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 **Sci**entific **D**iscovery through **A**dvanced **C**omputing (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 **O**ak Ridge **L**eadership **C**omputing **F**acility 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

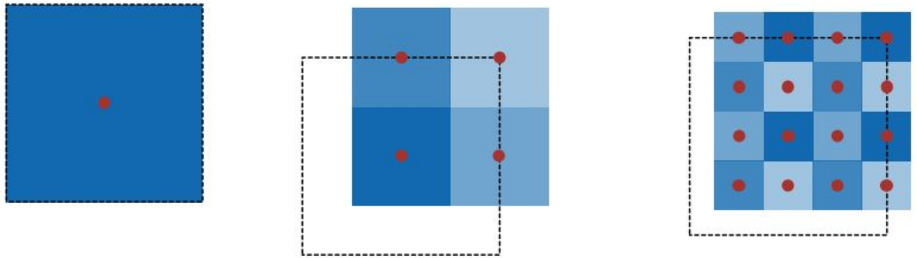
- What is DCA++? -- Scientific software for Dynamical Cluster Approximation
- Build threading abstraction layer using HPX
- Use GPUDirect to solve memory bound issue in DCA++

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.

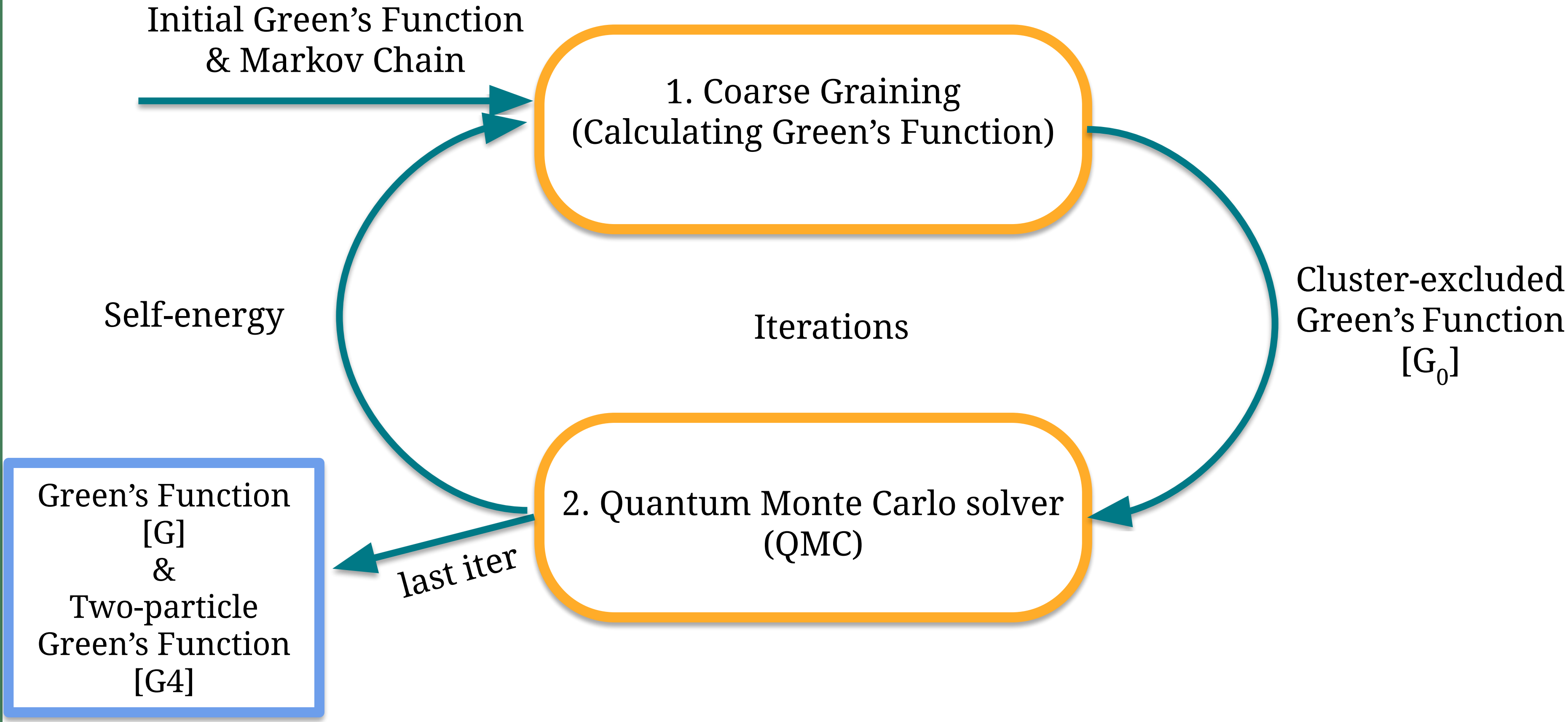


[1] DCA++ Github: https://github.com/CompFUSE/DCA
[2] Hähner, Urs R., *et al*. Computer Physics Communications 246 (2020): 106709.

OAK RIDGE National Laboratory

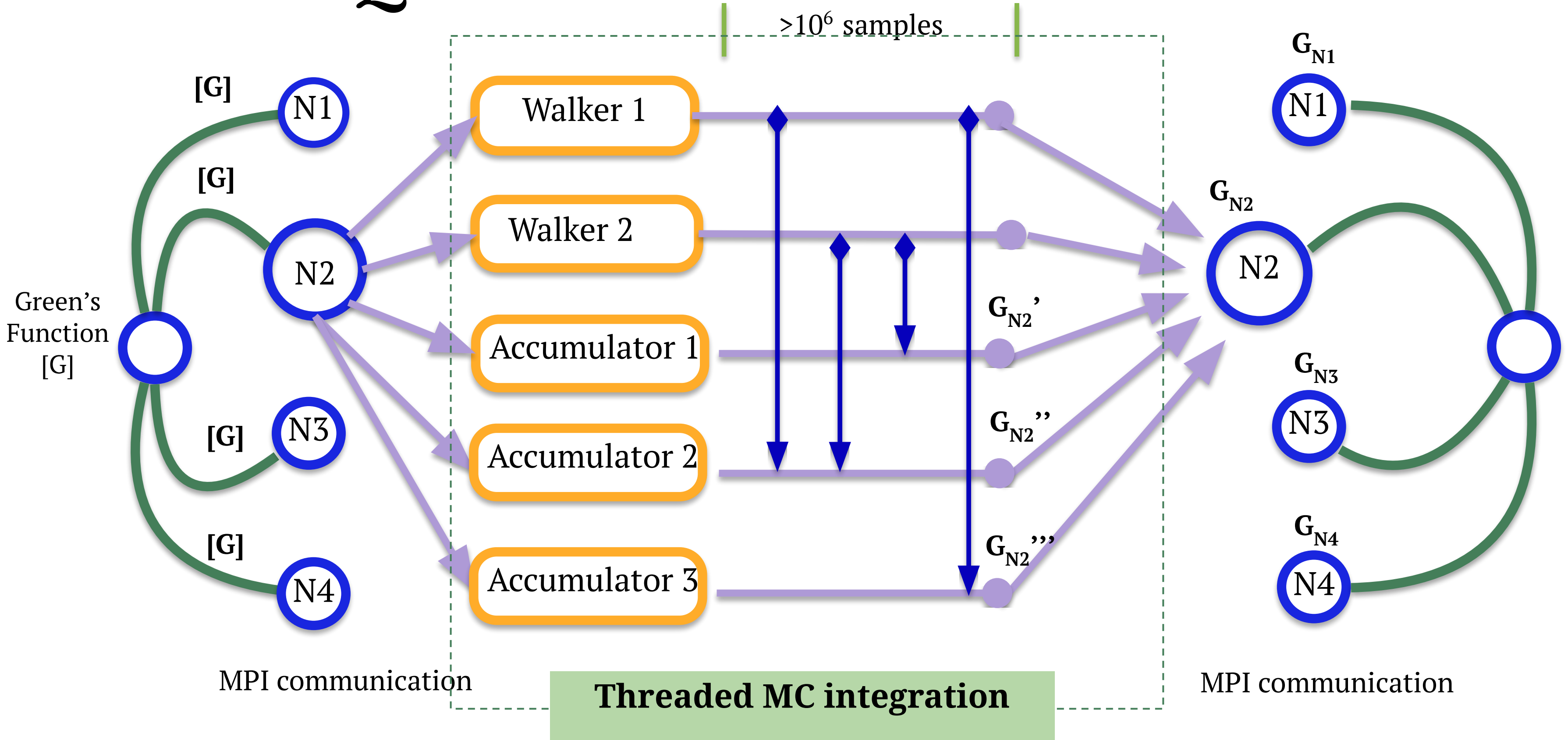
STE||AR GROUP

DCA++: Primary kernels workflow



Balduzzi, Giovanni, *et al.* In 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT) (pp. 433-444). IEEE.

DCA++ : Quantum Monte Carlo Solver



Walkers samples data, accumulator collects measurement

Balduzzi, Giovanni, *et al.* In 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT) (pp. 433-444). IEEE.

