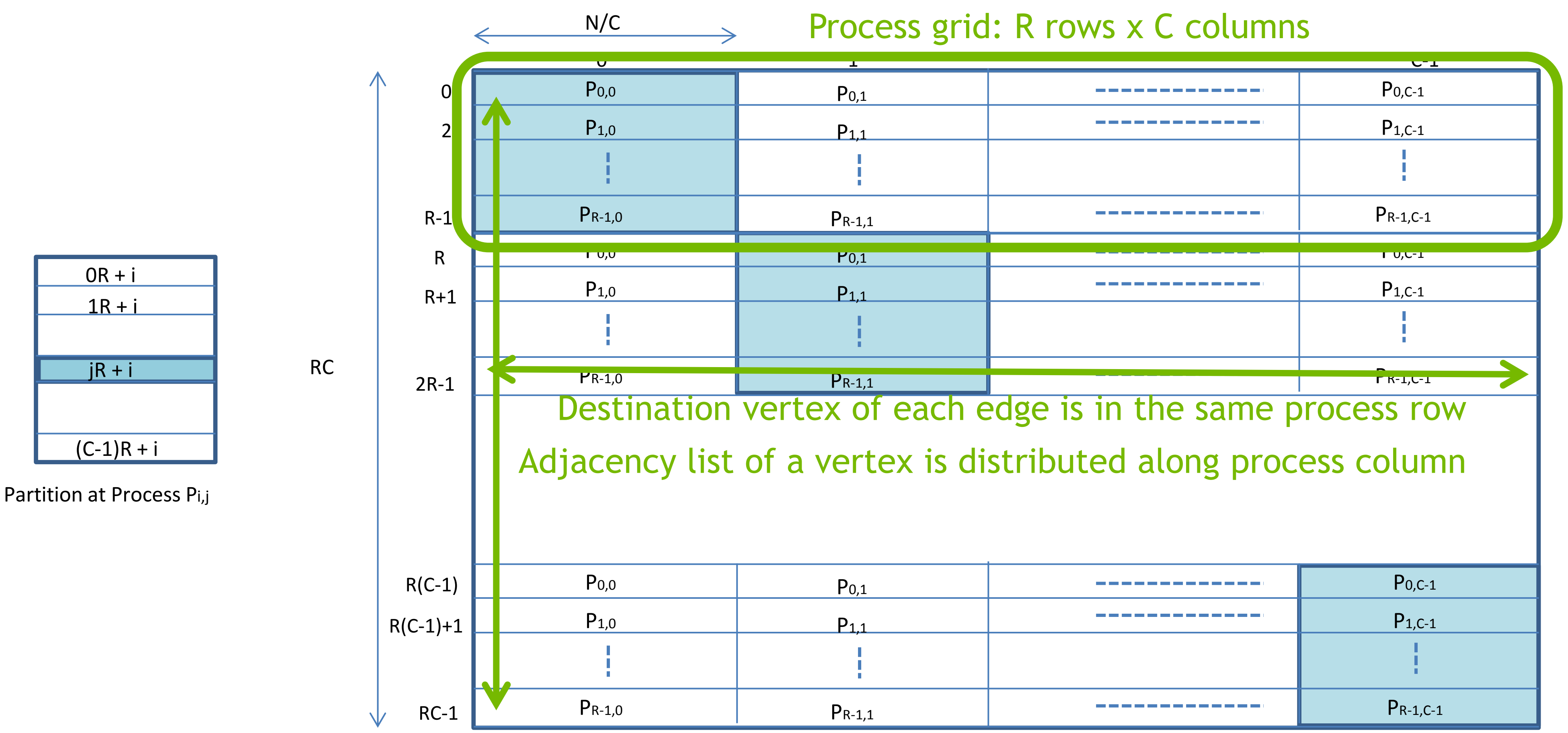
Loosely follows the Graph 500 specification

- R-MAT generator, mean performance over 64 BFS operations
- Uses 32-bit vertices instead of required 48-bit representation

Uses a parallel level-synchronous algorithm

- Frontier: list of vertices from which traversal starts in a give level
- Expand: traverse graph to find un-visited neighbors of frontier vertices
- Exchange: exchange list of newly found vertices
- Append: remove redundancy and build the frontier for next level

