

Welcome!

Due to my poor network,

1. Please be muted  

2. Please Q&A at the end, if possible

3. To follow my slides:
<https://tiny.cc/msdefense>

Enabling Parallel Abstraction Layer to DCA++ using HPX And GPUDirect

Weile Wei

Masters Student, Louisiana State University

ex-intern, ORISE, Oak Ridge National Lab (Fall 19)

wwei9@lsu.edu

Enabling High-Level Parallel Abstractions to DCA++ Using HPX and GPUDirect

Portion of this talk has been presented in:

- SCALA 2020, Feb 2020. Baton Rouge, LA. [Link](#)
- Theater Talk at SuperComputing 19, Nov 2019. Denver, CO. [Link](#)
- SciDAC CompFUSE annual all-hands meeting, Oct 2019. Oak Ridge, TN.

Accepted talk:

- P3HPC: 2020 Performance, Portability, and Productivity in HPC Forum.
TBD. [Link](#)

SciDAC: Computational Framework for Unbiased Studies of Correlated Electron Systems



I would like to thank:

Giovanni Balduzzi (ETH Zurich)
Hartmut Kaiser (LSU)
John Biddiscombe (CSCS)
Arghya Chatterjee (ORNL)

Thomas Maier (ORNL)
Oscar Hernandez (ORNL)
Ed F D'Azevedo (ORNL)
Peizhi Mai (ORNL)
Ying Wai Li (LANL) [former ORNL]

- *The parallel abstraction optimization was supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. DOE, Office of Science, Advanced Computing Scientific Computing Research (ASCR) and Basic Energy Sciences (BES), Division of Materials Science and Engineering.*
- *This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.*
- *This research was supported in part by an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education.*

Outline

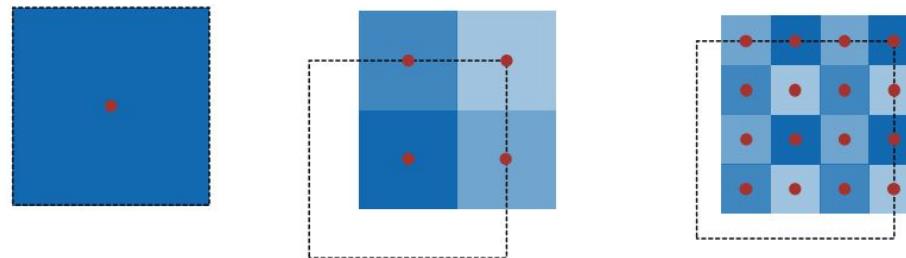
- What is DCA++? -- Scientific software for Dynamical Cluster Approximation
- Build threading abstraction layer using HPX
- Use GPUDirect to solve memory bound issue in DCA++

To follow my slides: <https://tiny.cc/msdefense>

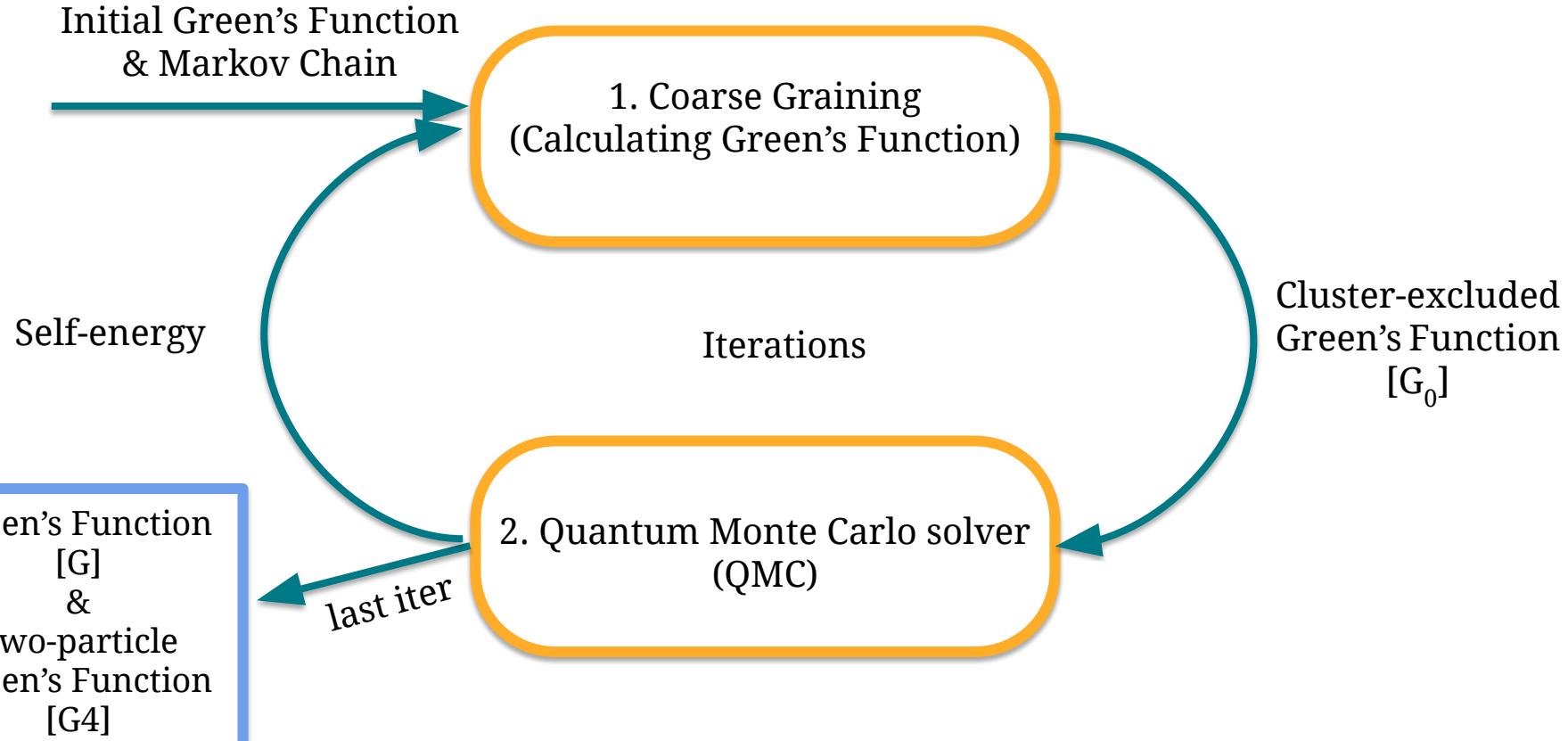
DCA ++ (Dynamical Cluster Approximation)

In the area of condensed matter physics, scientist would like to study some properties of materials, including high-temperature superconductivity, magnetism, and liquid behaviors. These behaviors are due to **strong electron-electron interaction** (mainly Coulomb repulsion).

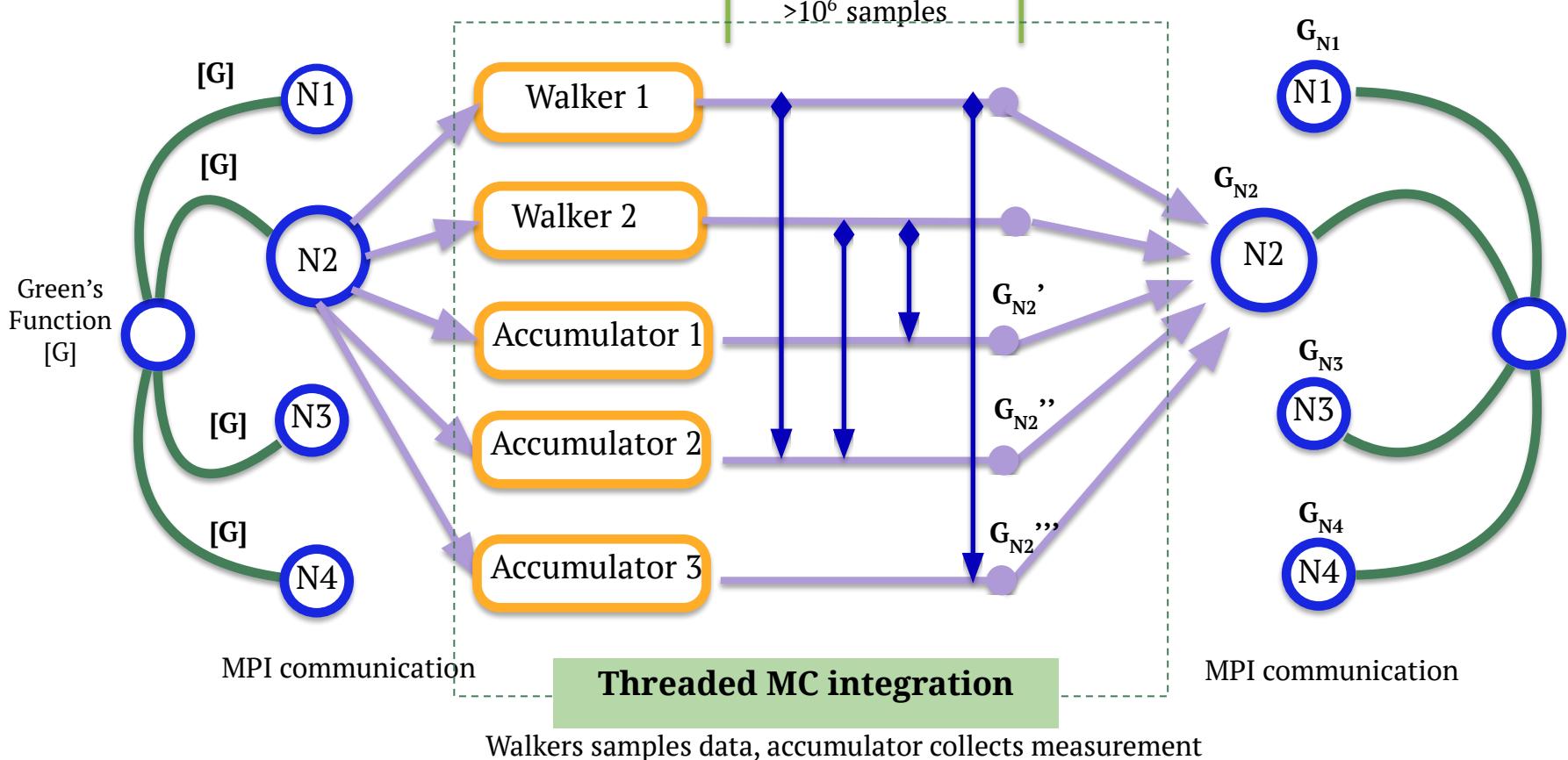
DCA++ is a numerical simulation software to solve correlated electron problems with Quantum Monte Carlo method.



