Computing System Trends and Performance Portability

Bálint Joó – OLCF, Oak Ridge National Laboratory

Lattice 2021 Virtual Meeting (MIT)

July 30, 2021

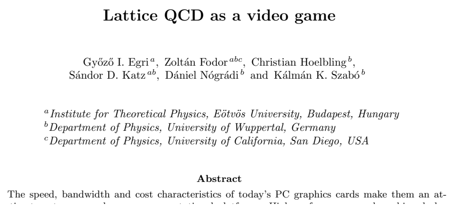
ORNL is managed by UT-Battelle LLC for the US Department of Energy

Contents

- Machine Trends
- Node Programming Models
- Porting Stories

Introduction – a brief history

- I recall at Lattice 2006 in Tuscon, first hearing about the idea of LQCD calculations on GPUs in a talk by Daniel Nogradi.
  - Programming was done using OpenGL.
  - I thought: "This looks fun, but it will never catch on!"
- I had to eat my words as I gave a talk about GPU accelerated computing in Squaw Valley at Lattice 2011 – 10 years ago.
- 2012-2018 was an era of 'friendly competition' between NVIDIA GPUs and Intel Knights
  - The Knights fought well, but were discontinued
  - Remaining fighting Knights are getting close to retirement
- OLCF Summit exceeded 1.88 ExaOps (in 32 and 16-bit precisions) on a Genomics Machine Learning Application in 2018, kicking off the Exascale era



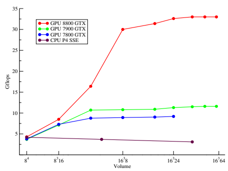
Lattice QCD as a video game

Győző I. Egri[a], Zoltán Fodor[abc], Christian Hoelbling[b], Sándor D. Katz[ab], Dániel Nógrádi[b] and Kálmán K. Szabó[b]

[a] *Institute for Theoretical Physics, Eötvös University, Budapest, Hungary*
[b] *Department of Physics, University of Wuppertal, Germany*
[c] *Department of Physics, University of California, San Diego, USA*

Abstract

The speed, bandwidth and cost characteristics of today's PC graphics cards make them an at-



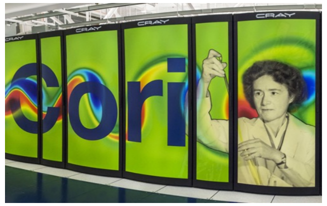
Egri et. al. "Lattice QCD as a video game" Comput. Phys. Commun. 177:631-639,2007 arXiv;hep-lat/0611022

JLab 9g cluster (2009)

JLab 10g cluster (2010)

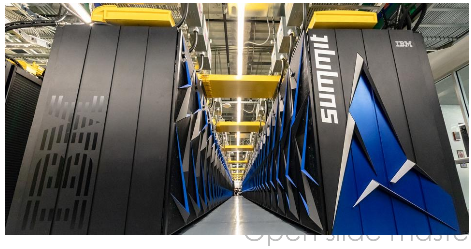
OLCF Titan (2012)

JLab 12k/m cluster (2012)



NERSC Cori KNL (2016)

JLab 16p and 18p KNL Cluster (2016-...)

