

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

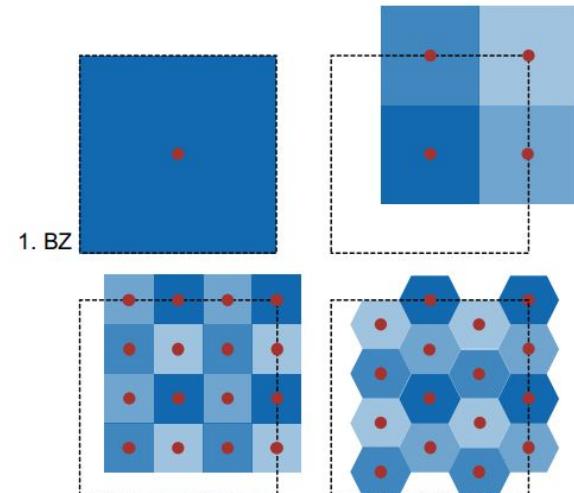
- What is DCA++? -- Scientific software for Dynamical Cluster Approximation
- Build threading abstraction layer using HPX
- Use GPUDirect to solve memory bound issue in DCA++

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).

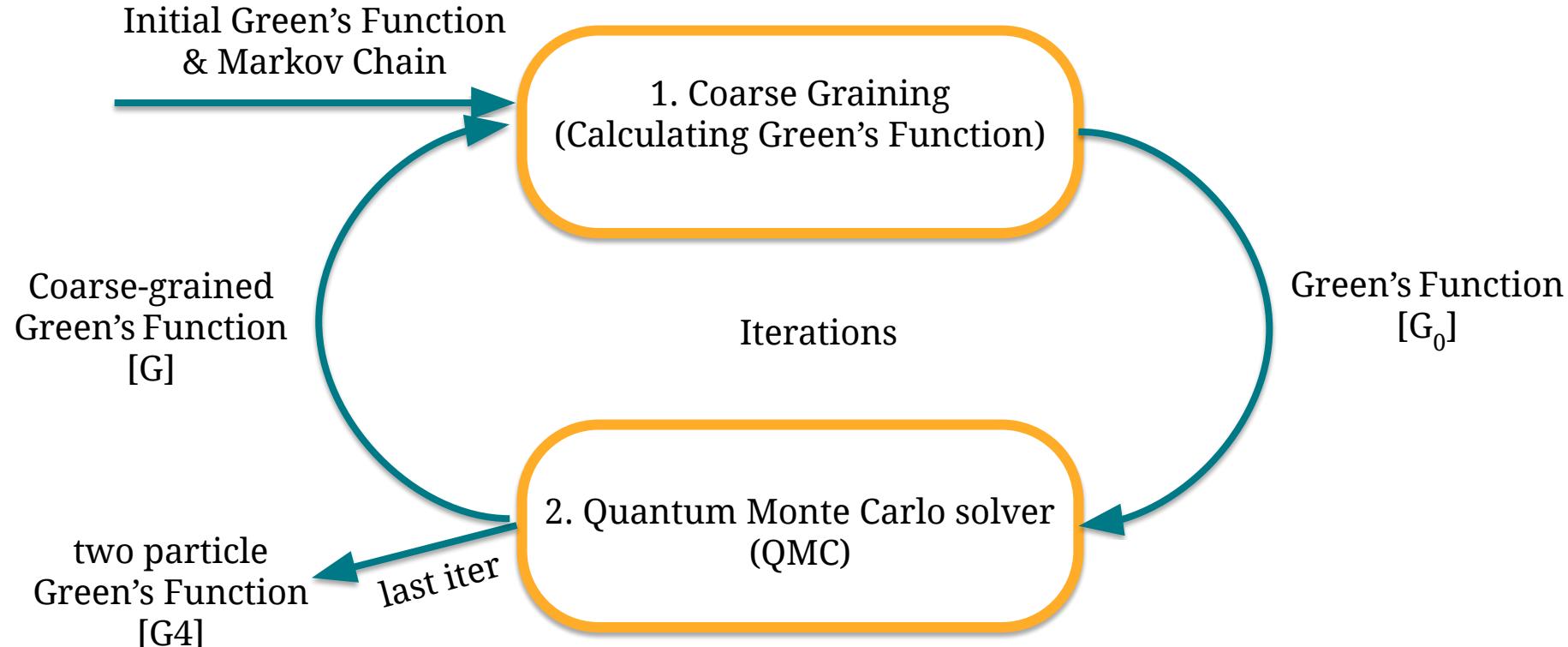
DCA++ is a numerical simulation software to solve correlated electron problems with Quantum Monte Carlo method.



[1] DCA++ 2019. Dynamical Cluster Approximation. <https://github.com/CompFUSE/DCA> [Licensing provisions: BSD-3-Clause]

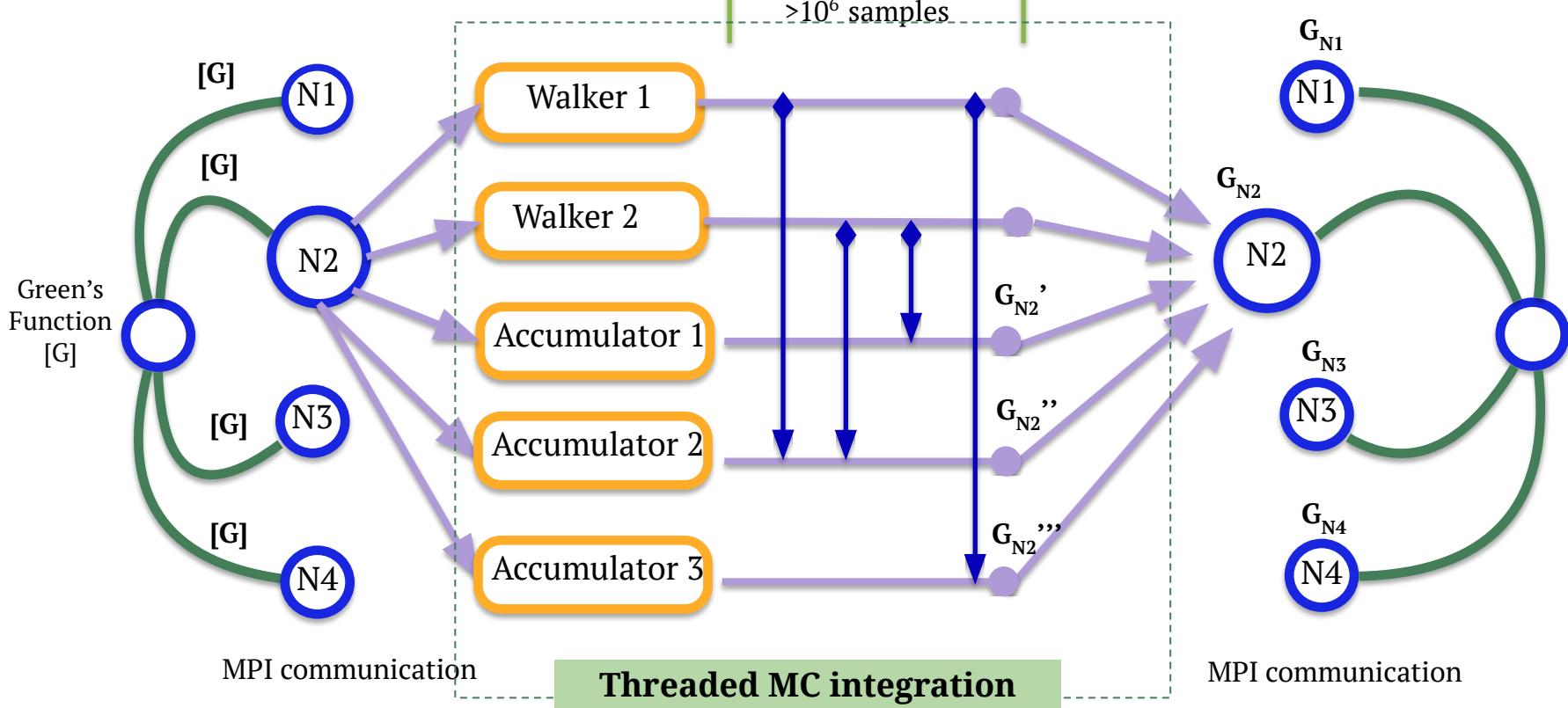
[2] Urs R. Hähner, Gonzalo Alvarez, Thomas A. Maier, Raffaele Solcà, Peter Staar, Michael S. Summers, and Thomas C. Schulthess, DCA++: A software framework to solve correlated electron problems with modern quantum cluster methods, *Comput. Phys. Commun.* 246 (2020) 106709.

DCA++: Primary kernels workflow



Baldazzi, Giovanni, Arghya Chatterjee, Ying Wai Li, Peter W. Doak, Urs Haehner, Ed F. D'Azevedo, Thomas A. Maier, and Thomas Schulthess. "Accelerating DCA++ (Dynamical Cluster Approximation) Scientific Application on the Summit Supercomputer." In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 433-444. IEEE, 2019.

