Challenges and Opportunities in Co-Design for High-Performance Computing Software Systems

Min Si

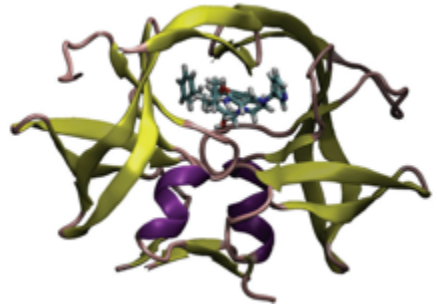
Argonne National Laboratory, USA

REBASE Talk at SPLASH 2020

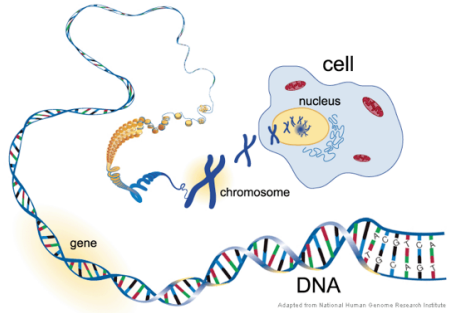
2020-11-20

HPC for Scientific Computing and Beyond

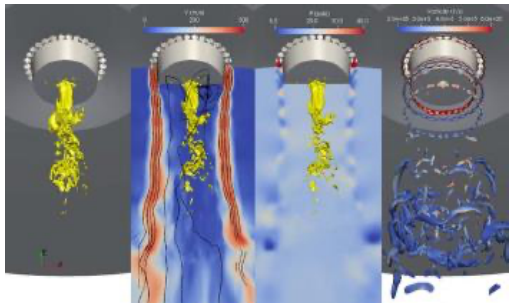
- High-performance computing (HPC) systems are designed for pursuing **extreme-scale parallelism and computational power**



- Supports computational-expensive simulations as well as data-intensive analysis for scientific discovery and industry innovation
  - Aerospace: airframes and jet turbines simulations
  - Energy: nuclear power, battery innovation
  - Quantum advances: molecular system simulations
  - Health care: precision medicine, genome analysis
  - Manufacturing: smart manufacturing, computational fluid dynamics simulations
  - …



Genome Analysis.
Image adapted from National Human Genome Research Institute

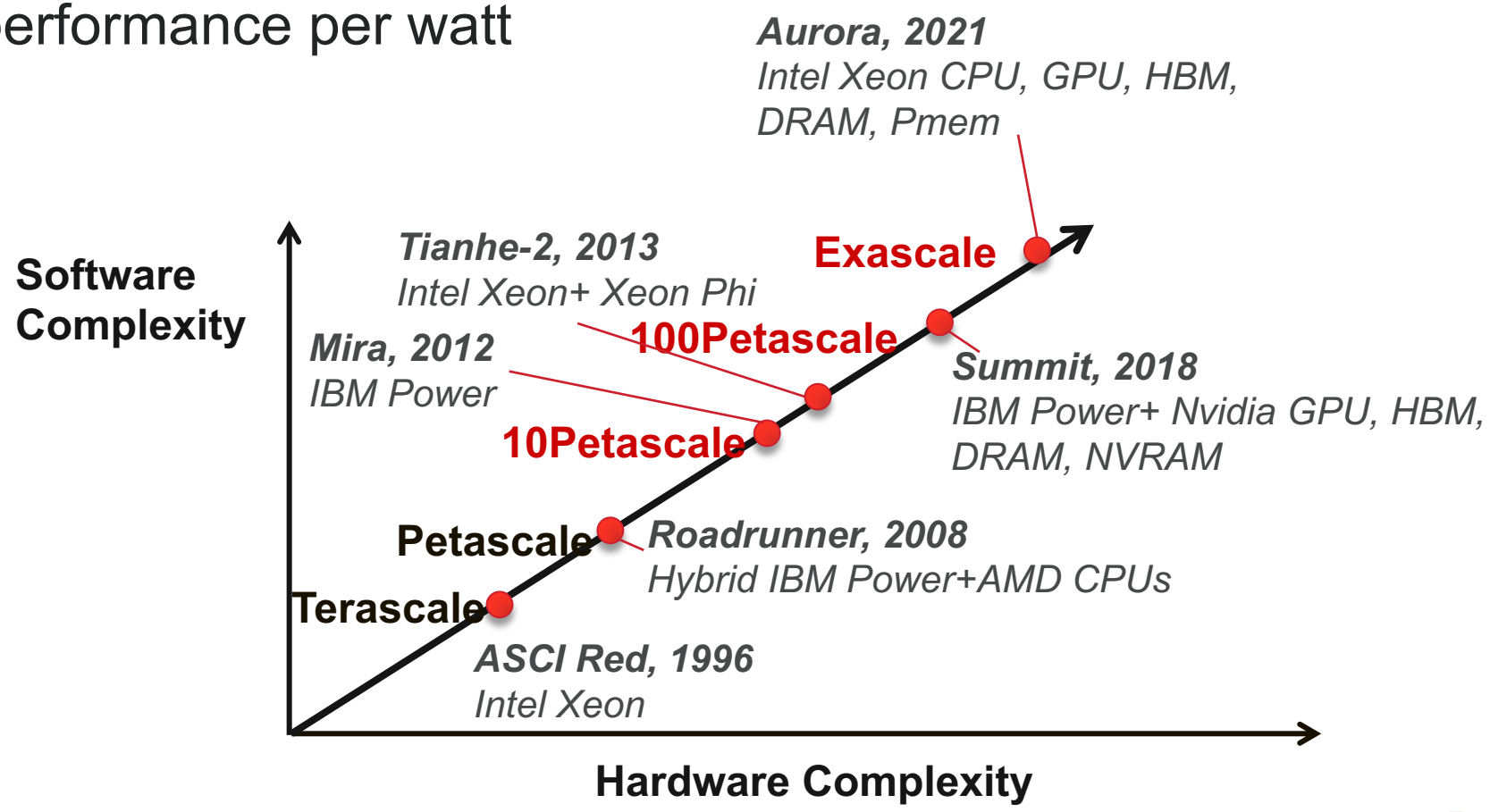


Simulation of the atomization process at energy production.
Image adapted from https://hpc4mfg.llnl.gov/

Aurora Supercomputer at ALCF. Image adapted from alcf.anl.gov

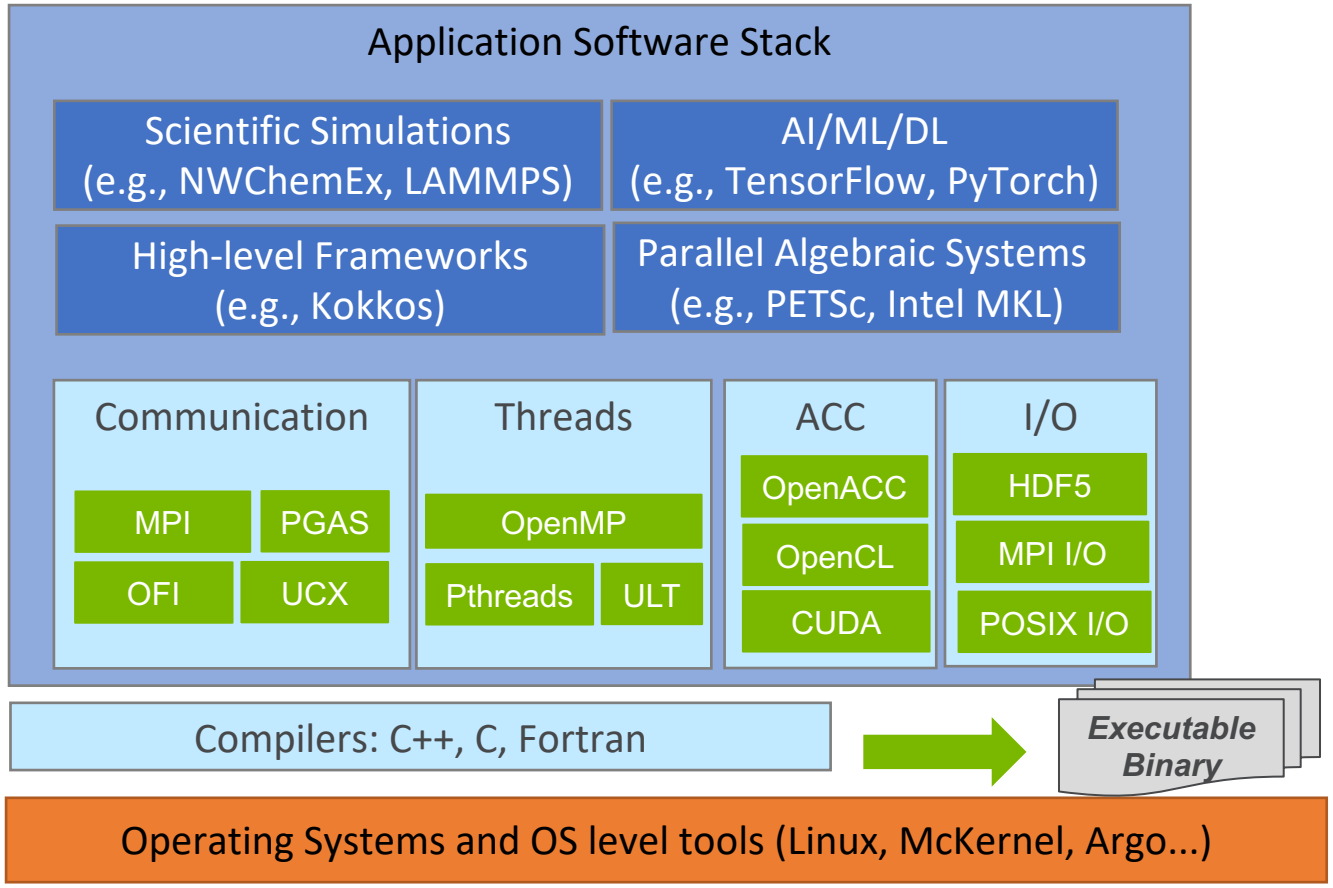
Trend of HPC Architecture

- End of Moore's Law
- From single-core frequency to multi/many-core parallelization
- From performance to performance per watt