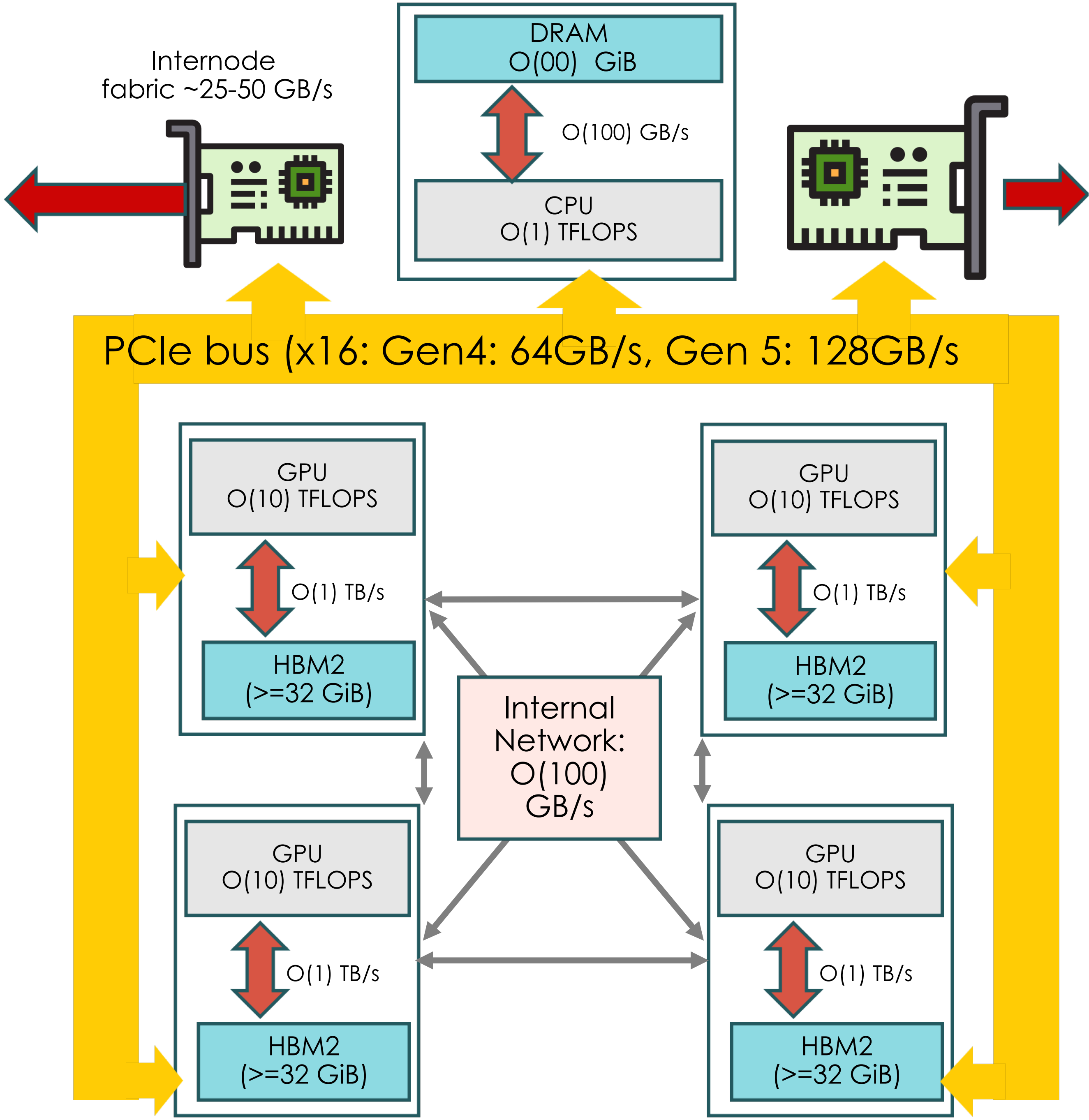


OLCF Summit 2017

Hardware

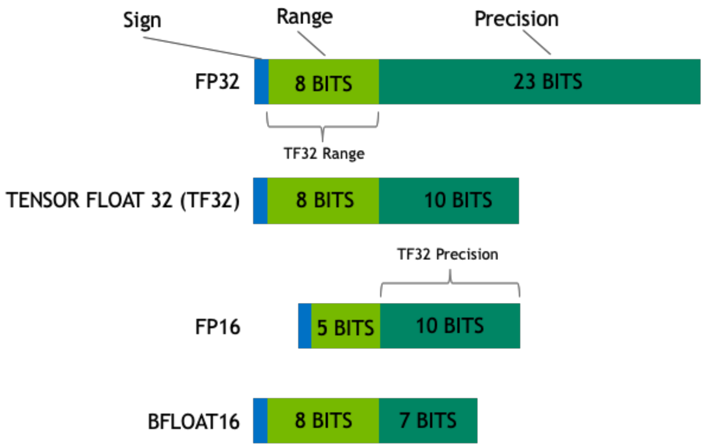
Typical Pre-Exascale Node

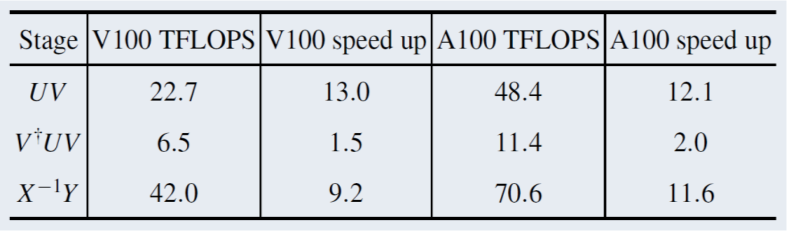
- Schematically not much has changed from Summit
  - Internal network
    - NVLink; 50GB/s per link, A100: up to 12 links=>600 GB/s
    - Infinity Fabric: 92 GB/s link, MI100: up 3 links=> 276 GB/sec
  - External Network
    - Cray/HPE Slingshot (Slingshot **10**: 12.5 + 12.5 = 25 GB/s)
    - Infiniband: (NDR: 50 GB/sec)
  - GPU compute ~ ~10x CPU compute
  - GPU mem-bandwidth ~10x CPU bandwidth
- GPUs have fast low precision matrix-matrix multiply capability
  - NVIDIA Tensor Cores (V100, A100)
  - AMD CDNA (MI100)
- In these accelerated nodes GPUs provide majority of performance
  - I will **blissfully ignore CPU details from here on in**
- Most (not all) LQCD stencils are memory bandwidth bound
  - 1EFLOPS peak => 100 PB/sec peak bandwidth
  - F/B ratio ~0.5-1 for Wilson like fermions in 64 Bit