DCA++ : Quantum Monte Carlo Solver



Baldazzi, Giovanni, Arghya Chatterjee, Ying Wai Li, Peter W. Doak, Urs Haehner, Ed F. D'Azevedo, Thomas A. Maier, and Thomas Schulthess. "Accelerating DCA++ (Dynamical Cluster Approximation) Scientific Application on the Summit Supercomputer." In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 433-444. IEEE, 2019.

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

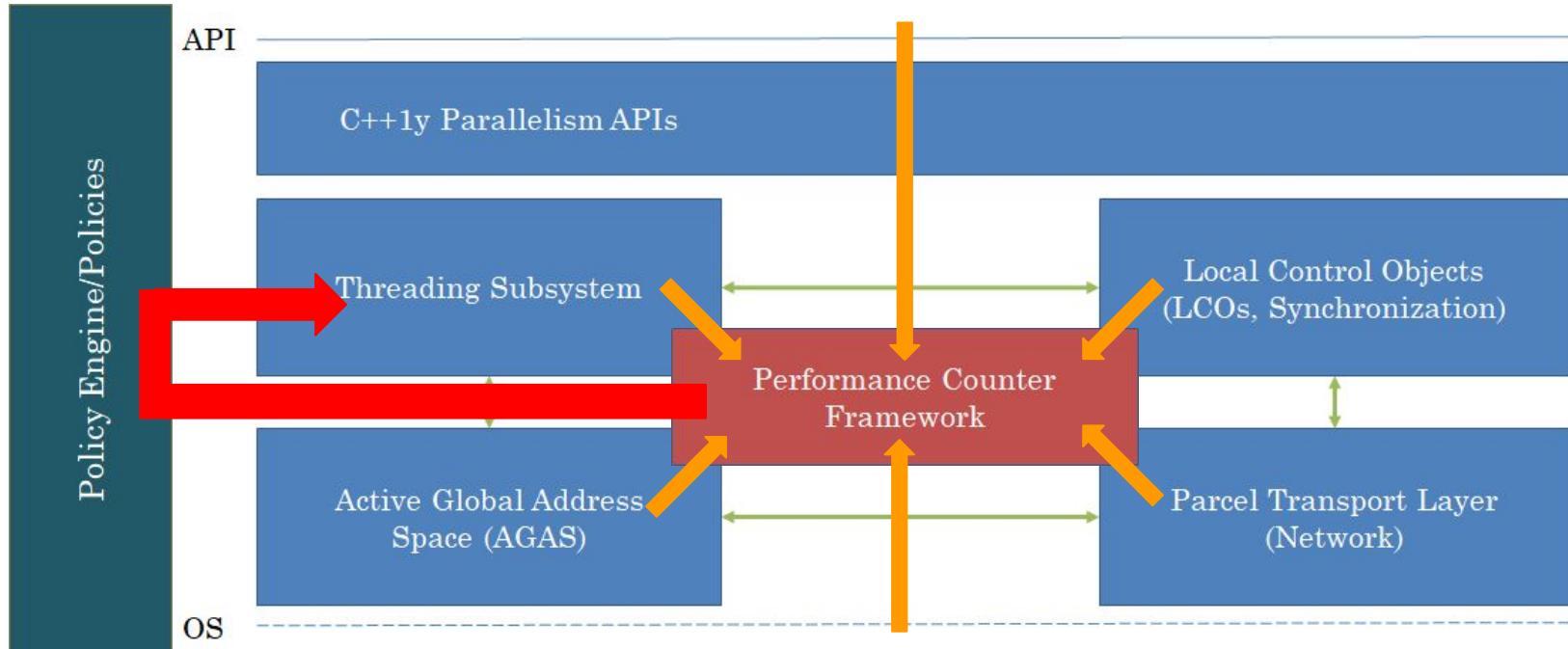
-DHPX_DIR=\$HPX_PATH
-DDCA_WITH_HPX=ON

switch by user-input at compile time

Threading Abstraction

DCA++

HPX - High-Performance ParalleX



Kaiser, Hartmut, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey. "Hpx: A task based programming model in a global address space." In Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, pp. 1-11. 2014.

HPX - A General Purpose Runtime System

- Widely portable
 - **Platforms:** x86/64, Xeon/Phi, ARM 32/64, **Power**, BlueGene/Q
 - **Operating systems:** Linux, Windows, Android, OS/X
- Unified and standard-conforming C++ API and more ...
- Explicit support for hardware accelerators and vectorization
- *Boost license* and has an open, active, and thriving developer community
- Backend support for legacy codes at DoE, DoD, NSF, and more ...
- Domains: Astrophysics, Coastal Modeling, Distributed Machine Learning

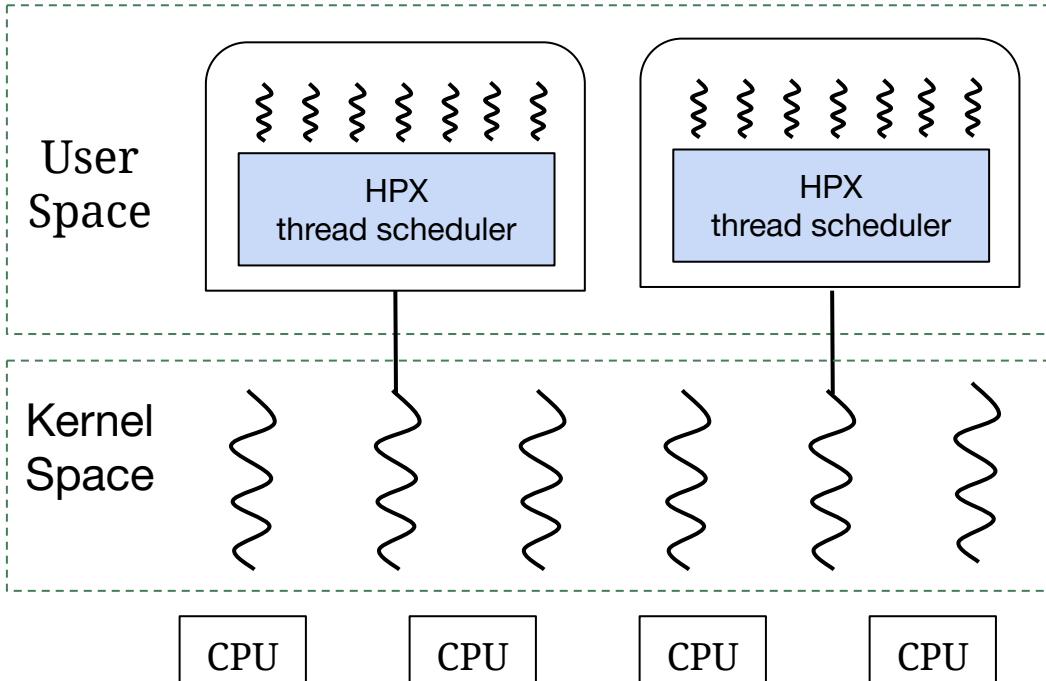
HPX - C++ standard compliant and more

- 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/ standard thread

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

auto& pool = dca::parallel::ThreadPool::get_instance();

for (int i = 0; i < thread_task_handler_.size(); ++i) {
    if (thread_task_handler_.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
std::vector<hpx::future<void>> futures;