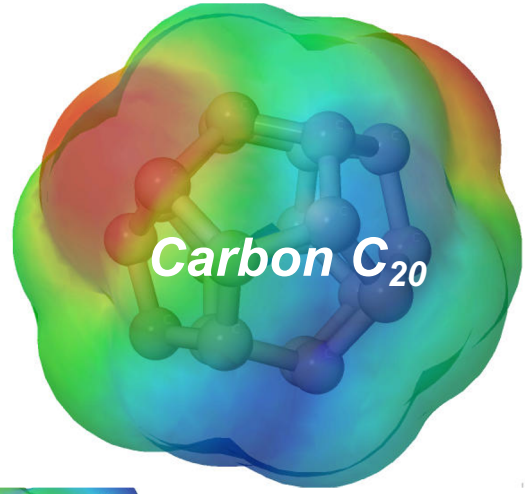
Deep and Wide Software Stack

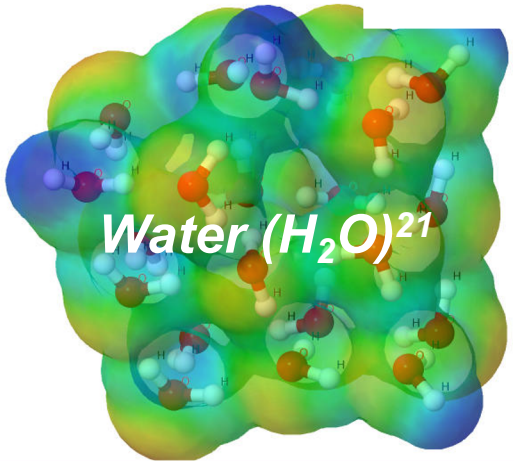
- Complex computational requirements from applications
  - Compute-bound v.s. memory-bound v.s. IO-bound
  - Bulk synchronous parallel v.s. data-driven
  - Simulation v.s. AI/ML/DL training & inference
- Diverse capacities provided by underlying hardware
  - Massive on-node parallelism
  - CPU+GPU
  - Interconnects protocol and topology
  - Parallel file systems



Deep HPC Stack: NWChem [1]

- High performance computational chemistry application suite
- Quantum level simulation of molecular systems
  - Very expensive in computation and data movement, so is used for small sy
  - Larger systems use molecular level simulations
- Composed of many simulation capabilities
  - Molecular Electronic Structure
  - Quantum Mechanics/Molecular Mechanics
  - Pseudo potential Plane-Wave Electronic Structure
  - Molecular Dynamics
- Very large code base
  - 4M LOC; Total investment of ~1B $ to date

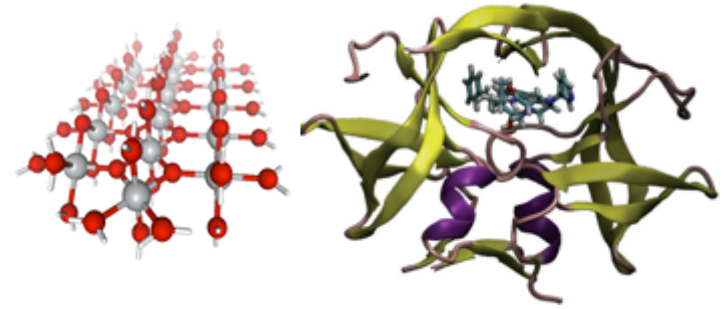


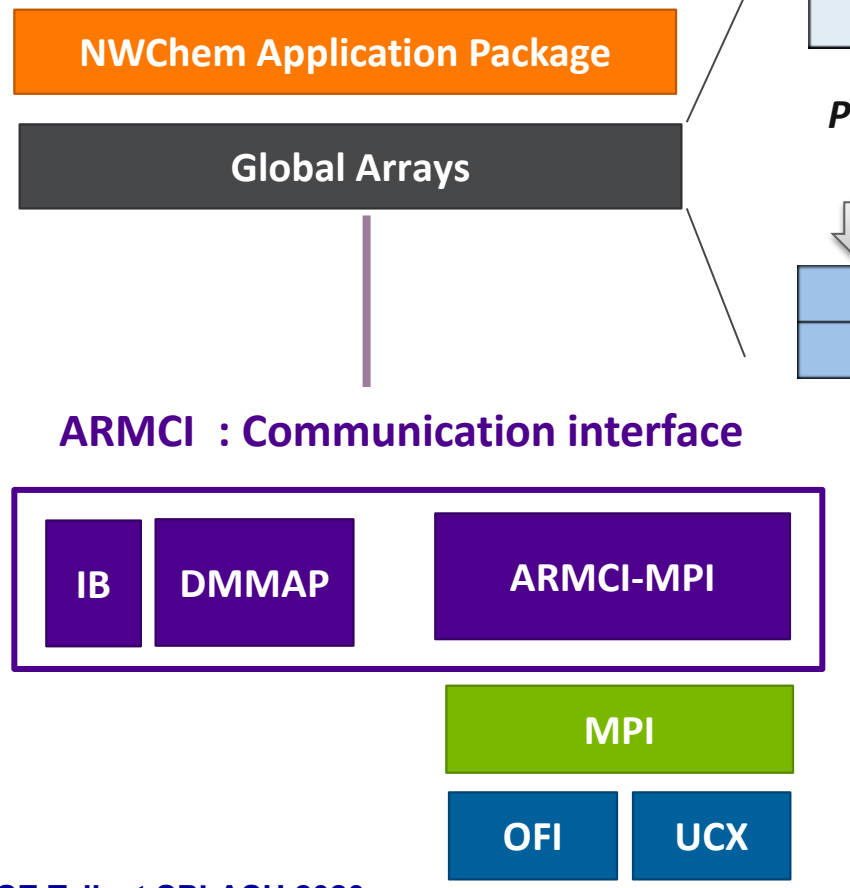


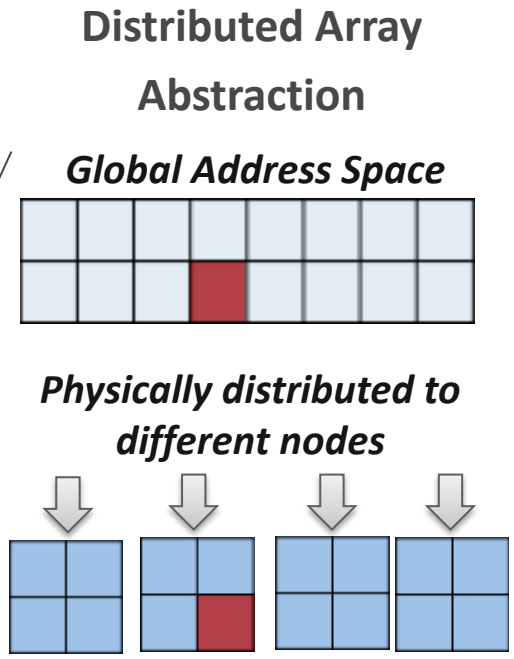
[1] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations" Comput. Phys. Commun. 181, 1477 (2010)

Deep HPC Stack: NWChem

- Deep software stack of NWChem
  - **Global Arrays** defines the abstraction of distributed arrays
  - **ARMCI** defines the multidimensional array-oriented communication interface for Global Arrays
  - **ARMCI-MPI** is a port of ARMCI based on MPI-3 RMA
  - **MPI** defines data elements-oriented data transfer
  - **OFI/UCX** provides low-level bytes-level data transfer for different interconnects