DATA PARTITIONING



BFS USING MPI

frontier list at one processor column is populated with the seed vertex

while (frontier not empty)
{
    Expand: local expansion
    Horizontal Exchange: **Alltoall** exchange newly discovered vertices (happens only along grid rows)
    Append: build new frontier vertex list
    Vertical exchange: **Allgather** frontier list along grid columns (happens only along columns)
    Convergence check: **Allreduce** frontier list size among all processors
}

EXCHANGE USING MPI

Vertex lists are exchanged in two forms:

- Using a vertex list (one integer per vertex)
- Using a bitmap (one bit per vertex)

Format is statically picked based on the level of traversal

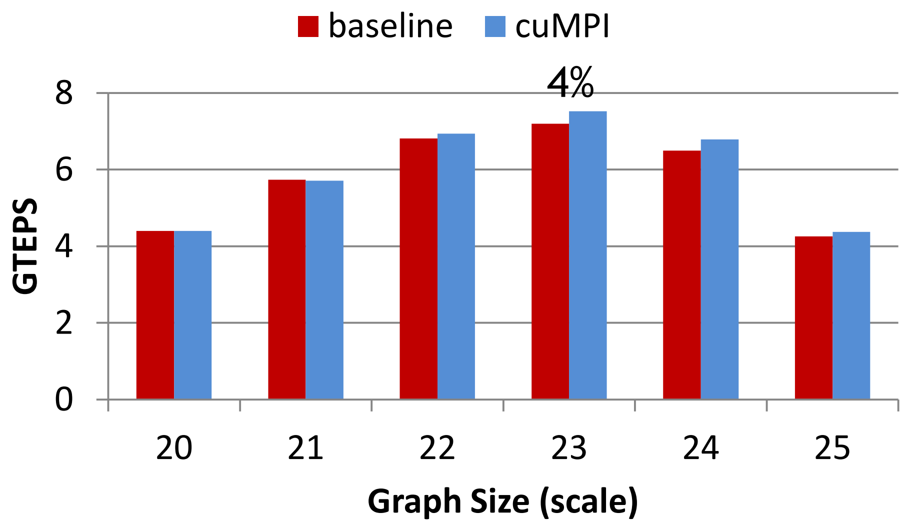
- bitmap when number of discovered vertices is expected to be large
- vertex list when number of discovered vertices is expected to be large

Communication implemented using non-blocking MPI sends and receives

Enhanced the base version to using CUDA-aware MPI

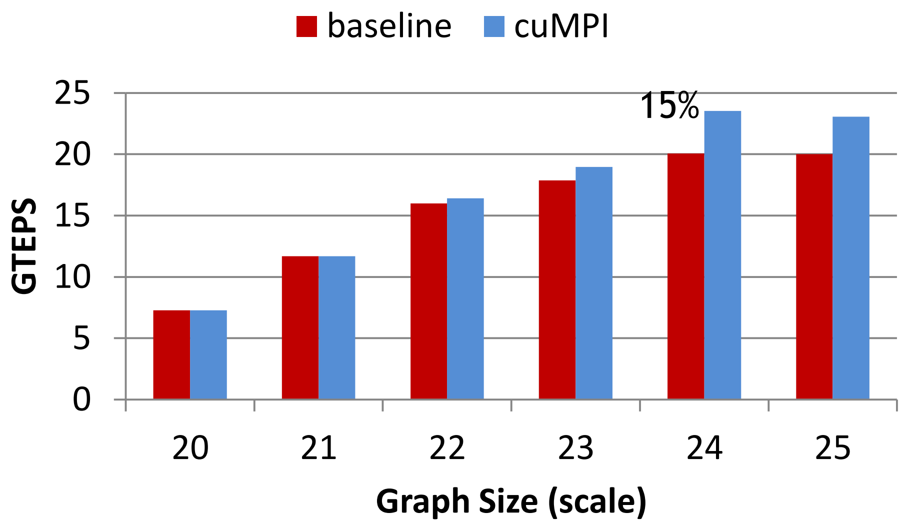
- internally uses CUDA IPC/P2P to take advantage of direct PCIe or NVLink connections

PERFORMANCE WITH CUDA-AWARE MPI



4 K40 GPUs + PCIe

Supermicro server w/ IvyBridge CPU and
Broadcom PCIe switch, CUDA 8.0, MVAPICH2 2.2



