OAK RIDGE National Laboratory

Performance Analysis of a Quantum Monte Carlo Application on Multiple Hardware Architectures Using the HPX Runtime

Weile Wei @weile_wei

Ph.D. student

Louisiana State University

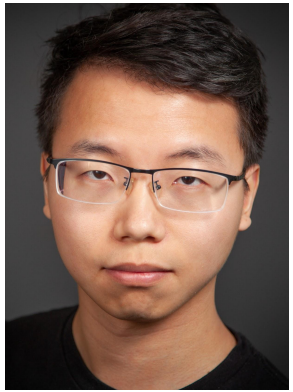
ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

ScalA20 @ SC20, Nov 12, 2020.
Preprint: https://arxiv.org/abs/2010.07098

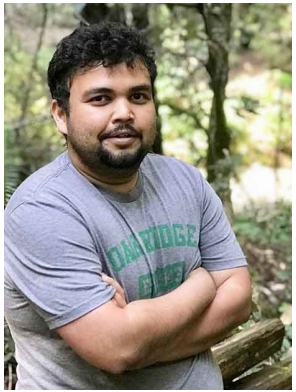
Performance Analysis of a Quantum Monte Carlo Application on Multiple Hardware Architectures Using the HPX Runtime

Authors:



Weile Wei

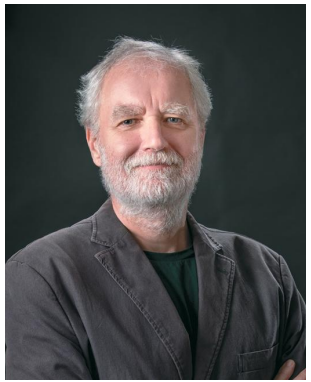
wwei9@lsu.edu



Arghya Chatterjee

Kevin Huck

Oscar Hernandez



Hartmut Kaiser

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)

- *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.*
- *Support for UO through the U.S. Department of Energy, Office of Science, ASCR, RAPIDS SciDAC Institute for Computer Science and Data under subcontract 4000159855 from ORNL.*

CompFUSE: https://compfuse.ornl.gov

Outline

- Quantum Monte Carlo solver application: DCA++
- Threading abstraction using HPX
  - HPX runtime system
  - Performance optimization using hpx::thread over C++ Standard threads
  - Porting code to multi architectures (x86_64, Arm V8, Power9, …)
  - APEX +HPX runtime analysis -> in depth visualization of kernels
- Ongoing efforts
  - Dynamic abstraction layer using GPUDirect RDMA on and across nodes

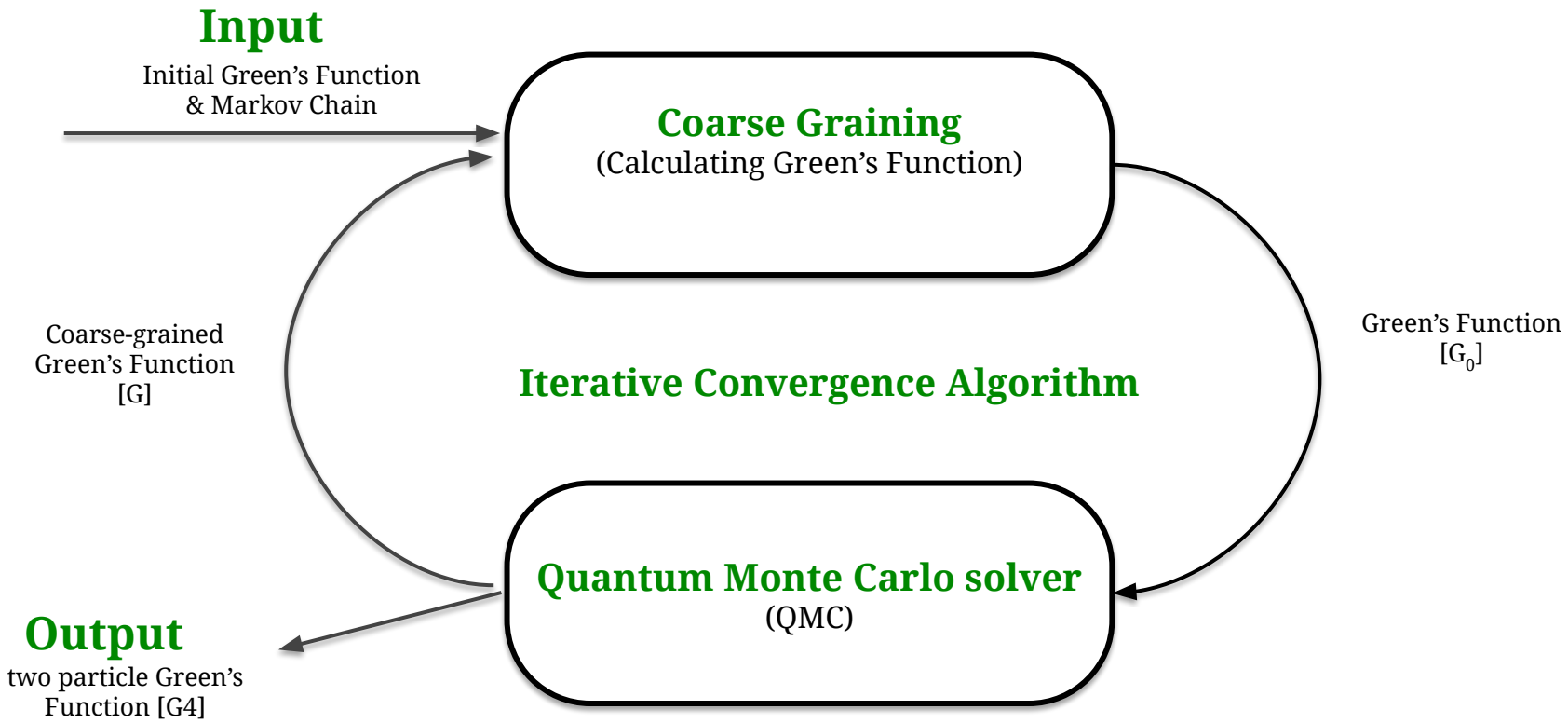
Dynamical Cluster Approximation (DCA++)