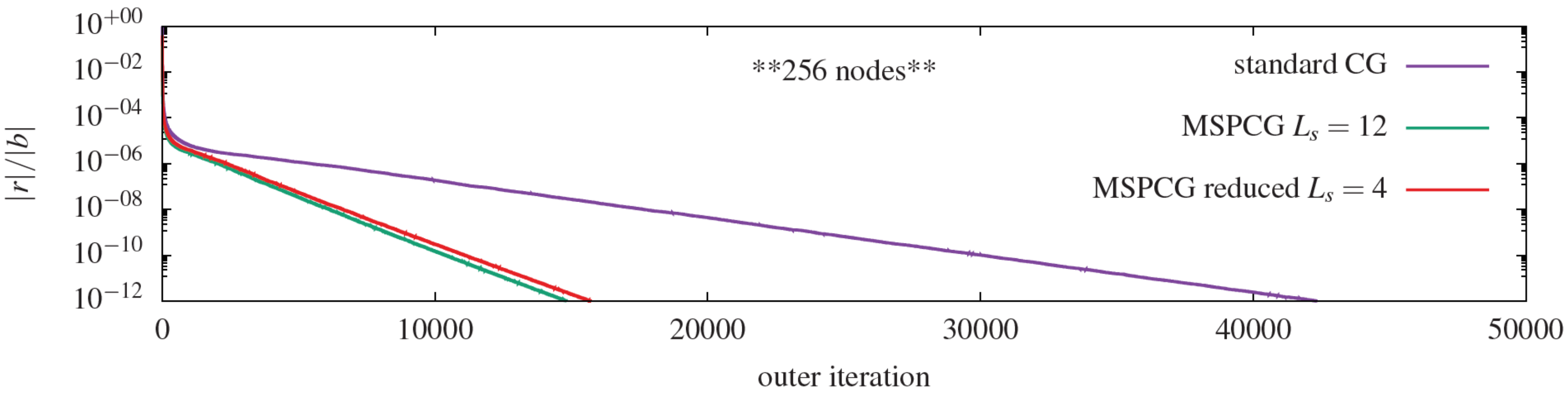
QUDA and tensor core

- Speed up compute intensive kernels in QUDA with tensor cores that accelerate MMA operations on NVIDIA GPUs [poster D9]
  - [WIP] Multi-BLAS
  - m5inv in MSPCG/additive Schwarz: **1.5x** time-to-solution speed up [arXiv:2104.05615, J. Tu's parallel 07/28 14:30]
  - Multi-grid setup: as high as **13x** speed up for `computeUV` on Volta
- [WIP] MMA support for all four precisions: double (fp64), single (fp32), half (16-bit fixed-point), quarter (8-bit fixed-point)
  - (hi + lo) * (hi + lo) = hi * hi + hi * lo + lo * hi + ~~lo * lo~~
  - Use 3xTFLOAT32 for single, and 3xBFLOAT16 for half





Stage	V100 TFLOPS	V100 speed up	A100 TFLOPS	A100 speed up
UV	22.7	13.0	48.4	12.1
$V^{\dagger}UV$	6.5	1.5	11.4	2.0
$X^{-1}Y$	42.0	9.2	70.6	11.6



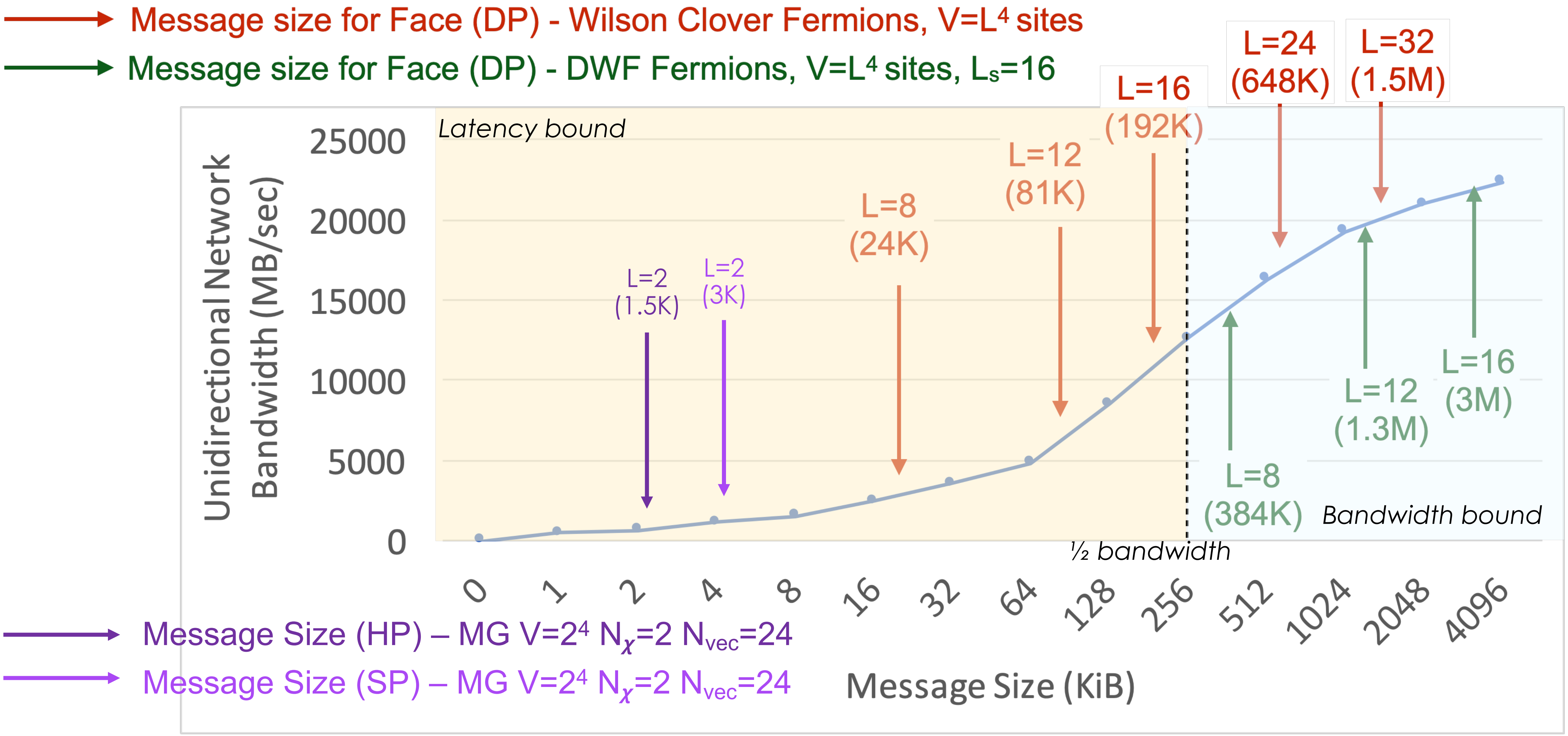
577.6 seconds

458.3 seconds

381.5 seconds

Slide Courtesy of Jiqun Tu, NVIDIA

Latency Or Bandwidth? Both Please...