4 P100 GPUs + NVLink

DGX-1 w/ Broadwell CPU, CUDA 8.0, MVAPICH2 2.2

BFS USING NVSHMEM

Fuse communication into CUDA kernels

expand <- exchange of newly discovered vertices

append <- exchange frontier vertices

```
while (frontier not empty)
{
   Expand on GPU: local expansion
   Append on GPU: build new frontier vertex list
   Convergence check on CPU: Allreduce frontier list size among all processors
}
```

```
atomicOr(sbuf + r[k]/BITS(msk), m[k]);
```

↓

```
int peer = r[k] / drow_bl;
uint32_t off = disp + r[k]%drow_bl;
q[k] = shmem_int_or ((int *)(msk + off/BITS(msk)), m[k], peer);
```

gets rid of the MPI exchange code on the host

WORKAROUND FOR PCIE

GPU-GPU atomics are not supported over PCIE

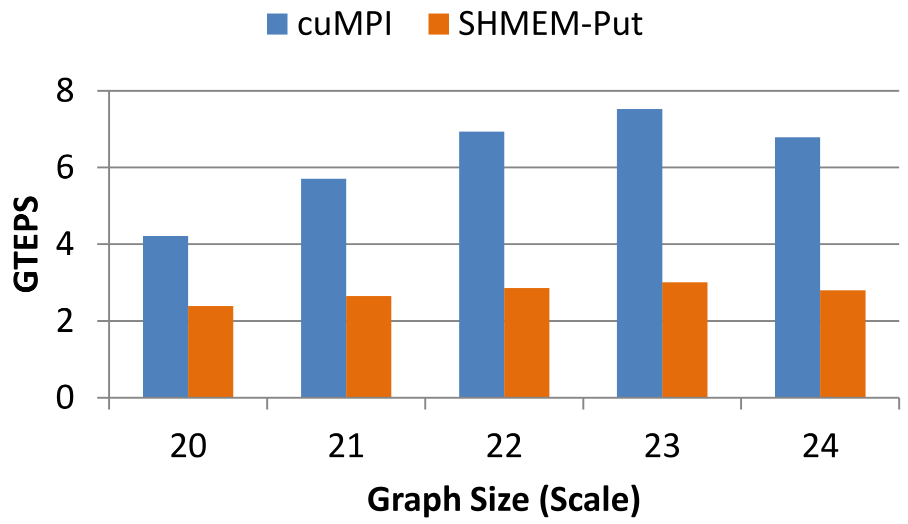
We take advantage of fact that 32-bit writes are atomic

Vertex list is represented as integers, marked with shmem_int_p

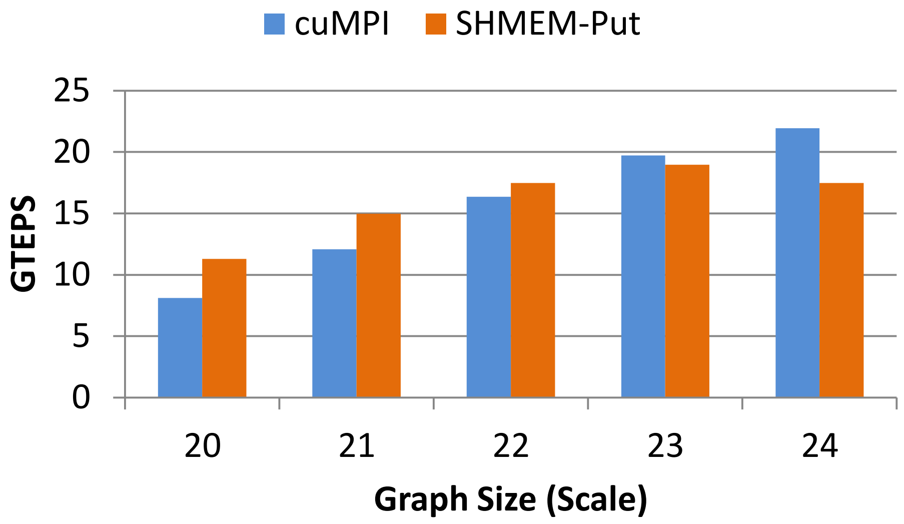
Higher processing overhead as the vertex map is 32x larger

