# Building Parallel Abstractions to DCA++ Scientific Software by Taking Advantage of HPX and GPUDirect

*Weile Wei*[1], A. Chatterjee[2], O. Hernandez[2], H. Kaiser[1]

1. Louisiana State University
2. Oak Ridge National Laboratory

# SciDAC: Computational Framework for Unbiased Studies of Correlated Electron Systems (CompFUSE)

Authors would like to thank:

Giovanni Balduzzi (ETH Zurich)
John Biddiscombe (CSCS)

Thomas Maier (ORNL)
Ed. D'Azevedo (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, **A**dvanced **S**cientific **C**omputing **R**esearch (**ASCR**) and **B**asic **E**nergy **S**ciences (**BES**), Division of Materials Science and Engineering.*

- *This research used resources of the **O**ak Ridge **L**eadership **C**omputing **F**acility (**OLCF**) 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.*

**CompFUSE: [https://compfuse.ornl.gov](https://compfuse.ornl.gov)**

# Outline

- Quantum Monte Carlo solver application: DCA++
- Threading abstraction using HPX to parallelize computations
  - HPX runtime system
  - Performance implications of using HPX over C++ Standard threads
- Optimized distributed computing abstraction on & across Summit nodes
  - Using GPUDirect RDMA (NVLink)
  - To address memory bound challenges in DCA++
- Ongoing efforts
  - Multi platform support for QMC applications
  - Using APEX + HPX Runtime → in depth visualization of kernels

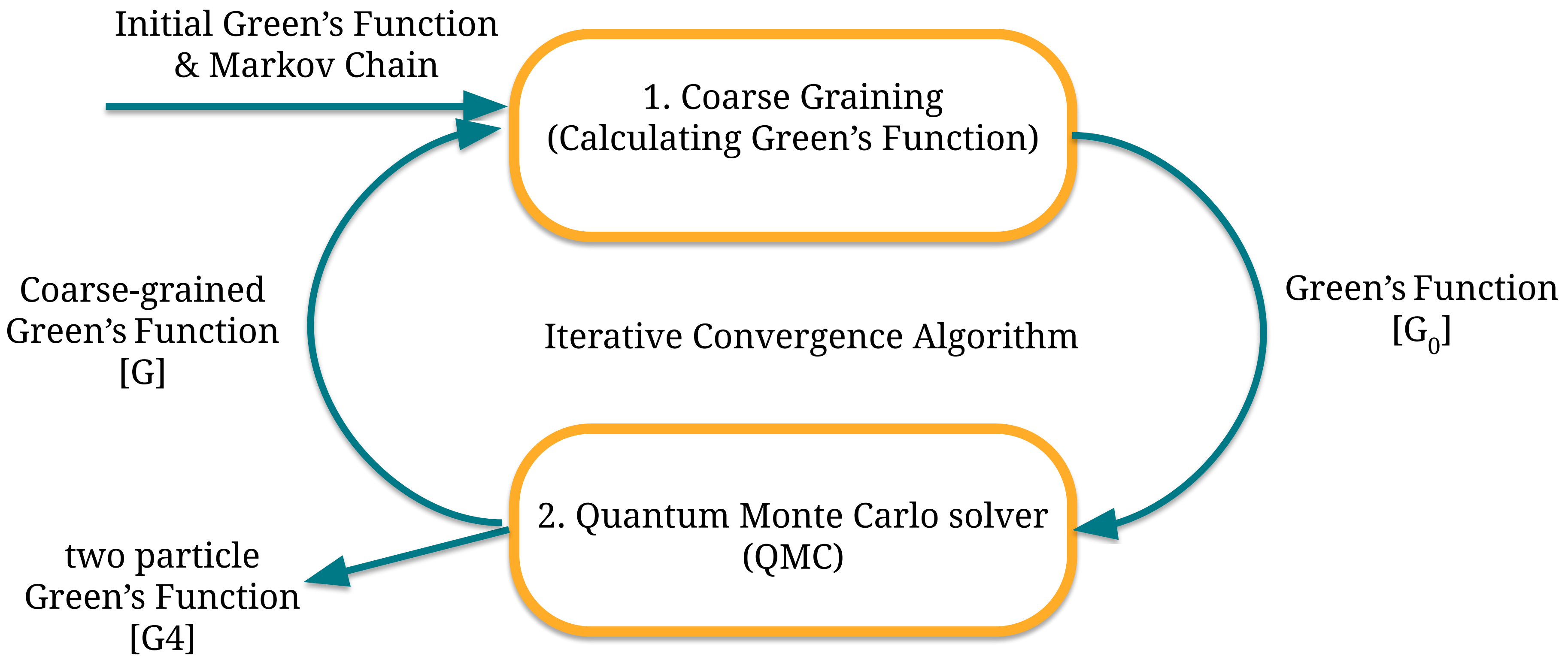
# DCA ++ (Dynamical Cluster Approximation)

- Scientific software for solving quantum many-body problems
- A numerical simulation tool to predict behaviors of co-related quantum materials (such as **superconductivity**, **magnetism**)
- Ported to world's largest supercomputers, e.g. Titan, Summit, Cori, Piz Daint (CSCS) sustaining many petaflops of performance.



[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 workflow



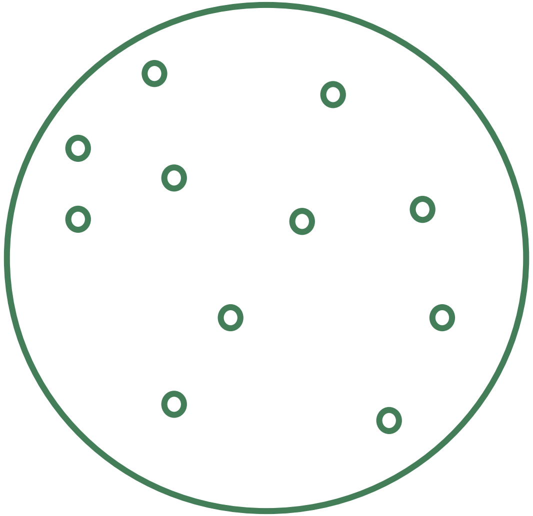
# DCA++ : Quantum Monte Carlo Solver

Imagine: 2D space with lots of points on it (measurements)