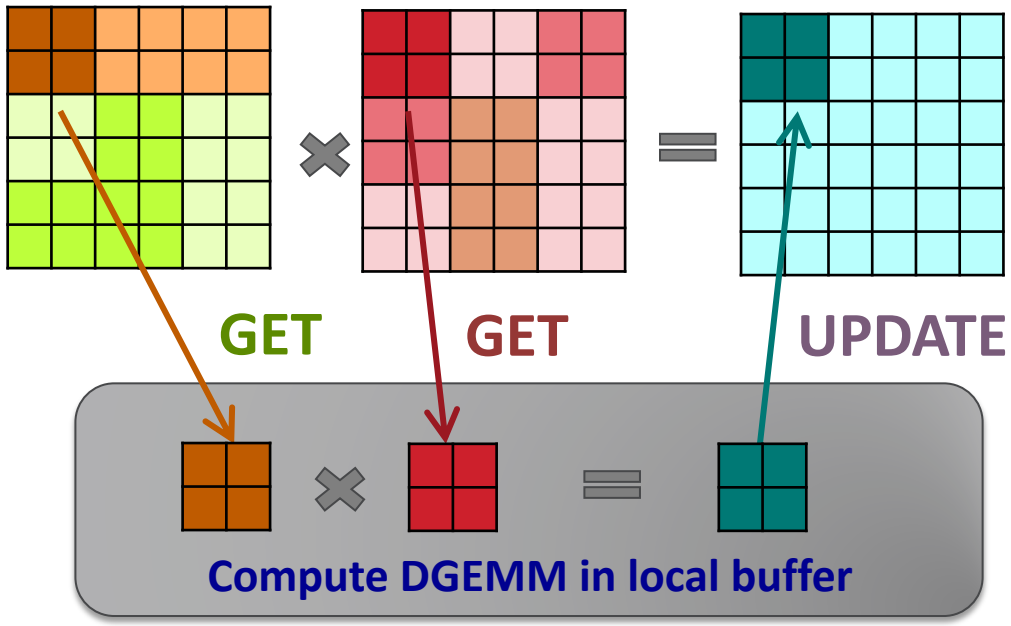




Performance Challenges in NWChem Deep Stack

- Typical Get-Compute-Update algorithm for matrix-matrix multiplication
- Applied to multi-dimensional tensors in NWChem

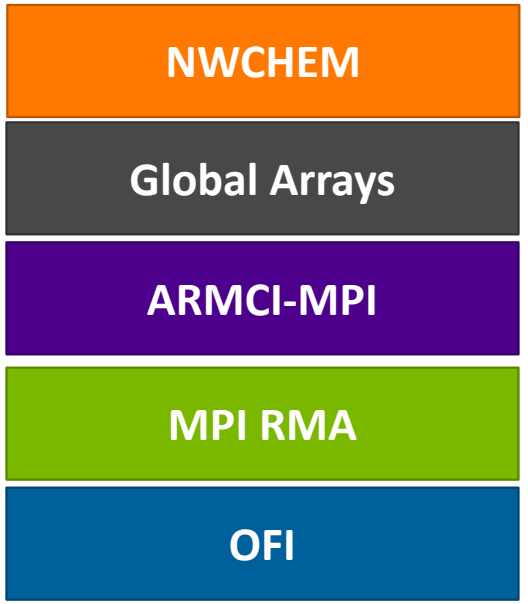
Demonstrating Get-Compute-Update mode on 2D matrix
(Note: NWChem computes on multi-dimensional tensors)



Pseudo code of C += A x B

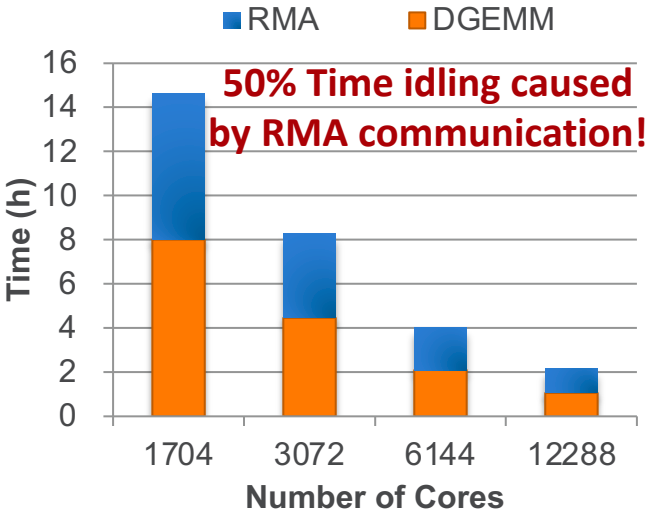
```
for i in I blocks:
  for j in J blocks:
    for k in K blocks:
      GET block a from A
      GET block b from B
      c += a * b /*computing*/
    end do
    Update block c to C
  end do
end do
```

Software Stack

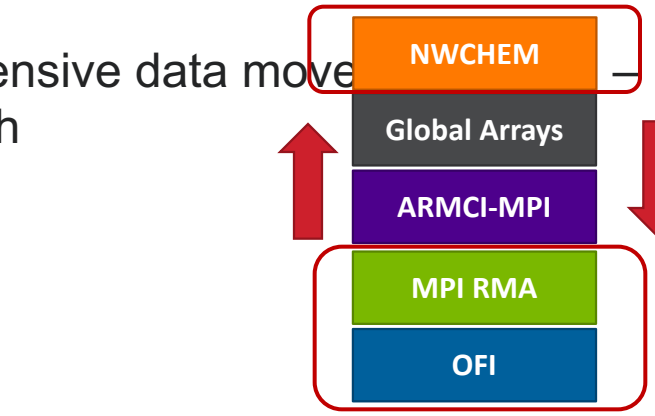


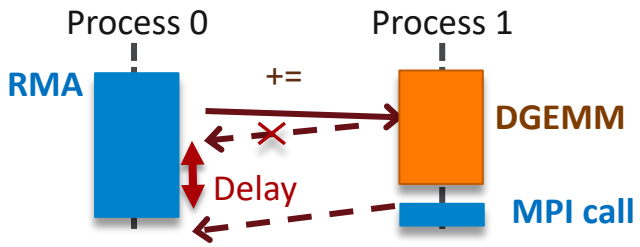
Performance Challenges in NWChem Deep Stack

- Insufficient implementation attributes exposed from low level runtime
  - Unaware of software-emulated RMA in MPI/OFI
  - Caused extremely expensive data move in NWChem critical path
- Insufficient computation/communication characteristics shared by application
  - Some app phases involve dense data movement, but some others are sparse
  - Runtime is unaware of such characteristics, cannot make fine-grained resource adjustment

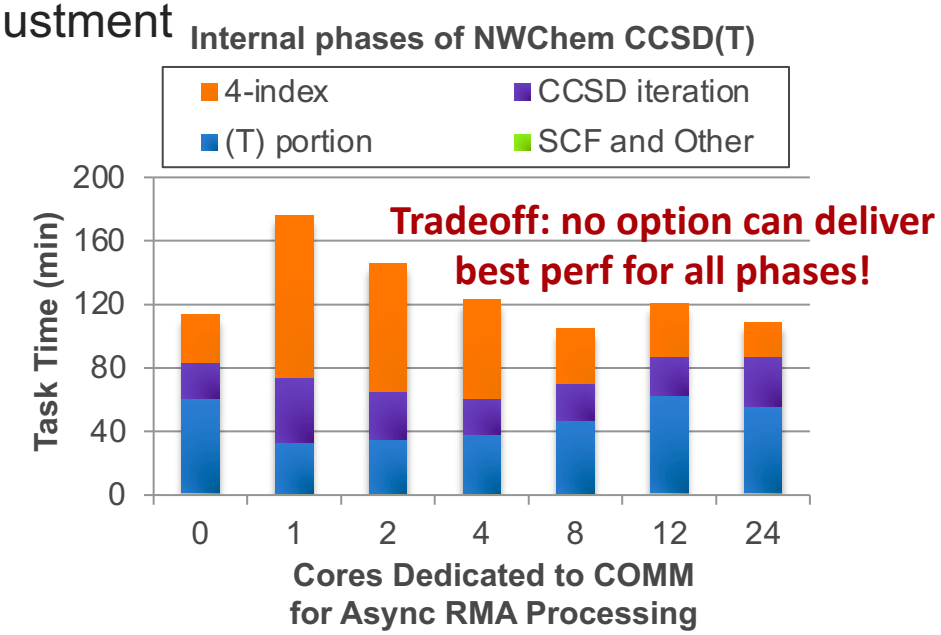


Overhead breakdown for computation-intensive O(N7) tensor contraction in "Gold standard" CCSD(T) with large Water-21 problem (NERSC Edison)





Software-emulated MPI RMA



Suboptimal perf of CCSD(T) with varying number of COMM-dedicated cores with Tetracene problem (NERSC Edison)

Wide HPC Stack: QMCPACK [1][2]

- Quantum Monte Carlo Simulation for Material Science
- Enable multiple simulation capabilities for electronic structure problems
  - Auxilary Field Monte Carlo
  - Variational Monte Carlo
  - Diffusion Monte Carlo
  - Backflow wavefunctions
  - All electron and non-local pseudopotential calculations
  - ...