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

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

    // else if handle other conditions...
}

}
```

QMC solver w/ threading abstraction

```
162
163 - std::vector<std::future<void>> futures;
164
165     dca::profiling::WallTime start_time;
166
167 - auto& pool = dca::parallel::ThreadPool::get_instance();
168     for (int i = 0; i < thread_task_handler_.size(); ++i) {
169         if (thread_task_handler_.getTask(i) == "walker")
170 -         futures.emplace_back(pool.enqueue(&ThisType::startWalker, this, i));
171         else if (thread_task_handler_.getTask(i) == "accumulator")
172 -         futures.emplace_back(pool.enqueue(&ThisType::startAccumulator, this, i));
173         else if (thread_task_handler_.getTask(i) == "walker and accumulator")
174 -         futures.emplace_back(pool.enqueue(&ThisType::startWalkerAndAccumulator, this,
175             i));
176         else
177             throw std::logic_error("Thread task is undefined.");
178     }
179
180     @@ -188,8 +188,9 @@ void StdThreadQmciclusterSolver<QmcicSolver>::integrate() {
181
182     };
183
184     try {
185         for (auto& future : futures)
186             future.get();
187
188     }
189
190     catch (std::exception& err) {
191         print_metadata();
192
193     }
194
195     catch (std::exception& err) {
196         print_metadata();
197
198     }
199
200     std::vector<dca::parallel::thread_traits::future_type<void>> futures;
201
202     dca::profiling::WallTime start_time;
203
204     // auto& pool = dca::parallel::ThreadPool::get_instance();
205     for (int i = 0; i < thread_task_handler_.size(); ++i) {
206         if (thread_task_handler_.getTask(i) == "walker")
207             futures.emplace_back(hpx::async(&ThisType::startWalker, this, i));
208         else if (thread_task_handler_.getTask(i) == "accumulator")
209             futures.emplace_back(hpx::async(&ThisType::startAccumulator, this, i));
210         else if (thread_task_handler_.getTask(i) == "walker and accumulator")
211             futures.emplace_back(hpx::async(&ThisType::startWalkerAndAccumulator, this, i));
212
213     }
214
215     else
216         throw std::logic_error("Thread task is undefined.");
217
218     }
219
220     hpx::wait_all(futures);
221
222 }
```

Threading Abstraction

std::thread

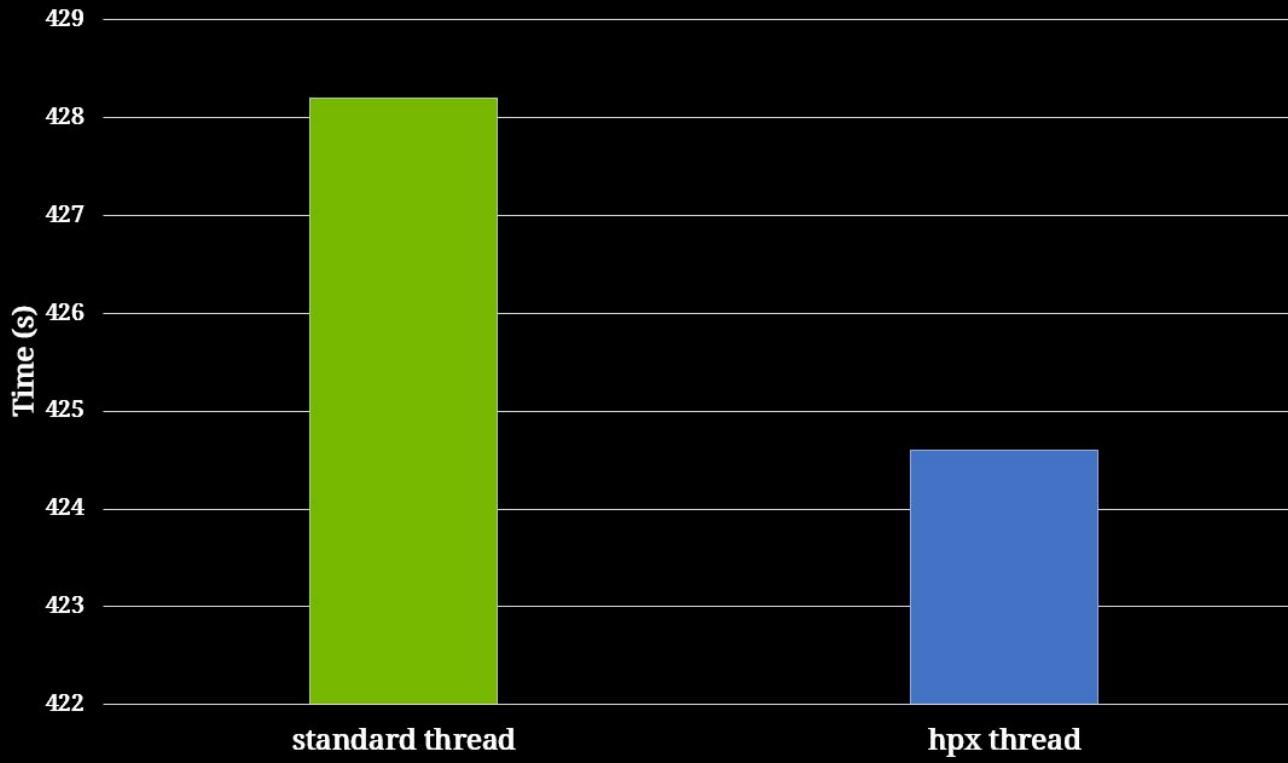
```
namespace dca {  
namespace parallel {  
  
struct thread_traits {  
    template <typename T>  
    using future_type      = std::future<T>;  
    using mutex_type       = std::mutex;  
    using condition_variable_type =  
        std::condition_variable;  
    using scoped_lock      =  
     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

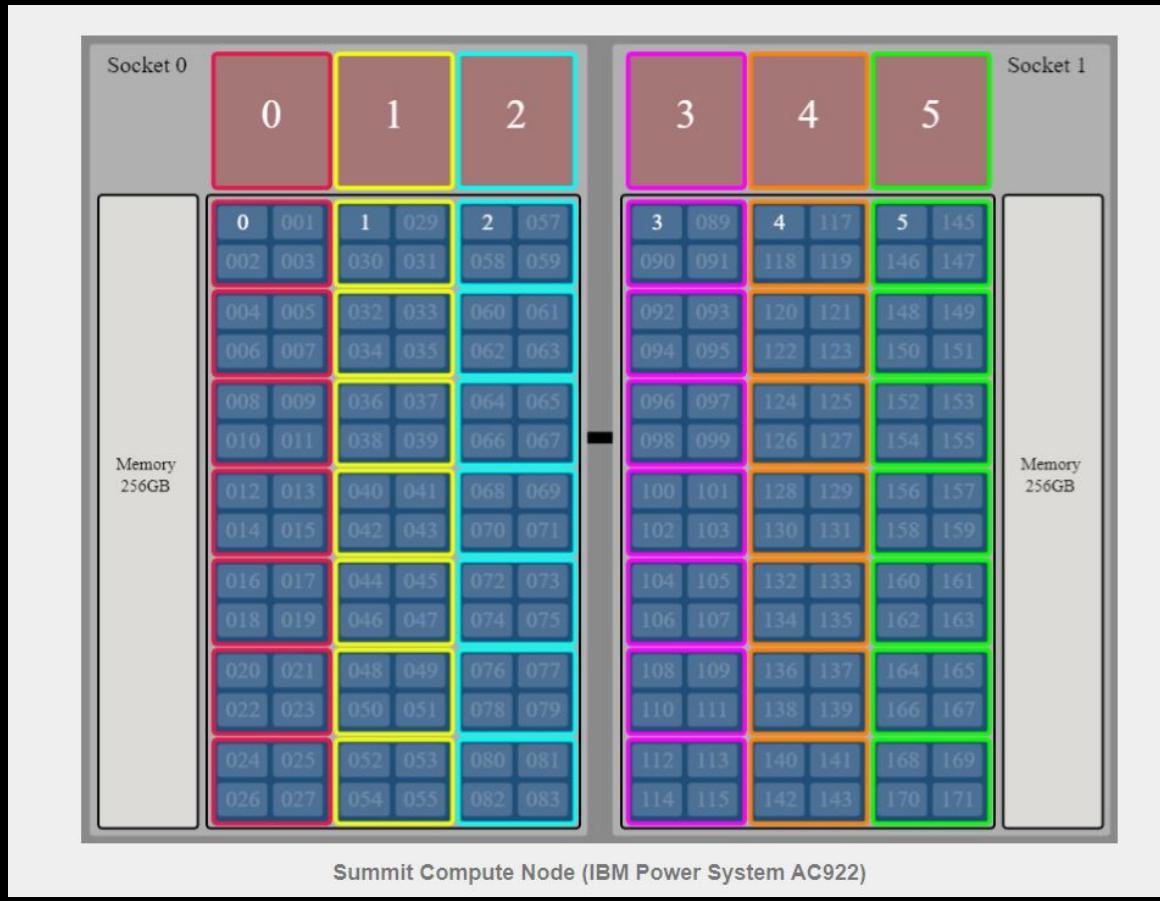
HPX thread

```
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
```