DCA++: Primary kernels workflow



DCA++ : Quantum Monte Carlo Solver



Threading Abstraction

Threading abstraction for QMC Solver

Original Implementation

custom-made thread pool using std::thread

DCA++

New Implementation

custom-made thread pool using std::thread

HPX thread pool

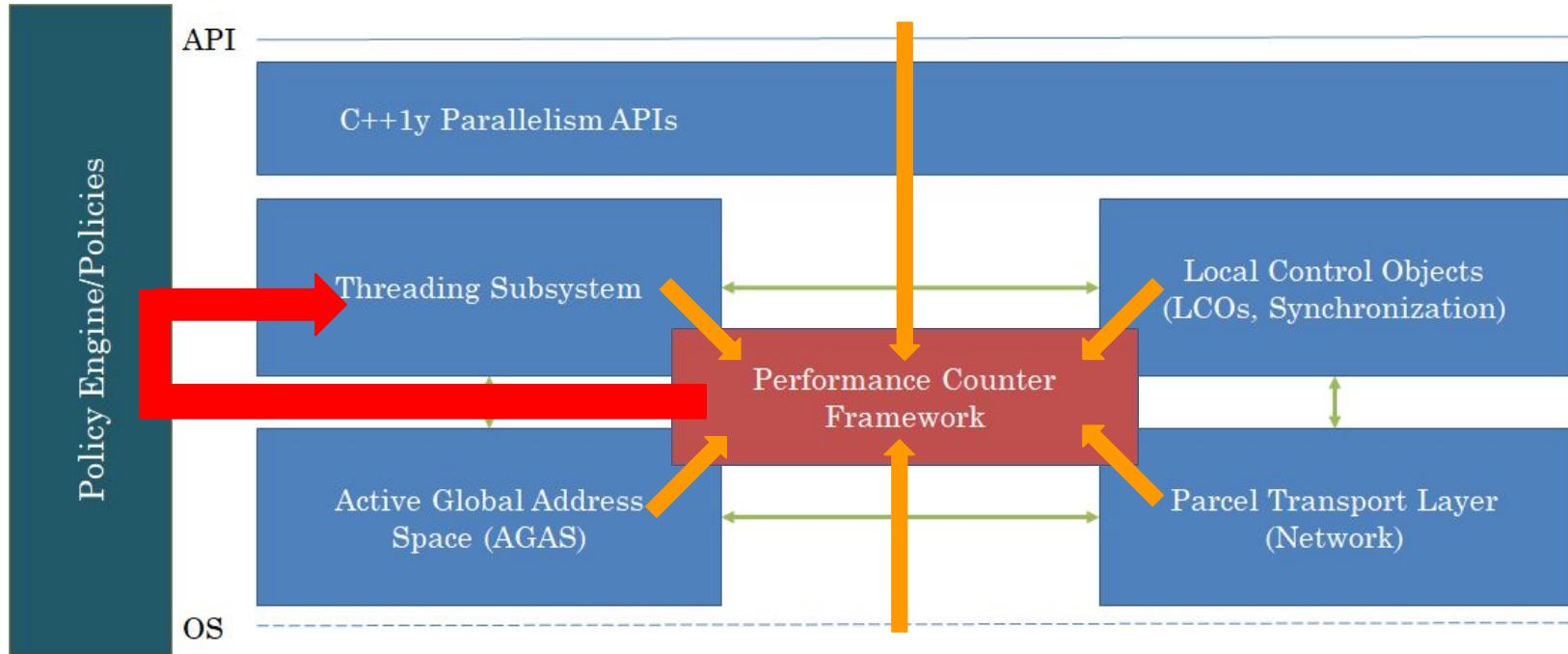
-DHPX_DIR=\$HPX_PATH
-DDCA_WITH_HPX=ON

switch by user-input at compile time

Threading Abstraction

DCA++

HPX - High-Performance ParalleX



HPX - A General Purpose Runtime System

- Widely portable (raspberry pi → different OS → supercomputers)
- C++ standard compliant
- Opensource: 100+ developers over a decade
<https://github.com/STELLAR-GROUP/hpx>
- Supported Distributed Machine Learning, Astrophysics, Coastal Modeling

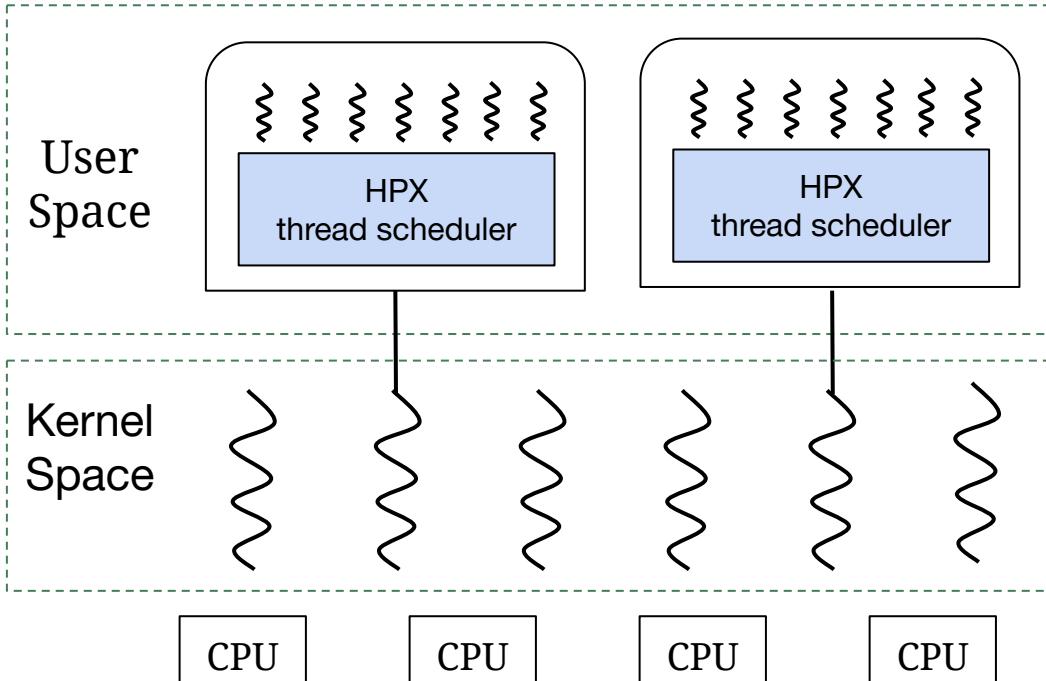
HPX - C++ standard compliant and more

- As close as possible to C++ standard library:

• std::thread	hpx::thread
• std::mutex	hpx::mutex
• std::future	hpx::future
• std::async	hpx::async
• std::bind	hpx::bind
• std::function	hpx::function
• std::tuple	hpx::tuple
• std::any	hpx::any
• std::parallel::for_each, etc	hpx::parallel::for_each
• std::cout	hpx::cout
• std::vector	hpx::vector, hpx::partitioned_vector

- Extend standard APIs where needed (compatibility is preserved)

HPX thread pool



Nanosecond level

HPX thread is a lightweight user-level thread

- ~1000x faster context switch than OS thread

Microsecond level

QMC solver w/ std::thread

```
// original implementation w/ standard thread
vector<std::future<void>> futures;

auto& pool = ThreadPool::get_instance();

for (int i = 0; i < tasks.size(); ++i) {
    if (tasks.getTask(i) == "walker")
        futures.emplace_back(pool.enqueue(&ThisType::startWalker, this, i));

    // else if handle other conditions...
}
```

QMC solver w/ hpx

```
// new implementation w/ hpx
vector<hpx::future<void>> futures;

// auto& pool = ThreadPool::get_instance();

for (int i = 0; i < tasks.size(); ++i) {
    if (tasks.getTask(i) == "walker")
        futures.emplace_back(hpx::async(&ThisType::startWalker, this, i));

    // else if handle other conditions...
}
```