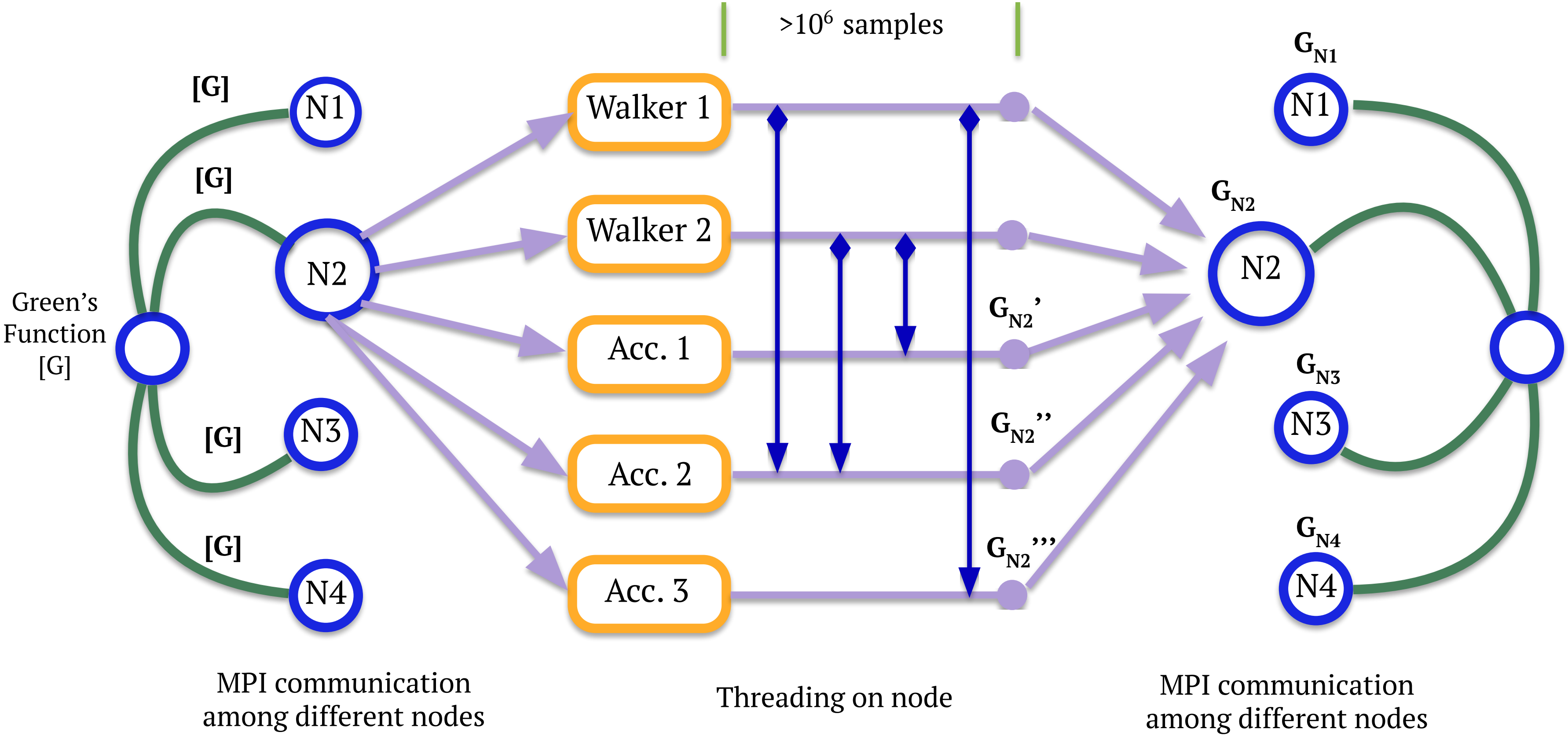
Walkers → 1. picks these measurements at random
2. performs computation (mostly DGEMMs)
3. sends matrices to accumulator (*Producer*)

Accumulators → 1. Feeds in the matrices from the walkers
2. Computes $[G_2]$ for next iteration (*Consumer*)
3. Also computes G4. → $[G_2] * [G_2]$

[all computation happens on both GPU and CPU sides]



# DCA++ : Threaded QMC solver





OAK RIDGE National Laboratory
STE||AR GROUP

# Threading abstraction w/ HPX runtime system

# Threading abstraction for QMC Solver

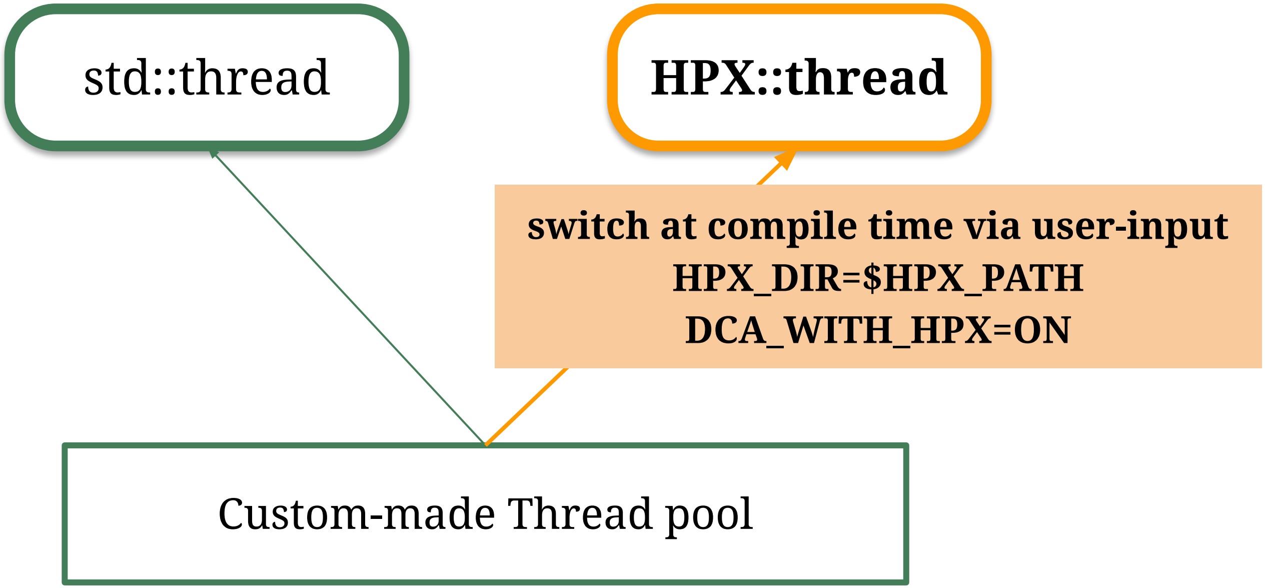



**Figure: workflow of thread-pool. Adding hpx option does not change API of custom-made thread pool in DCA++ due to HPX is C++ standard compliant**

# HPX - A General Purpose Runtime System

- Widely portable (Platforms / Operating System)

- 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
- Domains: Astrophysics, Coastal Modeling, Distributed Machine Learning
- Funded through various agencies:

**Yes, we accept Pull Requests !!!**

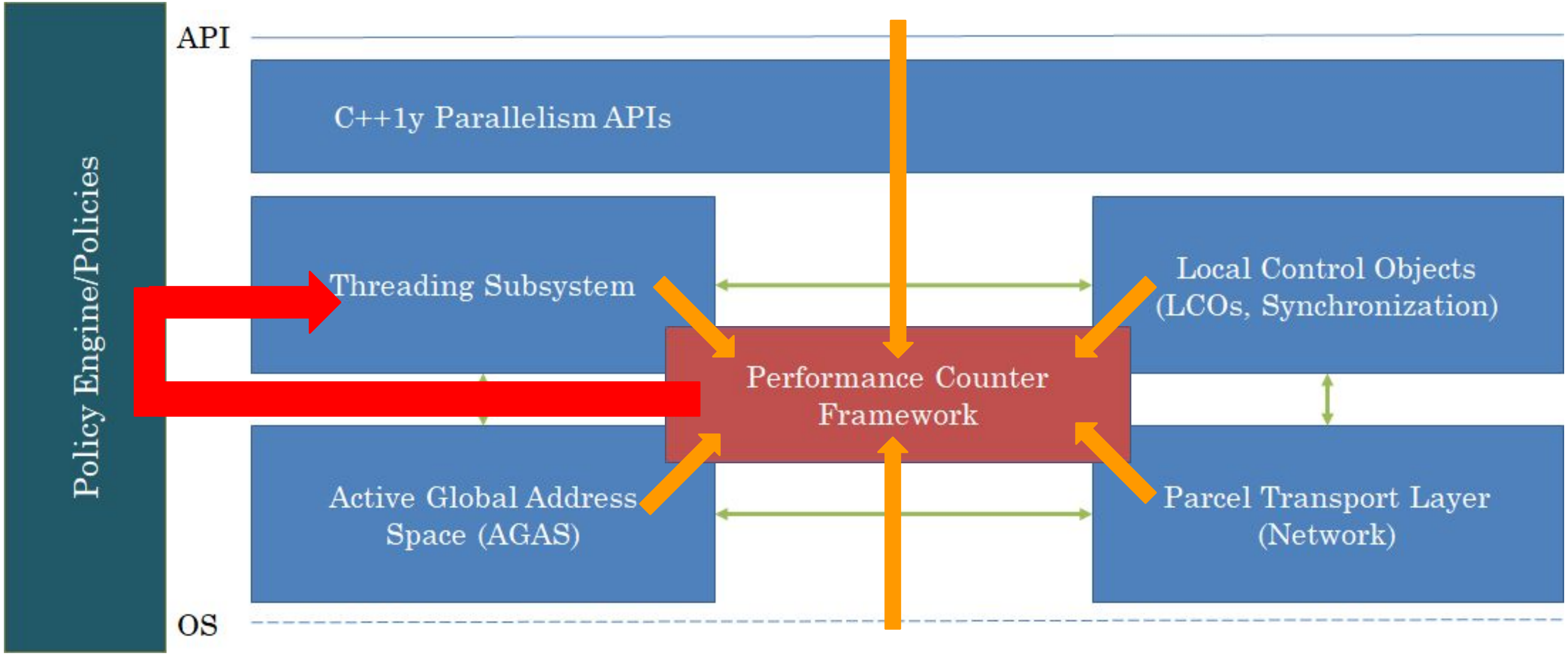
[https://github.com/STEllAR-GROUP/hpx](https://github.com/STEllAR-GROUP/hpx)