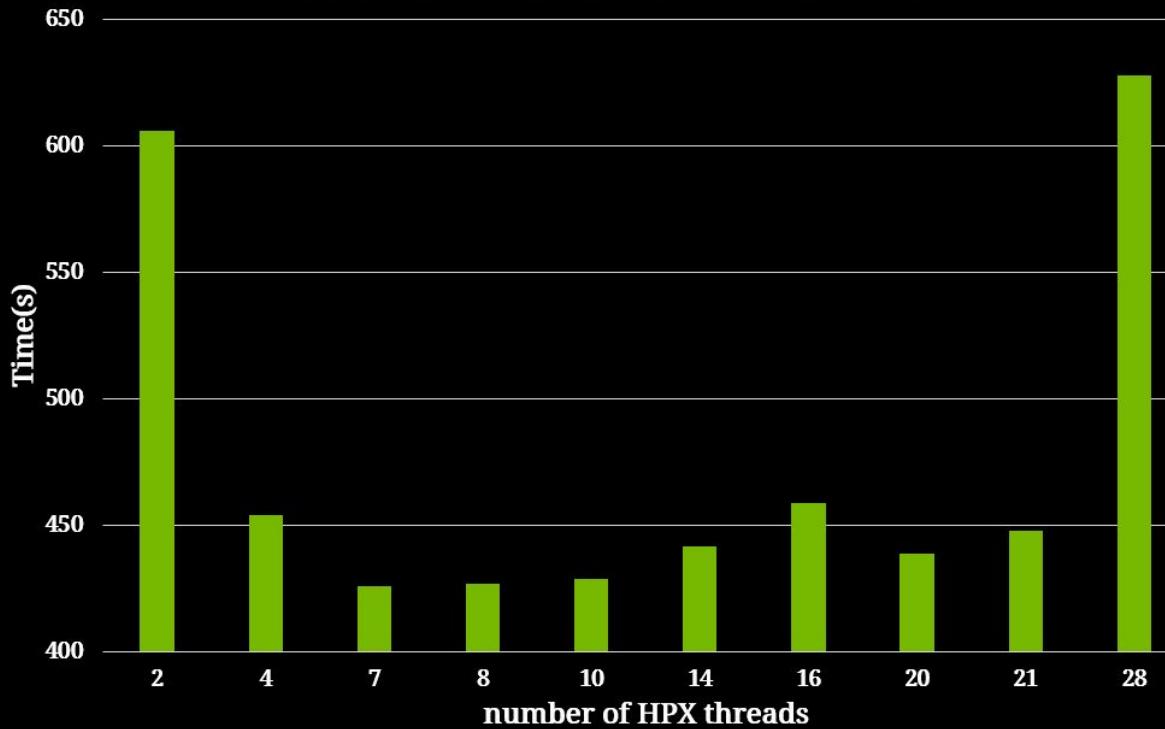
Performance Measurement

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





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



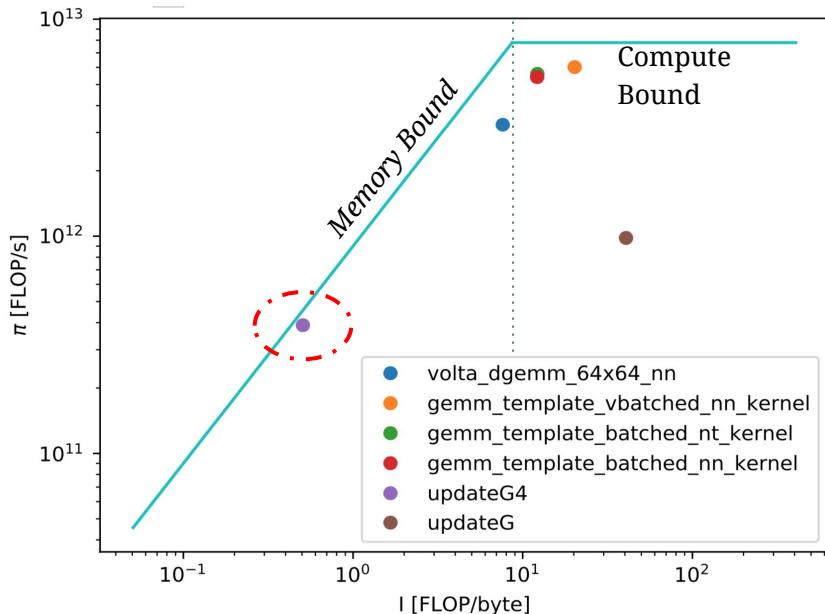
Sub-summary

- **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

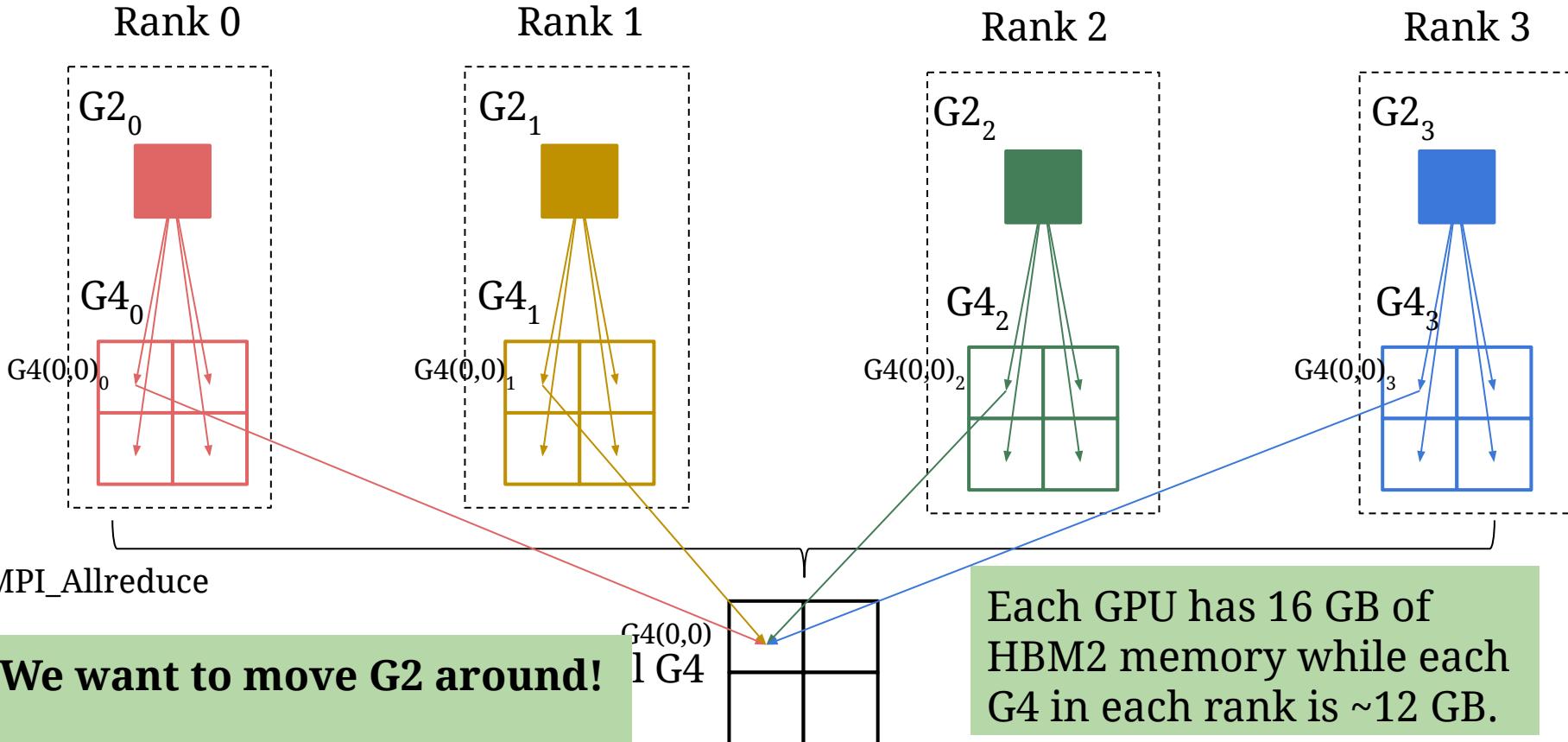


Solution: *broadcasting* each $G_2[][]$ matrix to all other ranks:

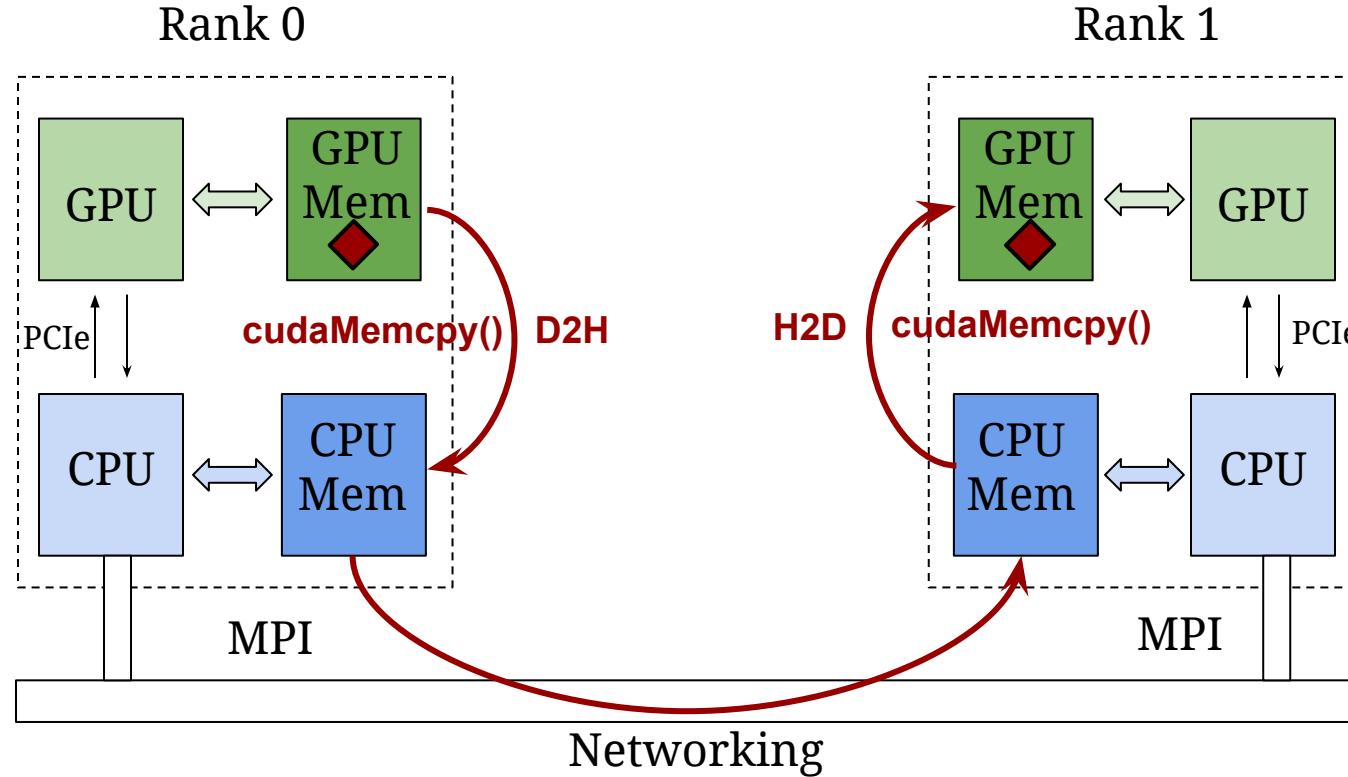
- Regular MPI method
- NVLink method

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

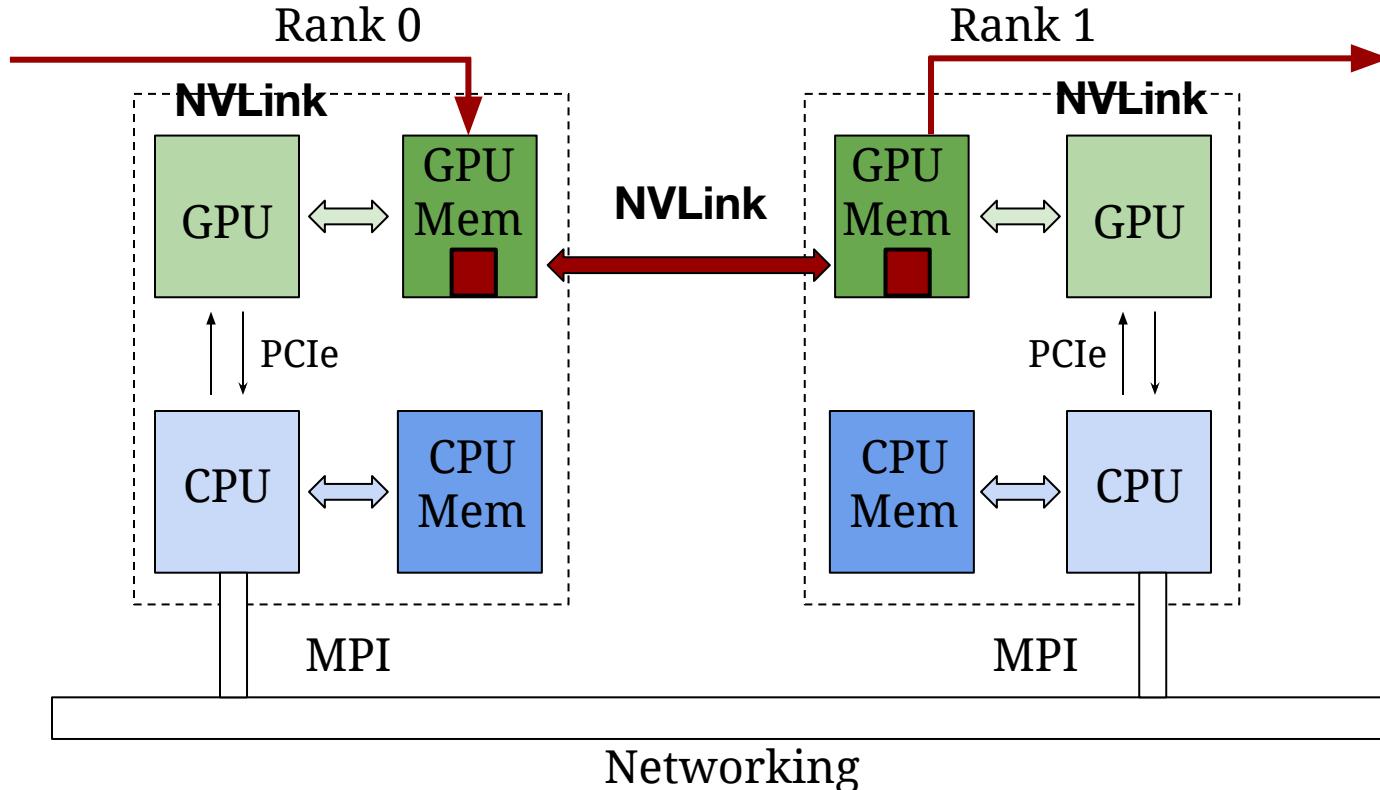
Memory bound issue w/ G4



Moving G2 around: Regular MPI method