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

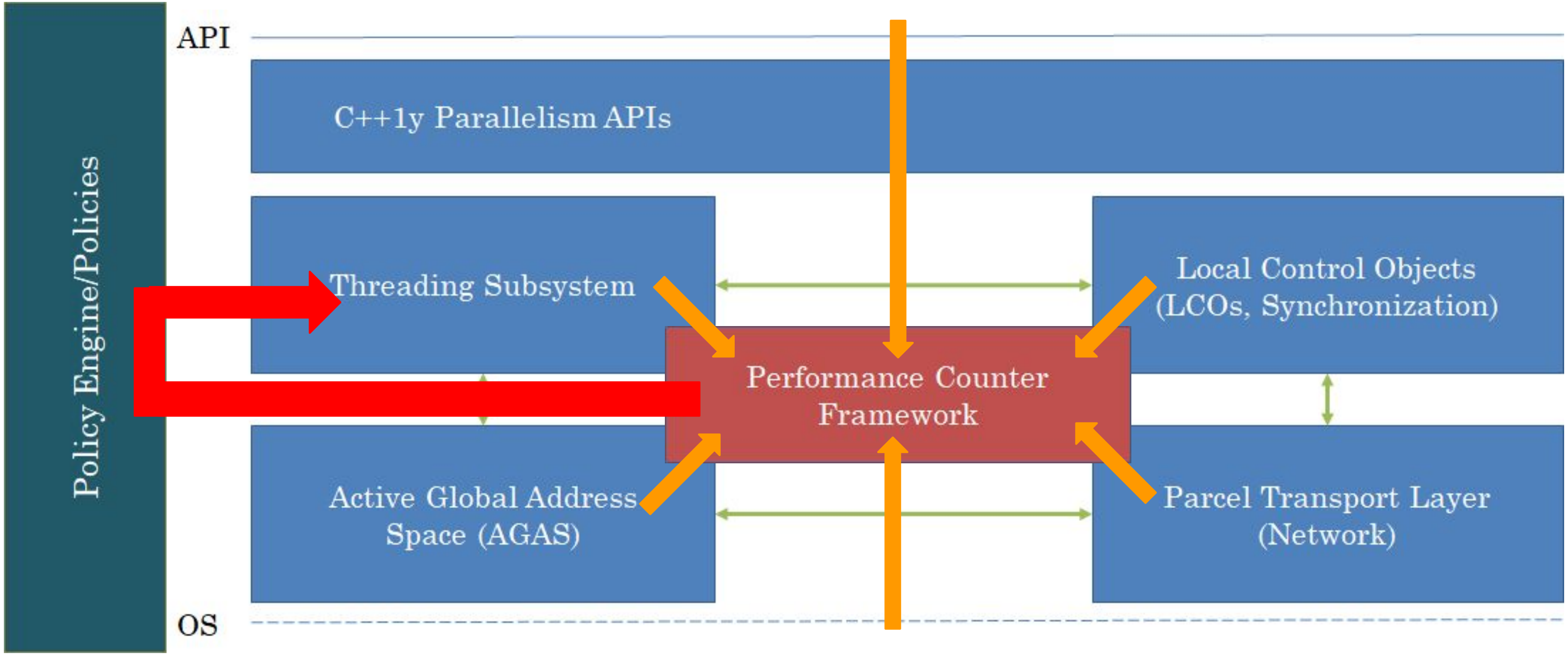
-DHPX_DIR=$HPX_PATH
-DDCA_WITH_HPX=ON

switch by user-input at compile time

Threading Abstraction

DCA++

HPX - High-Performance ParalleX



Kaiser, Hartmut, *et al.* 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 (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

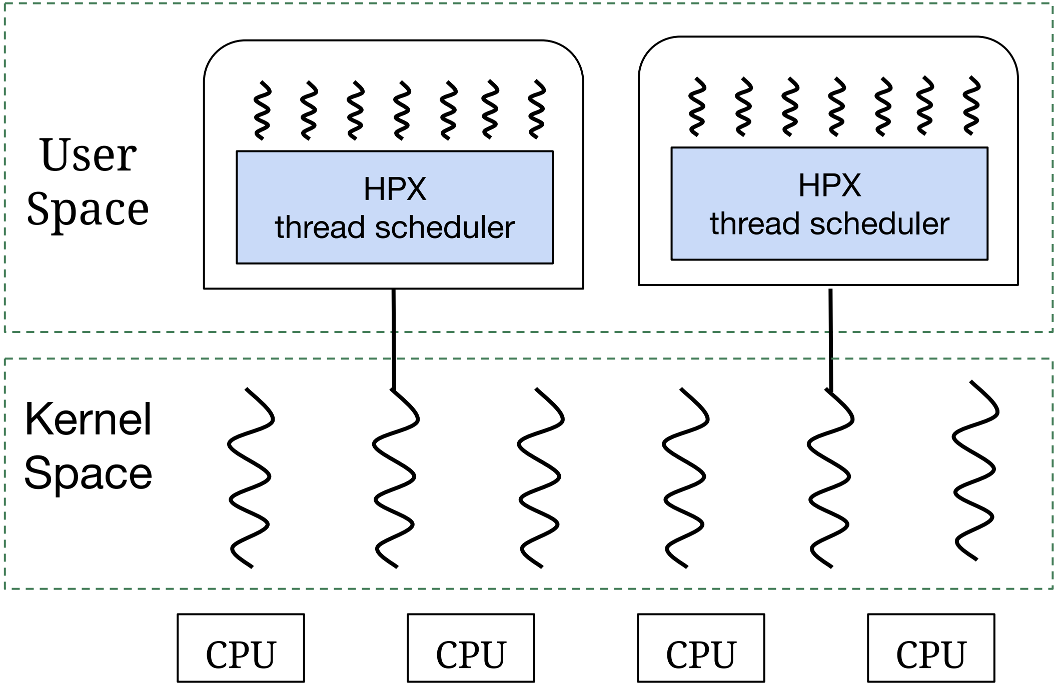
HPX - C++ standard compliant and more

- 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

Zhang, Tianyi, *et al*. In Proceedings of the International Workshop on OpenCL, pp. 1-10. 2019.

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...
}
```


OAK RIDGE
National Laboratory

STE||AR GROUP

Threading Abstraction

std::thread

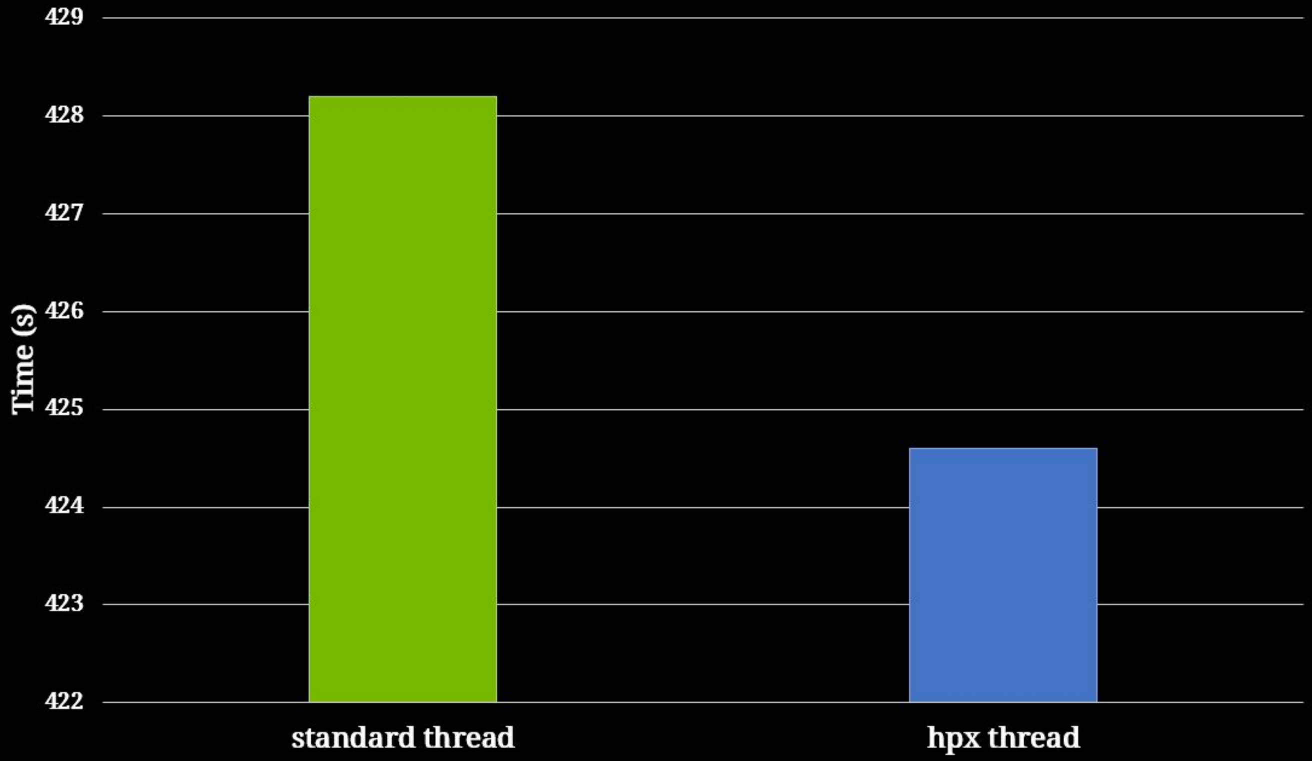
```
namespace dca {
namespace parallel {