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.
- **Gordon Bell Prize Winner 2008**, a highly scalable application

[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.
[3] DCA++ ran on Titan – 18600 nodes at16 Petaflop rate (peak), sustained 1.3 Petaflop rate [Gordon Bell 2008]

DCA++: Primary workflow



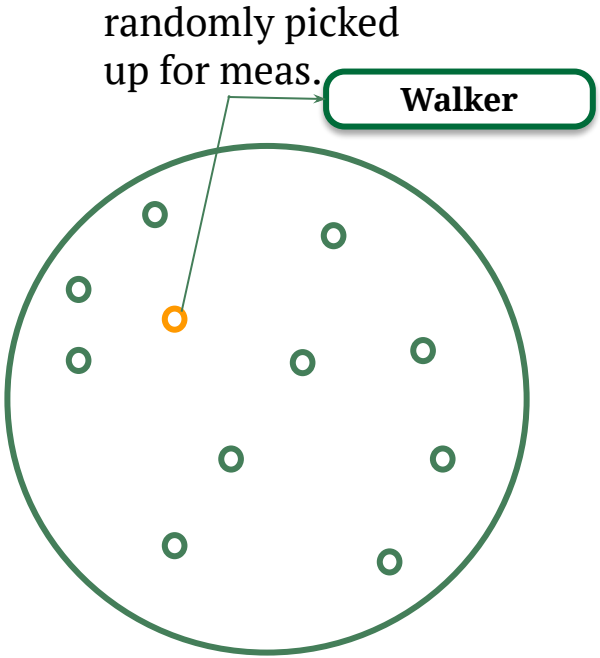
DCA++ : Quantum Monte Carlo Solver

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*)

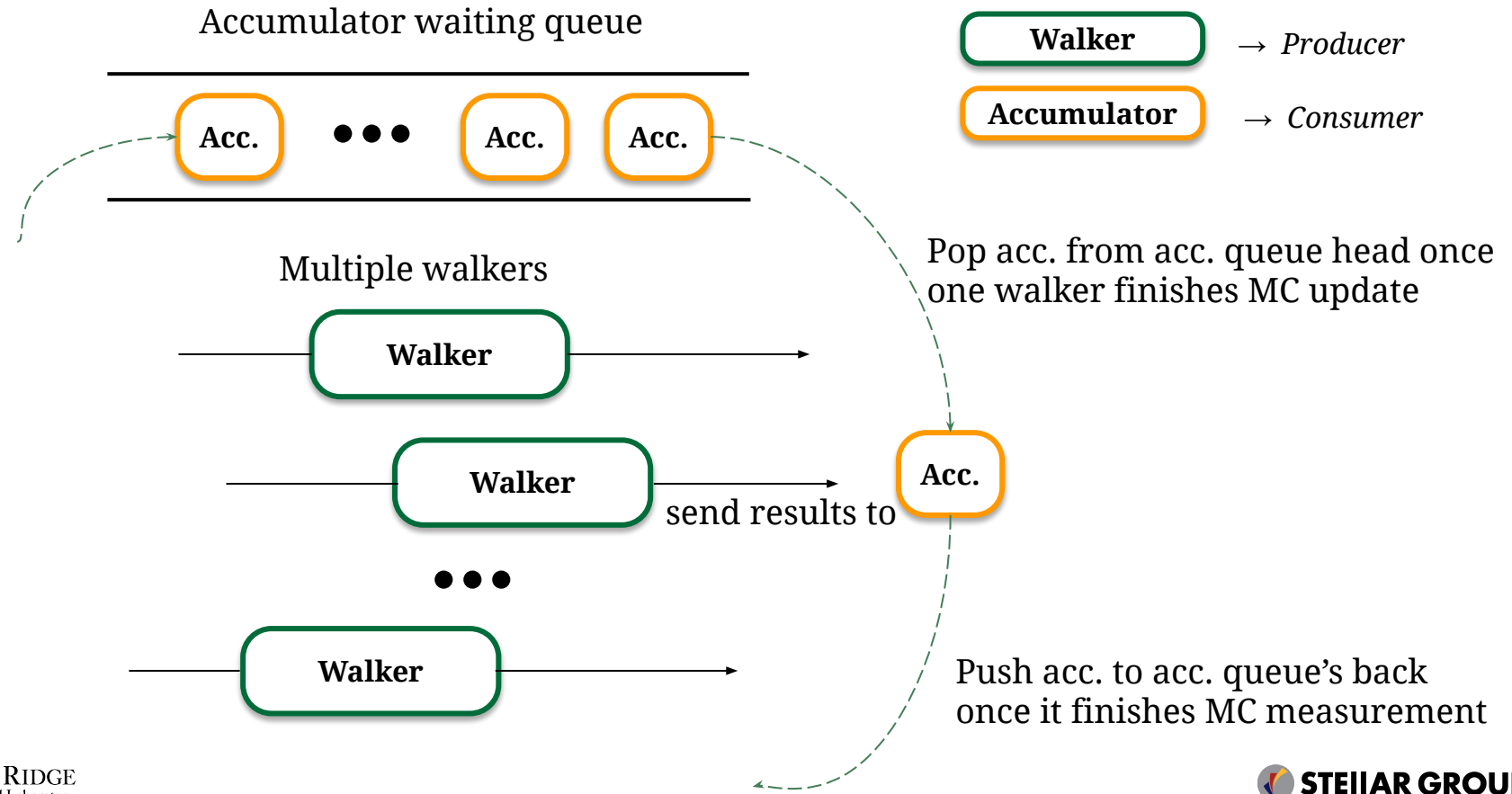
Accumulators → 1. Feeds in the matrices from the walkers
2. Computes $[G_2]$ for next iteration (*Consumer)*
3. Also computes G4. → $[G_2] * [G_2]$