Message size for Face (DP) - Wilson Clover Fermions, V=L^4 sites

Message size for Face (DP) - DWF Fermions, V=L^4 sites, L$_s$=16

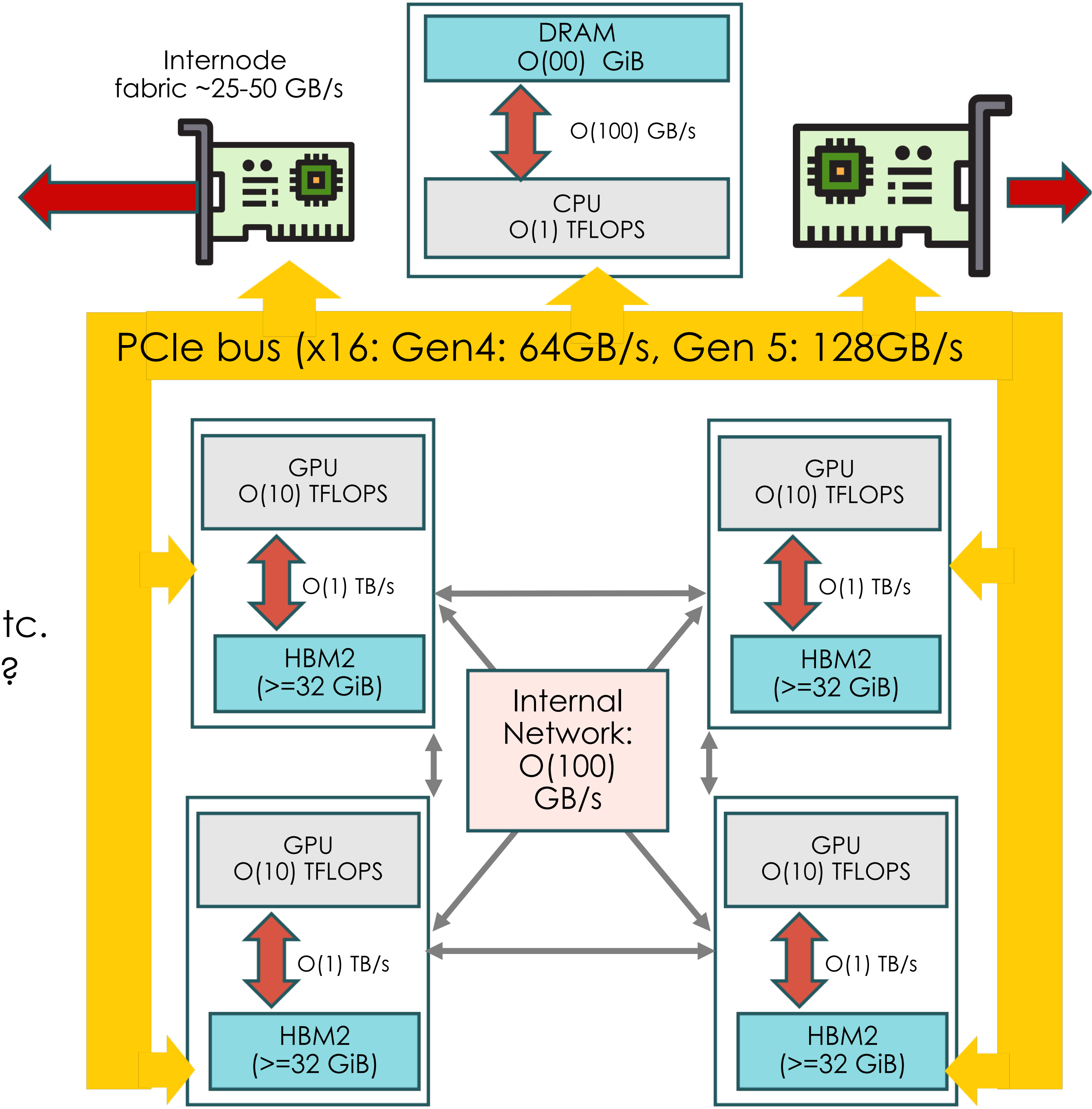
Message Size (HP) – MG V=2^4 N$_\chi$=2 N$_{vec}$=24

Message Size (SP) – MG V=2^4 N$_\chi$=2 N$_{vec}$=24

Data for Summit from SummitWorkshop-SummitNodePerformane-WJ talk by W. Joubert

And so, nuances matter

- Hardware capability
  - Direct path from GPUs to NIC?
  - Can GPUs initiate communications?
  - Number of hops in internode networks
  - Amount of GPU memory & H2D2H transfers
  - Unified/Managed Memory Capability?
- Programming model & implementation
  - Kernel launch latency
  - GPU aware MPI, Interfaces for P2P, SHMEM etc.
  - Persistent kernels, synchronizing with atomics?
  - Access to Unified/Managed Memories
- Algorithmic Pushes
  - MG & Reduced precision -> latency
  - Multi-RHS/Split Grid algorithms -> B/W
  - DD algorithms -> device locality