QMCPACK

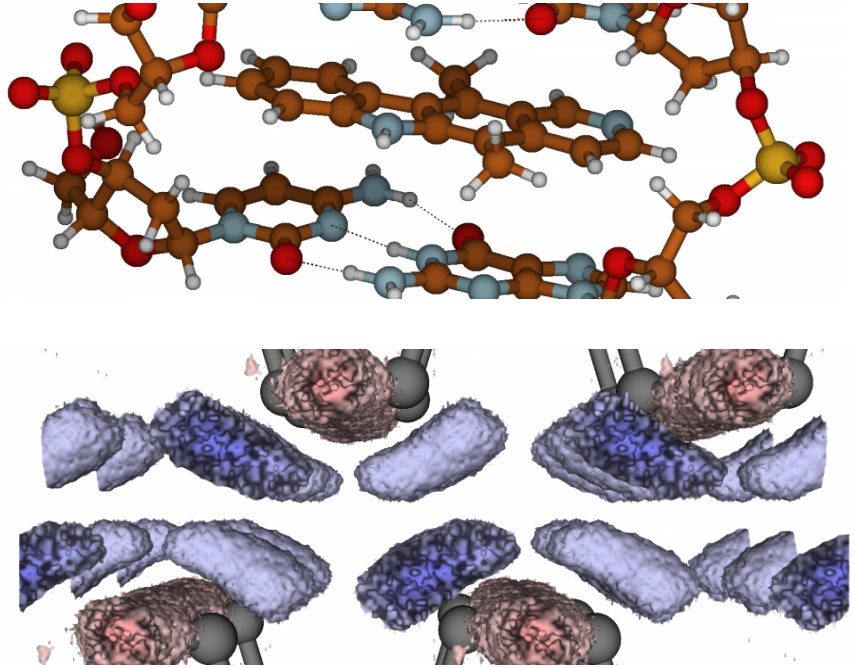


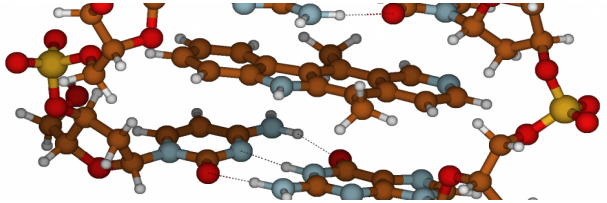
Image adapted from qmcpack.org

[1] J. Kim et al. QMCPACK: an open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules and solids. Journal of Physics: Condensed Matter (2018).
[2] qmcpack.org

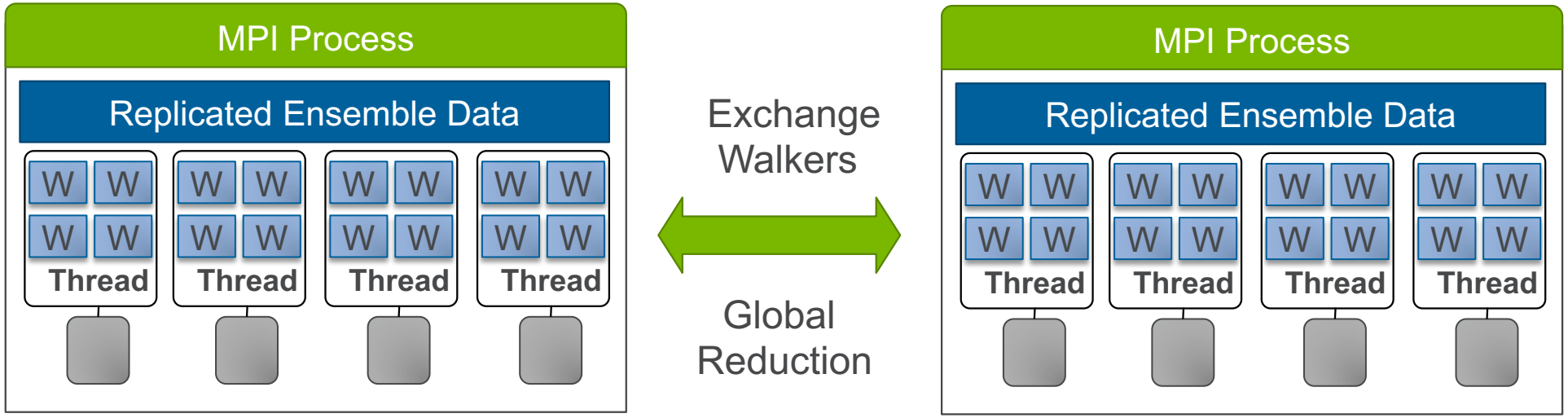
Wide HPC Stack: QMCPACK

- Size of physical system to simulate is bound by memory capacity
  - Memory space dominated by large replicated ensemble tables (Gbytes and up)
  - Regular communication pattern
    - All processes finish their local computation, and then exchange data
- **Hybrid MPI+OpenMP+CUDA**
  - Share large ensemble table among threads on single node
  - Use MPI processes for inter-node communication
  - Recent updates for leveraging NVIDIA GPU acceleration

QMCPACK

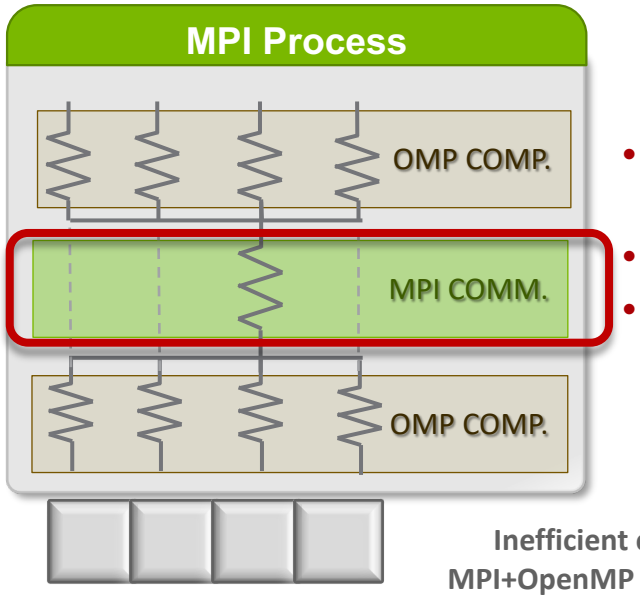


MPI | OpenMP | CUDA



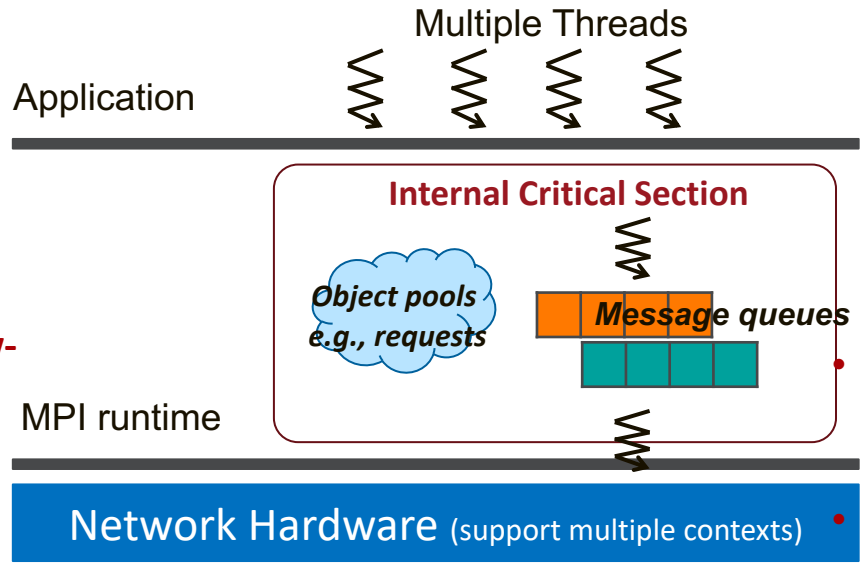
Performance Challenges in QMCPACK Wide Stack

- Lack of interoperability between runtime libraries
  - MPI and OpenMP runtimes in the QMCPACK example
  - Similar issues also exist between many other on-node parallelism and off-node communication libraries too!
  - Convenient MPI Funneled mode delivers suboptimal communication
  - MPI Multithreading mode still cannot enable parallelism in communication due to limitation of mainstream MPI implementations



- **Funneled mode: only master thread can call MPI**
- **Large amount of IDLE threads**
- **Single core delivers poor performance especially on low-freq core (e.g., Xeon Phi)**

Inefficient core utilization in MPI+OpenMP with funneled mode



- **Expensive lock contention**
- **Only limit #cores can post data concurrently**
- **low utilization of network resources**

Serialized communication in MPI+OpenMP with multithreading mode

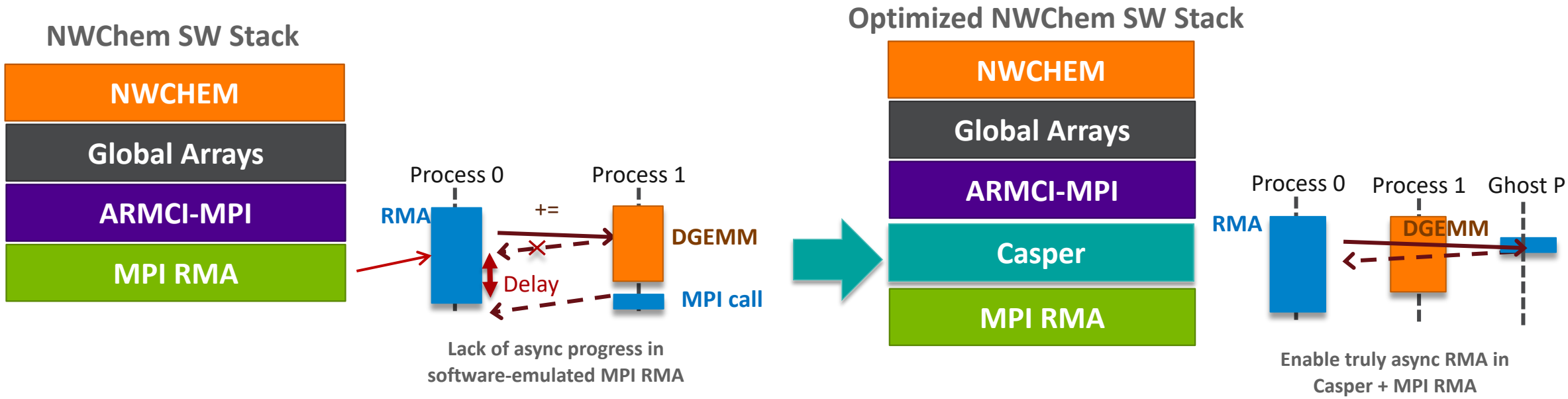
Deep and Wide HPC Stack: Co-Design is the Key