[most computation happens on GPU]



Walkers and accumulators are threaded

Threaded QMC workflow



Threading abstraction w/ HPX runtime system

Threading abstraction for QMC Solver

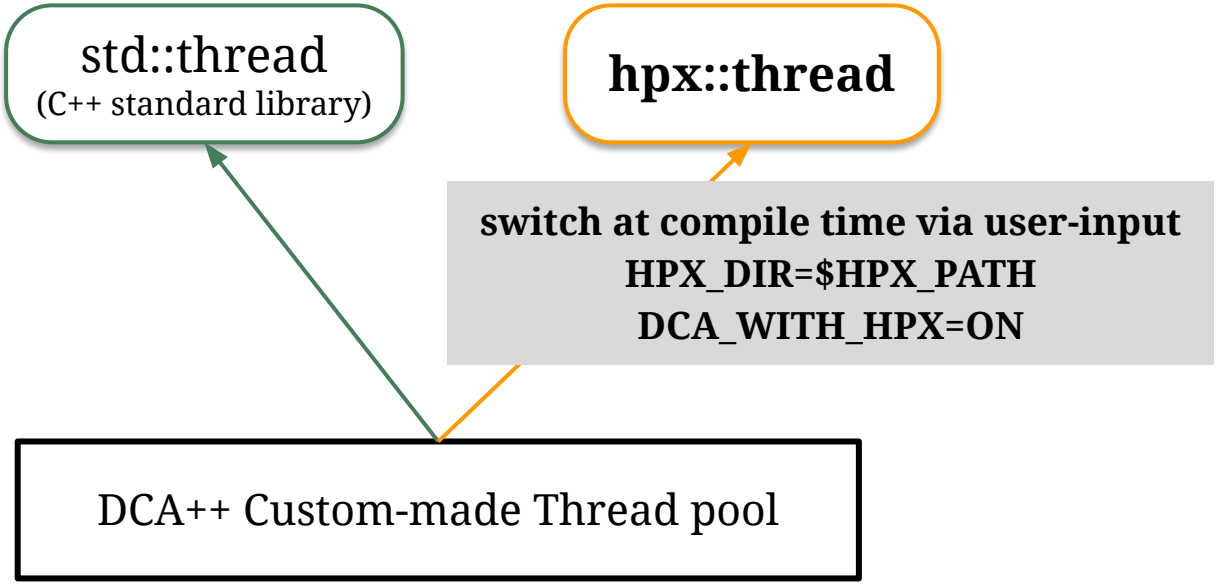


Figure: workflow of thread-pool.

Note, adding hpx support **does not change API and workflow** of custom-made thread pool in DCA++ due to **HPX is C++ standard compliant**

HPX - A General Purpose Runtime System

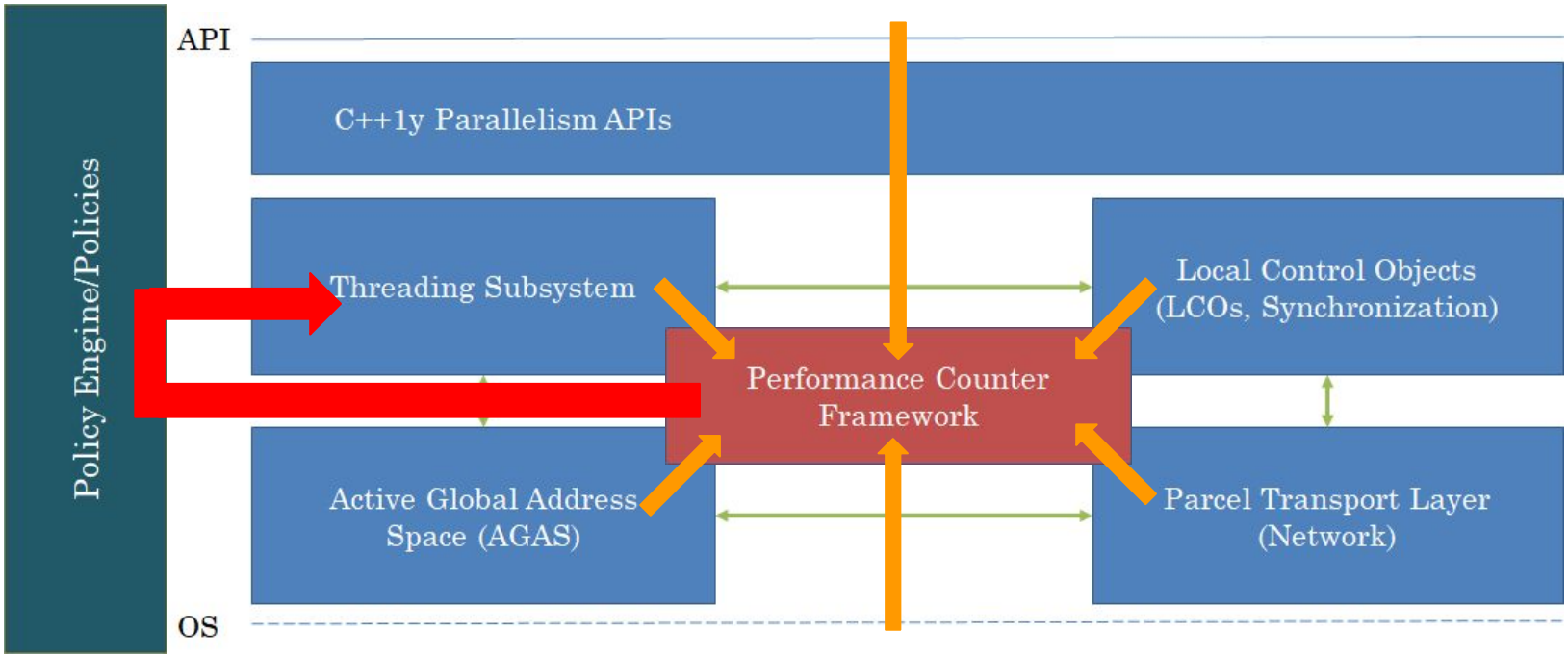
- Widely portable (Platforms / Operating System)