Threading Abstraction

std::thread

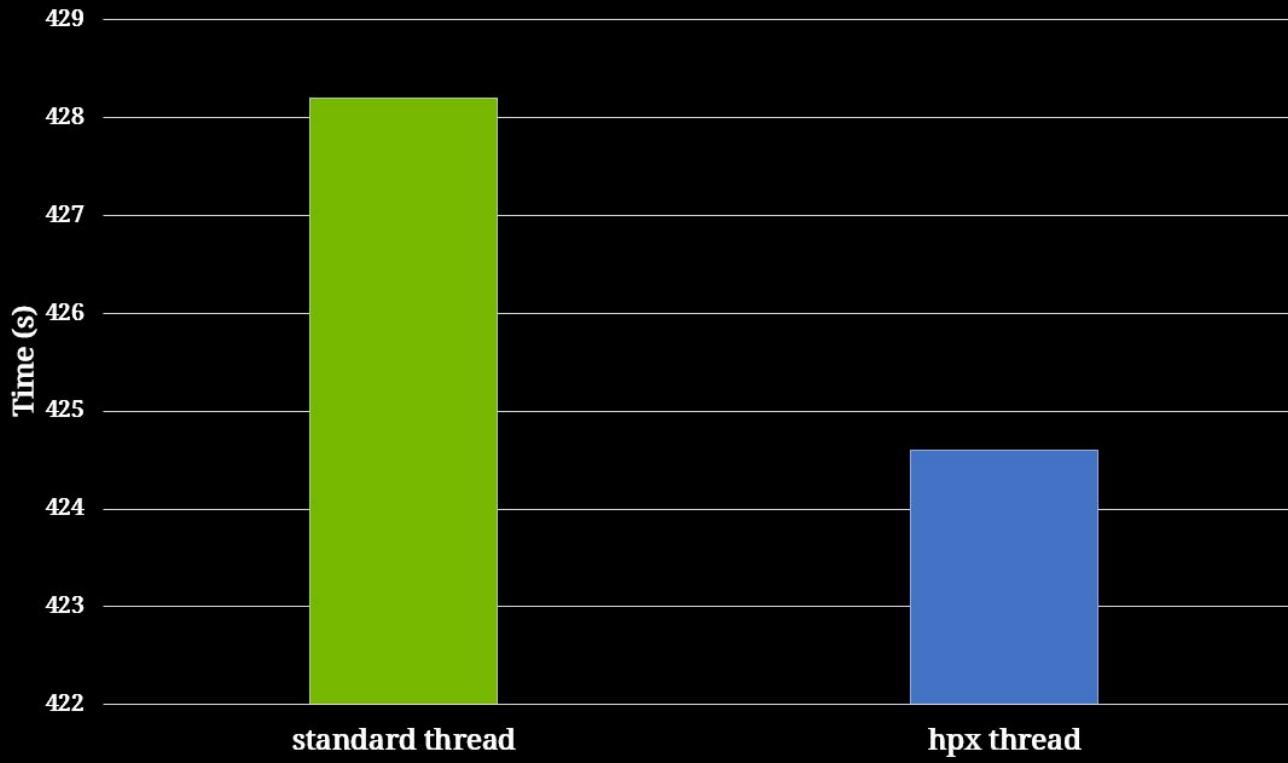
```
namespace dca {  
namespace parallel {  
  
struct thread_traits {  
    template <typename T>  
    using future_type      = std::future<T>;  
    using mutex_type       = std::mutex;  
    using condition_variable_type =  
        std::condition_variable;  
    using scoped_lock      =  
        std::lock_guard<mutex_type>;  
    using unique_lock      =  
        std::unique_lock<mutex_type>;  
}  
}  
} // namespace parallel  
}; // namespace dca
```

HPX thread

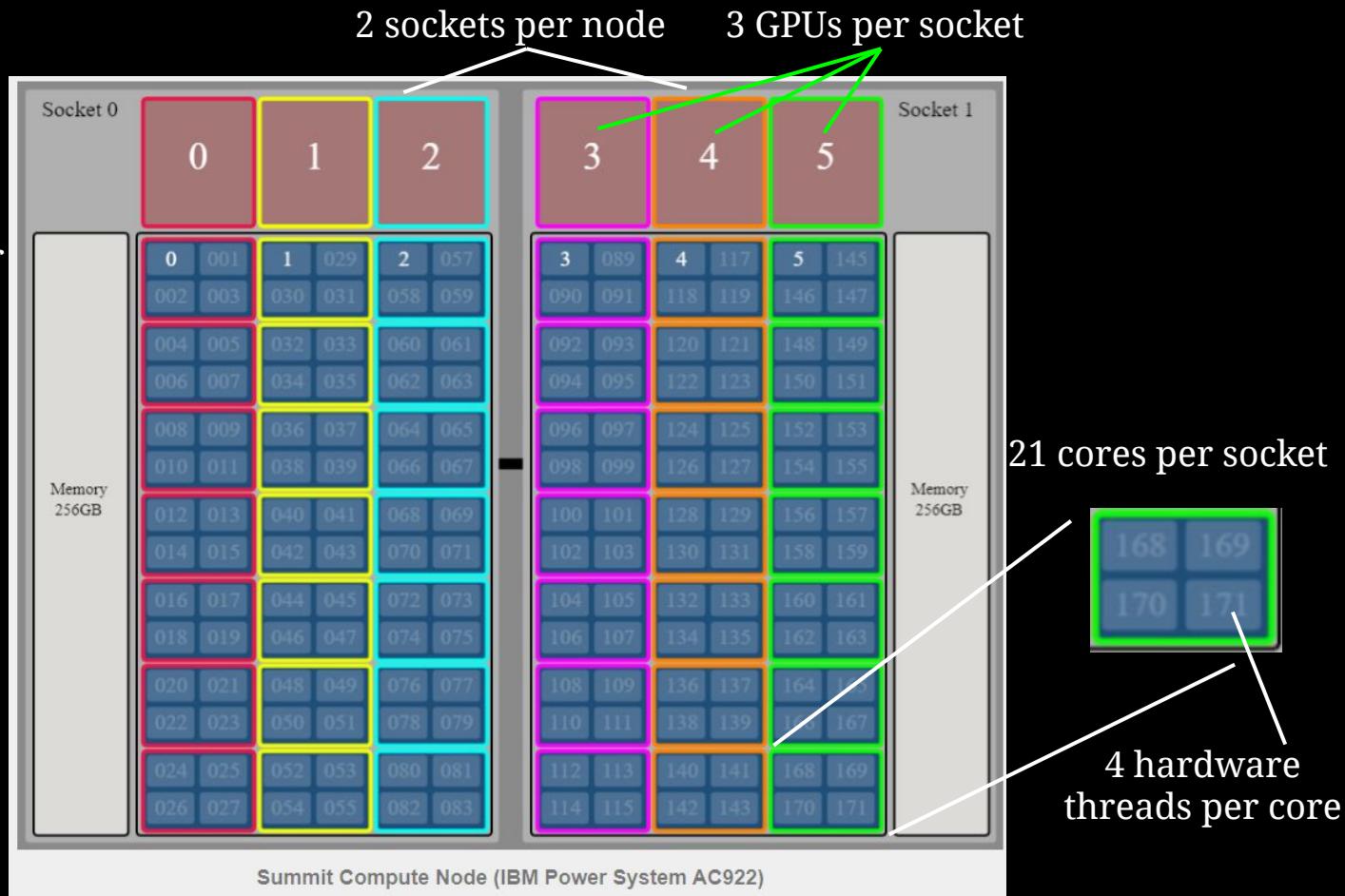
```
namespace dca {  
namespace parallel {  
  
struct thread_traits {  
    template <typename T>  
    using future_type      = hpx::future<T>;  
    using mutex_type       = hpx::mutex;  
    using condition_variable_type =  
        hpx::condition_variable;  
    using scoped_lock      =  
        std::lock_guard<mutex_type>;  
    using unique_lock      =  
        std::unique_lock<mutex_type>;  
}  
}  
} // namespace parallel  
}; // namespace dca
```

Performance Measurement

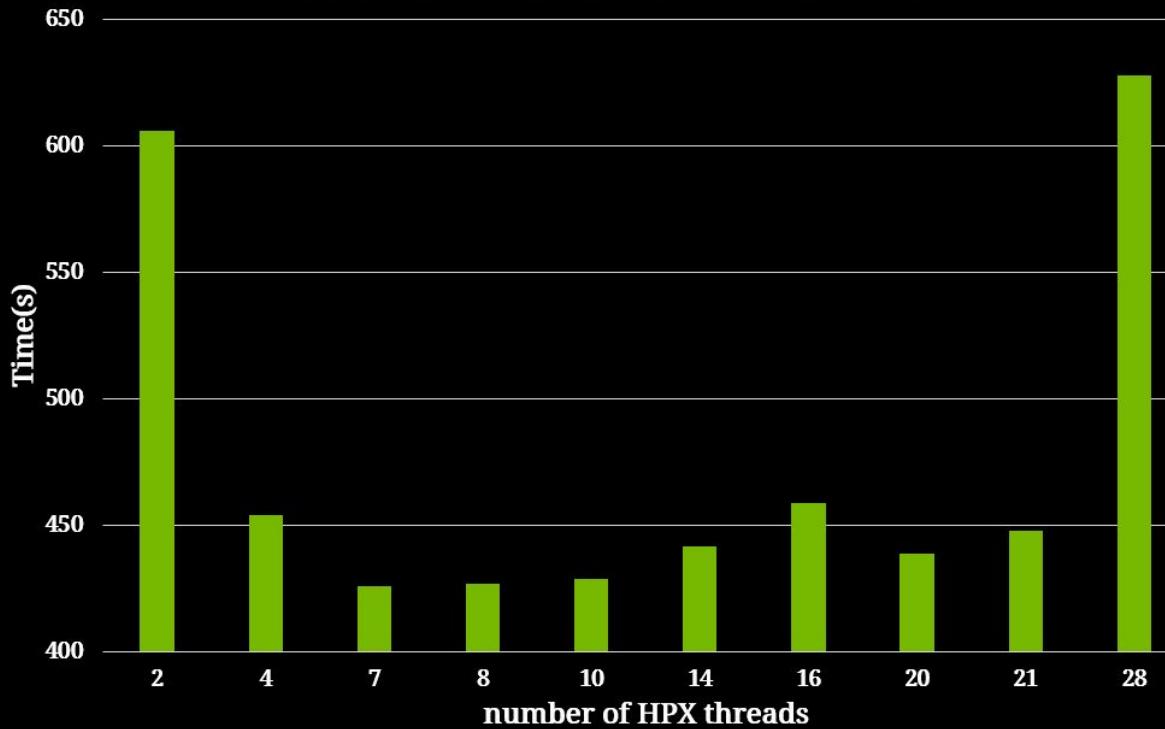
Compare DCA++ execution time between standard thread and hpx thread (threads=7)