Latencies affect strong scaling

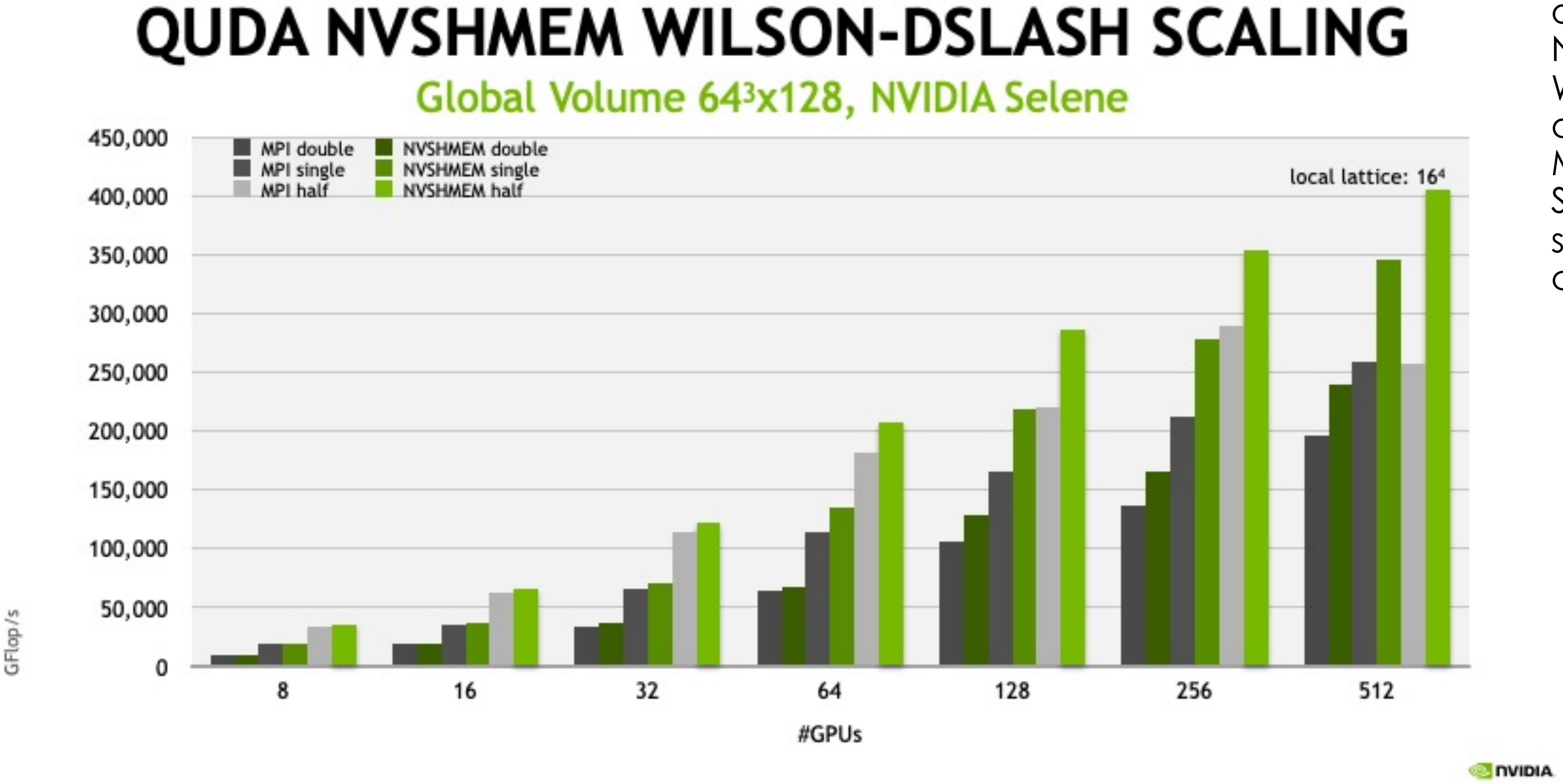
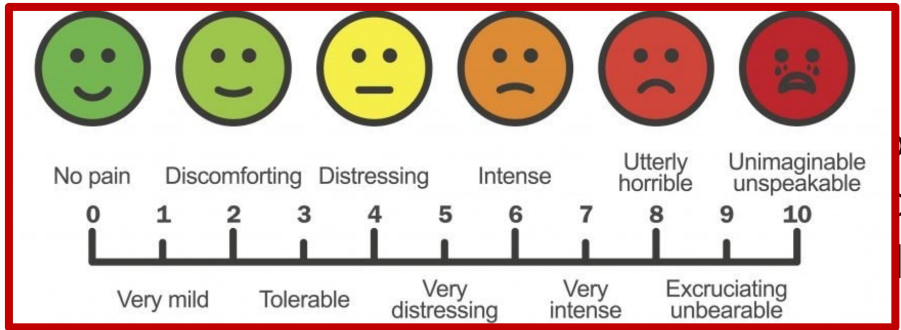


Figure Courtesy of M. Wagner, NVIDIA (Talk on Wednesday afternoon in Machines and Software session in this conference)