- Unified and standard-conforming
- 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:

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

https://github.com/STEllAR-GROUP/hpx

HPX Runtime System



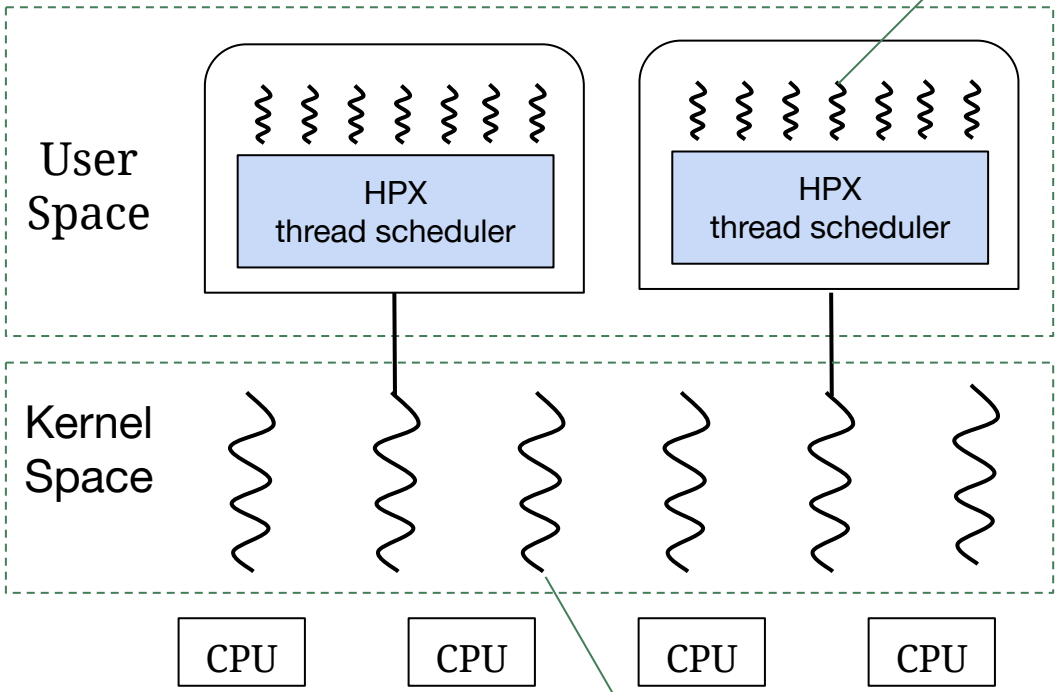
HPX - C++ standard compliant and more

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

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



hpx::thread is user-level thread

Nanosecond level

HPX thread is a lightweight user-level thread

- ~1000x faster context switch than OS thread

Microsecond level

std::thread is kernel-level thread (or pthread)

QMC solver w/ custom-made thread pool

```
1.  // original implementation w/ custom thread pool
2.  std::vector<std::future<void>> futures;
3.
4.
5.  auto& pool = dca::parallel::ThreadPool::get_instance();
6.
7.
8.  for (auto& task: thread_task_handler) {
9.    if (task.type() == "walker")
10.     futures.emplace_back(pool.enqueue(&ThisType::startWalker,
                                          this, task.id() ));
11.
12.   // else if handle other conditions...
13. }
```

QMC solver w/ threading abstraction

```
1.  // new implementation w/ threading abstraction
2.  std::vector<dca::parallel::thread_traits::future_type<void>> futures;
3.  // switch to std::future or hpx::future at compile time
4.
5.  auto& pool = dca::parallel::ThreadPool::get_instance();
6.
7.
8.  for (auto& task: thread_task_handler) {
9.    if (task.type() == "walker")
10.     futures.emplace_back(pool.enqueue(&ThisType::startWalker,
                                          this, task.id() ));
11.
12.   // else if handle other conditions...
13. }
```