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

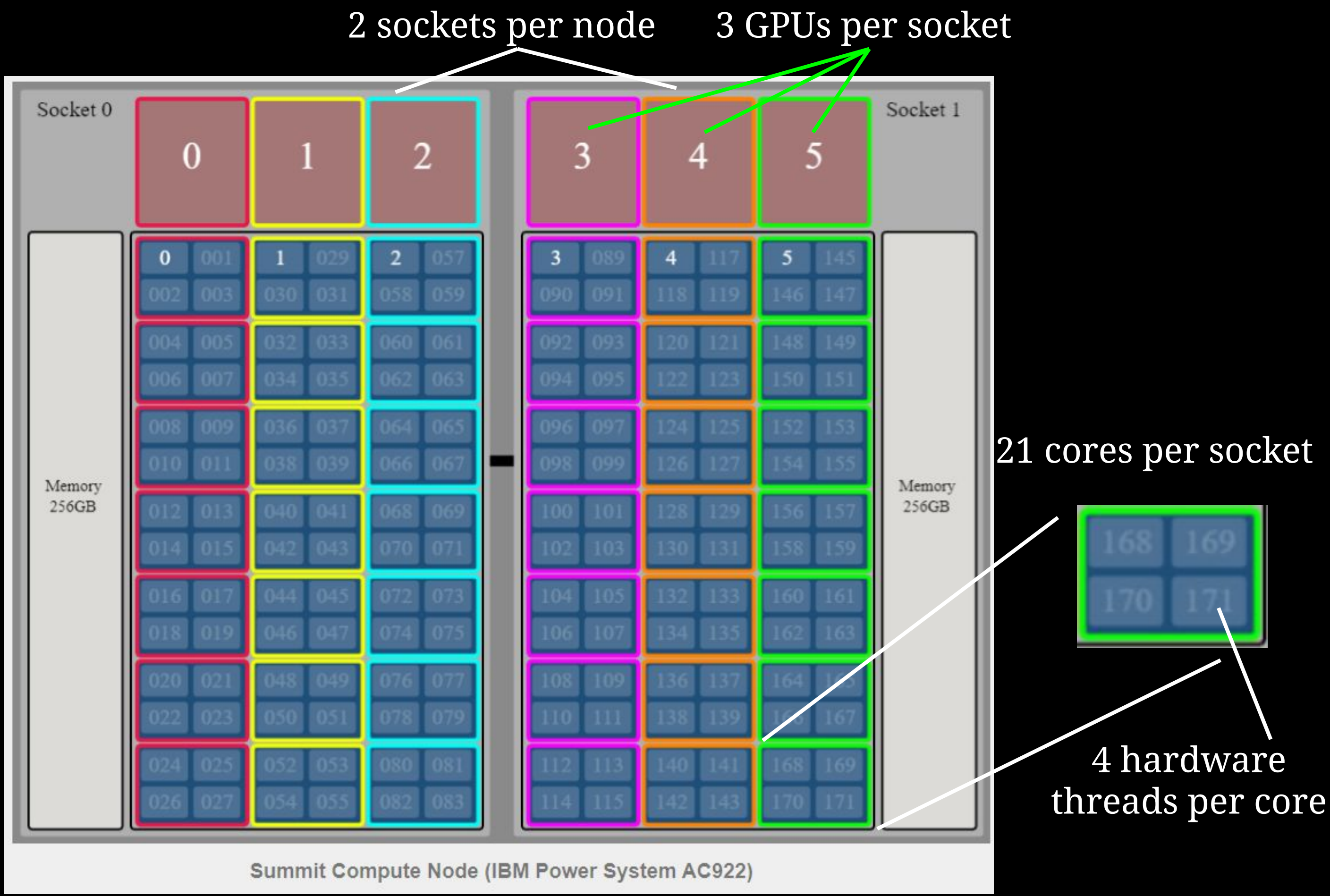
```
namespace dca {
namespace parallel {

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)



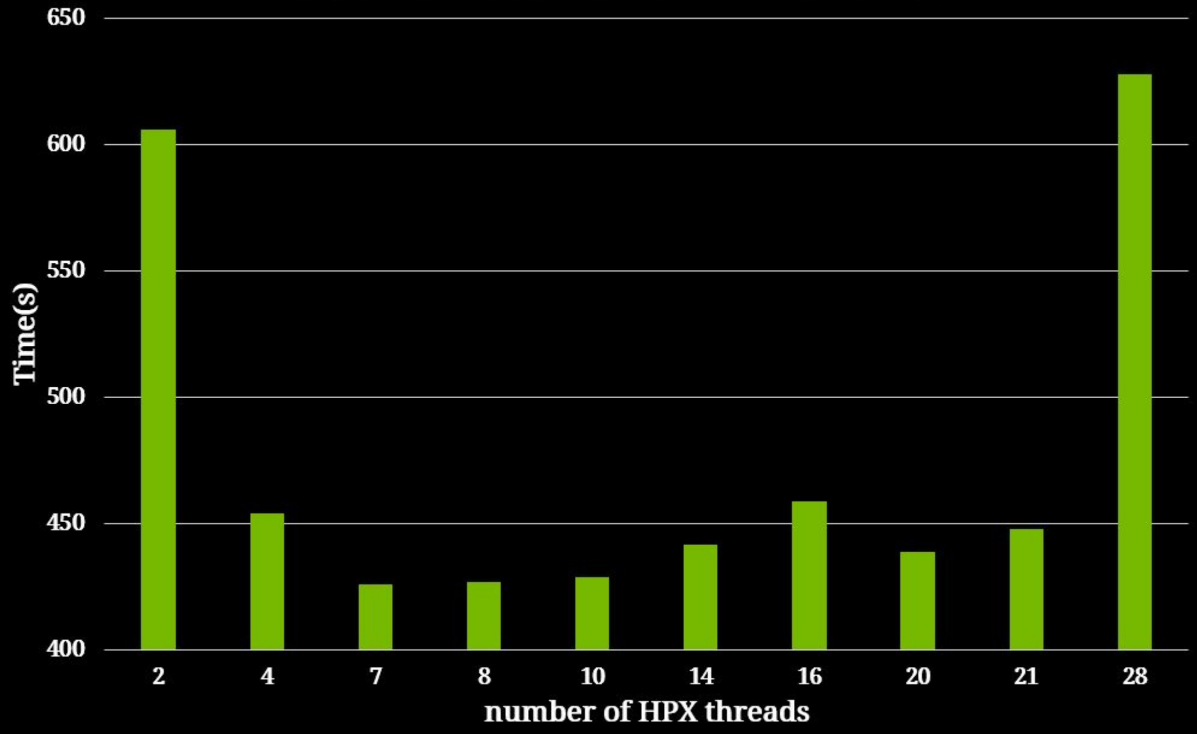
- DCA only allows 1 GPU per rank
- DCA strategy:
    - (1 gpu + 7 cpus) per rank
    - 7 threads only



https://jsrunvisualizer.olcf.ornl.gov/?s1f1o01n6c7g1r11d1b21l0=

https://docs.olcf.ornl.gov/systems/summit_user_guide.html

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

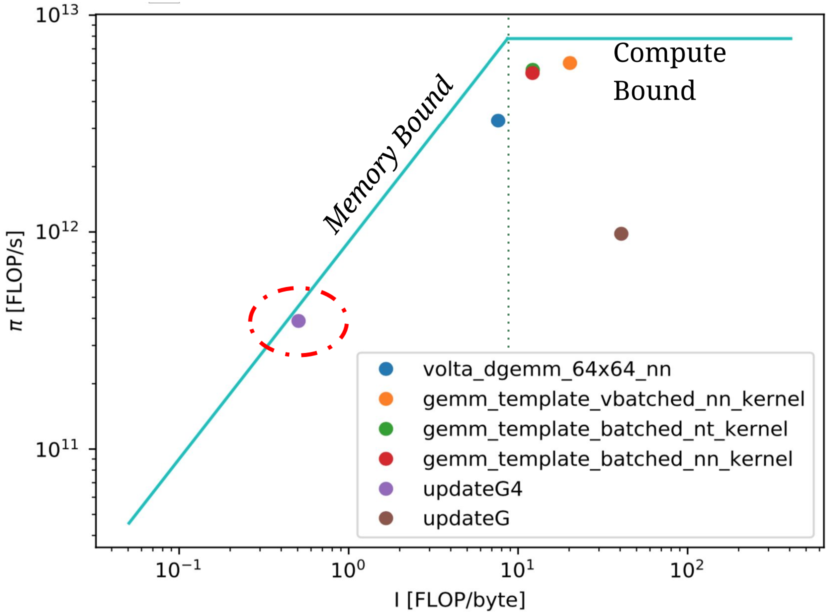


Fig. : Roofline plot of a NVIDIA V100 GPU running DCA++ at production level on Summit (OLCF)