- DCA only allows 1 GPU per rank
- DCA strategy:
 - (1 gpu + 7 cpus) per rank
 - 7 threads only



Compare DCA++ execution time among different numbers of HPX threads



Sub-summary

- **Threading abstraction w/ HPX light-weight threads:**
 - Added threading abstraction
 - Profiled performance
 - Future work
 - To add more tasks continuation to fully utilize hardware resources

Memory Bound Issue and Solution

Memory bound issue and solution

- Memory bound issue caused by the size of G4

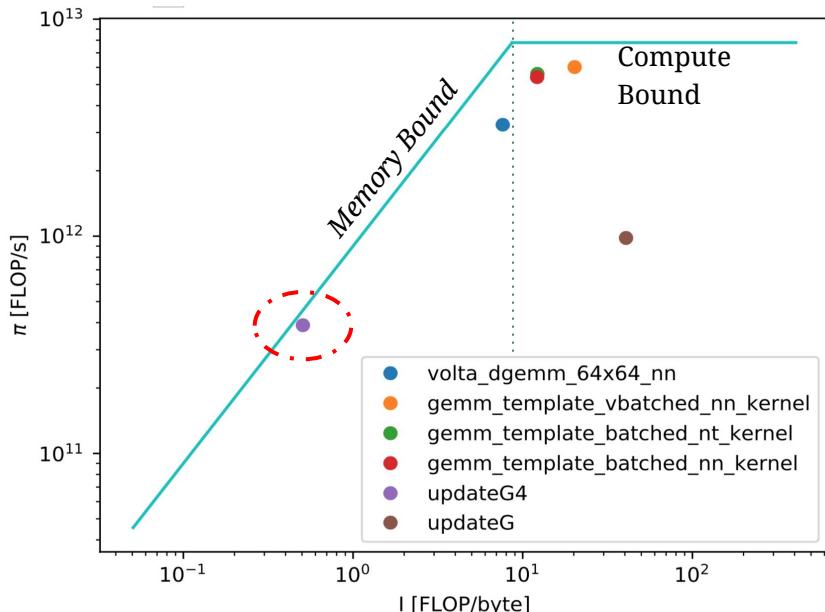
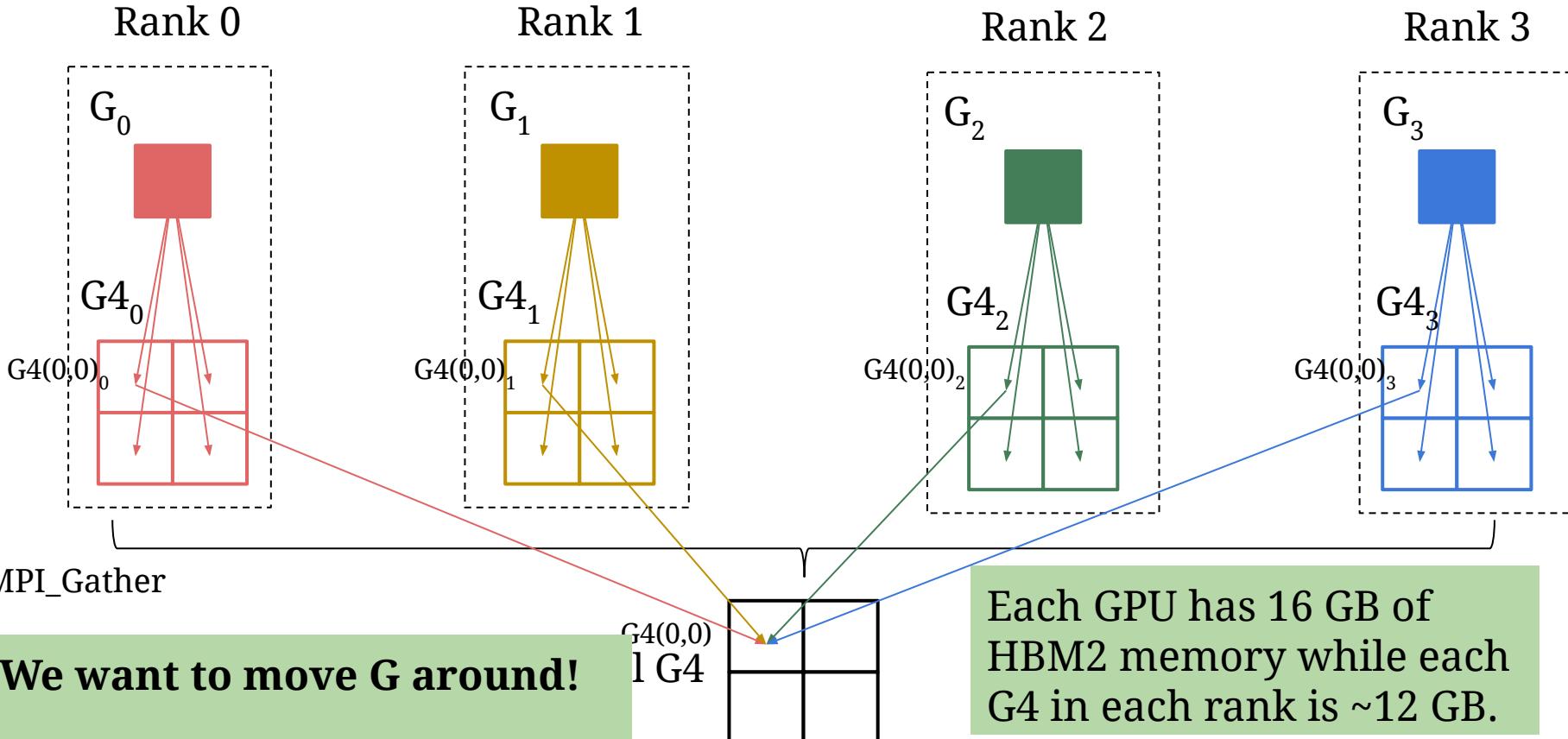


Fig. : Roofline plot of a NVIDIA V100 GPU running DCA++ at production level on Summit (OLCF)

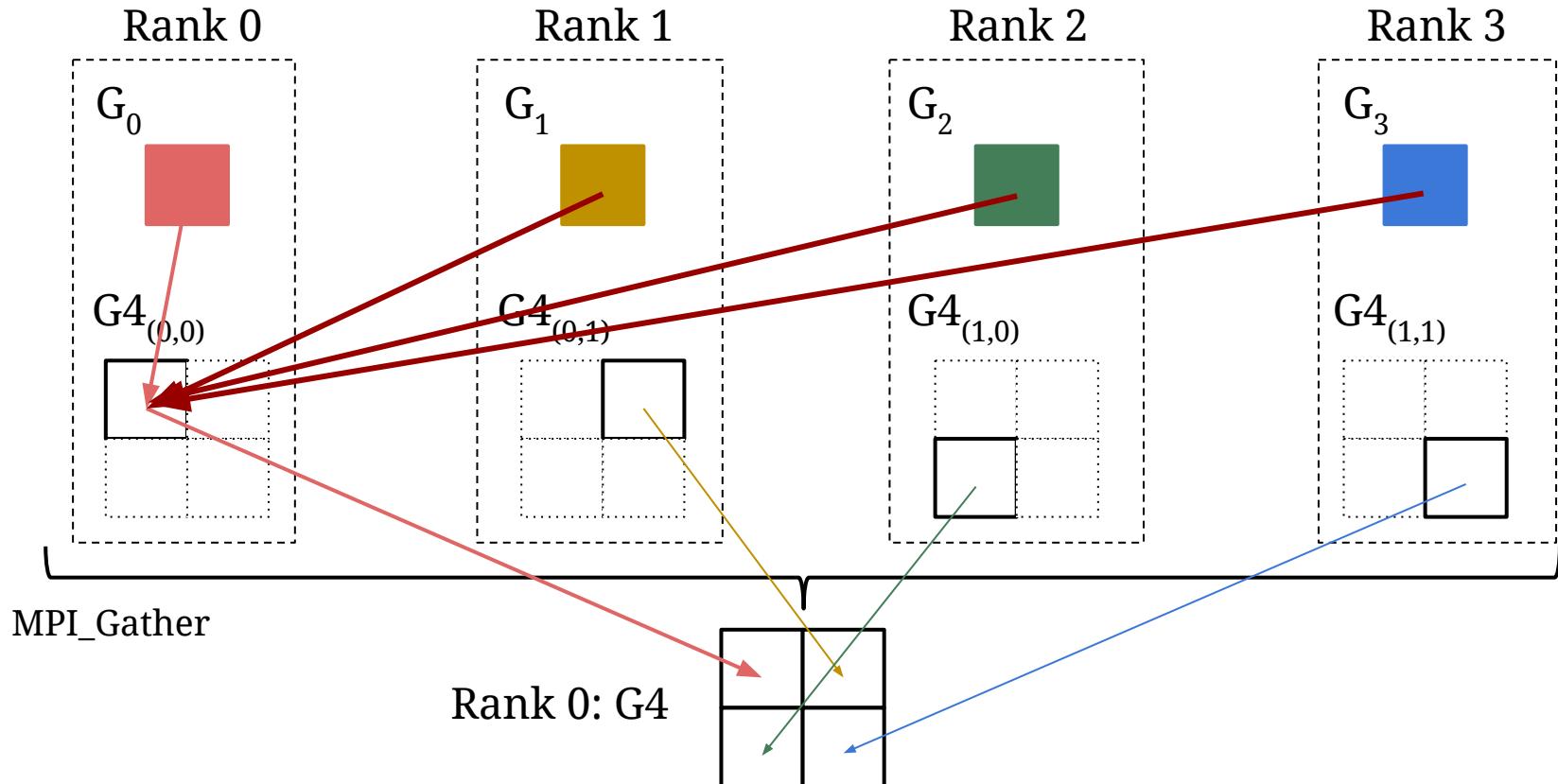
Solution: *broadcasting* each G matrix to all other ranks:

- Regular MPI method
- NVLink method

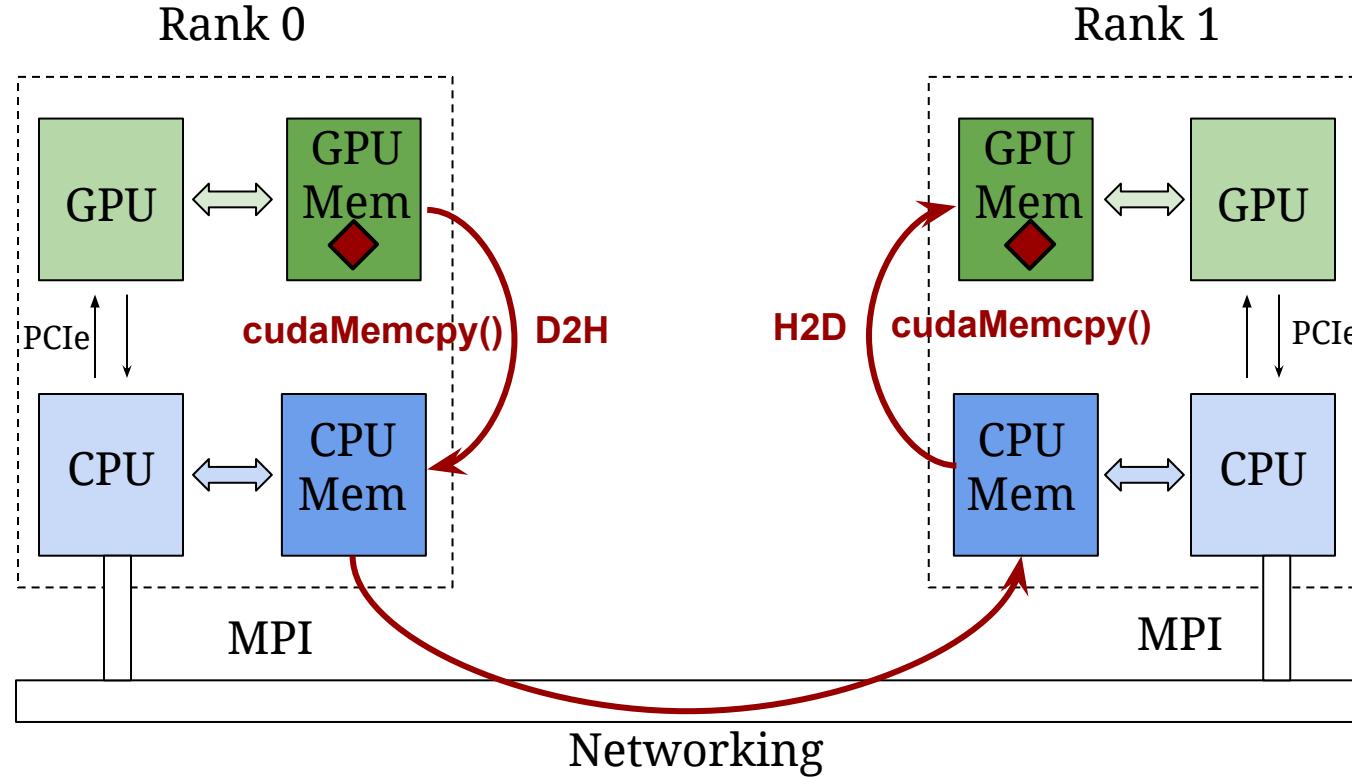
Memory bound issue w/ G4



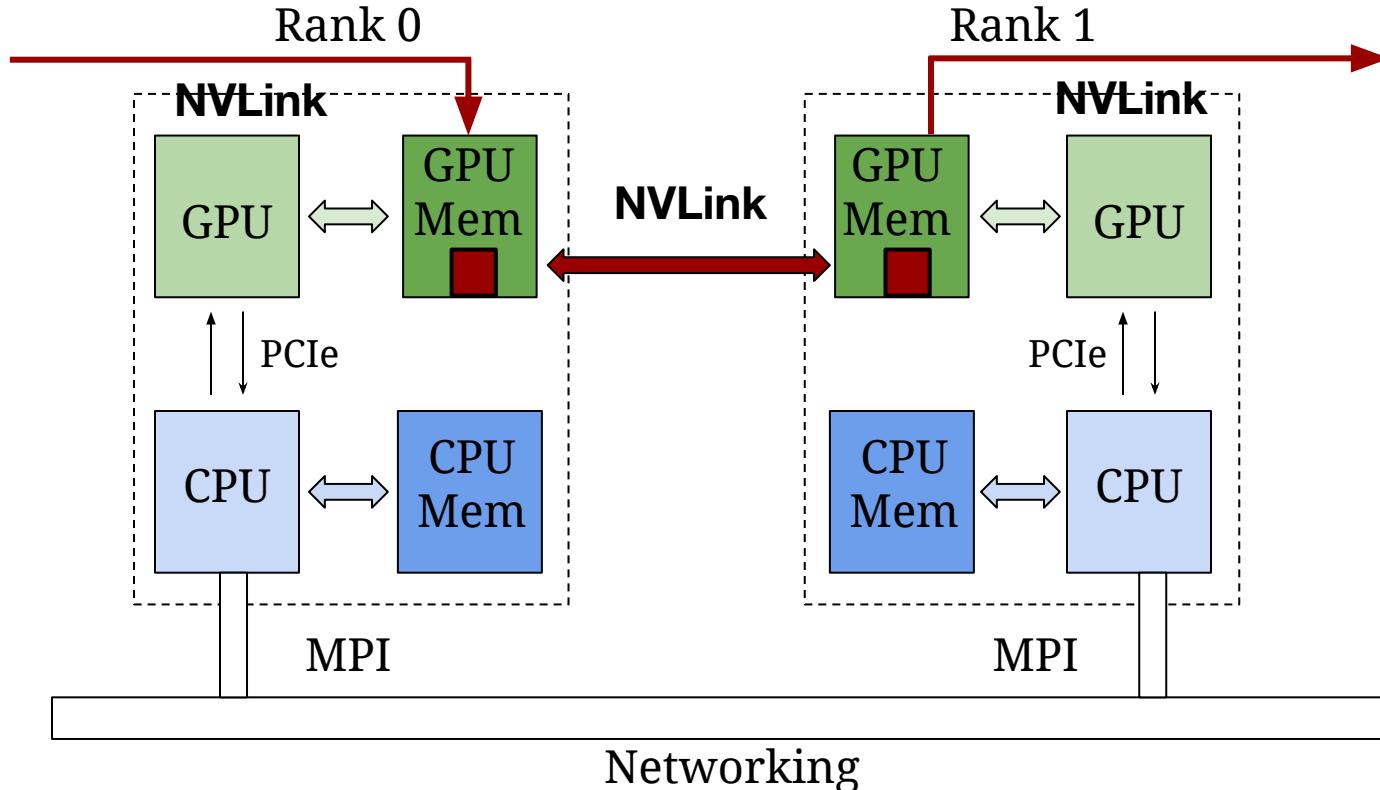
Move G2 around



Moving G around: Regular MPI method



Moving G around: NVLink method

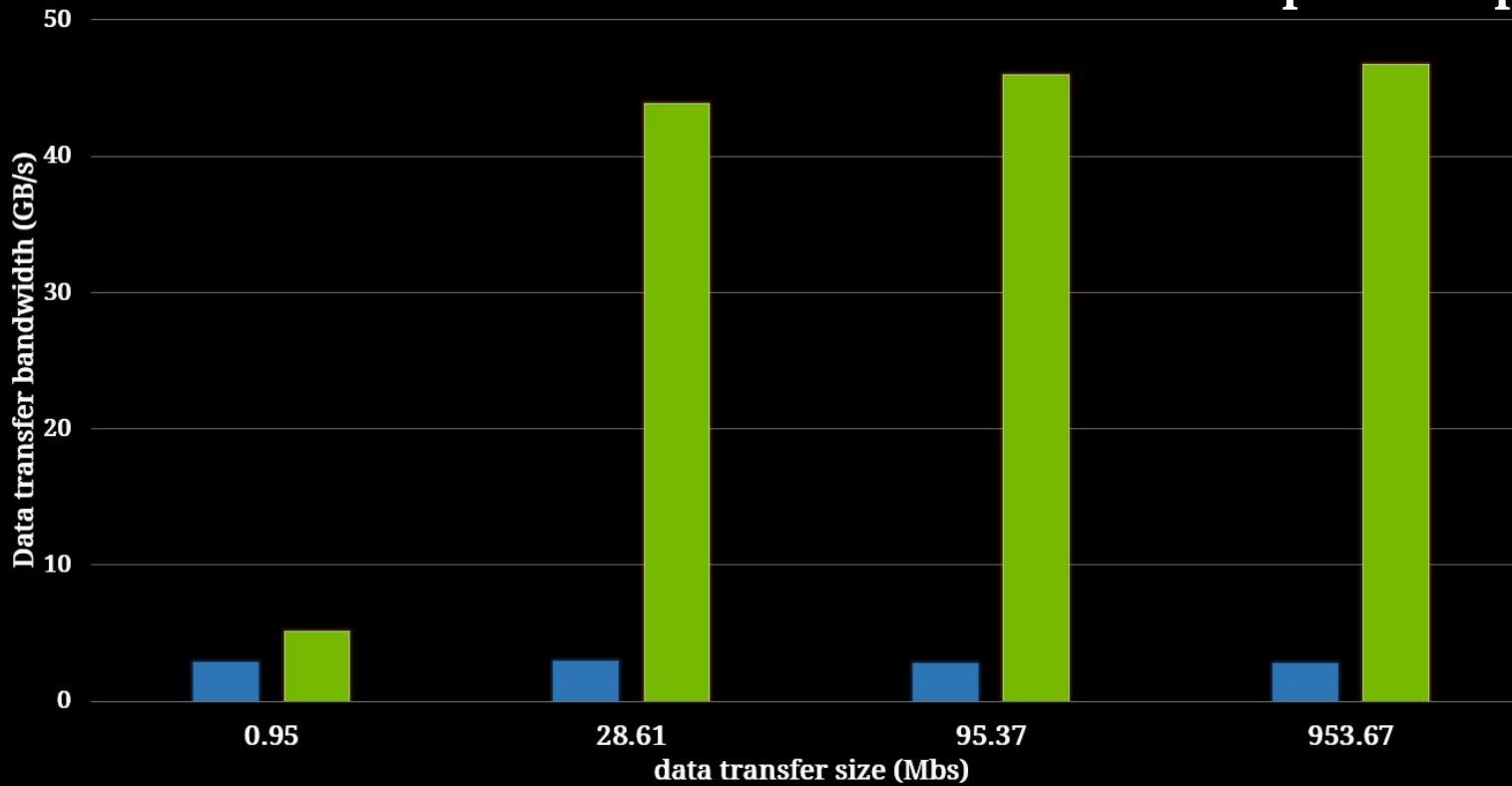


Bandwidth Measurement