- Abstraction allowed developers and researchers to concentrate on innovations of individual software components
  - **Good aspects**: simplify problem, achieve generalization/component reuse
  - **Drawbacks**: lack of information sharing/coordination caused suboptimal performance
- Software-level exascale and beyond truly relies on full utilization of all computation/communication resources!
- **Co-design** is the key to minimize suboptimal resource utilization, thus enabling system full scale performance

CO-DESIGN SHOWCASES IN SOFTWARE STACK

Showcase 1: Co-Design for Communication Resource Management

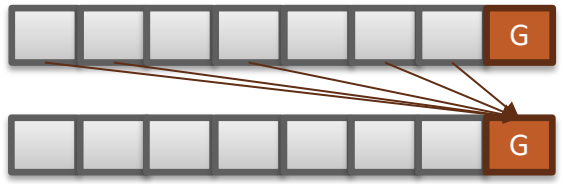
- Application – Communication Runtime Co-Design
  - Address suboptimal communication in NWChem
    - Application is unaware of software-emulated RMA in MPI/OFI
    - Caused extremely expensive data movement in critical path
  - **Casper: a portable progress management layer on top of MPI**
    - Manage communication progression in software, deliver same level of performance as that from network HW-handled RDMA



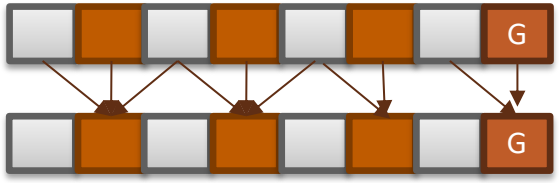
Adaptive Communication Resource Management in Casper

- Most data movement today still relies on CPU resources due to limited capability of network hardware (e.g., NIC)
  - E.g., cannot process communication-driven computation with noncontiguous data layout
  - The software has to tradeoff CPU resources for computation and communication requests
    - Root cause of inefficient data movement in NWChem!
- Managing CPU resources in multiphase applications is challenging!
  - **Computation-heavy phase** prefers to assign only a few number of cores to communication, thus the other cores can be used to speedup heavy computation
  - **Communication-heavy phase** requires more cores for communication progress; otherwise COMM. bottleneck may occur
  - **Such application performance characteristics is not exposed to runtime!**

Heavy COMP Phase
Utilize only single progress core for COMM

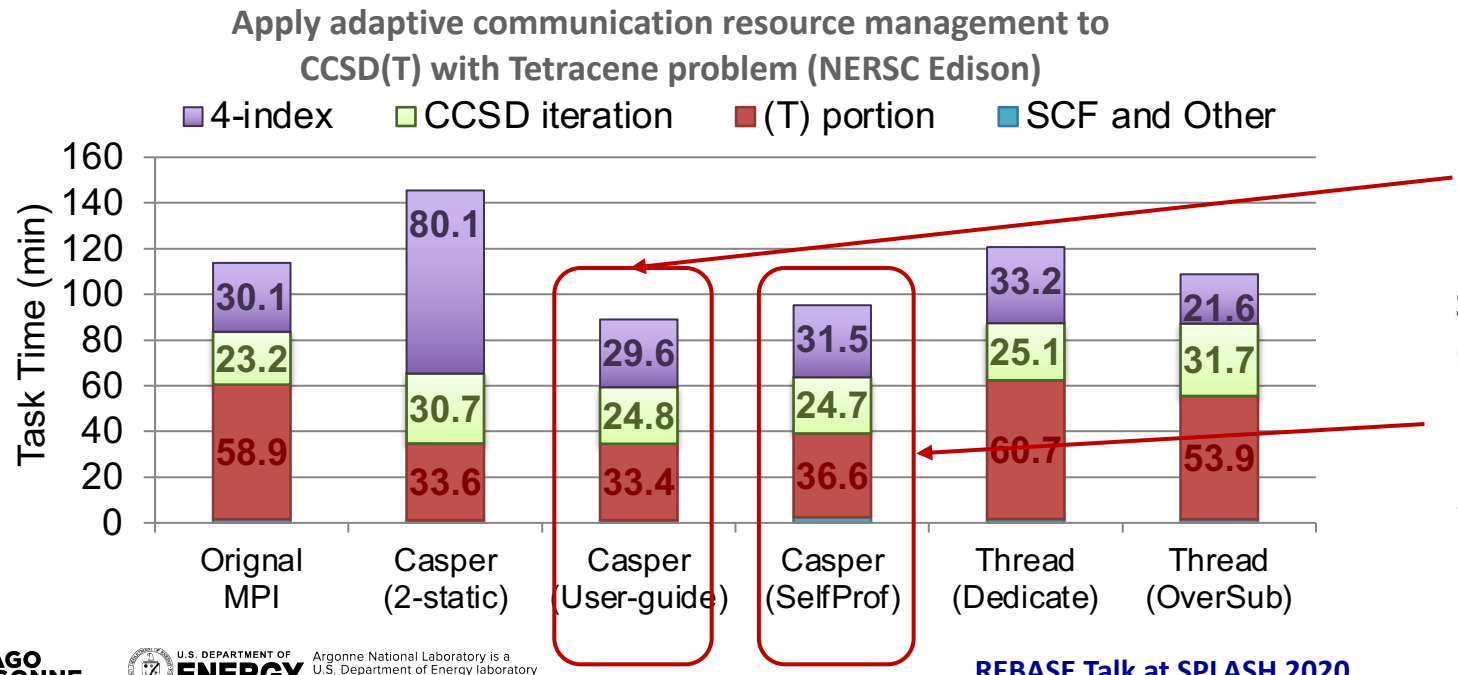


Heavy COMM Phase
Assign more progress cores for COMM



NWChem with Adaptable Communication Resource Management

- Studied two approaches:
  - Application-guided approach: application specifies the need of communication progress
  - Automatic runtime profiling approach: runtime inserts timers to predict the need of communication progress
- Both worked for NWChem but app-guided approach delivers the best performance



App-guided adaptation:

- Immediate adaptation
- Optimal result

Self-profiling-based adaptation:

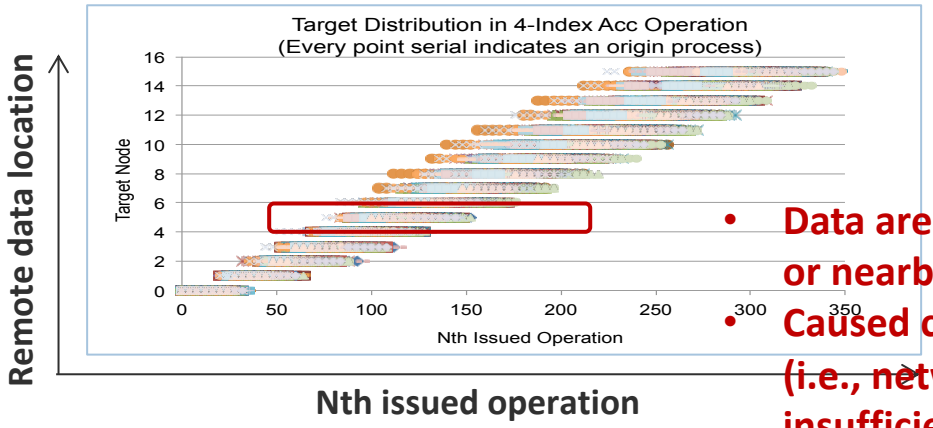
- Might be delayed because it relies on user synchronization point, or periodical global synchronization
- Additional overhead caused by background synchronization/profiling (already highly optimized runtime)

What Can Be Done Better?

- Suboptimal communication patterns (or data locality) can be detected by runtime techniques
  - Aggregated data transfer was detected in NWChem (a high-optimized application package)
  - Too complex to ensure perfect data locality in application algorithms
- Can runtime coordinate with the application layers to dynamically improve data locality?
  - Co-design is essential!