And so, Nuances matter

- Hardware capability



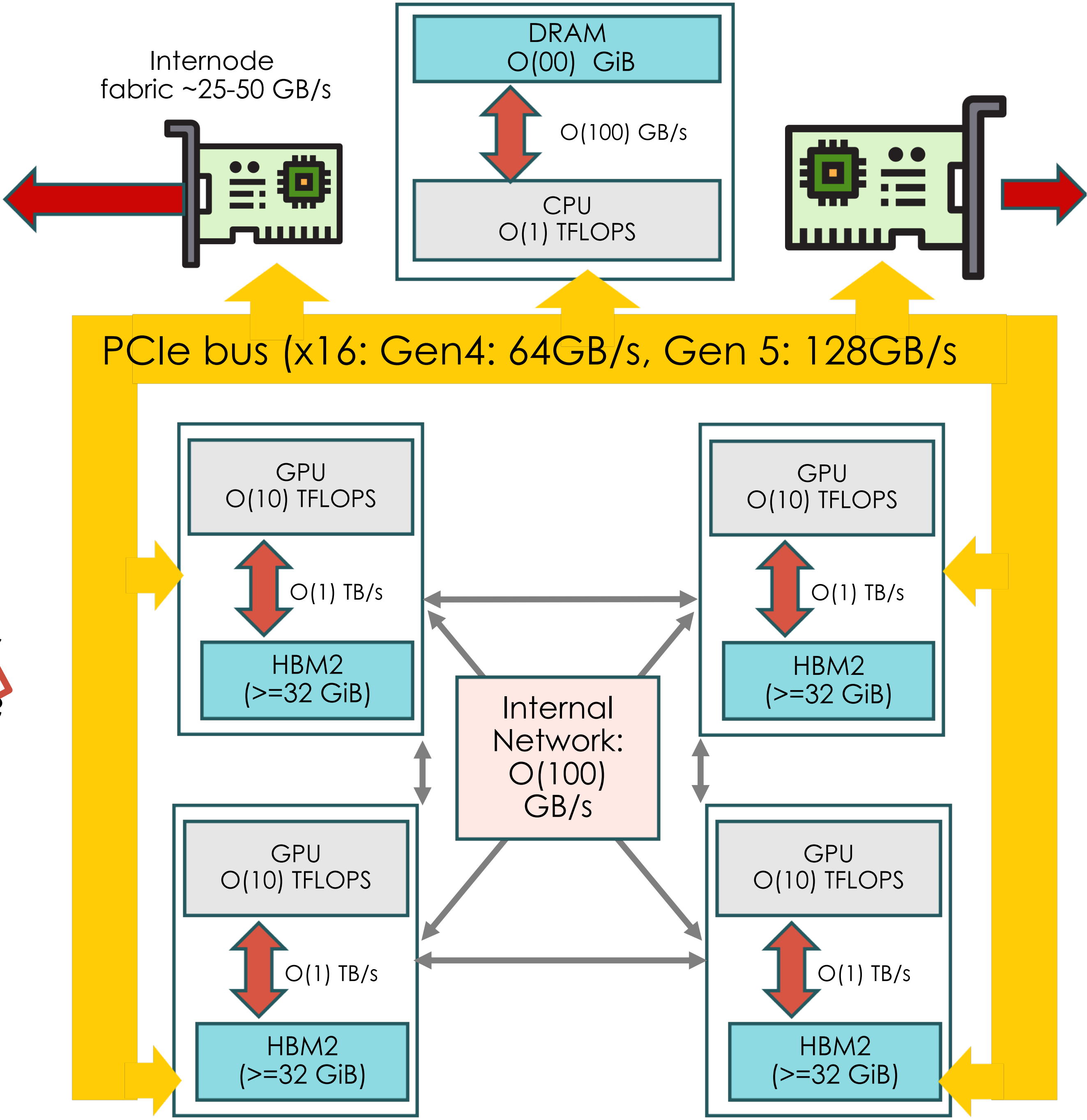
rks

ansfers)

ity?

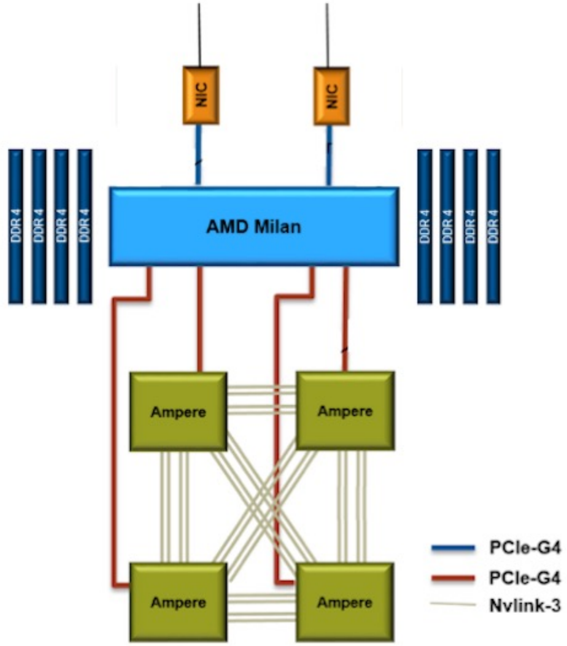
- Programming model & implementat n
  - Kernel launch latency
  - User access to Hardware c P, SHMEM etc)
  - Persistent kernels, s s?
  - Access to Unif s
- Algorithm
  - M -> latency
  - lgorithms -> B/W
  - -> device locality

YOUR LEVEL OF PAIN MAY VARY !!!



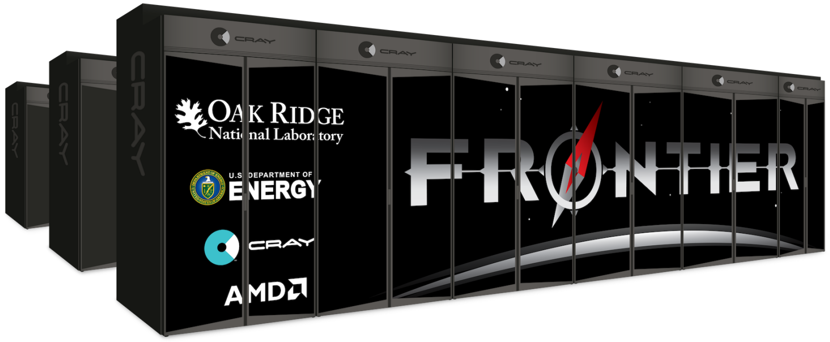
US Exascale and Pre-Exascale Systems: NERSC Perlmutter

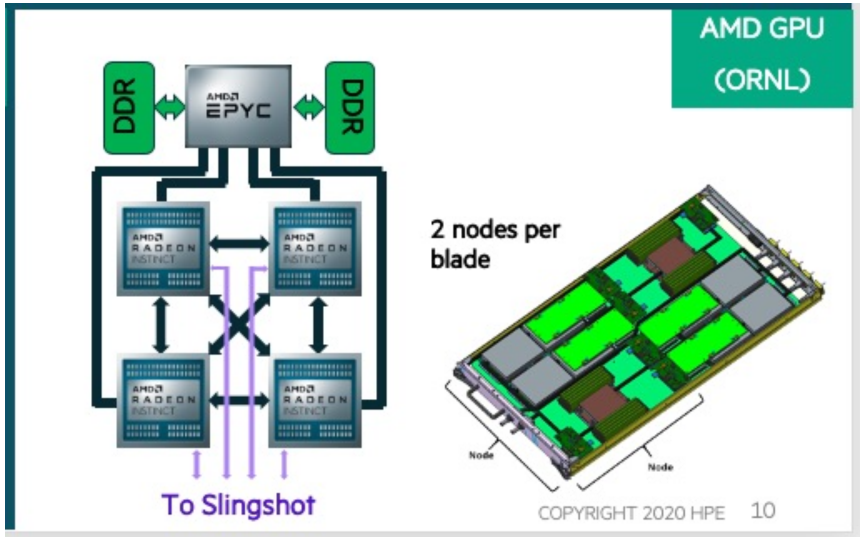
- Perlmutter is the New HPE/Cray “Shasta” system recently arrived at NERSC
  - #5 on Top 500 List: 89.8 PF (Rpeak), 64.5 PF (Rmax)
- Accelerators are NVIDIA A100 (Ampere) GPUs
  - Memory B/w: 1555.2 GB/sec
  - FLOPs:
    - 19.5 TF (FP32)   9.7 TF (FP64)
    - Tensor ops: 311.9 (FP16), 155.9 (TF32), 19.5 (FP64)
- NVLink 3
  - 4 GPUs on node have all-to-all connection
    - 100 GB/s/direction between a pair of GPUs
    - 600 GB/sec total into/out-of a GPU
- HPE/Cray Slingshot-10 Interconnect
- System is expected to open to users soon
- Programming Models
  - MPI + CUDA, OpenMP-5 offload, NVSHMEM
- https://www.nersc.gov/systems/perlmutter/