Compare on-node bandwidth of data transfer on Summit between NVLink and Regular MPI GPU to Remote GPU method

■ Regular MPI GPU to Remote GPU ■ NVLink

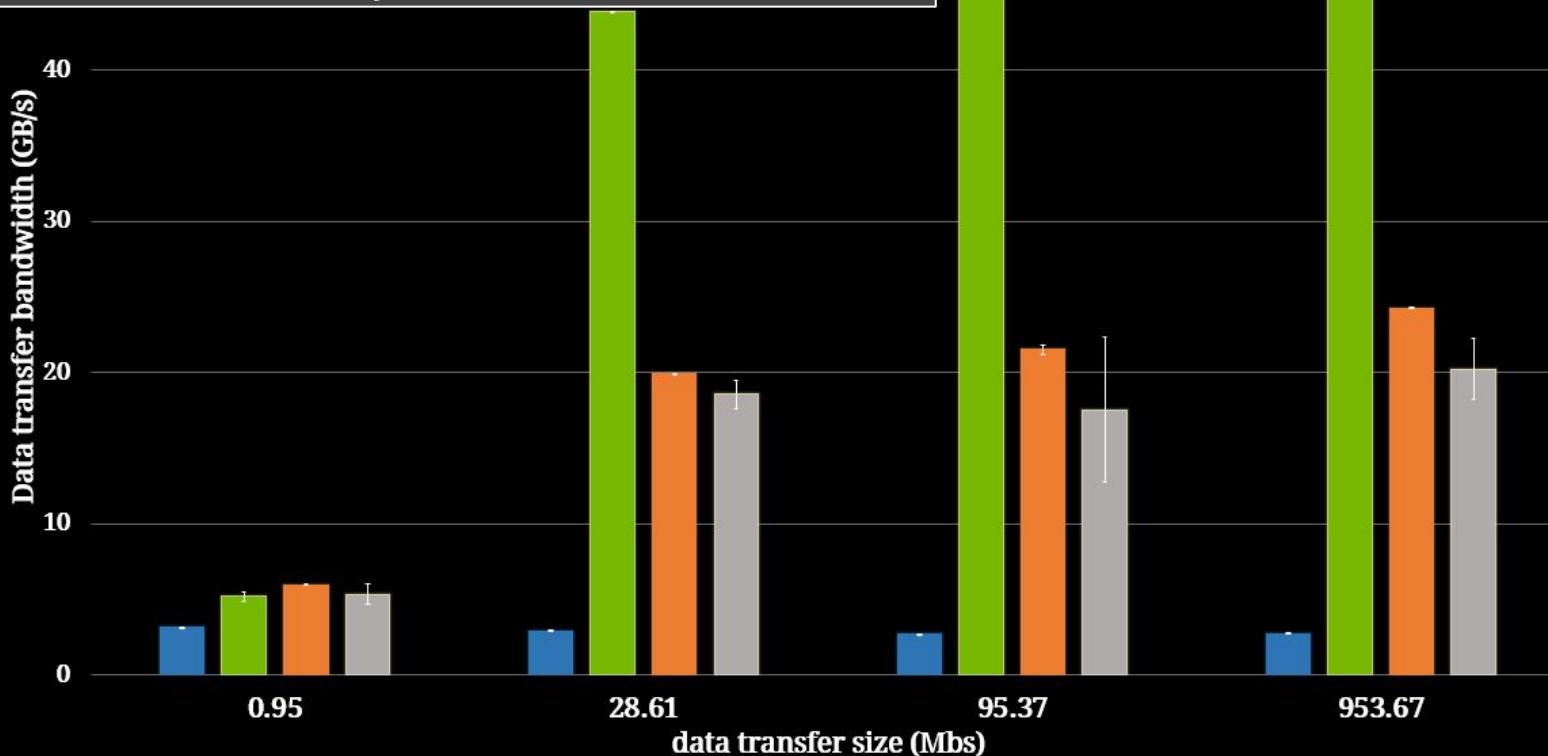
up to 17x speedup



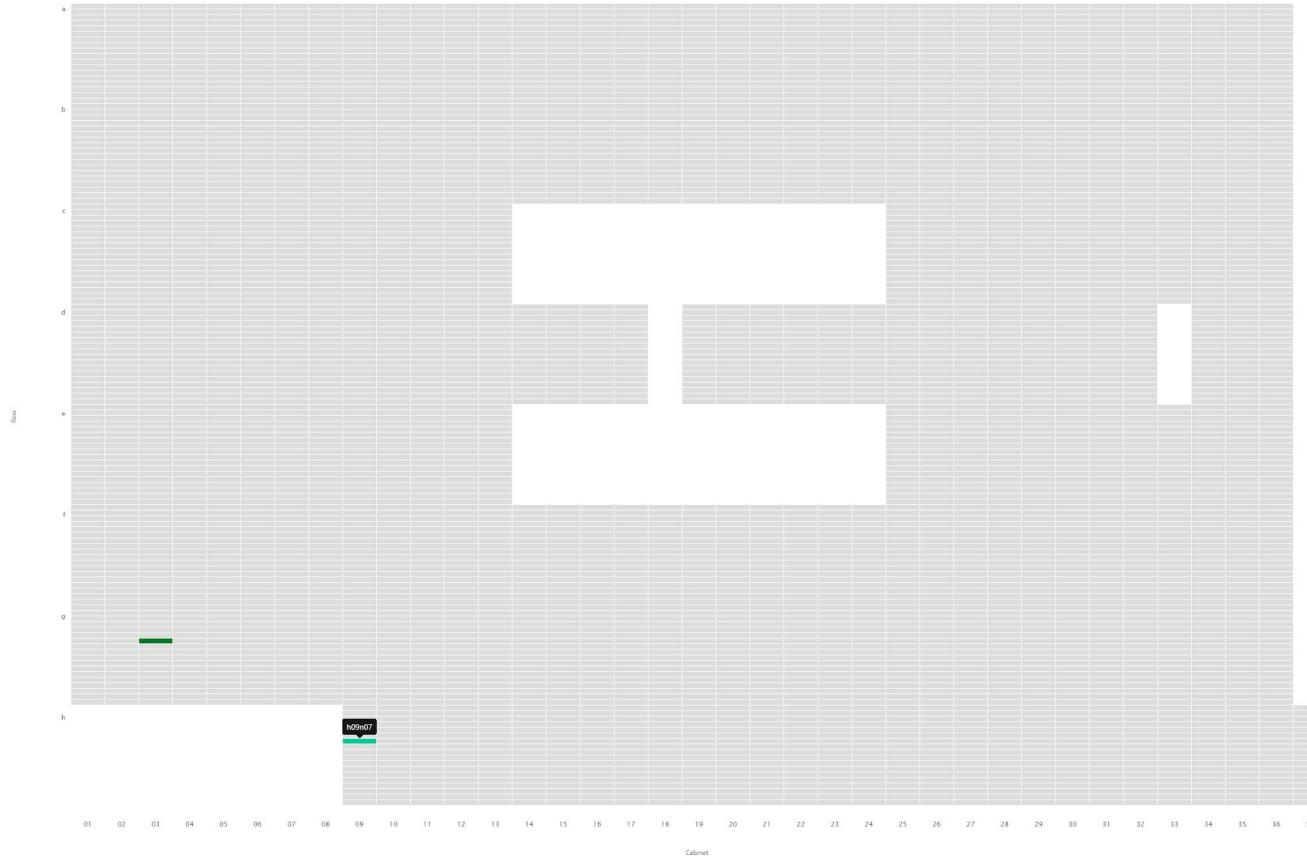
Compare bandwidth of data transfer on Summit using NVLink