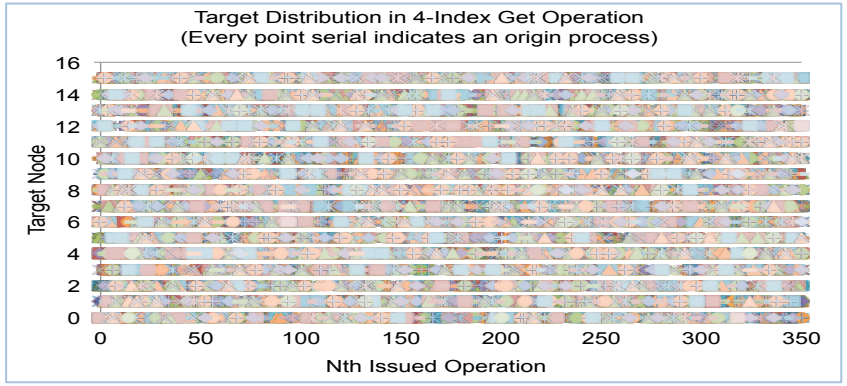
Suboptimal Data Accessing/Computing

(Every point serial indicates an MPI process)



- **Data are transferred to the same or nearby nodes**
- **Caused communication hotspots (i.e., network congestion, insufficient CPU resources)**

Evenly Distributed (Optimal) Data Accessing

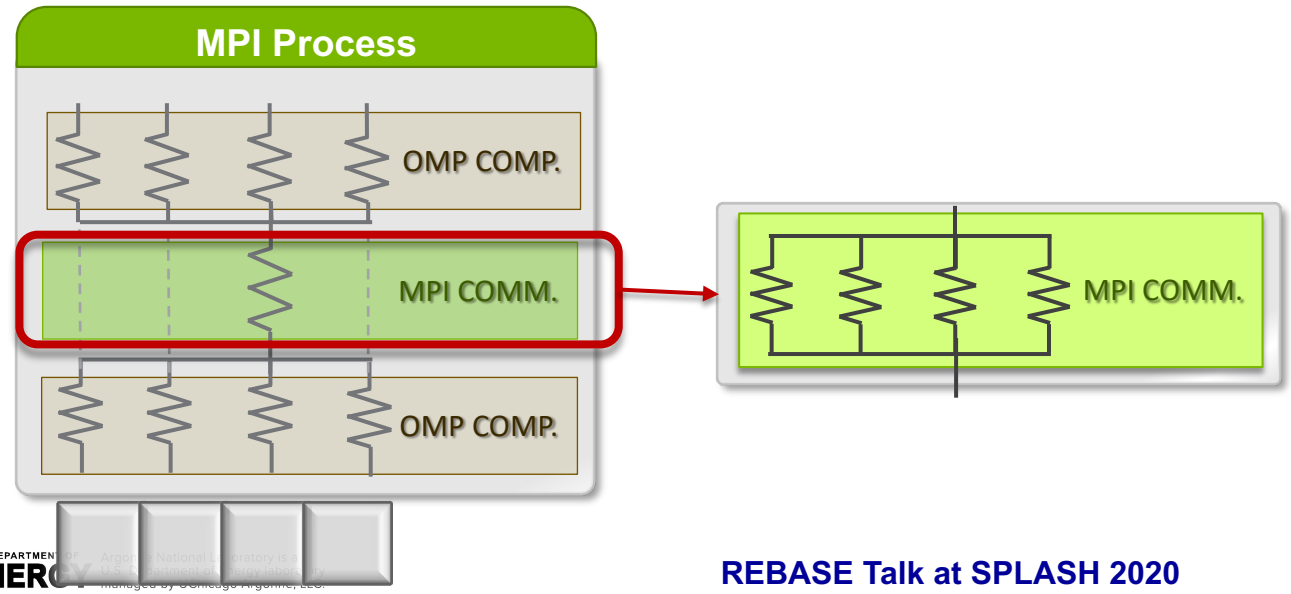


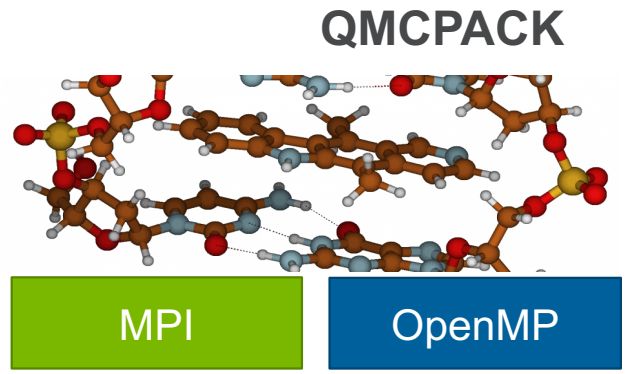
Take Away from Communication Resource Management

- Efficient utilization of HPC resource requires full understanding and coordination between multiple software layers
  - Understand the behavior of application (e.g., data distribution of global array, communication/computation workload)
  - Understand cost of communication in low level runtime (e.g., HW Get/Put, SW ACC on RDMA network)
    - Limited by hardware capabilities (e.g., RDMA, network topology, computing power on NIC)
- Smart NIC will be the ideal solution?
  - All communication-driven computation can be offloaded to powerful computing units on NIC
  - Requirement to network hardware innovation

Showcase 2: Co-Design for MPI and Threads Runtime Coordination

- Runtime – Runtime Co-Design in hybrid MPI + X mode
  - Focus on MPI+OpenMP runtime coordination in the QMCPACK example
  - **MT-MPI: Parallelizing MPI internal processing by utilizing user IDLE threads shared from application**
    - Opportunity: Many MPI internal tasks can be parallelized by threads (e.g., using OMP)
    - Key challenge: MPI runtime is unaware of the idleness of threads during the call of MPI!
    - **Require thread idleness info (e.g., how many threads are idle?) exposed from OpenMP**





Thread idleness detection in MT-MPI

- **Algorithm trade off**
  - Modify MPI internal algorithms for better parallelization
  - Parallel algorithms might not be efficient with insufficient #threads
  - Need tradeoff between parallel / sequential versions according to **number of IDLE threads**
- **Nested parallelism**
  - Nested parallel region if MPI call is inside user parallel region
  - Simply creates new Pthreads, and offloads thread scheduling to OS
  - Need manual guidance to avoid thread oversubscription using **number of IDLE threads**
- **Number of IDLE threads at MPI runtime is UNKNOWN!**
- **Our solution: keep track on thread idleness at OMP runtime and expose such info to MPI**

```
MPI_Function (…)
{
  /* Parallel algorithm */
  #pragma omp parallel for
  {…}
}
```

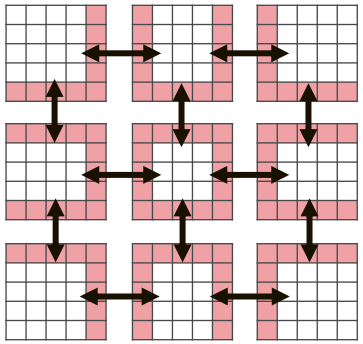
```
if ( num_idle_threads < N )
{   /* Sequential algorithm */ }
else
{   /* Parallel algorithm */ }
```

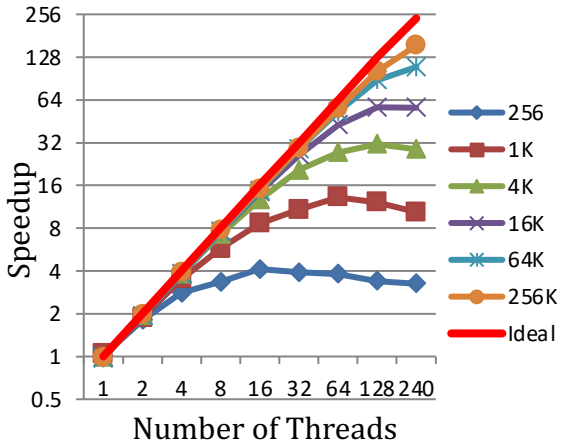
```
#pragma omp parallel        Creates N Pthreads !
#pragma omp single
{
   MPI_Function()
   {
                            Creates N Pthreads !
      #pragma omp parallel
      {…}
   }
}
      num_threads (num_idle_threads)
```