Moving G2 around: NVLink method

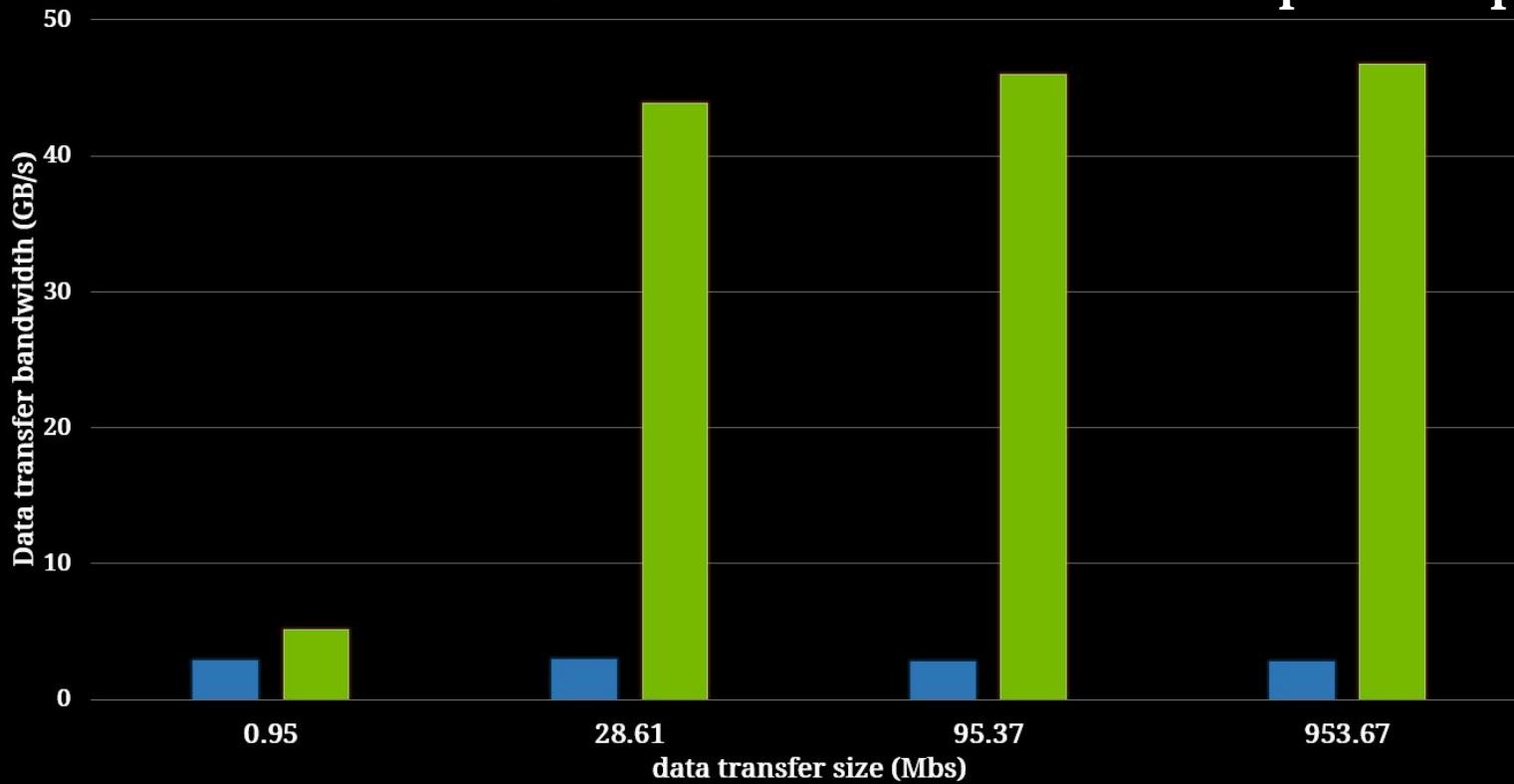


Bandwidth Measurement

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

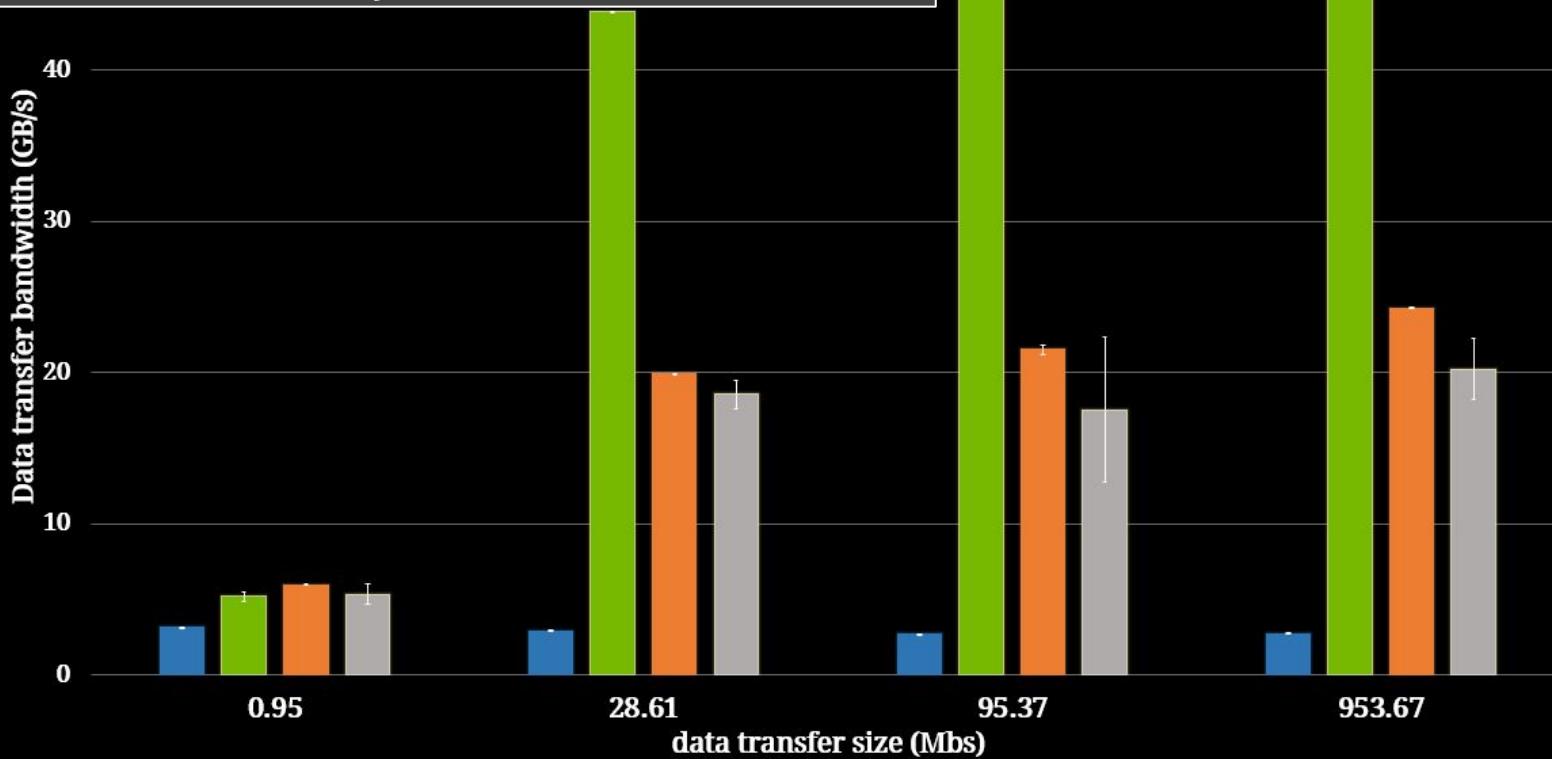
■ Regular MPI GPU to Remote GPU ■ NVLink