US Exascale and Pre-Exascale Systems: OLCF Frontier

- Frontier will be the New HPE/Cray “Shasta” system at Oak Ridge Leadership Computing Facility
  - >1.5 Exaflops peak performance
- Accelerators will be AMD Radeon Instinct GPUs
- AMD Infinity Fabric
  - 4 GPUs on node to have all-to-all connection
- HPE/Cray Slingshot
  - “Multiple Slingshot NICs providing 100GB/sec network bandwidth”
  - GPUs will have direct connection to Slingshot
- Programming Models
  - MPI, AMD HIP, OpenMP-5-offload, ...
- https://www.olcf.ornl.gov/frontier

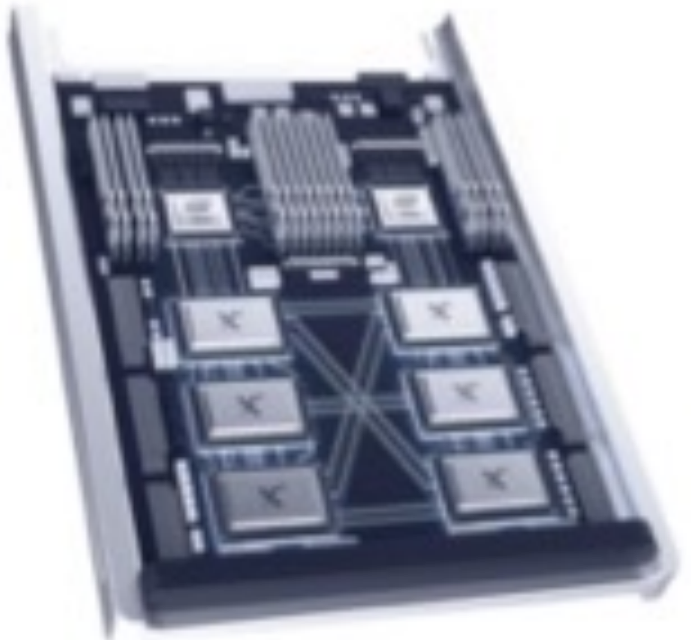




US Exascale and Pre-Exascale Systems: ALCF Aurora

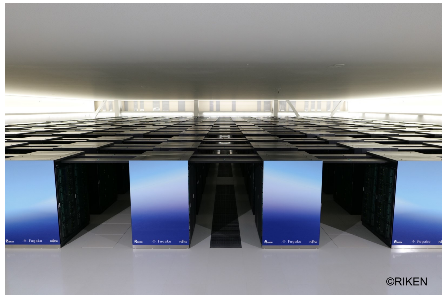
- Aurora will be the New HPE/Cray “Shasta” system at Argonne Leadership Computing Facility
  - >1 Exaflop peak (DP) performance
- Accelerators will be Intel Xe architecture based ”Ponte Vecchio” GPUs
- Unified Memory Architecture
  - Across GPU and CPU
- Low Latency, High Bandwidth all-to-all connectivity within Node
- HPE/Cray Slingshot interconnect
  - 8 fabric endpoints per node
- Programming Models
  - MPI, Intel oneAPI DPC++ (based on SYCL), OpenMP-5-offload, ...
- https://alcf.anl.gov/aurora

Argonne NATIONAL LABORATORY
U.S. DEPARTMENT OF ENERGY
intel
Hewlett Packard Enterprise
Aurora

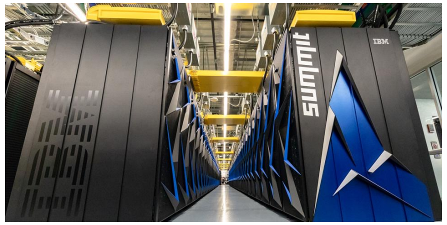


Other Noteworthy Systems for LQCD

- Supercomputer Fugaku, RIKEN CCS Japan
  - #1 Top 500 list: 537PF (Rpeak), 442 PF (Rmax)
  - Please see talk by Yoshifumi Nakamura!
- Summit at OLCF, U.S.A
  - #2 Top 500 list: 200PF (Rpeak), 148.6 PF (RMax)
- Sunway TaihuLight, National Supercomputing Center, Wuxi, China
  - #4 on on Top 500 list: 125.4 PF(Rpeak) 93PF(Rmax)
- Tianhe-2A, National Super Computer Center, Guangzhou, China
  - #7 on the Top 500 list; 100.6 PF (Rpeak) 61PF (Rmax)
- JUWELS Booster, Forschungszentrum Jülich, Germany
  - #8 on Top500 List: 71PF (Rpeak), 44 PF (Rmax)
  - 4 NVIDIA A100 GPUs, switched directly to HCAs
  - Infiniband Network in DragonFly+ configuration.