OAK RIDGE National Laboratory

STE||AR GROUP

# HPX Runtime System



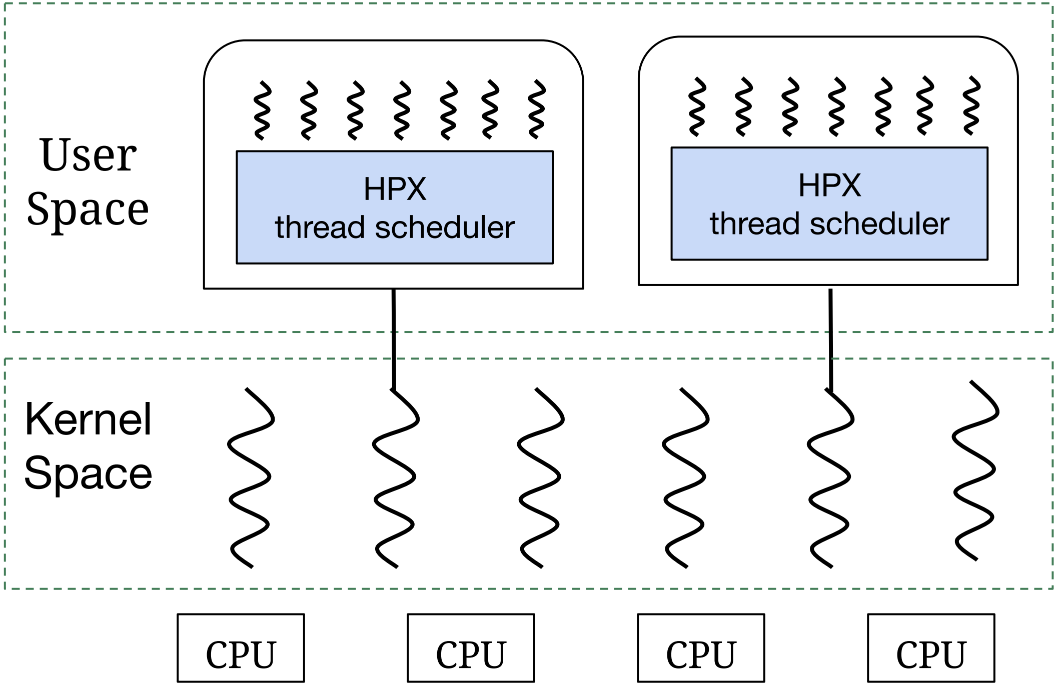
# HPX - C++ standard compliant and more

- C++ standard library API compatible: (selected)

| C++ | HPX |
|---|---|
| std::thread | hpx::thread |
| std::mutex | hpx::mutex |
| std::future | hpx::future |
| std::async | hpx::async |
| std::function | hpx::function |
| . | . |
| . | . |
| . | . |

- 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/ custom-made thread pool

```
// original implementation w/ custom thread pool
  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/ threading abstraction

```
// new implementation w/ threading abstraction
std::vector<dca::parallel::thread_traits::future_type<void>> futures;
// switch to std::future or hpx::future at compile time

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...
}
```

# Synchronization primitives in thread-pool class

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::lcos::future<T>;
    using mutex_type           =       hpx::lcos::local::mutex;
    using condition_variable_type  =
                                 hpx::lcos::local::condition_variable;
    using scoped_lock          =
                                 std::lock_guard<mutex_type>;
    using unique_lock          =
                                  std::unique_lock<mutex_type>;
}
} // namespace parallel
}; // namespace dca
```

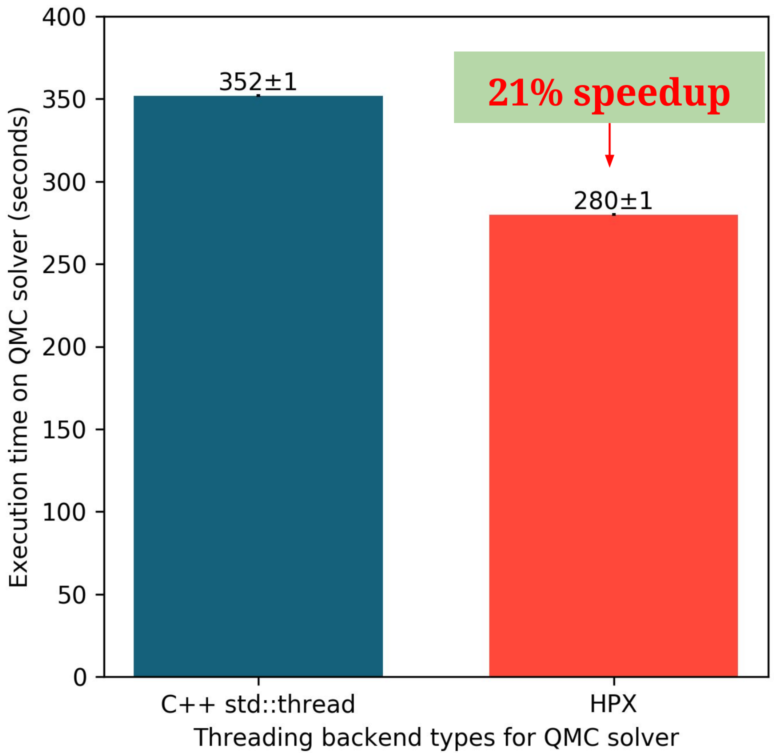
# Runtime Comparison

- Configuration: 1 Summit node
(6 MPI ranks; 7 CPUs + 1 GPU per rank)

- Results for 100k monte carlo measurements with error bars obtained from 5 independent executions.

- We observed **21% speedup using HPX threading** in DCA++ threaded QMC solver on Summit over C++ std threads.

- The speedup is due to **faster context switch and scheduler and less synchronization overhead** in HPX runtime system.