Performance of MPI Communication with Cross Runtime Coordination

- **Parallelizes** various aspects of the MPI processing

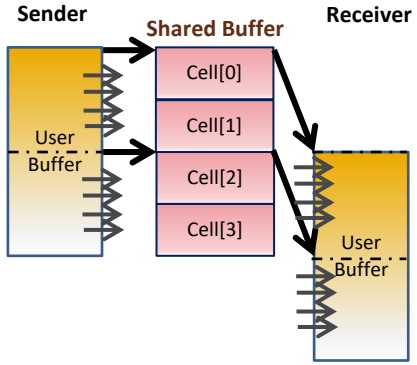
Derived Data Type Packing Processing

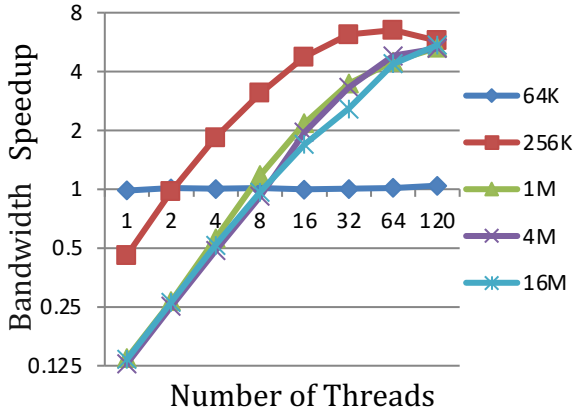




** 2D double elements matrix local packing*

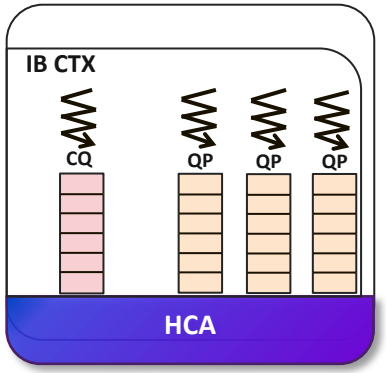
Intra-node Large Message Communication

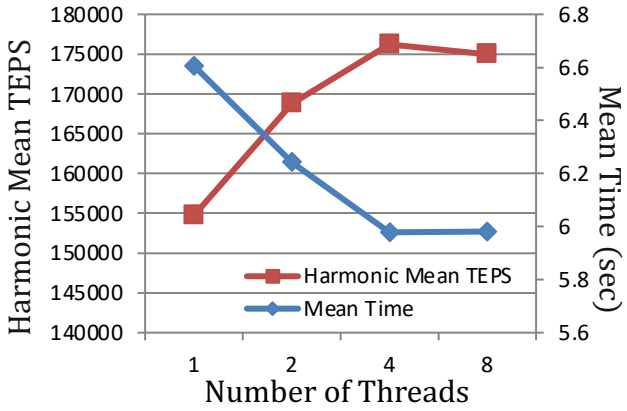




** OSU P2P benchmark*

Data Posting to IB Network





** Graph500 benchmark (MPI oneside version) using 9 MPI processes*

- 1 KNC per node
- 56 GB/s FDR IB
- KNC **native** mode
- 1 MPI process per KNC

Take Away from Cross-Runtime Coordination

- Computing resources (e.g., CPUs, GPUs) are shared by coexisting runtime subsystems
- Ways to make efficient use across libraries is under investigation
  - MPI + OpenMP
  - MPI + GPU
- Low-level resource management facility might be beneficial?
- Cross library/programming model interoperability plays the essential role
  - E.g., OpenMP runtime exposes thread idleness to the other runtime subsystem

Showcase 3: Co-Design for Low-Overhead Multithreading Communication

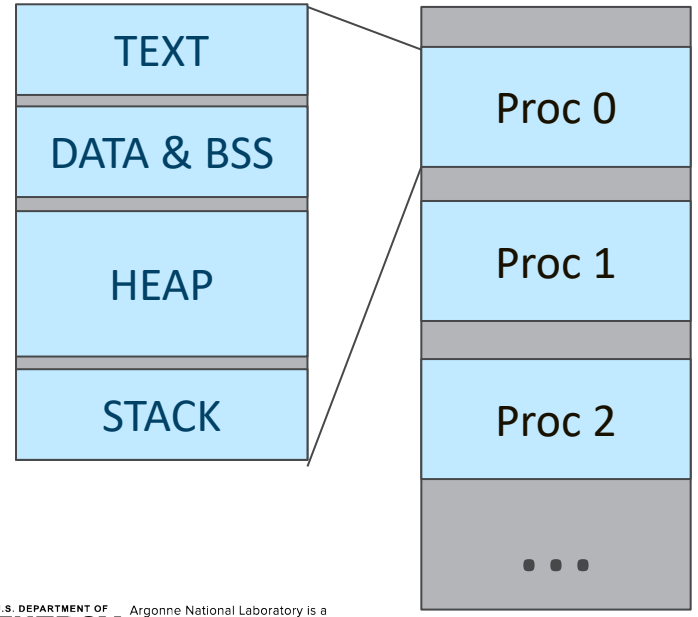
- Multiprocess model (e.g., MPI, PGAS)
  - **Independent Virtual Address Space (VAS)** and **privatized variable sets**
  - **Limitation**: Inefficient intercore communication & data sharing (e.g., message passing)
  - Techniques to enable memory mapping
    - **POSIX shmem**:
      - Explicit allocation (e.g., mmap)
      - Heap only
    - **XPMEM**:
      - Linux kernel module
      - Explicit & expensive exposing
- Multithread model (e.g., OpenMP)
  - **Shared VAS** and **shared variable sets**
  - **Limitation**: Contention between threads on shared variables
    - E.g., expensive multithreading safety when integrating with process-based model such as MPI

- **It is known in the HPC community that multithreading MPI does not perform well!**
  - **Have to use only single thread to call MPI**
  - **Inefficient core utilization as discussed in MT-MPI casestudy**

PiP: HPC-Specialized "Thread" Implementation

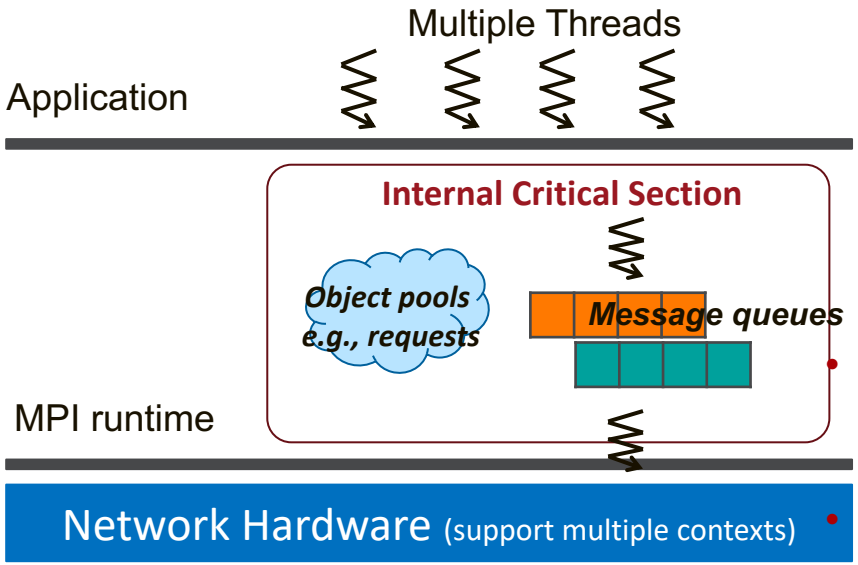
- Co-design of runtime system and low-level OS libraries
- Process-in-Process (PiP): a special combined execution model
  - A process can access data owned by other processes w/o overhead (shared VAS)
  - Variable sets are privatized by default, avoid unnecessary contention
  - Get the good aspects of both multiprocess and multithread modes!

Shared VAS in PiP



Using PiP to Address Multithreading Limitations in MPI

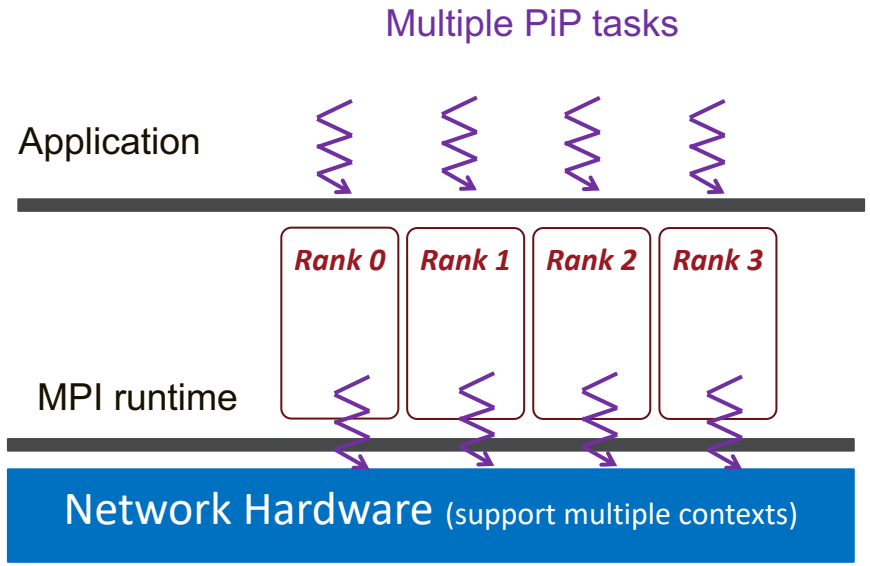
- Original Hybrid MPI+Threads
  - Easy data sharing across cores
  - **Drawback**: inefficient interaction with process-based runtime (e.g., MPI):



- **Expensive lock contention**
- **Only limit #cores can post data concurrently**
- **low utilization of network resources**

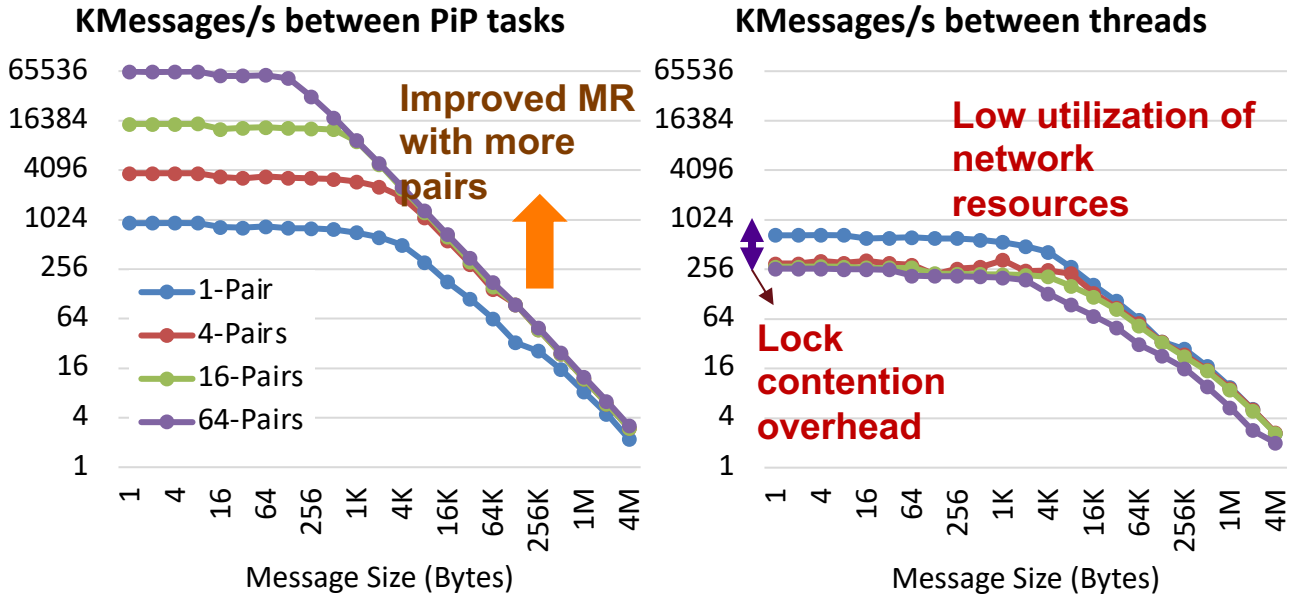
Serialized communication in MPI+OpenMP with multithreading mode