up to 17x speedup

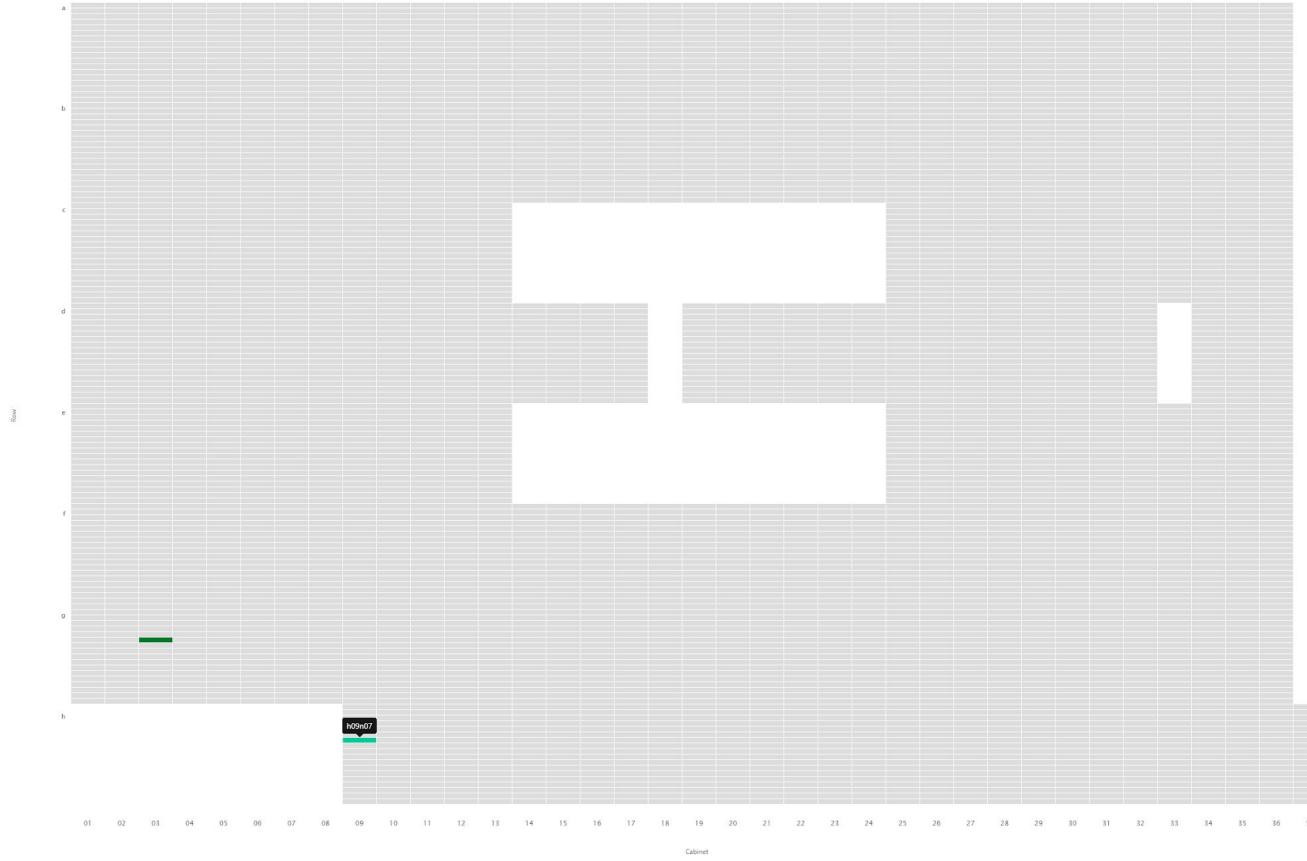


Compare bandwidth of data transfer on Summit using NVLink

- 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



Allocated: Used by job step

Allocated: Not used by job step

Not Allocated

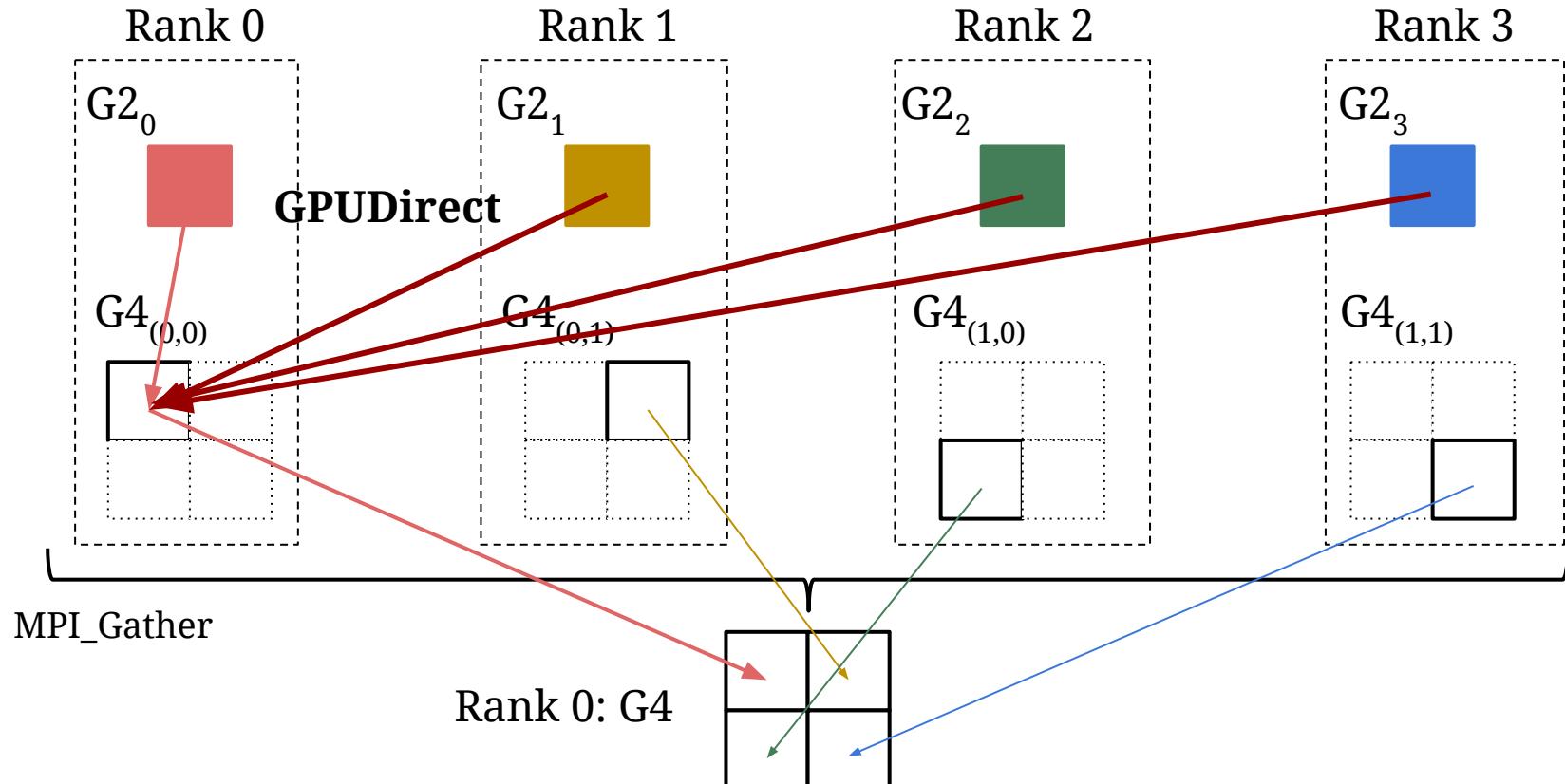
Nodes upgrade needed

Heterogeneous Scheme: row(cabinets[0]-18]

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

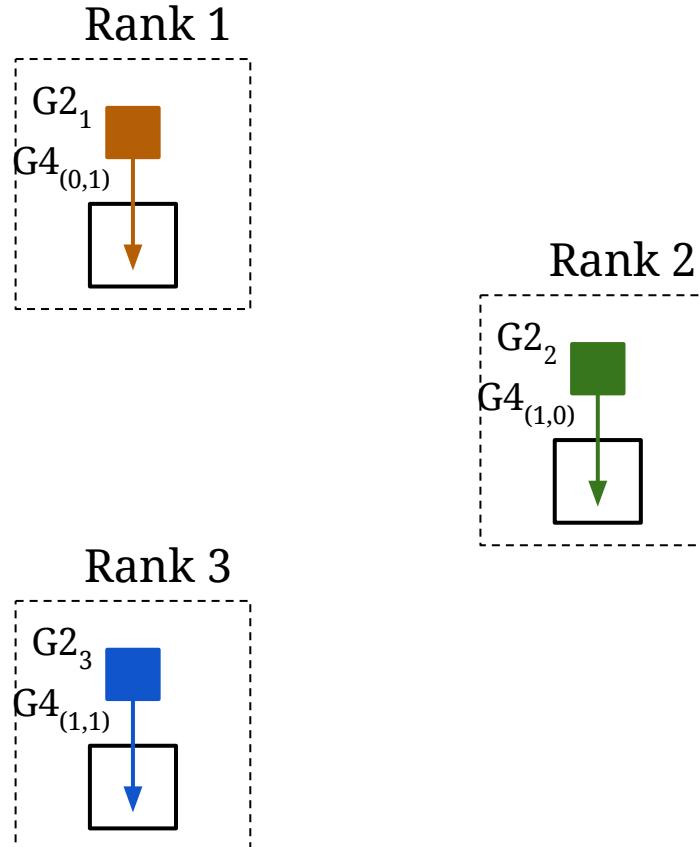
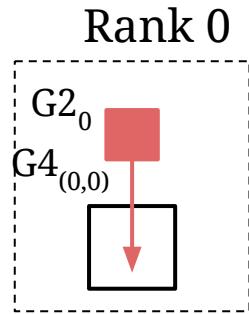
Powered by