# Optimized distributed computing with NVIDIA GPUDirect RDMA on Summit

# Memory bound challenge and solution

Focus: Memory usage of the two-particle Green's Function (G4) computation

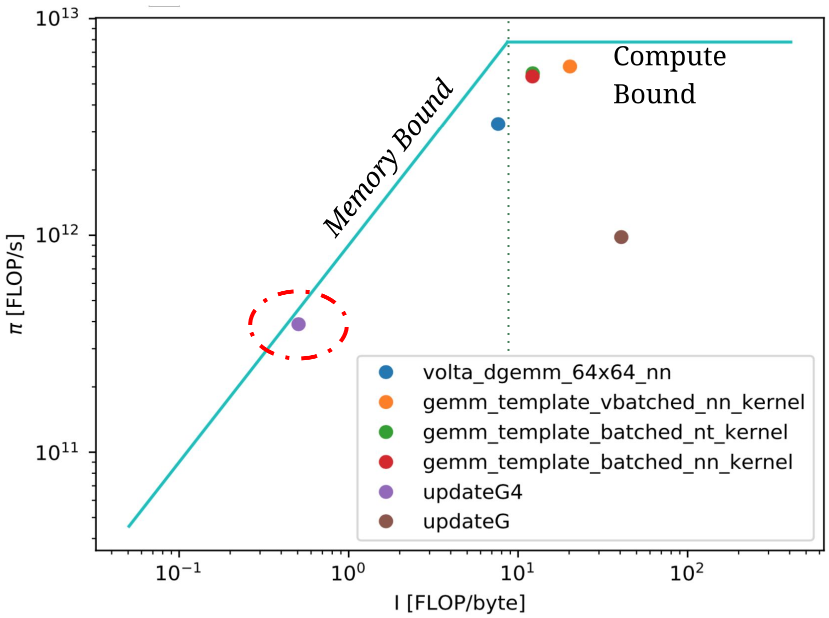


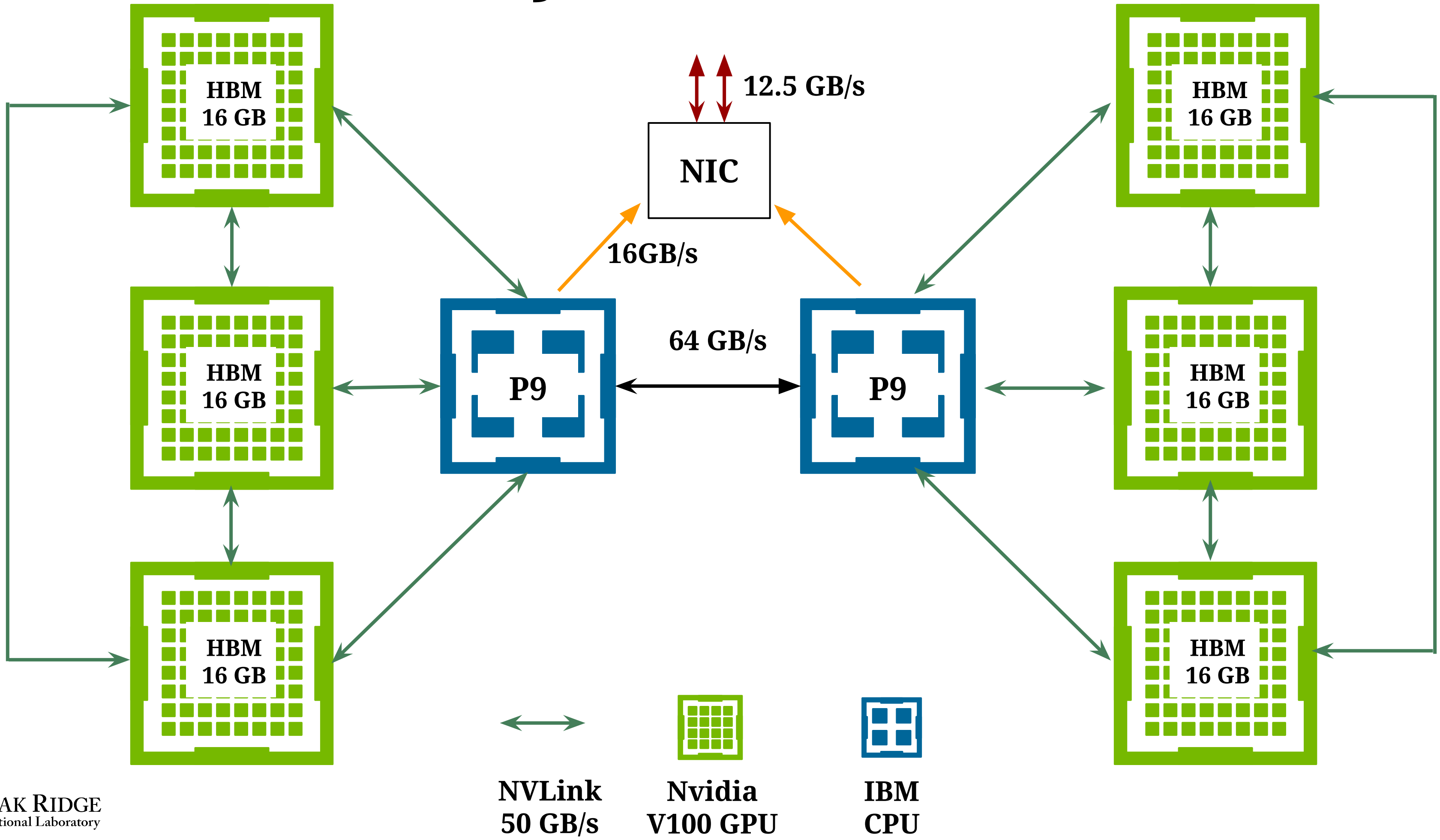

Fig. : Roofline plot of a single NVIDIA V100 GPU running DCA++ at production level on Summit (OLCF).

- In general, size of G2 is ~30 MB, while G4 is 12 GB.
- V100 HBM on Summit: 16 GB
- Solution: *Broadcasting* each $G_2$[][] matrix to all other ranks:
  - Traditional method
  - GPUDirect RDMA




G. Balduzzi, A. Chatterjee, Y.W. Li, P. Doak, U. Haehner, Ed. Azevedo, T. Maier, T. Schulthess. *Accelerating DCA++ (Dynamical Cluster Approximation) Scientific Application on the Summit supercomputer*. **PACT 2019**.

# Summit Node Layout

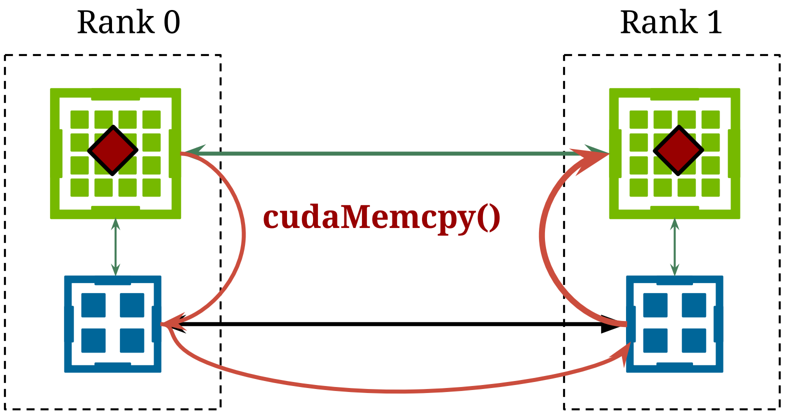





Oak Ridge National Laboratory

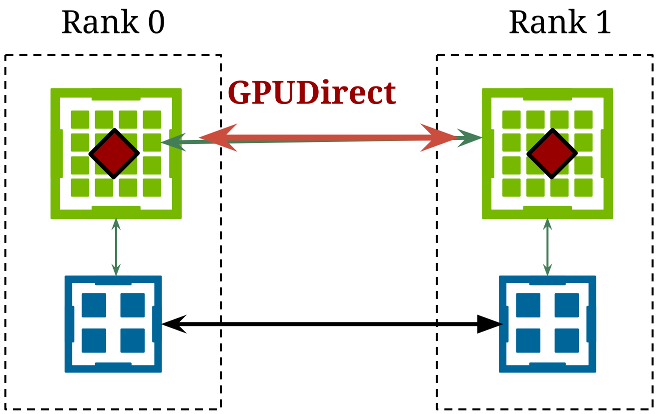
# Transfer G2 around

## Traditional method

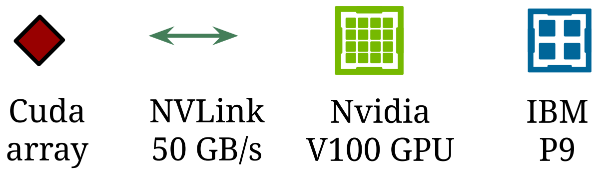



Memory copy everywhere: Device2Host, H2D, network transfer, etc.