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::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
```

https://github.com/STEllAR-GROUP/DCA/releases/tag/hpx_thread

Performance Analysis in-depth

Experiment setup

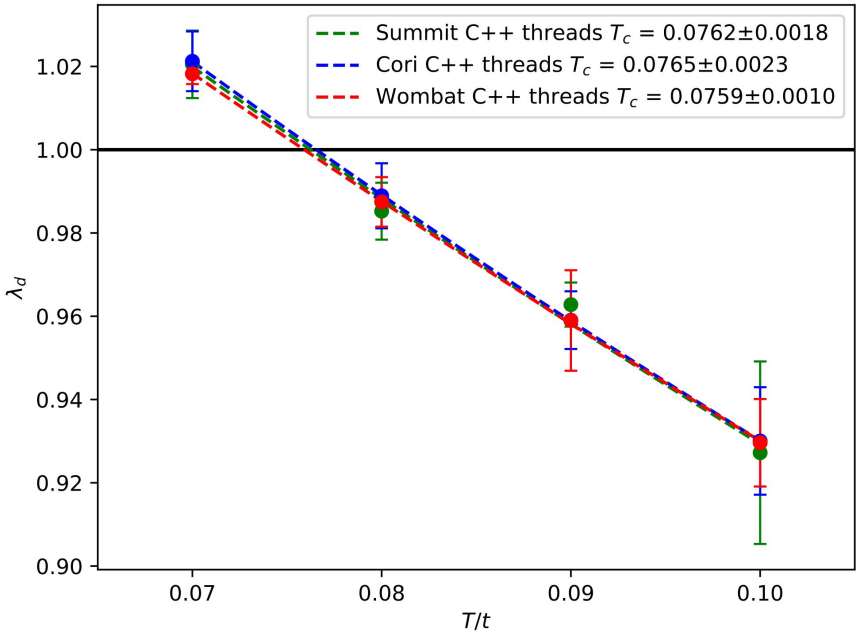
- Hardware configurations:

Configuration	Summit @ ORNL	Wombat @ ORNL	CoriGPU @ NERSC
GPU	NVIDIA Volta (6 per node)	NVIDIA Volta (2 per node)	NVIDIA Volta (8 per node)
CPU	IBM POWER9™ (2 Sockets / 21 Cores per socket)	Arm Cavium ThunderX2 (2 Sockets / 28 Cores per socket)	Intel Xeon Gold 6148 (2 sockets / 20 cores per socket)
CPU-GPU interconnect	NVIDIA NVLINK2 (50 GB/s)	PCIe Gen3 (16 GB/s)	PCIe Gen3 (16 GB/s)

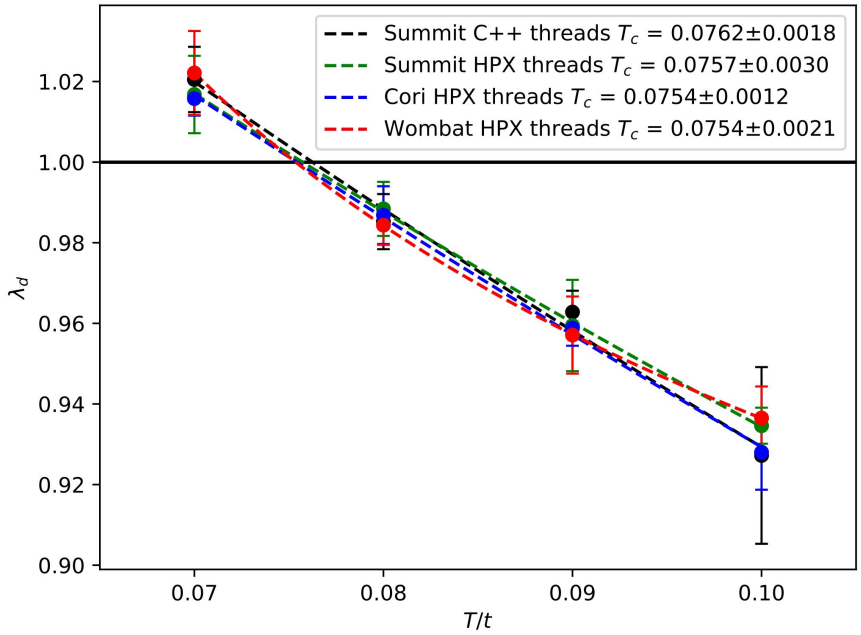
- For Summit run, each node has 6 MPI ranks; each rank has 7 CPUs + 1 GPU; each rank has 7 threads. This config. is the most optimal config. after parameters sweep.

Correctness on multi-architectures

- Obtained correct results from DCA++ runs on various architectures



(a) we validate our science case with C++ std :: thread implementation across three HPC platforms.



(b) Validation using the same case with hpx:: thread implementation across the same three systems. Additionally, we show the C++ std :: thread results on Summit as a reference.