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

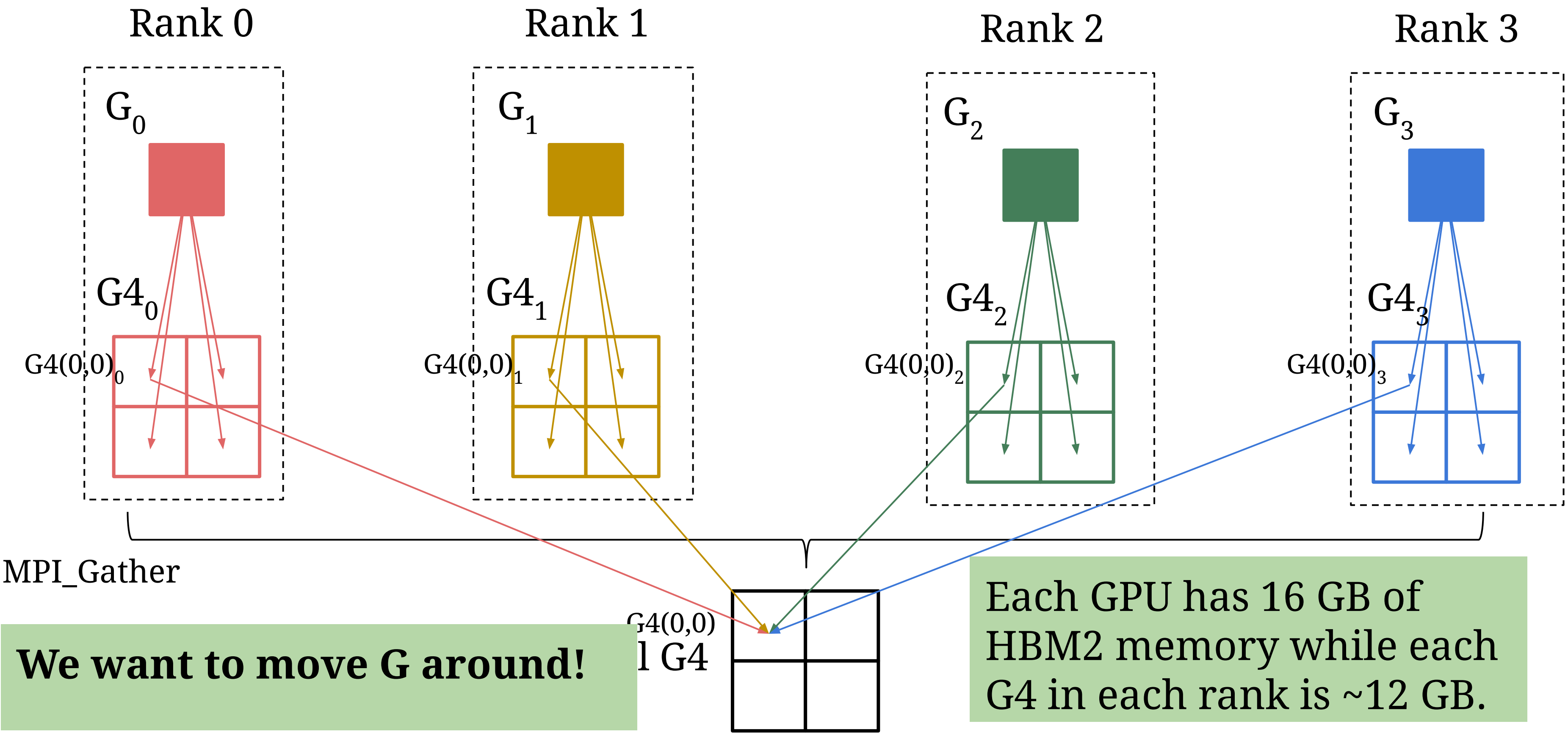
- Regular MPI method
- NVLink method

Oak Ridge National Laboratory

Balduzzi, Giovanni, *et al.* In 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT) (pp. 433-444). IEEE.

STE||AR GROUP

Memory bound issue w/ G4



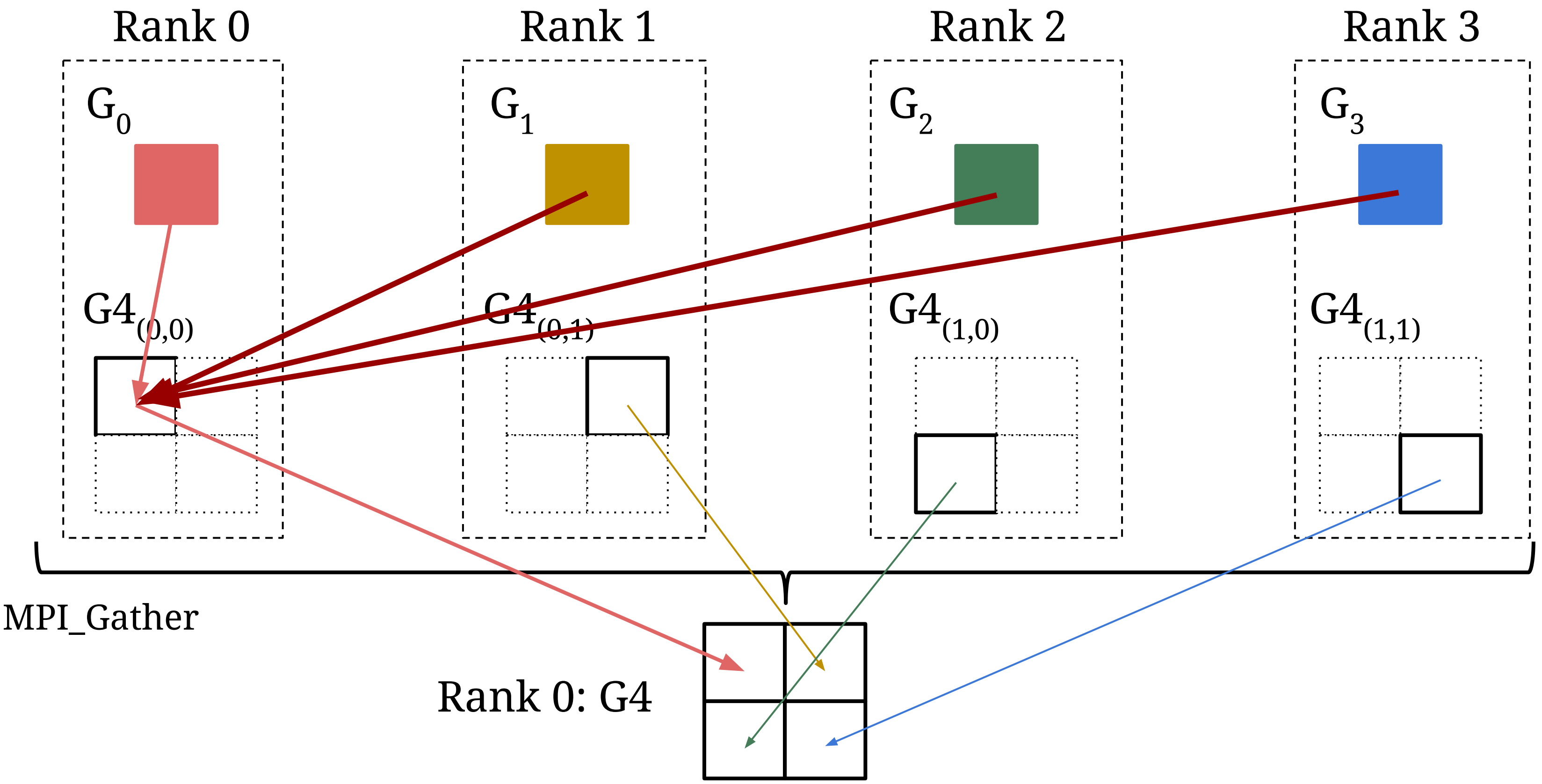
We want to move G around!

Each GPU has 16 GB of HBM2 memory while each G4 in each rank is ~12 GB.

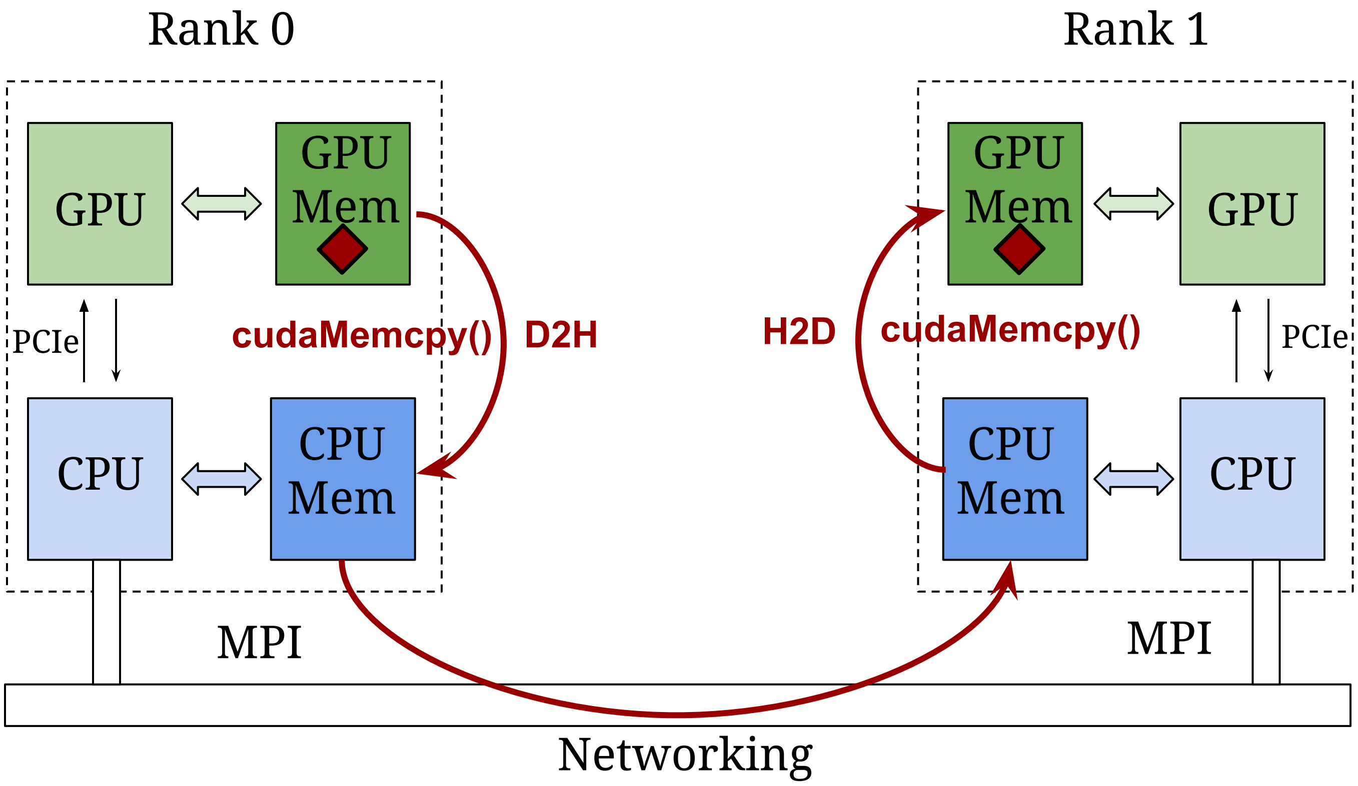
OAK RIDGE National Laboratory

STE||AR GROUP

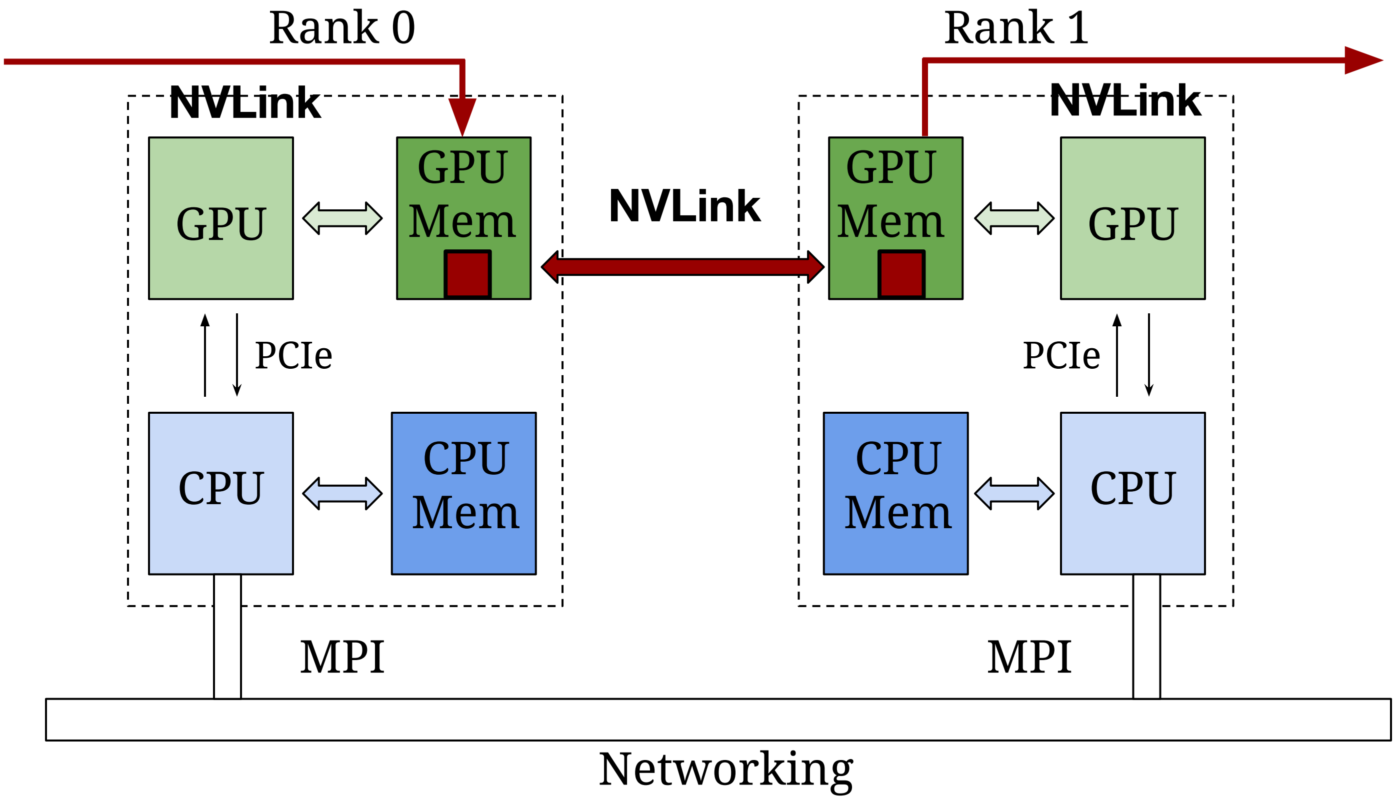