## GPUDirect method



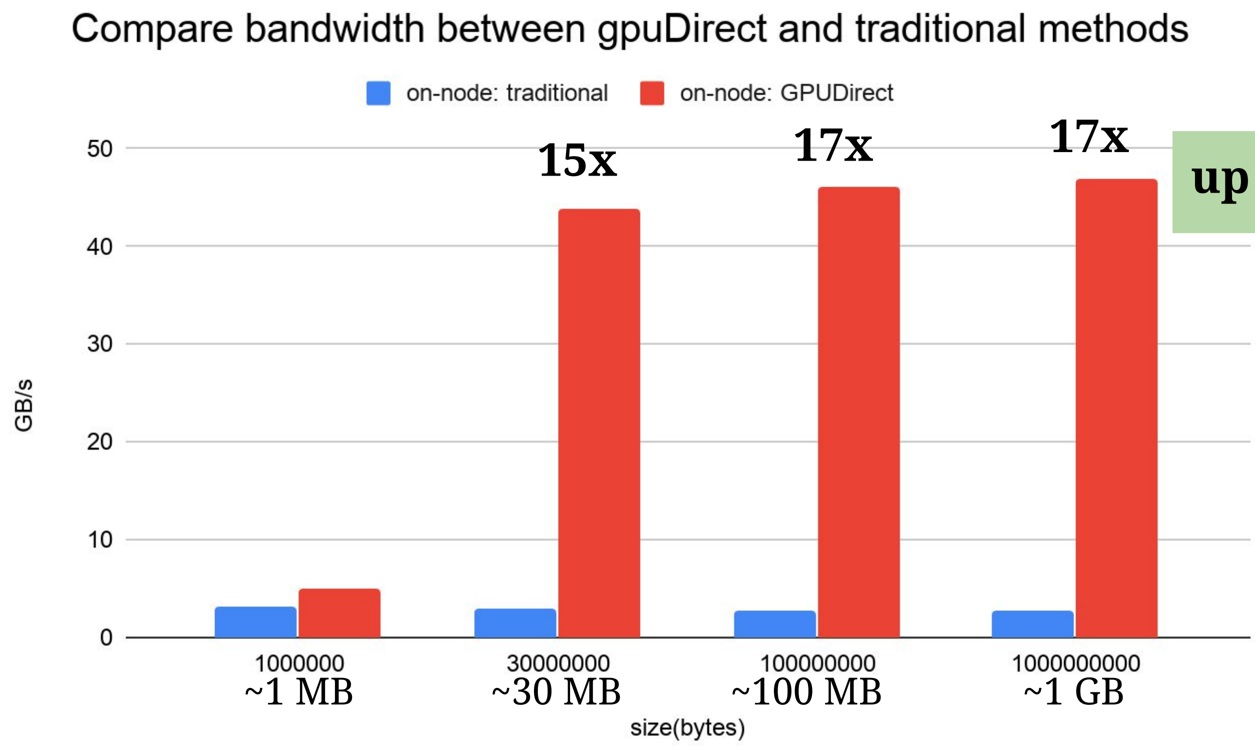

We avoid expensive memory copies and use high-speed network, the NVLink.





OAK RIDGE National Laboratory
STE||AR GROUP

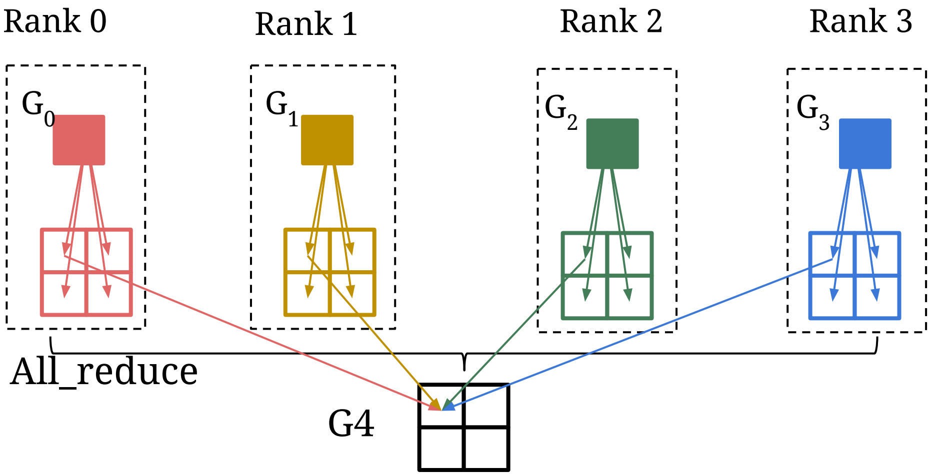
# Performance Comparison




**up to 17x speedup!**

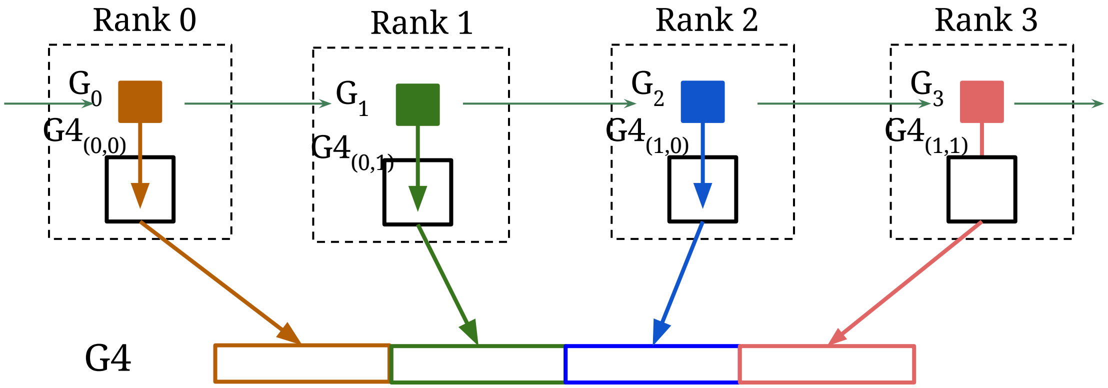
# Distributed G4 implementation

Original DCA++




Each rank keeps a private and full copy of G4.

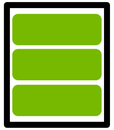
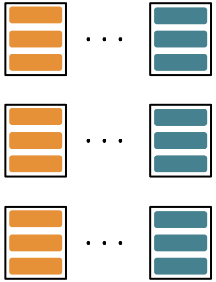
DistG4 solution




Memory usage reduced. Each rank keeps 1/n G4 (n = #ranks).