■ Regular MPI ■ on-node NVLink ■ off-node NVLink in same rack ■ off-node NVLink in different racks

- Drop off in bandwidth is due to network congestion
- 20-23 GB/s between any two nodes in the network



See Warnings

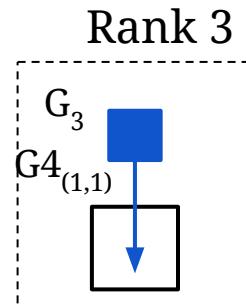
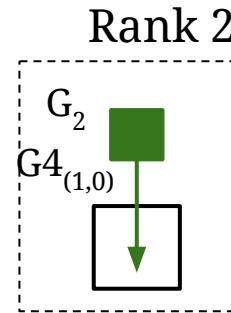
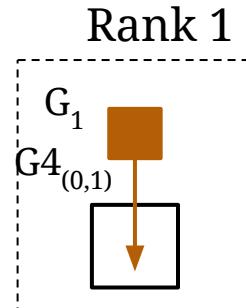
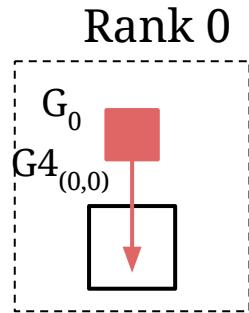
<https://jobstepviewer.olcf.ornl.gov/summit/952157-3/h09n07>

Powered by

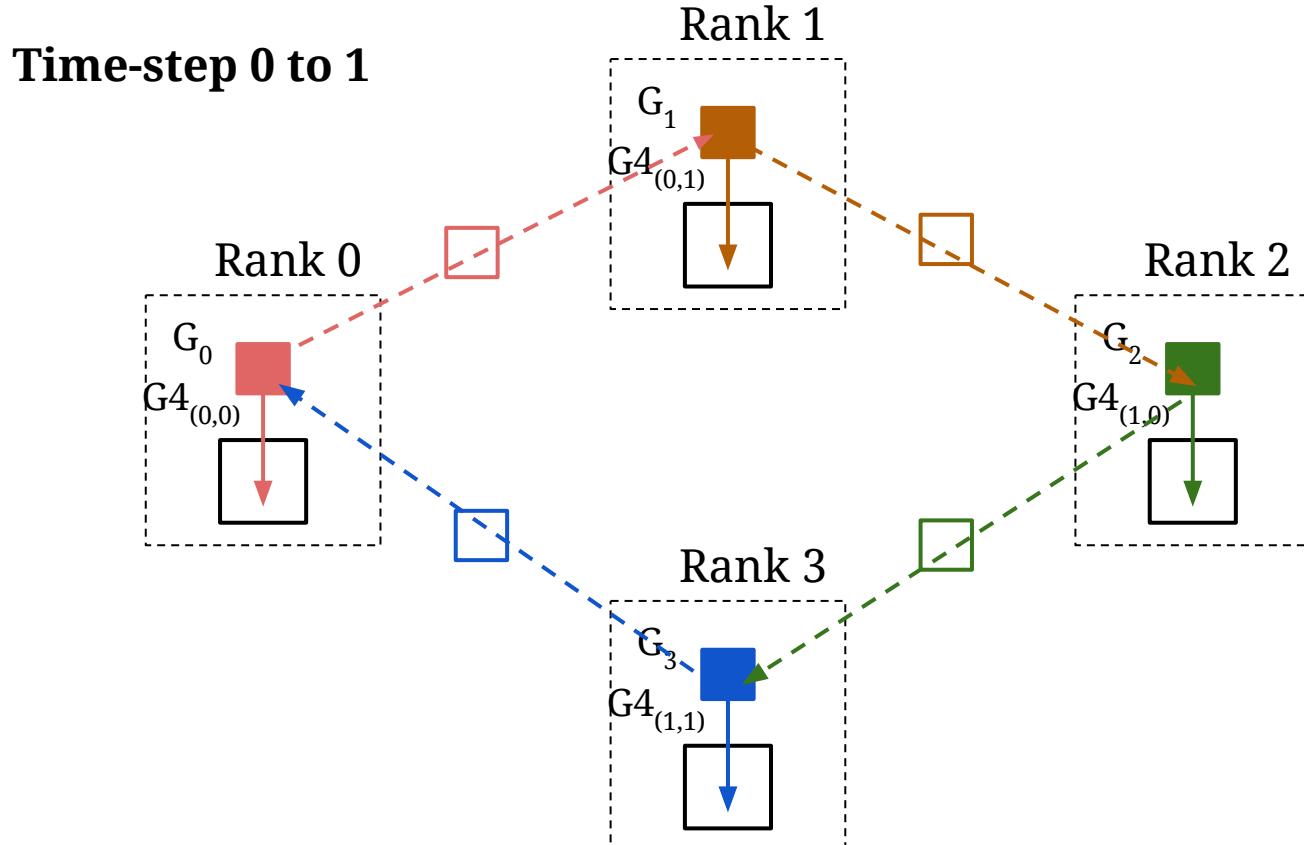
<https://jobstepviewer.olcf.ornl.gov/summit/952157-3>

Pipelined Ring Algorithm

Time-step 0

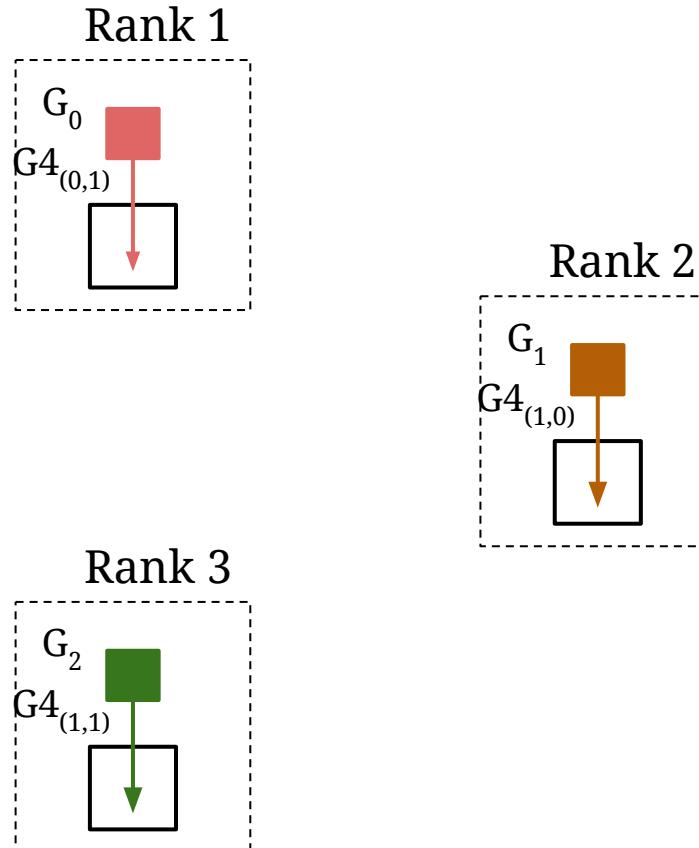
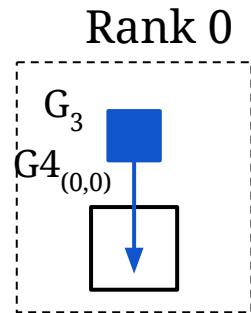