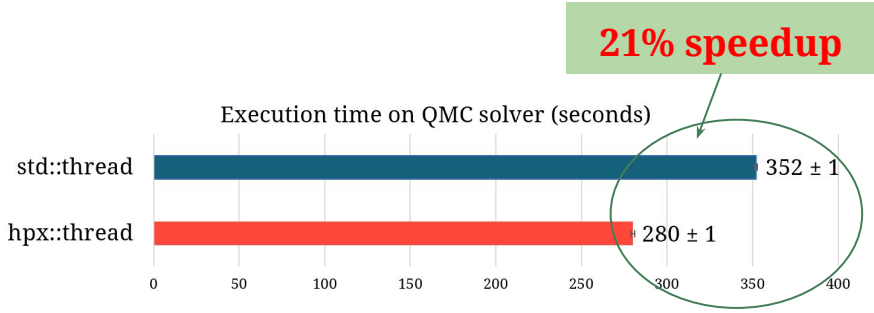
Runtime Comparison

- Configuration: 1 Summit node
- Each node has 6 MPI ranks; each rank has 7 CPUs + 1 GPU; each rank has 7 threads. This config. is the most optimal config. after parameters sweep.

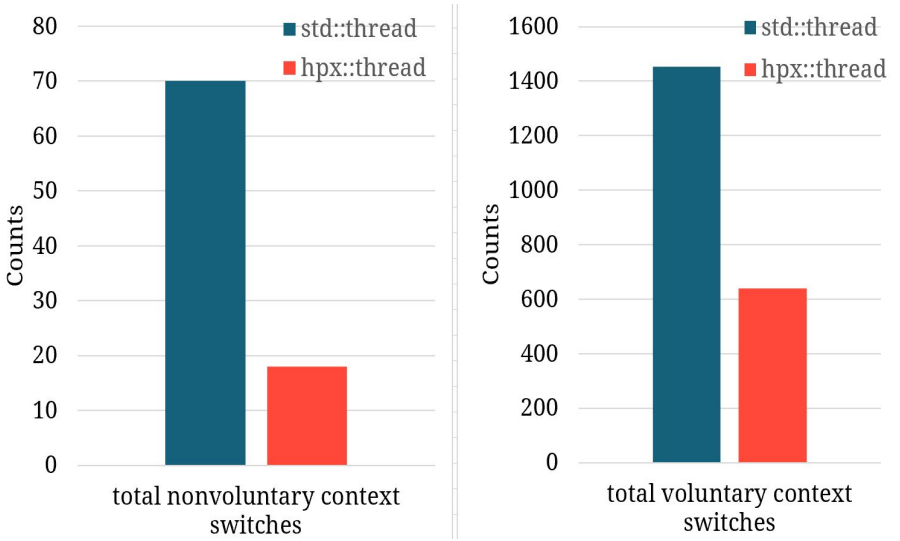
- Results for 100k monte carlo measurements with error bars obtained from 5 independent executions.

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

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



(a) Runtime comparison. Shorter is better.



(b) Context switch analysis. Lower is better.

NVIDIA Nsight System Profiling for CPU utilization

- hpx::thread uses hwloc to achieve thread-affinity
  - sets one hyper-thread per physical core.
  - Overhead ↘
  - CPU Utilization↗️
- std::thread spreads work over 4 hyper-threads per physical core.

OAK RIDGE National Laboratory

STE||AR GROUP

Autonomic Performance Environment for eXascale (APEX)

APEX

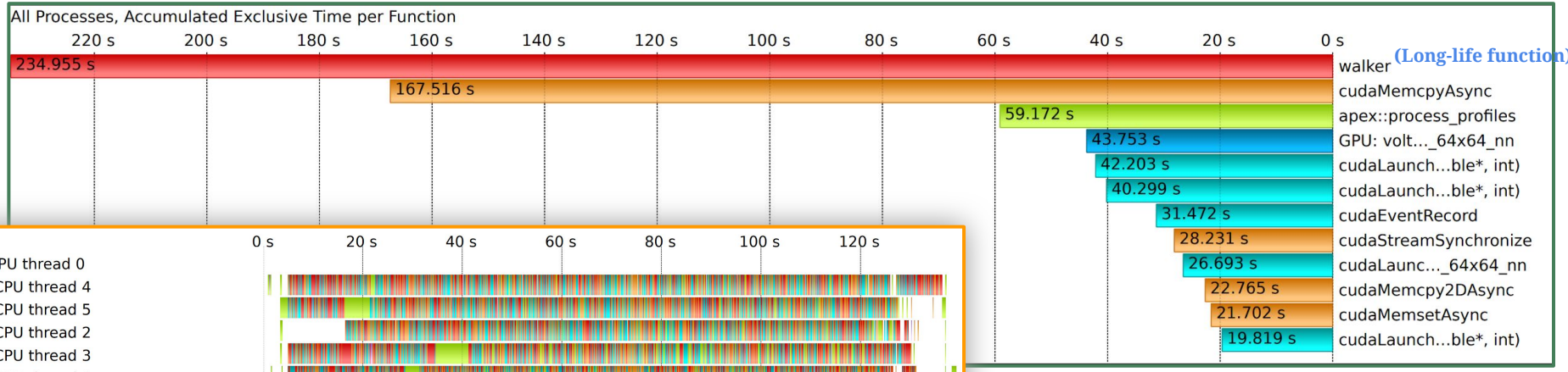
UNIVERSITY OF OREGON

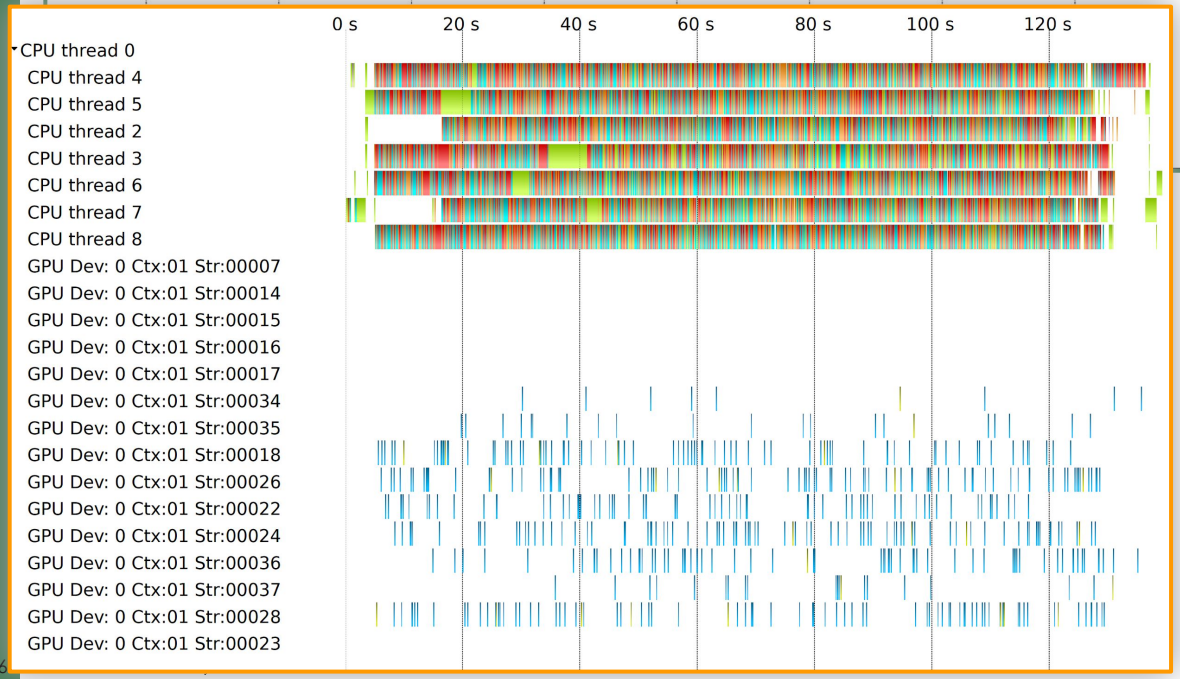
APEX

- Autonomic Performance Environment for eXascale
- *Performance Measurement* - designed for distributed, asynchronous tasking runtimes with task pre-emption and high concurrency (i.e. **HPX**)