OAK RIDGE National Laboratory

# Enabling MORE science

| | Original method | Distributed G4 w/ ← **same** science case | Distributed G4 w/ **larger** science case | What if much larger |
|---|---|---|---|---|
| Memory usage per GPU | Full, no more  | ⅙ is used, a lot left  | Each GPU is fully used  | X nodes  |
| Max. memory available for storing final G4 on one Summit node | 16 GB | 96 GB | | 96 * X (nodes) GB |

**6 X** more per node w. DistG4 implementation!

**Then much more!**

# Summary

- **HPX light-weight threads**
  - Added HPX threading support and maintained same API of the thread pool in DCA++
  - **21% speedup using HPX threading** in DCA++ threaded QMC solver on Summit over C++ std threads

- **GPUDirect RDMA**
  - Implemented ring algorithm using GPUDirect capability enabling us to explore **large and complex science** problems

# Ongoing work

- **Multi platform effort:**
  - Porting DCA++ w/ HPX to Arm64, Intel x86-64 and more…
- **HPX task continuation:**
  - Wrapping DCA++ cuda kernel into HPX future → overlapping communication and computation
- **APEX + HPX Runtime:**
  - Profiling DCA++ to identify bottlenecks and potential improvement in performance
- **GPUDirect RDMA:**
  - Adding bidirectional ring communication methods to utilize more bandwidth

# Building Parallel Abstractions to DCA++ Scientific Software by Taking Advantage of HPX and GPUDirect

Weile Wei

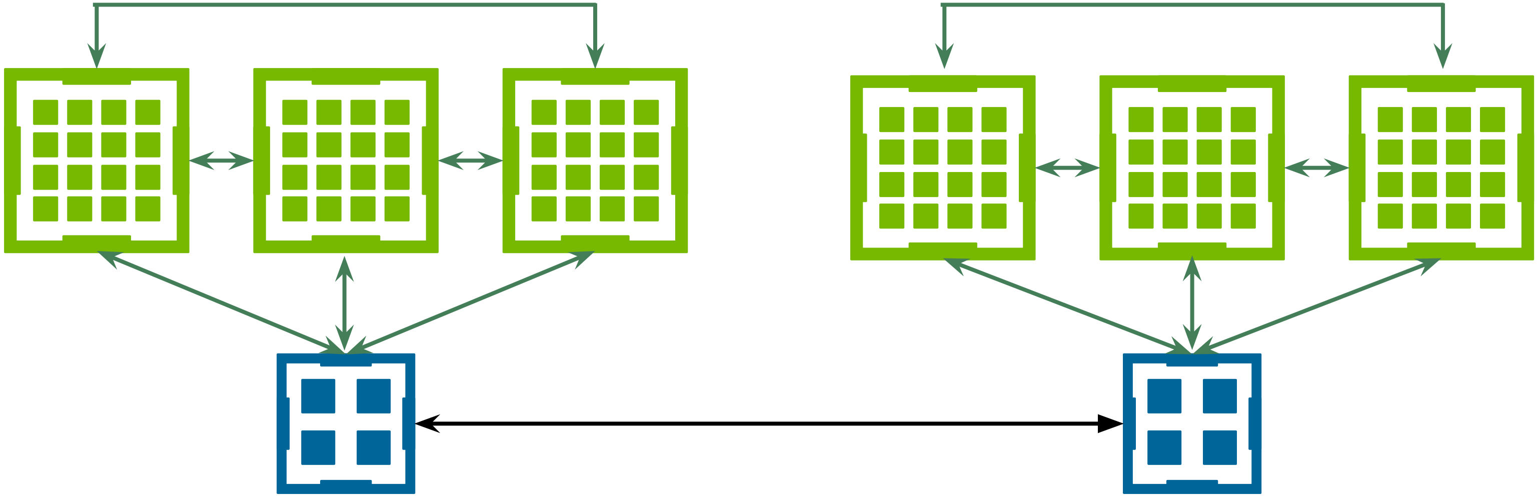
PhD Student, Louisiana State University

Research Collaborator, Oak Ridge National Laboratory

wwei9@lsu.edu

# Backup slides

# Enabling MORE science



**16GB HBM**



29

# 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
- Domains: Astrophysics, Coastal Modeling, Distributed Machine Learning
- Backend support for legacy codes at US DOE, DOD, NSF, and more …

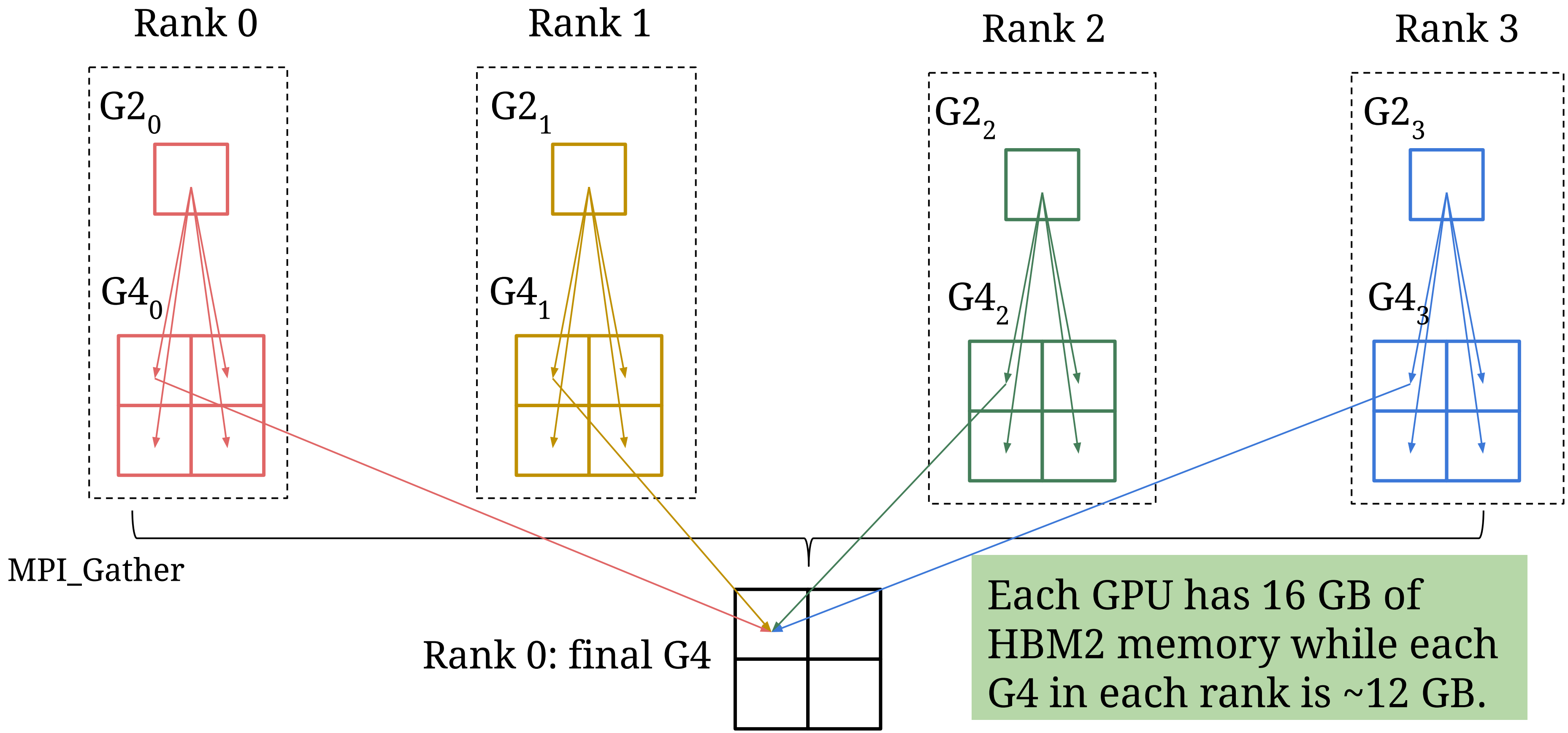
**Open-source at GitHub: [https://github.com/STEllAR-GROUP/hpx](https://github.com/STEllAR-GROUP/hpx)**