Pipelined Ring Algorithm



Pipelined Ring Algorithm

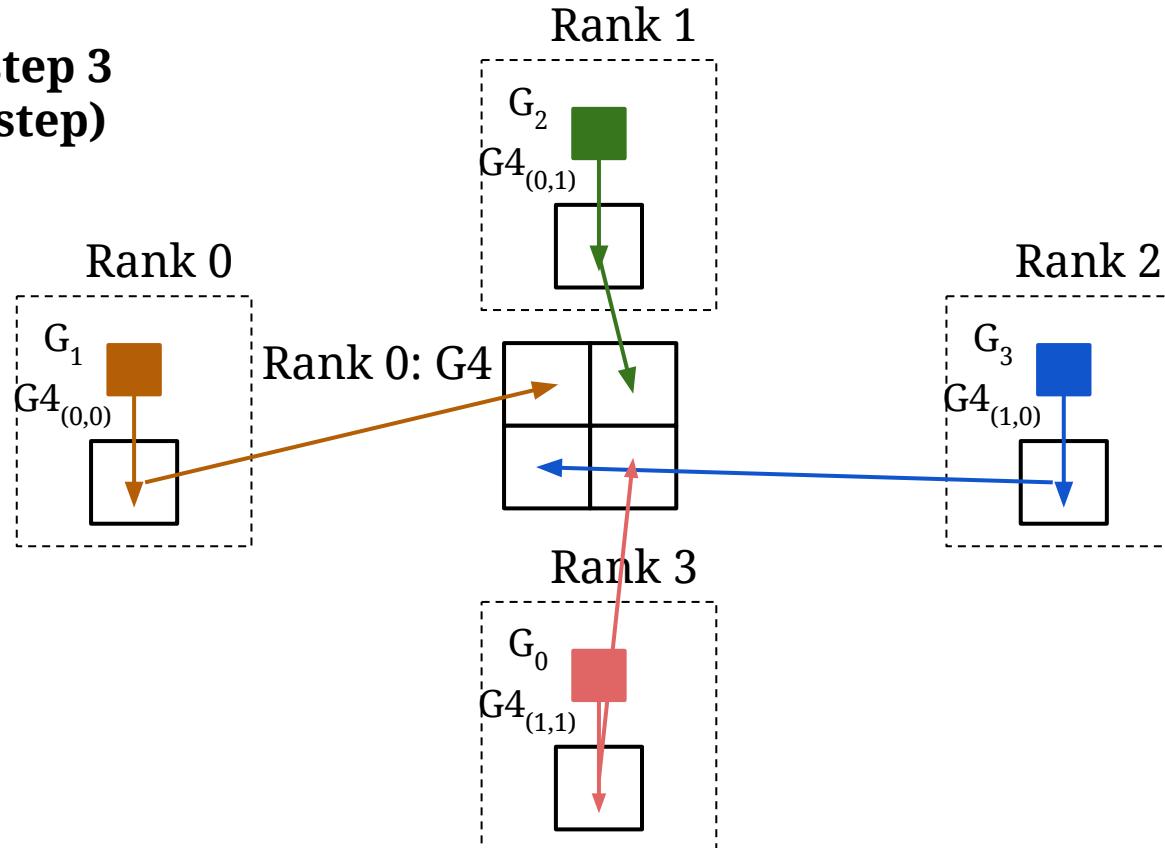
Time-step 1



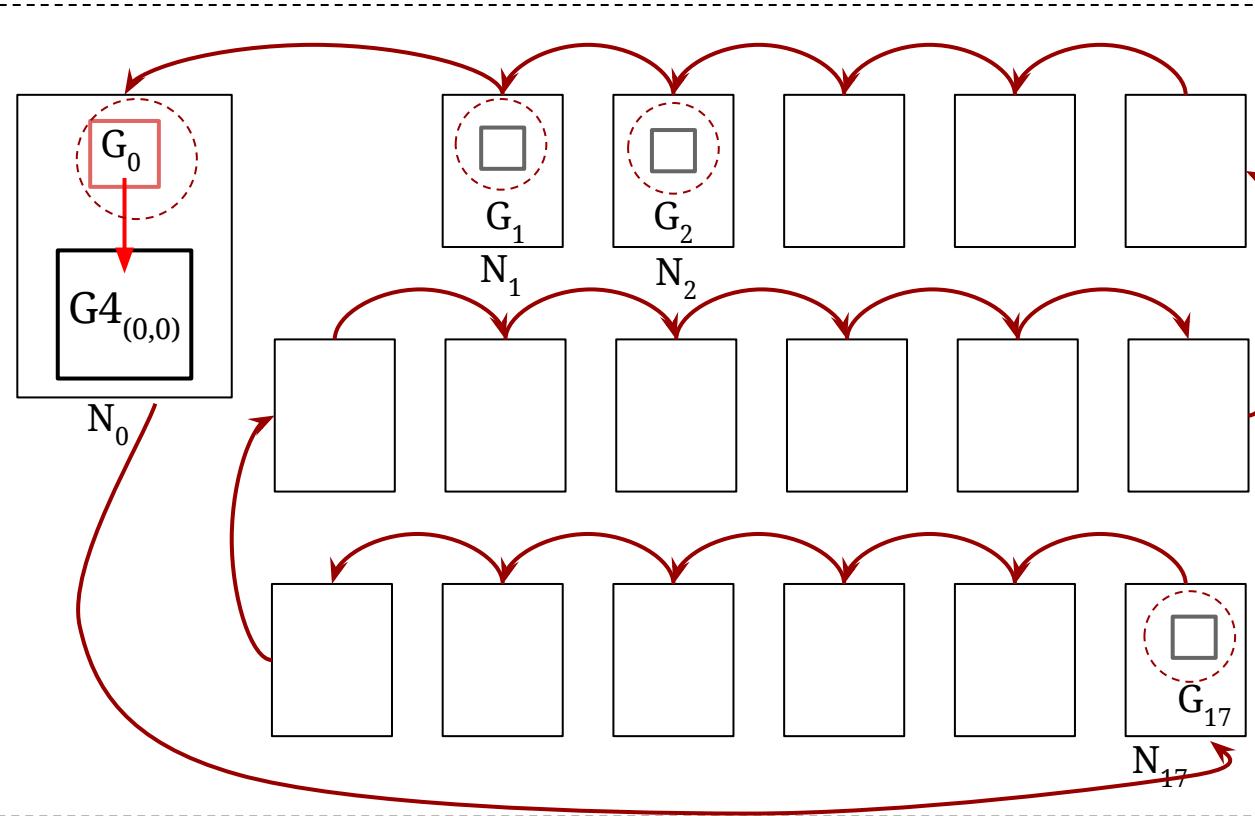
code: https://github.com/weilewei/Ring_example_MPI_CUDA/blob/master/G2_ring.cpp

Pipelined Ring Algorithm

Time-step 3
(final step)



Enabling MORE science



If G4 size remains same, say 12 GB, now each GPU now only carries
 $\frac{1}{\text{no. of GPUs}} * 12 \text{ GB}$

Amount of GPU memory in a rack we can tap into:
145x more if needed
(6 GPUs/node * 16GB * 18 nodes = 1.7 TB,
1.7 TB / 12 GB = 145x)

Conclusion & Future Work

- Threading abstraction w/ HPX light-weight threads
- **DCA++ w/ GPUDirect:**
 - Conducted NVLink bandwidth measurement
 - Future work
 - To apply GPUDirect in DCA++ to solve memory bound issue
 - Wrap GPUDirect into HPX future

Pipelined Ring Algorithm using HPX

```
24  
25 hpx::mpi::enable_user_polling enable_polling;  
26 hpx::mpi::executor exec(MPI_COMM_WORLD);  
27  
  
69 for(int icount=0; icount < (mpi_size-1); icount++)  
70 {  
71 // encode the originator rank in the message tag as tag = 1 + originator_irank  
72 int originator_irank = MOD(((rank-1)-icount + 2*mpi_size), mpi_size);  
73 int recv_tag = 1 + originator_irank;  
74 recv_tag = 1 + MOD(recv_tag-1, MPI_TAG_UB); // just to be safe, then 1 <= tag <= MPI_TAG_UB  
75  
76 hpx::future<int> f_send = hpx::async(exec, MPI_Irecv, recvbuff_G2, n_elems, MPI_FLOAT, left_neighbor,  
recv_tag);  
77 hpx::future<int> f_recv = hpx::async(exec, MPI_Isend, sendbuff_G2, n_elems, MPI_FLOAT, right_neighbor,  
send_tag);  
78  
79 f_recv.get();  
80 CudaMemoryCopy(G2, recvbuff_G2, n_elems);  
81 update_local_G4(G2, G4, rank, n_elems);  
82 f_send.get();  
83  
84 // get ready for send  
85 CudaMemoryCopy(sendbuff_G2, G2, n_elems);  
86 send_tag = recv_tag;  
87 }
```

HPX-MPI

• • •

19

```
61 for(int icount=0; icount < (mpi_size-1); icount++)  
62 {  
63 // encode the originator rank in the message tag as tag = 1 + originator_irank  
64 int originator_irank = MOD(((rank-1)-icount + 2*mpi_size), mpi_size);  
65 int recv_tag = 1 + originator_irank;  
66 recv_tag = 1 + MOD(recv_tag-1, MPI_TAG_UB); // just to be safe, then 1 <= tag <= MPI_TAG_UB  
67  
68 MPI_CHECK(MPI_Irecv(recvbuff_G2, n_elems, MPI_FLOAT, left_neighbor, recv_tag, MPI_COMM_WORLD,  
&recv_request));  
69 MPI_CHECK(MPI_Isend(sendbuff_G2, n_elems, MPI_FLOAT, right_neighbor, send_tag, MPI_COMM_WORLD,  
&send_request));  
70  
71 MPI_CHECK(MPI_Wait(&recv_request, &status));  
72 CudaMemoryCopy(G2, recvbuff_G2, n_elems);  
73 update_local_G4(G2, G4, rank, n_elems);  
74 MPI_CHECK(MPI_Wait(&send_request, &status)); // wait for sendbuf_G2 to be available again  
75  
76 // get ready for send  
77 CudaMemoryCopy(sendbuff_G2, G2, n_elems);  
78 send_tag = recv_tag;  
79 }
```

MPI

Enabling Parallel Abstraction Layer to DCA++ using HPX And GPUDirect

Weile Wei

Masters Student, Louisiana State University

ex-intern, ORISE, Oak Ridge National Lab (Fall 19)

wwei9@lsu.edu

To follow slides: <http://tiny.cc/msdefense>