  - Performance profiling & tracing support, task dependency graphs
  - Hardware / OS monitoring (utilization, power, **HPX**, others)
  - Native support for: OpenMP, NVIDIA CUDA, kokkos, OpenACC, C++
- *Policy Engine* - infrastructure for feedback/control, runtime optimization and auto-tuning (not used in this work)

https://github.com/khuck/xpress-apex

APEX Tracing HPX & CUDA, visualized in Vampir*



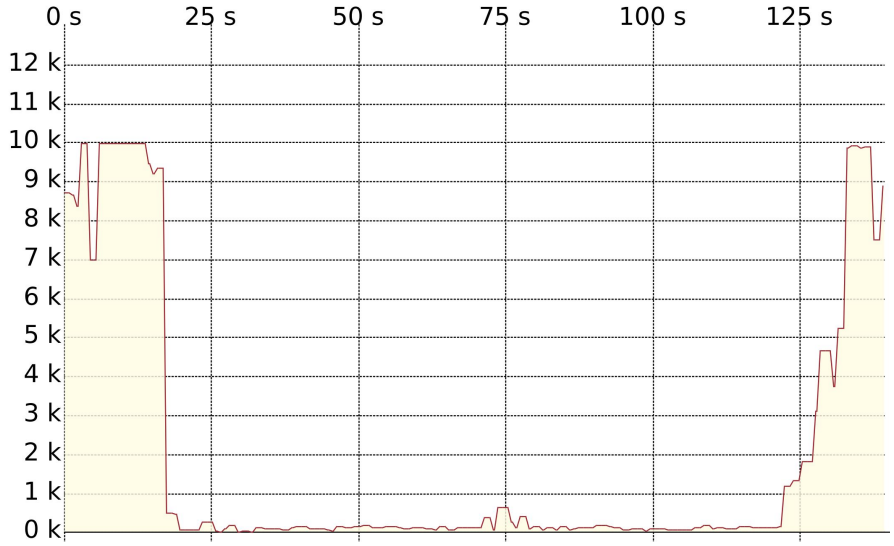


(above) Top 10 time consuming functions.

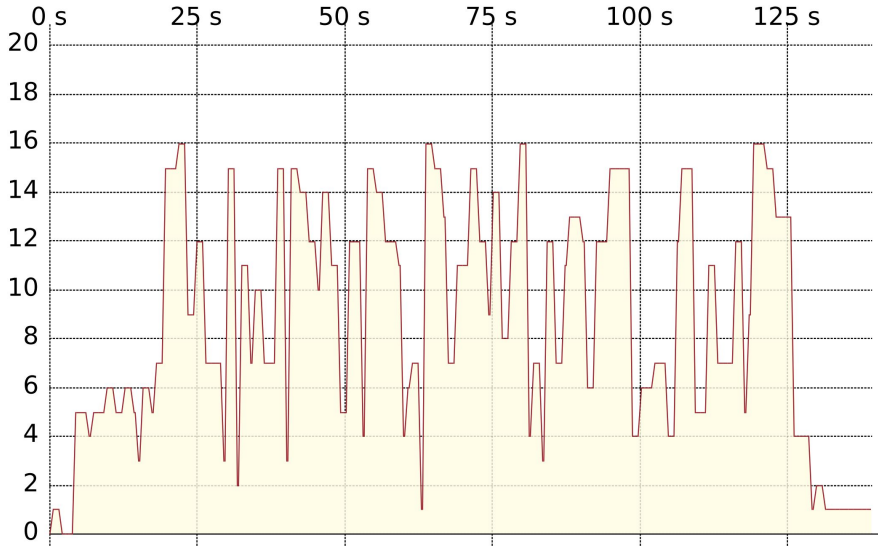
(left) Master timeline plot monitored events including CPU and GPU activities. APEX writes natively to OTF2, the trace format supported by Vampir. * https://vampir.eu

STE||AR GROUP

APEX monitoring of native HPX Performance Counters*



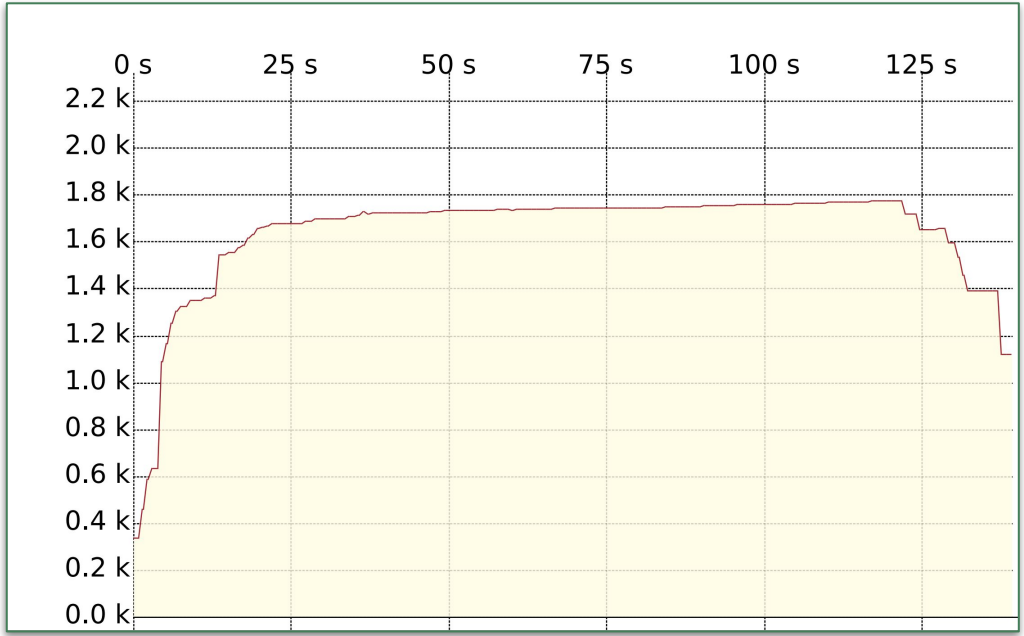
(a) HPX thread idle rate (unit: 0.01%), lower is better.



(b) HPX queue length (unit: number of available tasks), higher is better.

* https://hpx-docs.stellar-group.org/tags/1.3.0/html/manual/optimizing_hpx_applications.html

APEX monitoring of NVIDIA NVML data* & more



(a) APEX monitoring results on Summit summarizing device memory used (unit: megabytes) over time