Move G2 around



Moving G around: Regular MPI method



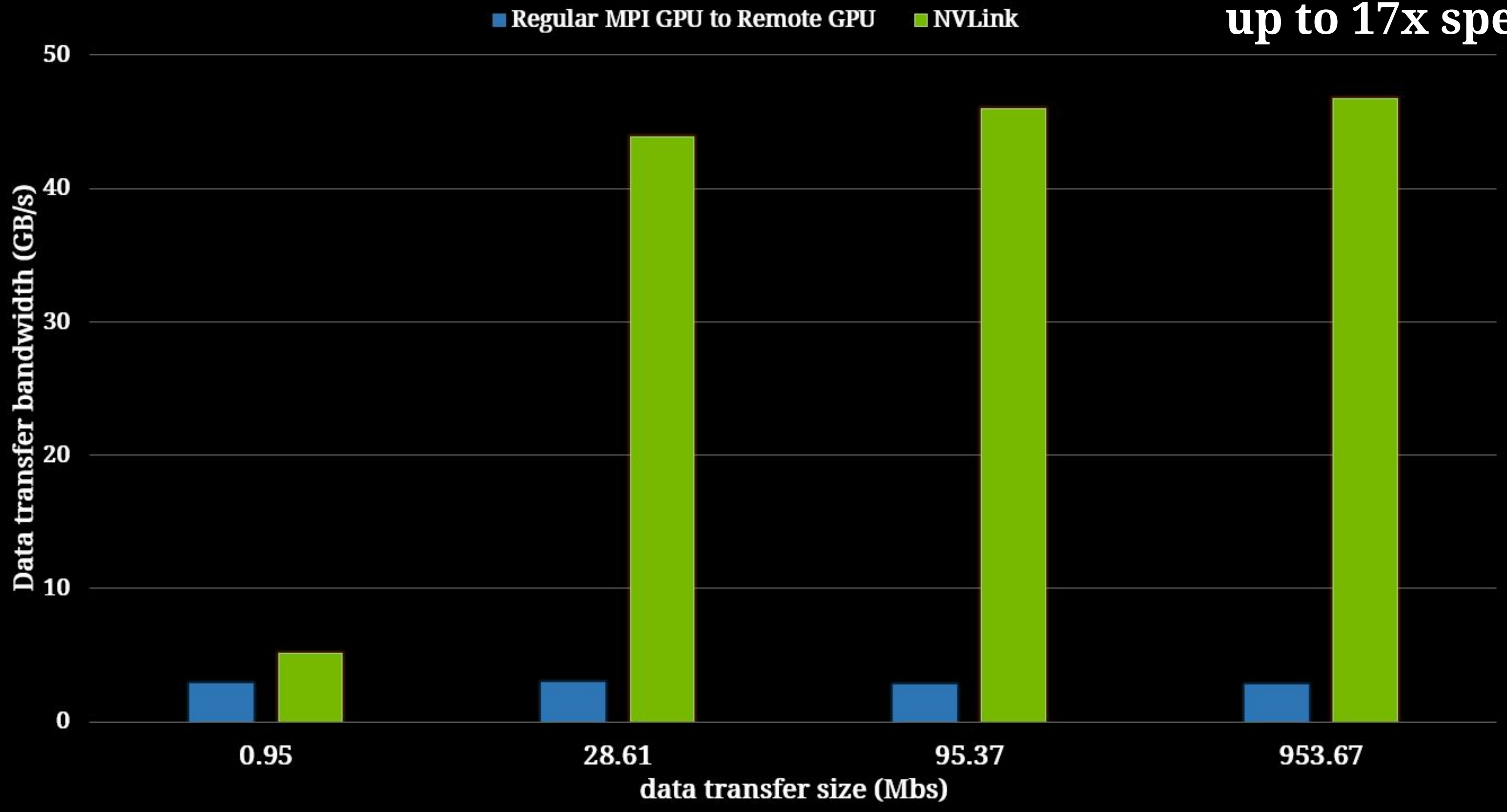
Moving G around: NVLink method



Bandwidth Measurement

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

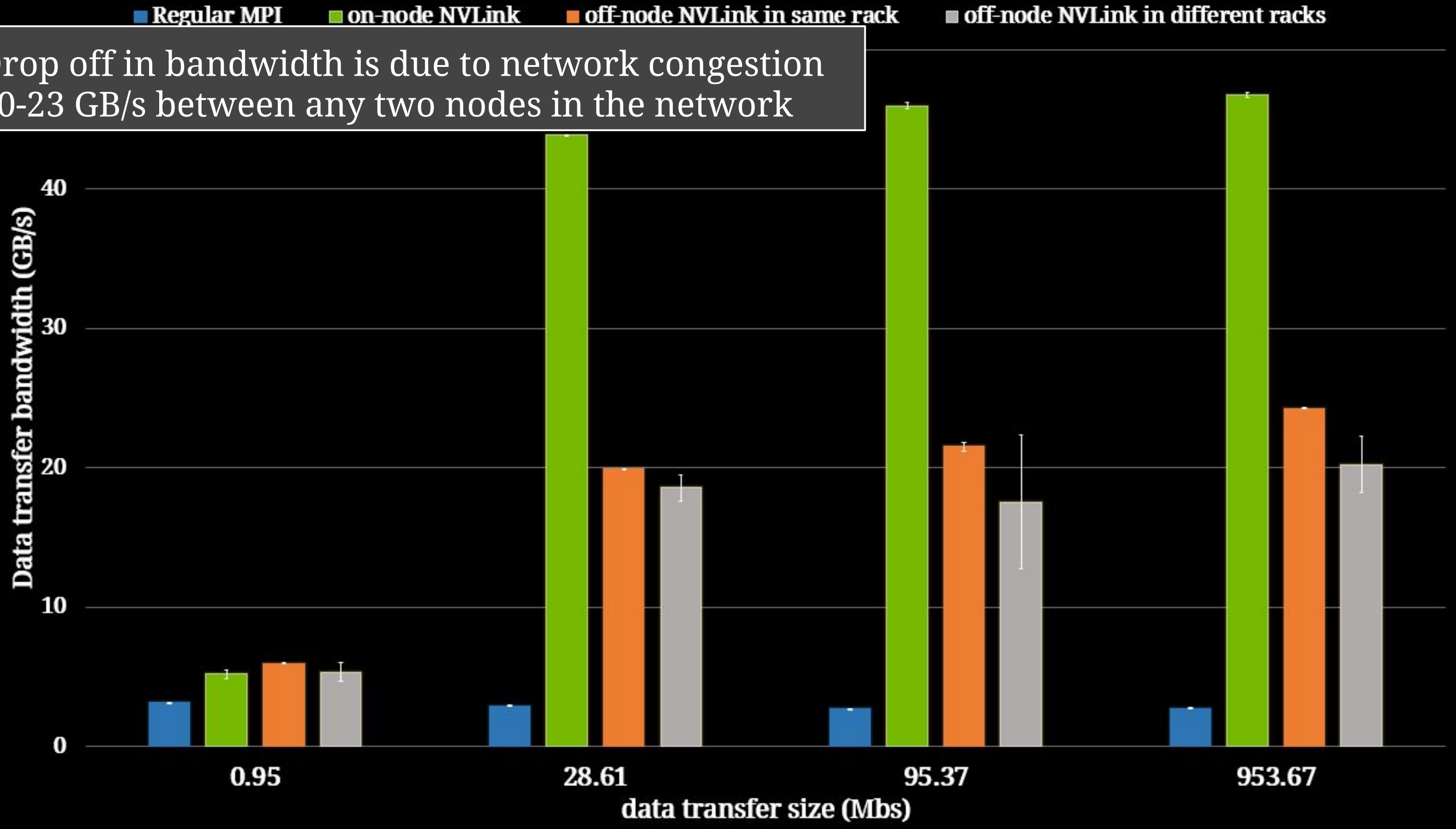
up to 17x speedup

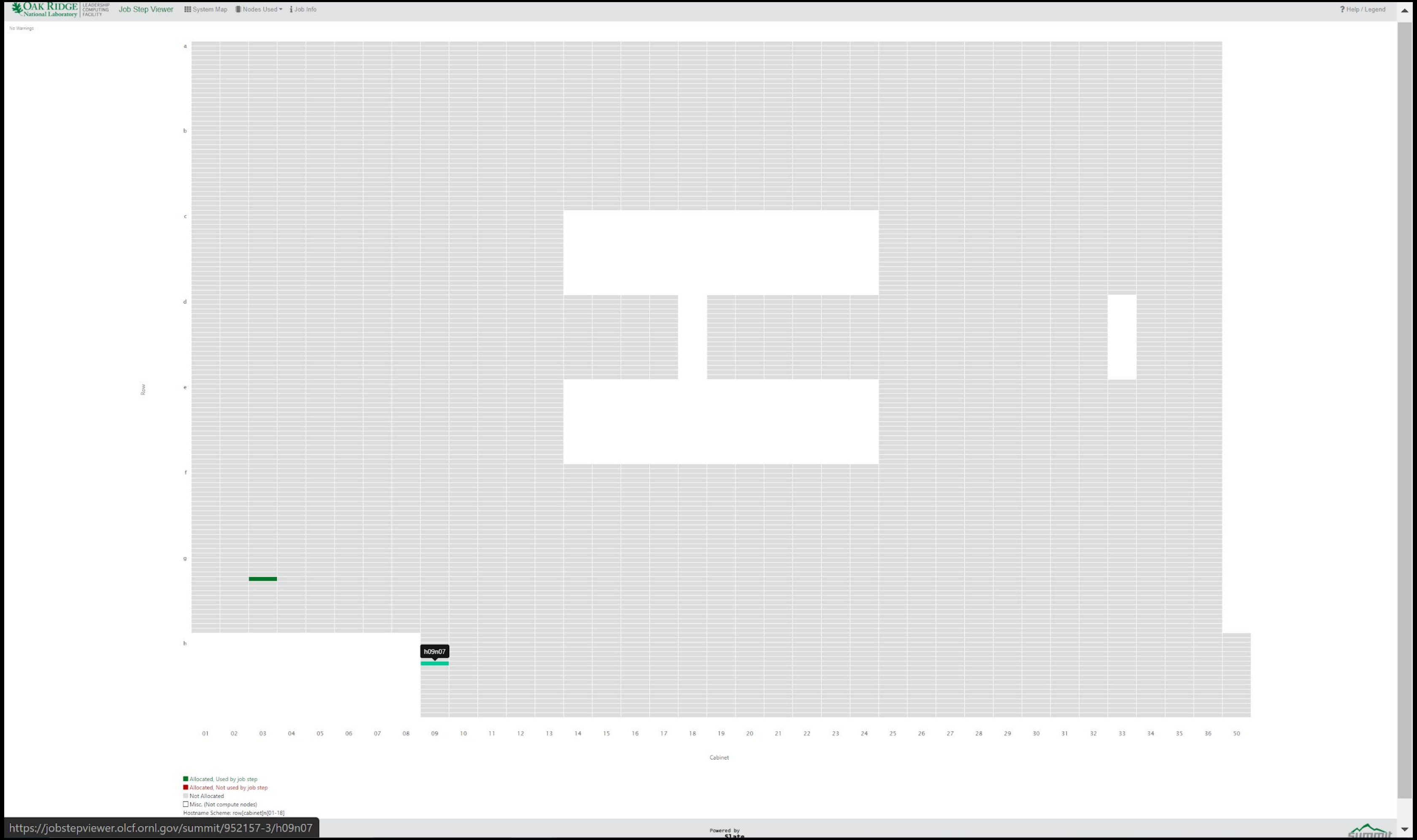


Ping-pong benchmark: https://github.com/weilewei/Ring_example_MPI_CUDA/blob/master/gpuDirect.cpp

Compare bandwidth of data transfer on Summit using NVLink

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

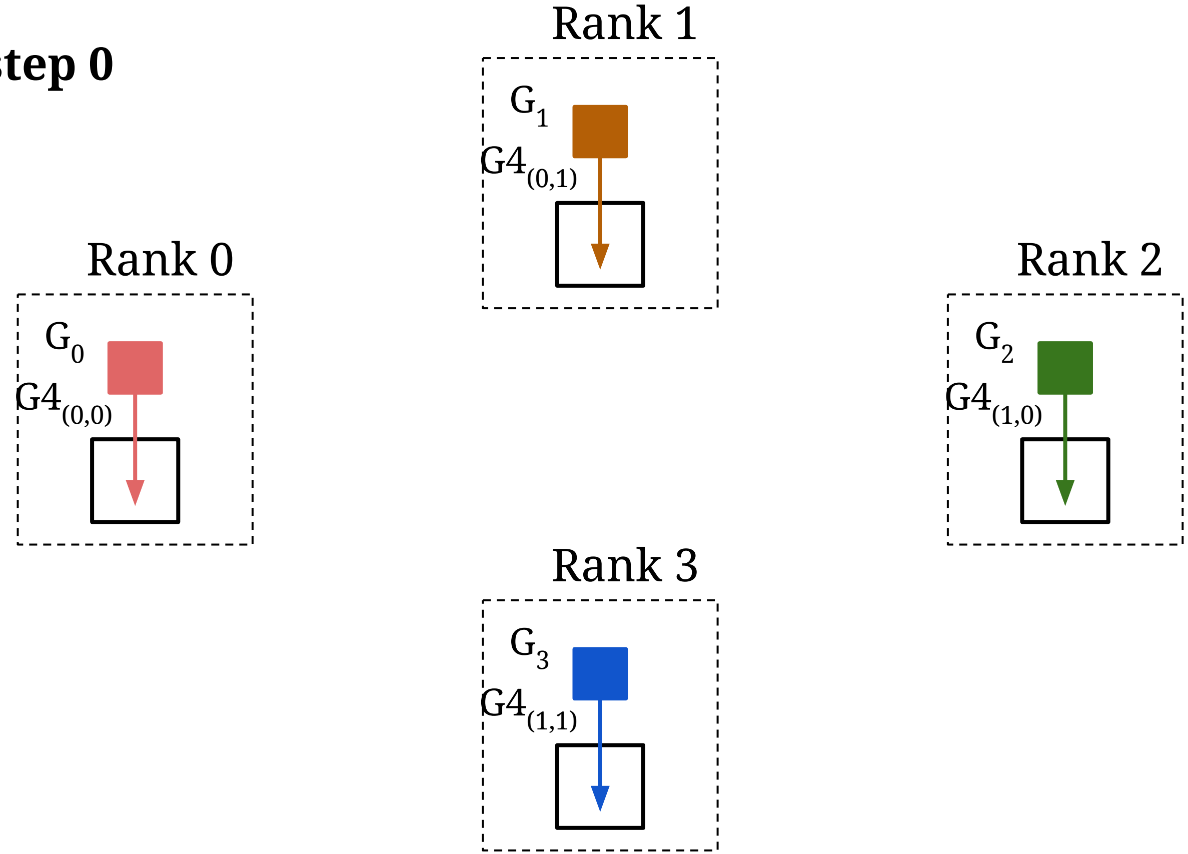




Job Step Viewer
System Map
Nodes Used
Job Info
? Help / Legend