Provides a common version to compare behavior over PCIe and NVLink

PERFORMANCE WITH PUTS

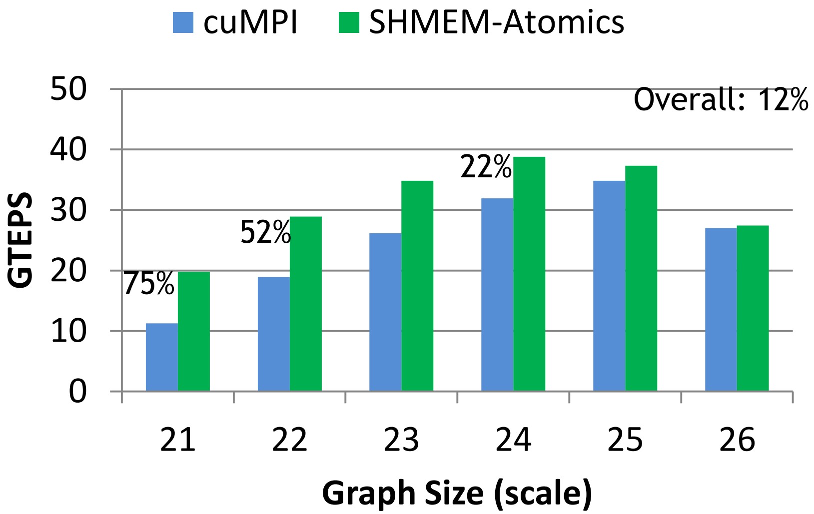


4 K40 GPUs + PCIe

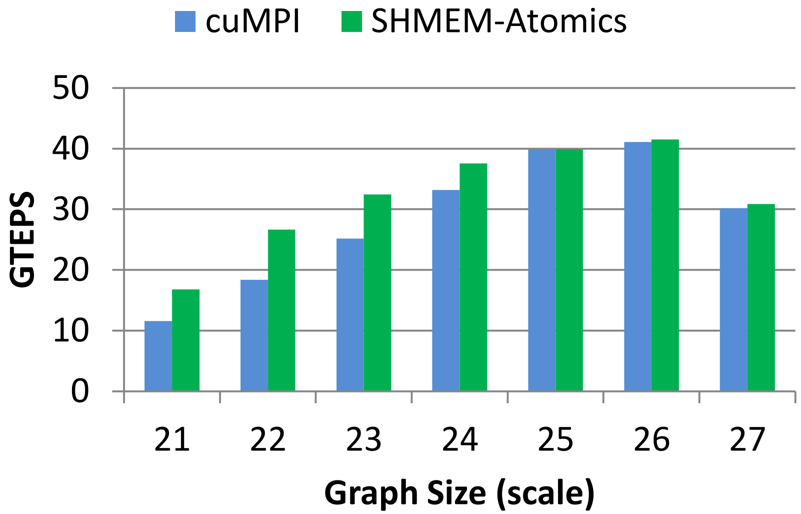


4 P100 GPUs + NVLink

PERFORMANCE WITH ATOMICS



8 P100 GPUs + NVLink (2x4)



8 P100 GPUs + NVLink (4x2)

FUTURE WORK

Fairer comparison by removing tracking of predecessor information

Hybrid vertex list/bitmap design like in the case of MPI

Can offload multiple levels of BFS using stream-based collectives in NVSHMEM

Possibility of kernel fusion, using kernel-side collectives in NVSHMEM

Evaluate the use of IB to extend NVSHEM across multiple nodes

CONCLUSION

NVLink provides a high-bandwidth memory-like fabric between GPUS

NVSHMEM provides GPU-side communication API based on OpenSHMEM

We presented a study of using GPU-initiated communication in BFS

Benefits from reduced overheads in strong scaling cases, compared to using MPI

Tradeoff with efficient network utilization for weak scaling

THANKS

We thank authors of the baseline BFS code

Mauro Bisson, Massimo Bernaschi, Enrico Mastrostefano: Parallel Distributed Breadth First Search on the Kepler Architecture

This has been supported in part by Oak Ridge National Lab under subcontract #4000145249.

NVIDIA