**Yes, we accept Pull Requests !!!**

# Summary

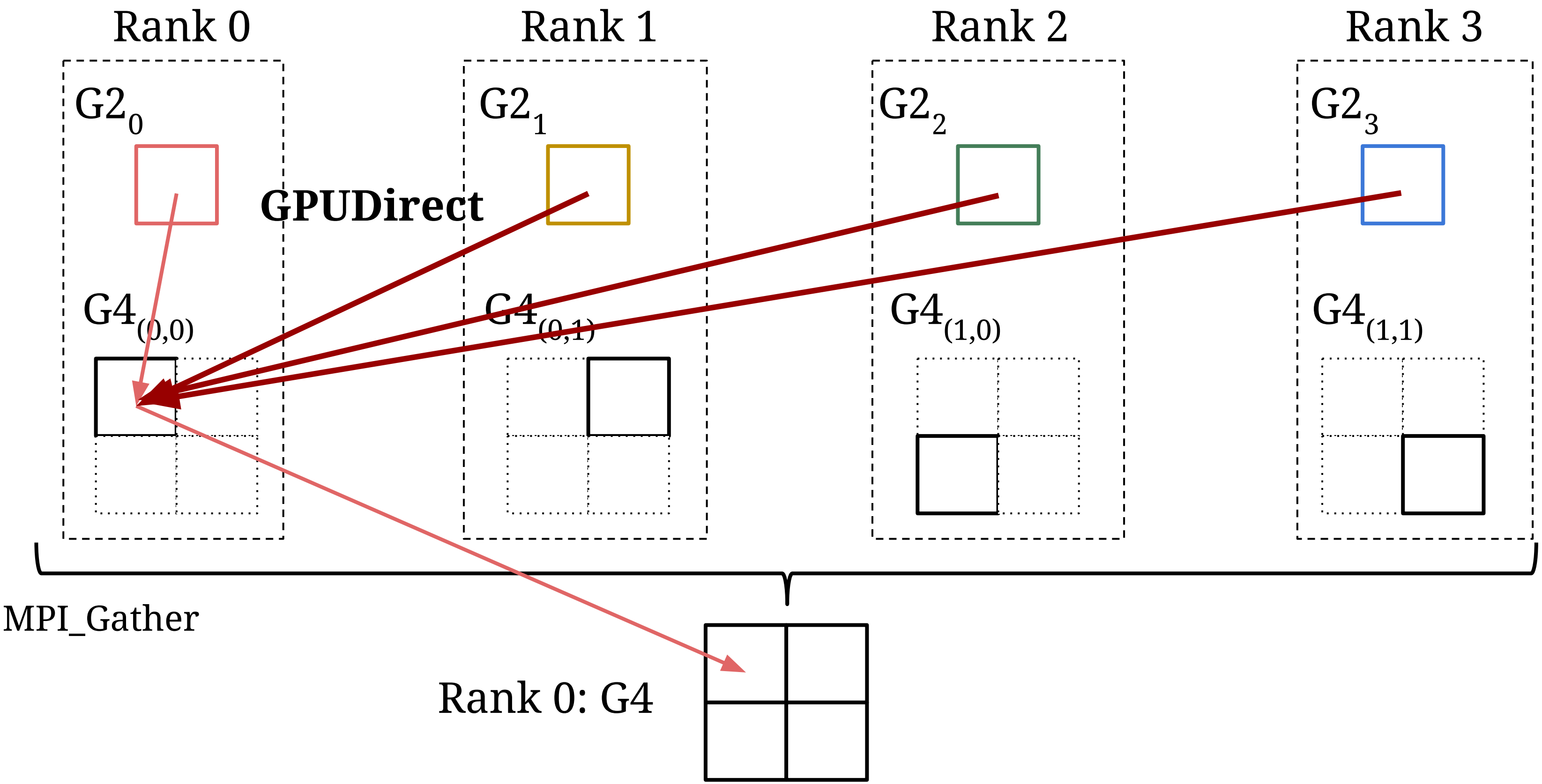
- **Threading abstraction w/ HPX light-weight threads:**
  - Added threading abstraction
  - Produced same results to custom thread pool
  - Future work
    - To profile performance
    - To add more tasks continuation