No Warnings
Row
a
b
c
d
e
f
g
h
h09n07
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 50
Cabinet
Allocated, Used by job step
Allocated, Not used by job step
Not Allocated
Misc. (Not compute nodes)
Hostname Scheme: row[cabinet]n[01-18]
https://jobstepviewer.olcf.ornl.gov/summit/952157-3/h09n07

Pipelined Ring Algorithm

Time-step 0

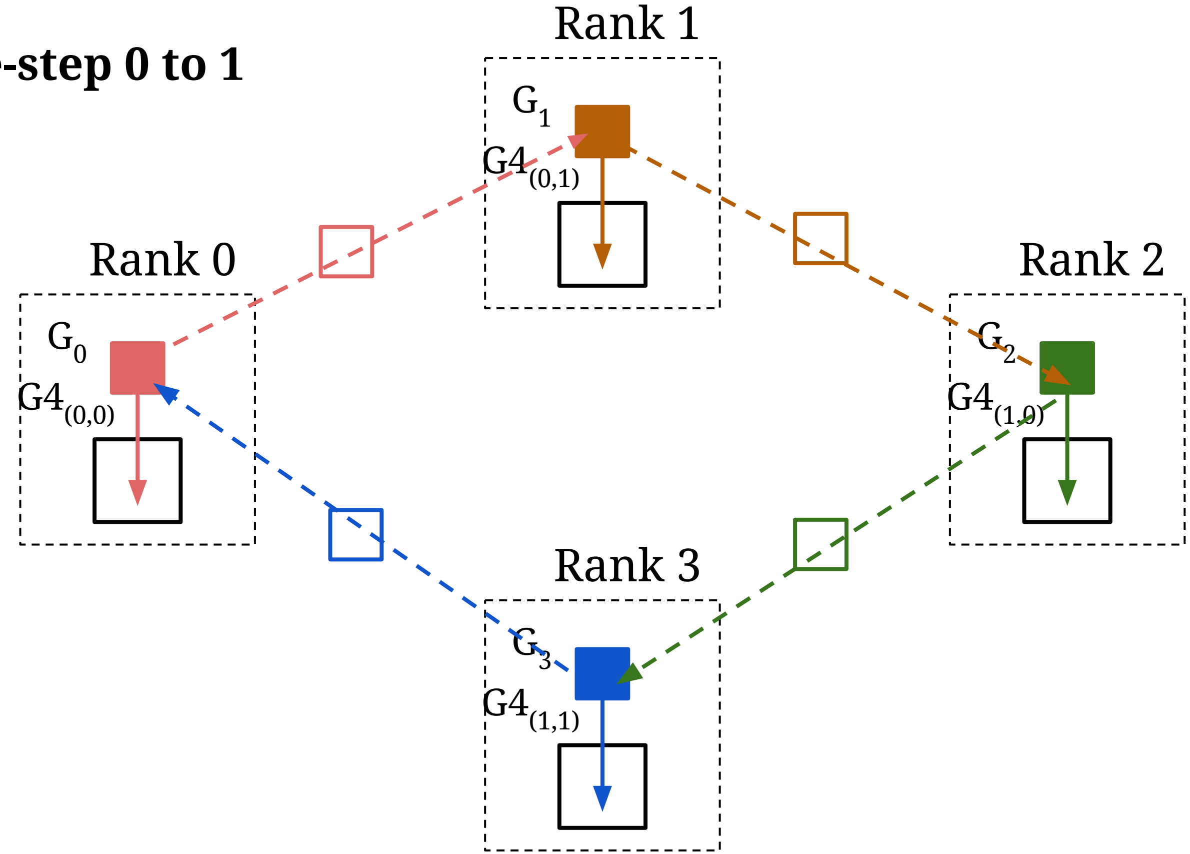


code: https://github.com/weilewei/Ring_example_MPI_CUDA/blob/master/G2_ring.cpp

OAK RIDGE National Laboratory
STE||AR GROUP

Pipelined Ring Algorithm

Time-step 0 to 1



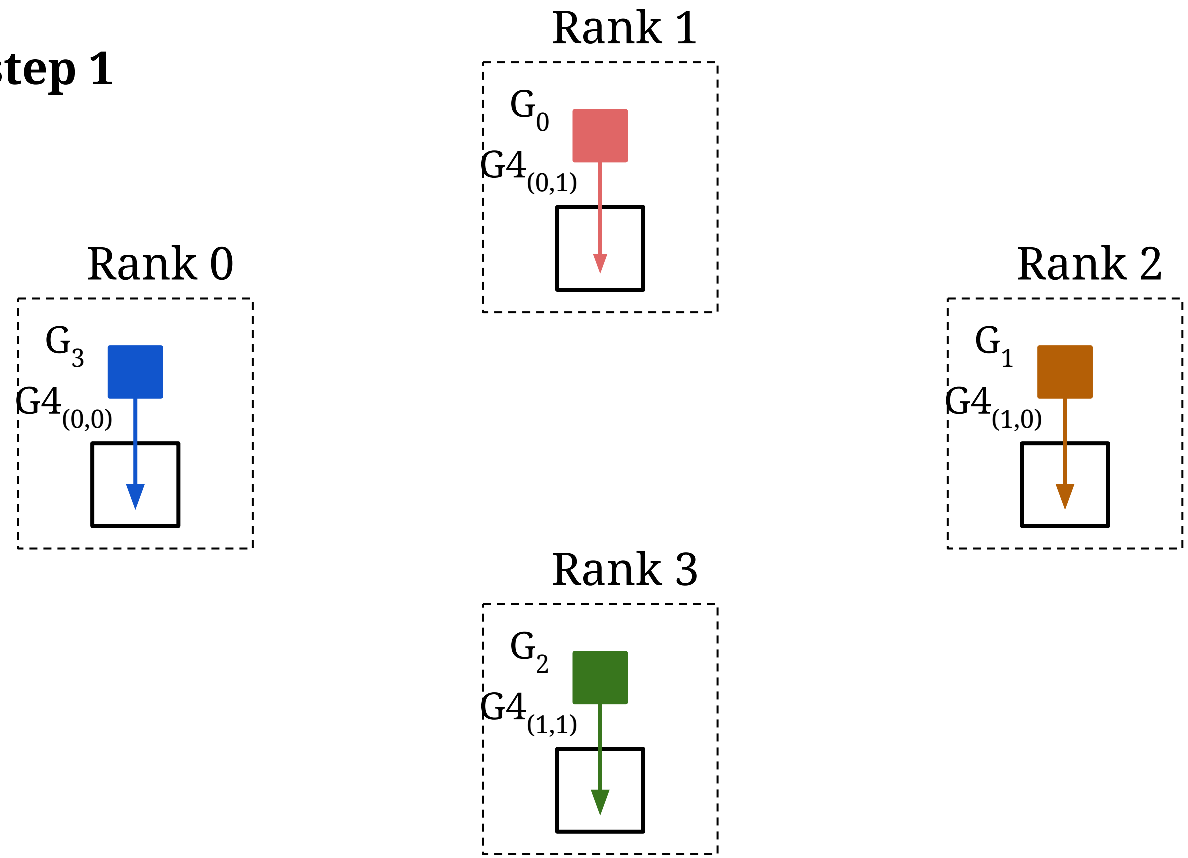
Oak Ridge National Laboratory