- Hybrid MPI+PiP
  - Use PiP tasks as the underlying support for multithreading runtime (e.g., OpenMP), similar to Pthreads
  - Isolated MPI stacks similar to processes

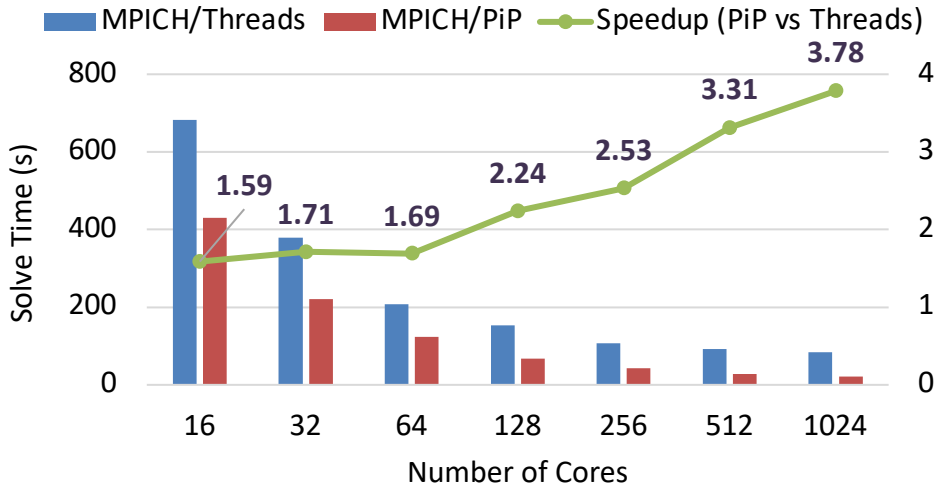


Performance Showcase with HPC-Specialized “Thread”

Multipair message rate (osu_mbw_mr) between two OFP nodes



PiP V.S. threads in hybrid MPI+OpenMP SNAP Particle Transport Proxy Application



Oakforest-PACS (Rank 9 in Top500 2017/11)

- Intel Xeon Phi 7250 68C 1.4GHz (KNL)
- Intel Omni-Path
- MPICH v3.3a3 (with two-level priority lock optimization)

Showcase 4: Specialized OS for HPC

- Mckernal (developed by RIKEN, Japan)
- Innovative OS for Fugaku, the Japanese exascale supercomputer
- Light-weight multi kernel operating system designed specifically for HPC
  - Scalable execution of large-scale parallelism
  - Provide efficient memory/device management to minimize resource contention and data movement
  - Eliminate OS noise by isolating OS services in Linux and provide jitter free execution on the light-weight kernel
  - Support full POSIX/Linux APIs by selectively offloading system calls to Linux

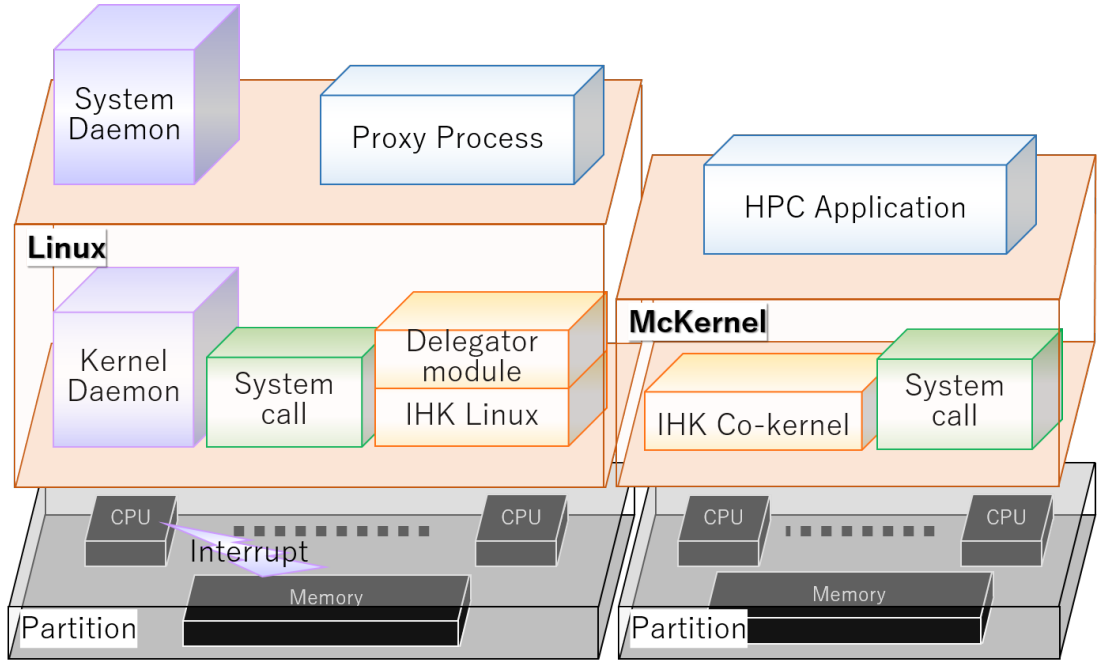
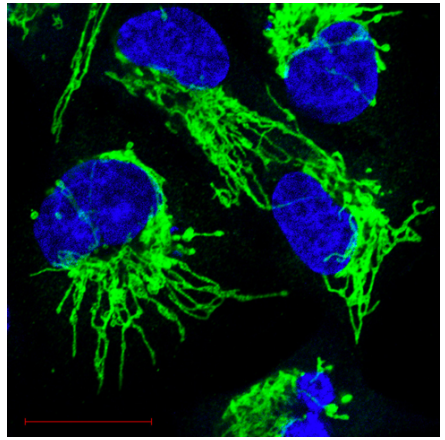


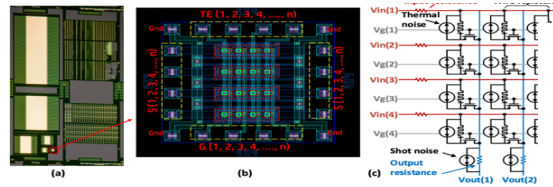
Image adapted from www.sys.r-ccs.riken.jp

Near Future in HPC: SW/HW Full-Stack Co-Design is Going to Be The Norm!

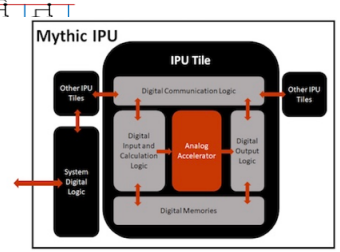
- Emerging convergence of HPC and AI
  - Example: the DOE-NCI CANDLE project
    - Integrate deep learning and simulation to enable precision medicine for cancer
    - Understand disease by using growing volumes and diversity of cancer related data
    - Provide guidance on treatments strategy for individual patients
    - Leverage supercomputing systems for large memory capacity and fast floating point computation
- Future HPC systems for HPC/AI
  - How much computational power do we expect for traditional scientific simulation?
  - How much for inference-like data processing?
  - Integrate AI-inspired HW into the classical HPC system?
  - Application, programming model, runtime?



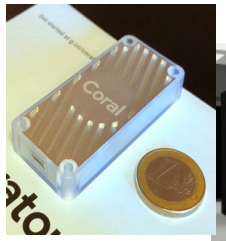
Ras-Driven Cancer. Image by David Kashatus/National Cancer Institute/Univ. of Virginia Cancer Center.



HPE DPE. Image adapted from [1].



Mythic IPU. Image adapted from silvertonconsulting.com



Google Edge TPU. Image adapted from medium.com

Hybrid chip for HPC ?

Challenges and Opportunities in Co-Design for HPC Software

- HPC is powerful but brings one of the most complex software/hardware stack in computer science
- The architecture is going to be more and more heterogeneous due to diverse demands from applications and the limitations of performance, performance per watt, performance per cost...
- Co-design has shown early success and will be the norm in Exascale Computing era and beyond!

Exascale Computing Project Co-design Centers: integrate the rapidly developing software stack with emerging hardware technologies while developing software components that embody the most common application motifs.