```
Counter
-----------------------------------------
                    1 Minute Load average
    Bytes Allocated for GPU <- cudaMalloc
Bytes Allocated for GPU <- cudaMallocHost
                              CPU Guest %
                           CPU I/O Wait %
                                CPU IRQ %
                               CPU Idle %
                               CPU Nice %
                              CPU Steal %
                             CPU System %
                               CPU User %
                            CPU soft IRQ %
          Device 0 GPU Clock Memory (MHz)
              Device 0 GPU Clock SM (MHz)
            Device 0 GPU Memory Free (MB)
            Device 0 GPU Memory Used (MB)
        Device 0 GPU Memory Utilization %
           Device 0 GPU NvLink Link Count
           Device 0 GPU NvLink Speed MB/s
       Device 0 GPU NvLink Utilization C0
       Device 0 GPU NvLink Utilization C1
                    Device 0 GPU Power (W)
             Device 0 GPU Temperature (C)
               Device 0 GPU Utilization %
       Device 0 PCIe RX Throughput (MB/s)
       Device 0 PCIe TX Throughput (MB/s)
       GPU: Bandwidth (GB/s) <- <unknown>
GPU: Bandwidth (GB/s) <- Memory copy DtoD
GPU: Bandwidth (GB/s) <- Memory copy DtoH
GPU: Bandwidth (GB/s) <- Memory copy HtoD
GPU: Bandwidth (GB/s) <- Memory copy HtoH
                  GPU: Bytes <- <unknown>
           GPU: Bytes <- Memory copy DtoD
           GPU: Bytes <- Memory copy DtoH
           GPU: Bytes <- Memory copy HtoD
           GPU: Bytes <- Memory copy HtoH
```

(b) more GPU/CPU profiling options

* https://developer.nvidia.com/nvidia-management-library-nvml

Ongoing work

- **HPX task continuation:**
  - Wrapping DCA++ cuda kernel into HPX future → overlapping communication and computation
- **GPUDirect RDMA:**
  - Building ring abstraction over GPUDirect to optimize distributed quantum monte carlo solver
- **Arm A64fx evaluation:**
  - Evaluate DCA++ performance on Arm A64fx cluster (Wombat @ORNL)

Thanks!
Q&A section
Performance Analysis of a Quantum Monte Carlo Application on Multiple Hardware Architectures Using the HPX Runtime