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

STE||AR GROUP

Pipelined Ring Algorithm

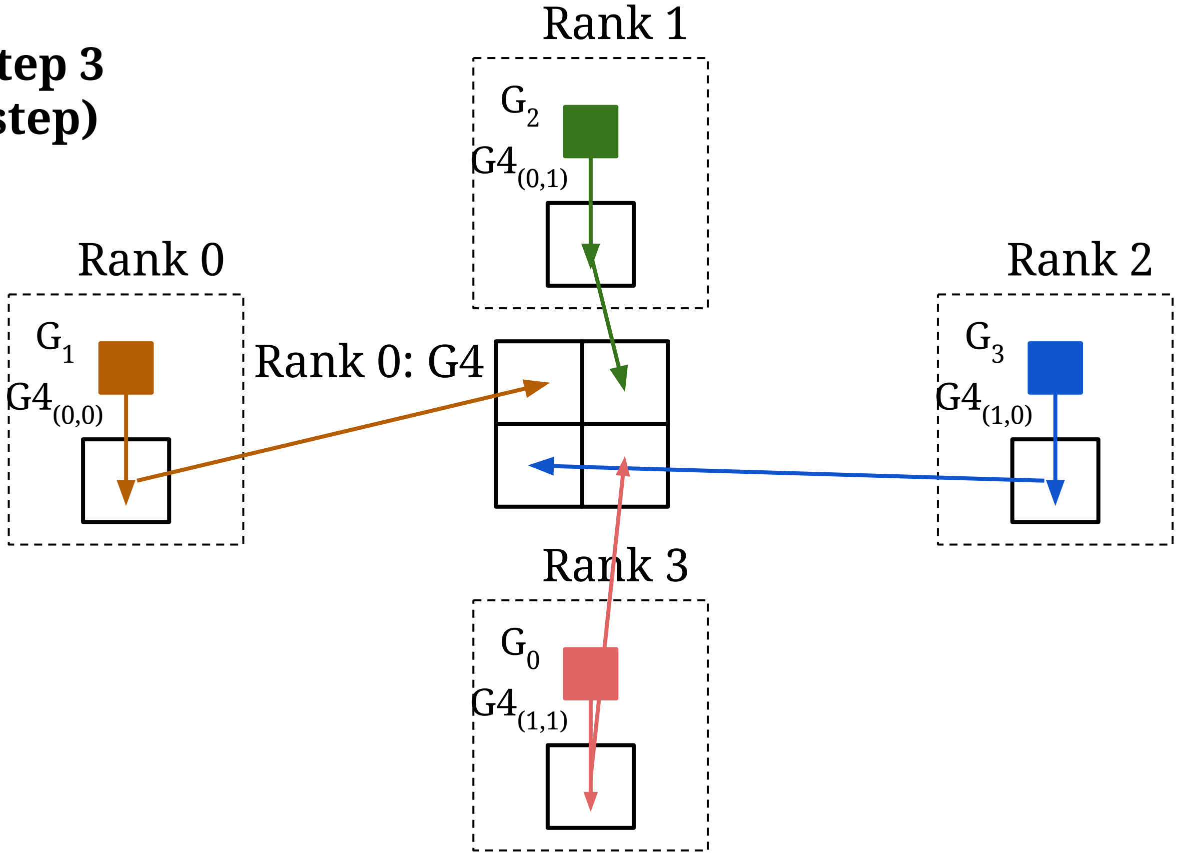
Time-step 1



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

Pipelined Ring Algorithm

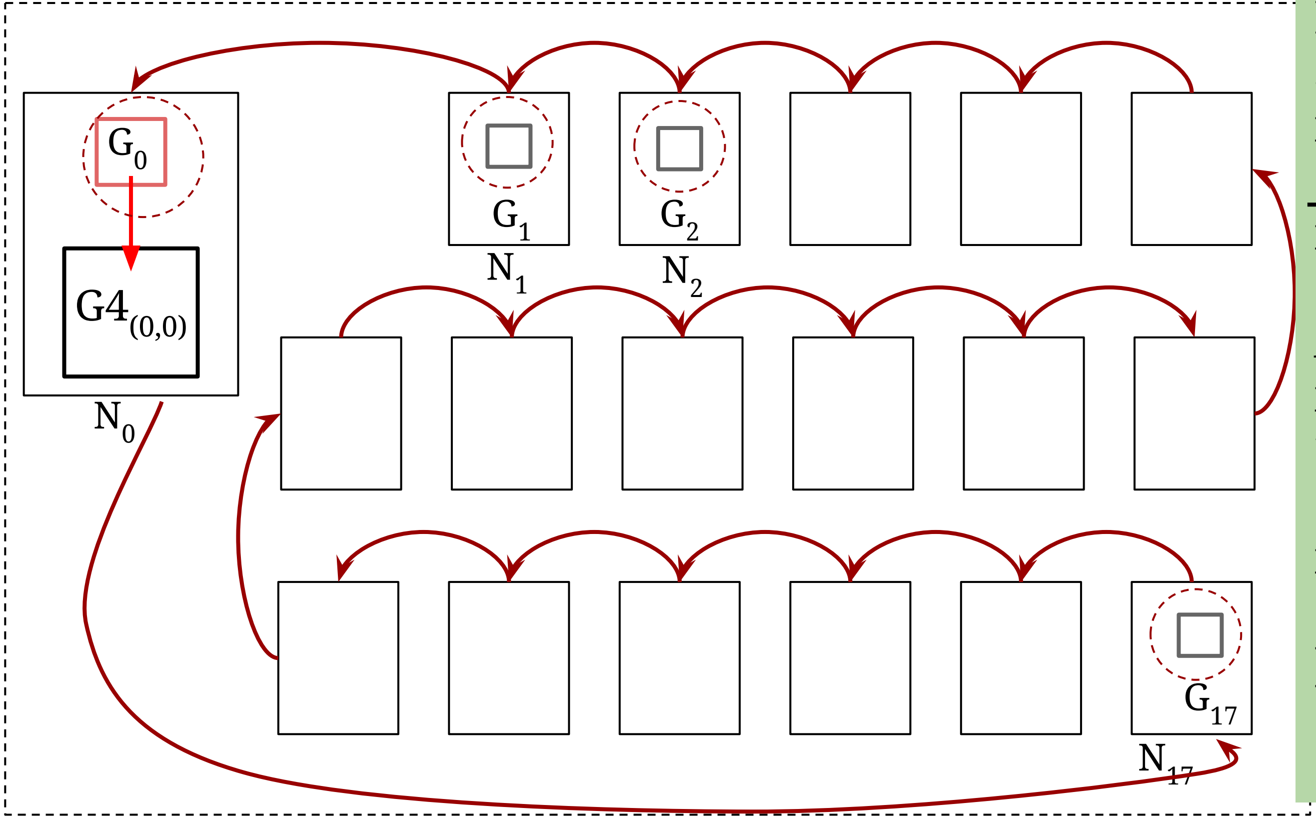
Time-step 3 (final step)



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

OAK RIDGE National Laboratory
STE||AR GROUP

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
)

Oak Ridge National Laboratory

<u>**Ring Fashion**</u>

Cabinet 0 (18 nodes)

STE||AR GROUP

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

HPX-MPI