G4 w/ NVIDIA's NVLink enabled

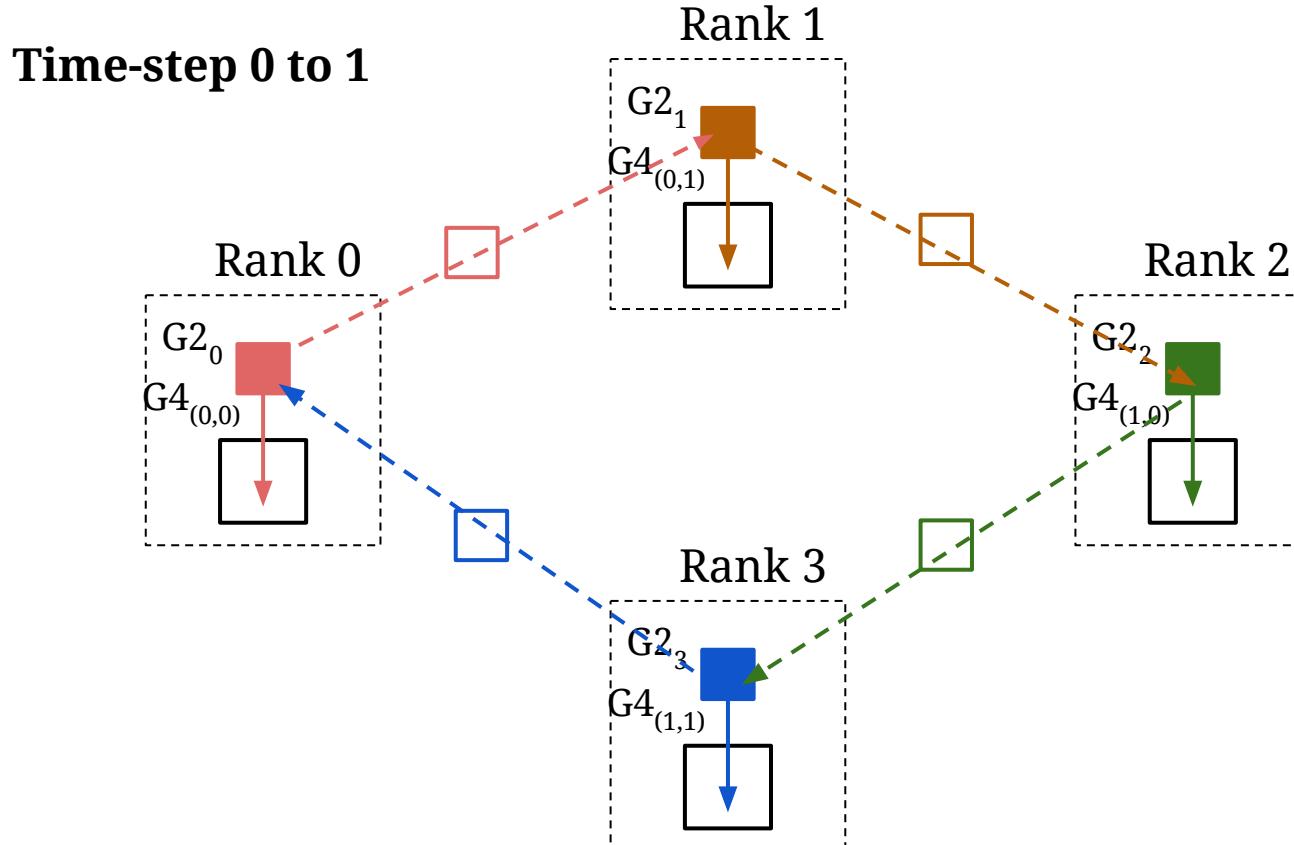


Pipelined Ring Algorithm

Time-step 0

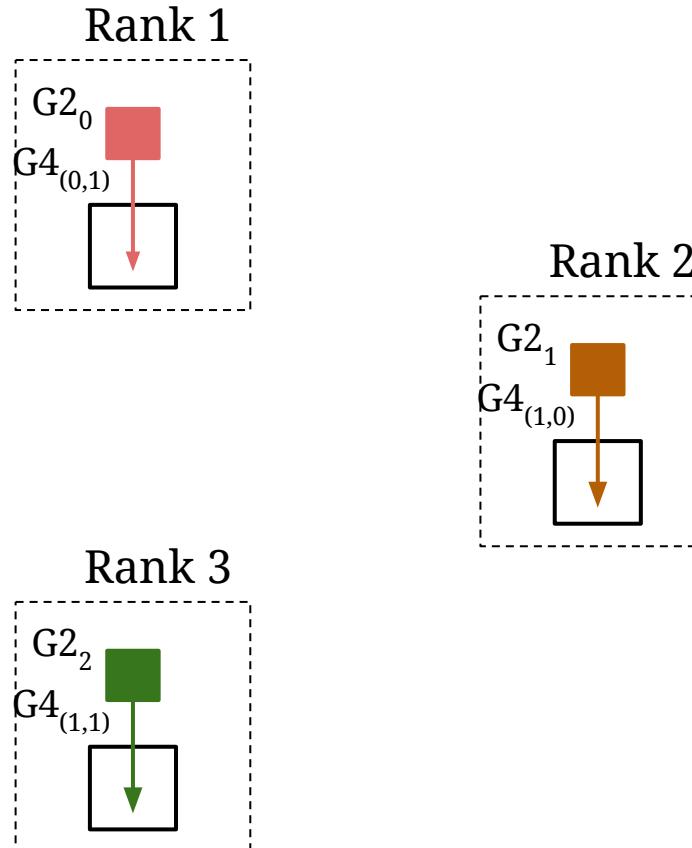
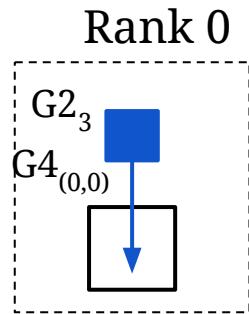


Pipelined Ring Algorithm



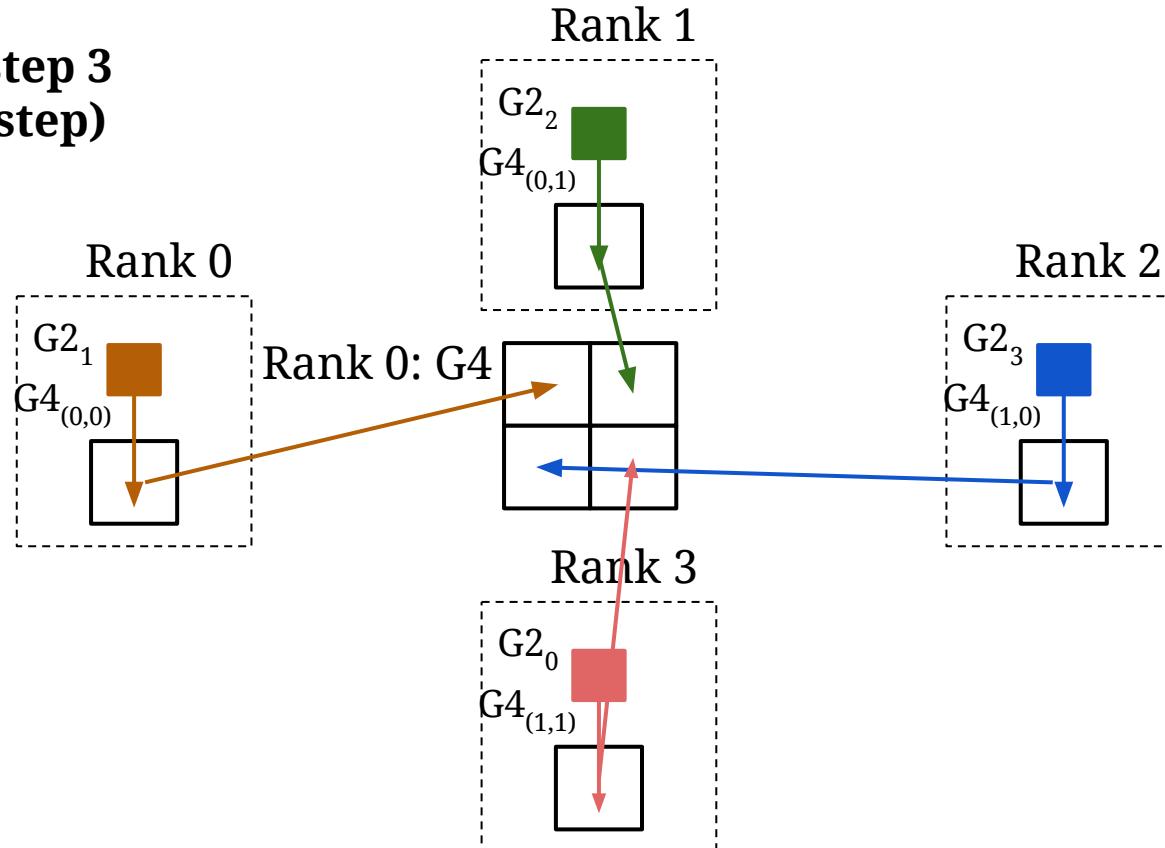
Pipelined Ring Algorithm

Time-step 1

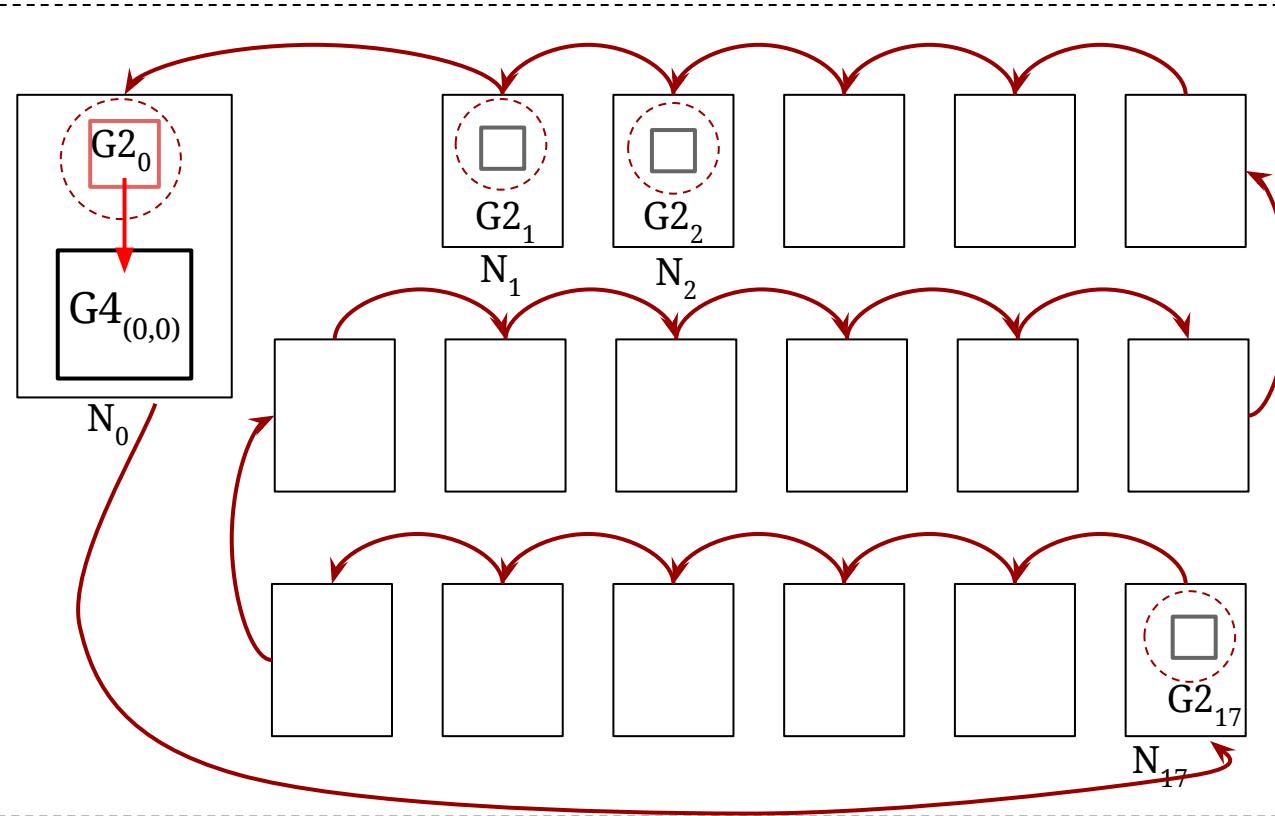


Pipelined Ring Algorithm

Time-step 3
(final step)



Enabling MORE science



Each G4 cell/rank
only carries

$$\frac{1}{\text{no. of GPUs}} * 12 \text{ GB}$$

Amount of GPU memory
we can tap into:

144x more if needed
(6*16GB*18 → 1.7 TB)

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

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

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