# Memory bound issue w/ G4




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

# G4 w/ NVIDIA's NVLink enabled

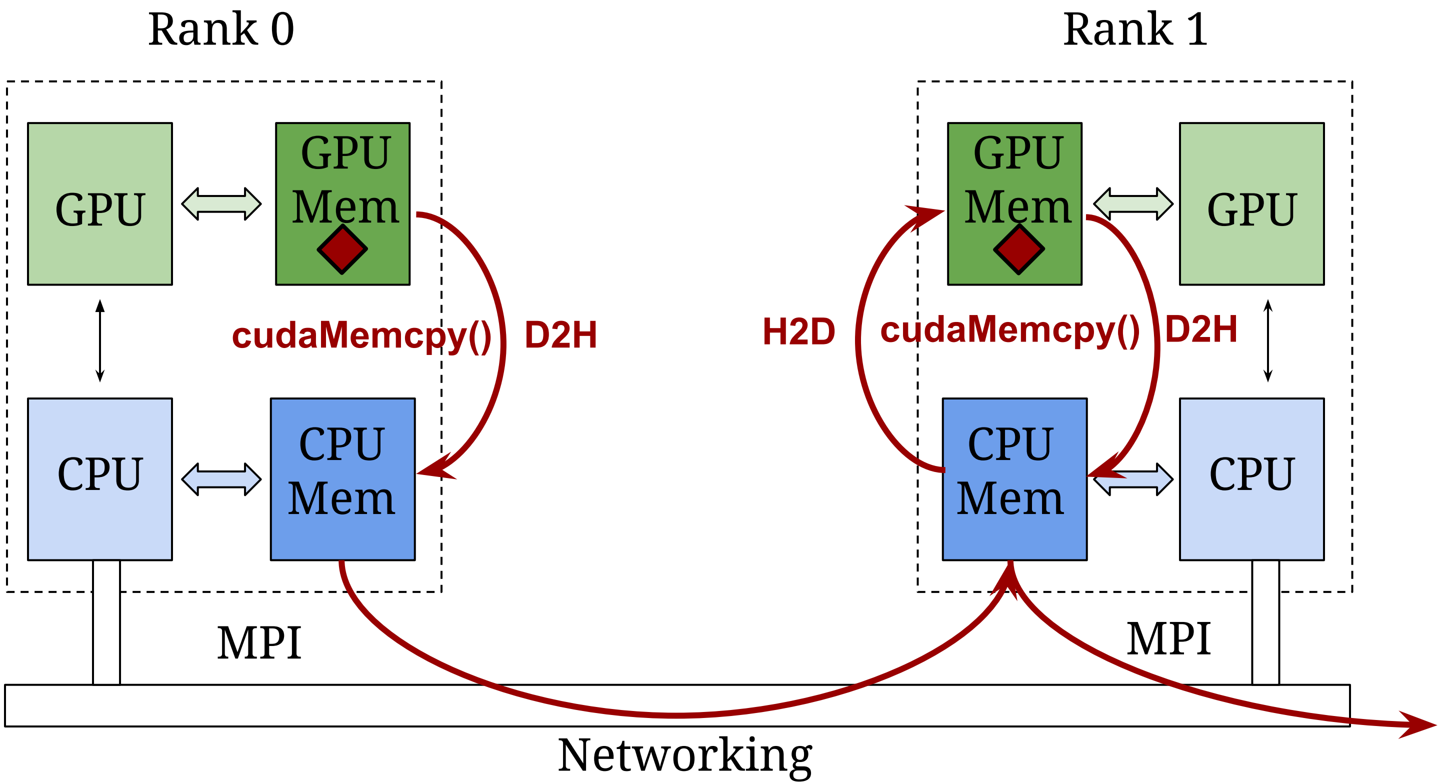





Oak Ridge National Laboratory

STE||AR GROUP

# Transfer G2 around via traditional method

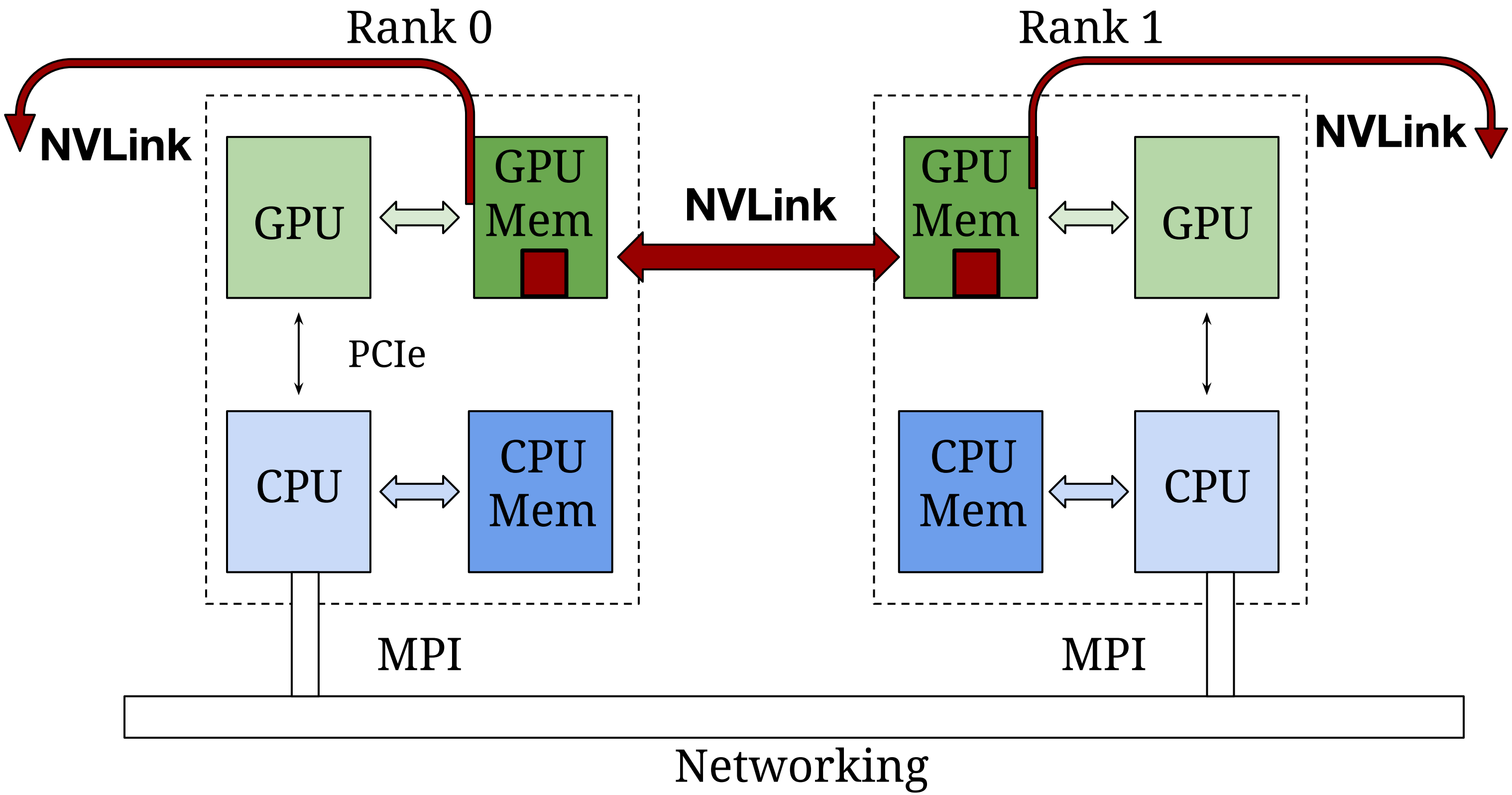




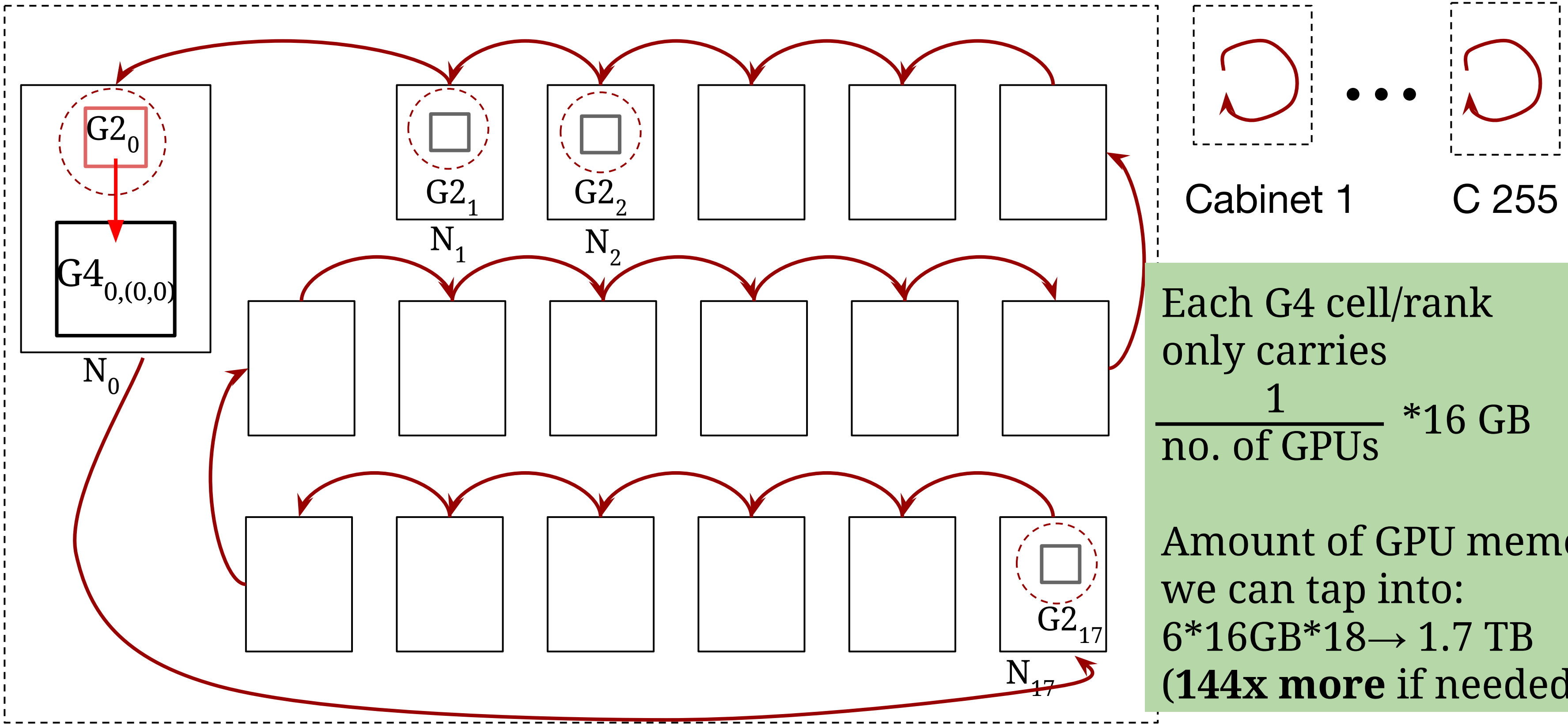

OAK RIDGE National Laboratory

STE||AR GROUP

# Transfer G2 around via GPUDirect



# Enabling MORE science




Each G4 cell/rank only carries

$$\frac{1}{\text{no. of GPUs}} * 16 \text{ GB}$$

Amount of GPU memory we can tap into:
6*16GB*18→ 1.7 TB
(**144x more** if needed)

**Ring Fashion**

Cabinet 0 (18 nodes)



OAK RIDGE National Laboratory

STE||AR GROUP

36