```
hpx::mpi::enable_user_polling enable_polling;
hpx::mpi::executor exec(MPI_COMM_WORLD);

for(int icount=0; icount < (mpi_size-1); icount++)
{
// encode the originator rank in the message tag as tag = 1 + originator_irank
int originator_irank = MOD(((rank-1)-icount + 2*mpi_size), mpi_size);
int recv_tag = 1 + originator_irank;
recv_tag = 1 + MOD(recv_tag-1, MPI_TAG_UB); // just to be safe, then 1 <= tag <= MPI_TAG_UB

hpx::future<int> f_send = hpx::async(exec, MPI_Irecv, recvbuff_G2, n_elems, MPI_FLOAT, left_neighbor, recv_tag);
hpx::future<int> f_recv = hpx::async(exec, MPI_Isend, sendbuff_G2, n_elems, MPI_FLOAT, right_neighbor, send_tag);

f_recv.get();
CudaMemoryCopy(G2, recvbuff_G2, n_elems);
update_local_G4(G2, G4, rank, n_elems);
f_send.get();

// get ready for send
CudaMemoryCopy(sendbuff_G2, G2, n_elems);
send_tag = recv_tag;
}
```

. . .

MPI

```
for(int icount=0; icount < (mpi_size-1); icount++)
{
// encode the originator rank in the message tag as tag = 1 + originator_irank
int originator_irank = MOD(((rank-1)-icount + 2*mpi_size), mpi_size);
int recv_tag = 1 + originator_irank;
recv_tag = 1 + MOD(recv_tag-1, MPI_TAG_UB); // just to be safe, then 1 <= tag <= MPI_TAG_UB

MPI_CHECK(MPI_Irecv(recvbuff_G2, n_elems, MPI_FLOAT, left_neighbor, recv_tag, MPI_COMM_WORLD, &recv_request));
MPI_CHECK(MPI_Isend(sendbuff_G2, n_elems, MPI_FLOAT, right_neighbor, send_tag, MPI_COMM_WORLD, &send_request));

MPI_CHECK(MPI_Wait(&recv_request, &status));
CudaMemoryCopy(G2, recvbuff_G2, n_elems);
update_local_G4(G2, G4, rank, n_elems);
MPI_CHECK(MPI_Wait(&send_request, &status)); // wait for sendbuf_G2 to be available again

// get ready for send
CudaMemoryCopy(sendbuff_G2, G2, n_elems);
send_tag = recv_tag;
}
```

code: https://github.com/weilewei/Ring_example_MPI_CUDA/blob/hpx_mpi_async/G2_ring_hpx.cpp

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