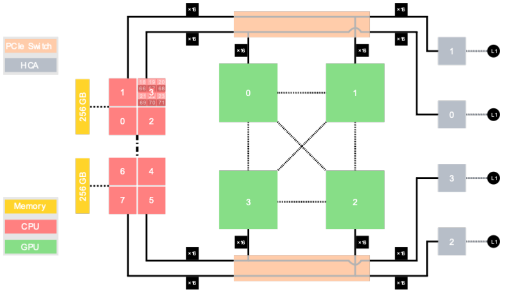


Supercomputer Fugaku



OLCF Summit

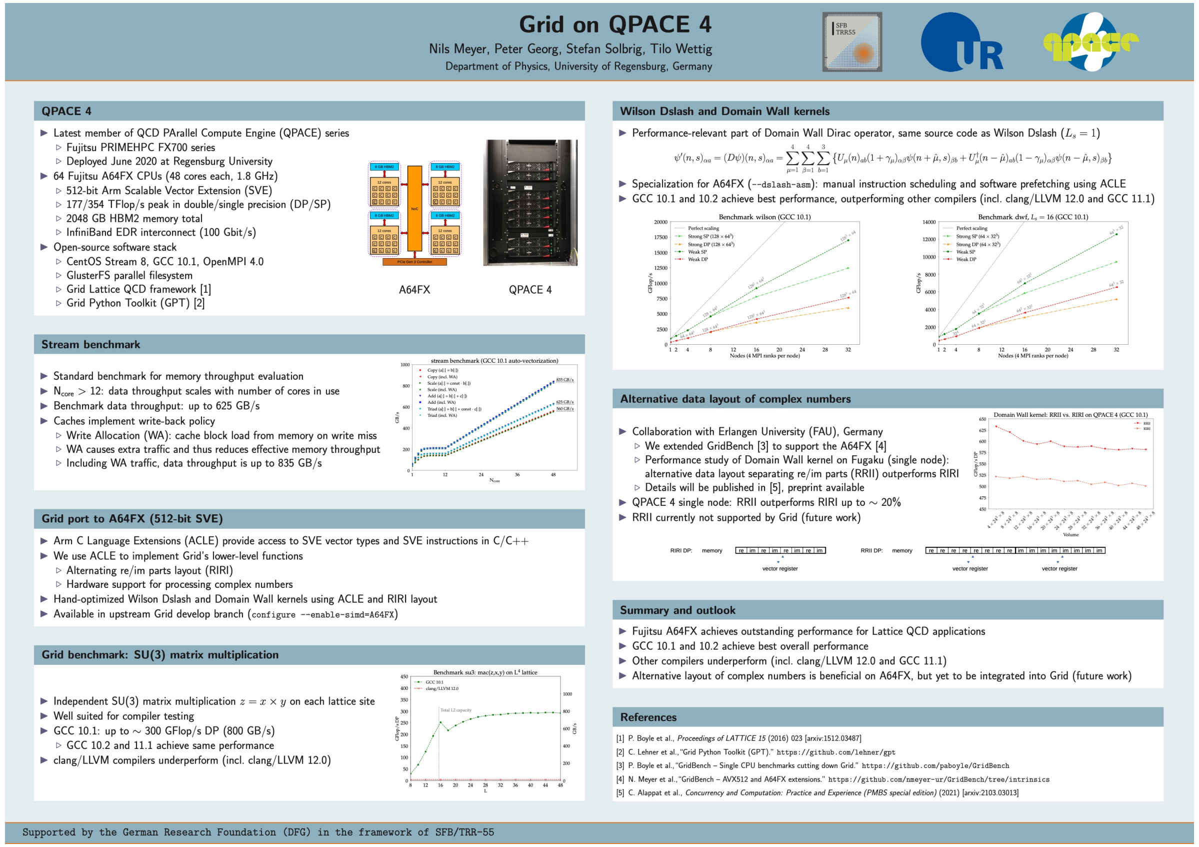
TaihuLight



JUWELS Booster Node Architecture

Custom QCD Systems?

- QPACE 4: 177/354 TFlop/s peak
  - Please see poster by the Regensburg group: N. Meyer, P.Georg, S. Solbrig & T. Wettig
- A64FX CPUs - Fugaku like
  - 48 cores/CPU 1.8 GHz
  - 512-bit Arm SVE SIMD: 2.76 TF/cpu
  - 32 GB HBM2 memory per CPU
- InfiniBand EDR interconnect
- Open-source software stack
  - no assistant cores like Fugaku
  - lots of kernel tuning to reduce O/S noise.



Grid on QPACE 4

Nils Meyer, Peter Georg, Stefan Solbrig, Tilo Wettig
Department of Physics, University of Regensburg, Germany

QPACE 4

- Latest member of QCD PArallel Compute Engine (QPACE) series
  - ▷ Fujitsu PRIMEHPC FX700 series
  - ▷ Deployed June 2020 at Regensburg University
- 64 Fujitsu A64FX CPUs (48 cores each, 1.8 GHz)
  - ▷ 512-bit Arm Scalable Vector Extension (SVE)
  - ▷ 177/354 TFlop/s peak in double/single precision (DP/SP)
  - ▷ 2048 GB HBM2 memory total
  - ▷ InfiniBand EDR interconnect (100 Gbit/s)
- Open-source software stack
  - ▷ CentOS Stream 8, GCC 10.1, OpenMPI 4.0
  - ▷ GlusterFS parallel filesystem
  - ▷ Grid Lattice QCD framework [1]
  - ▷ Grid Python Toolkit (GPT) [2]

A64FX

QPACE 4

Stream benchmark

- Standard benchmark for memory throughput evaluation
- $N_{core} > 12$: data throughput scales with number of cores in use
- Benchmark data throughput: up to 625 GB/s
- Caches implement write-back policy
  - ▷ Write Allocation (WA): cache block load from memory on write miss
  - ▷ WA causes extra traffic and thus reduces effective memory throughput
  - ▷ Including WA traffic, data throughput is up to 835 GB/s

Grid port to A64FX (512-bit SVE)

- Arm C Language Extensions (ACLE) provide access to SVE vector types and SVE instructions in C/C++
- We use ACLE to implement Grid's lower-level functions
  - ▷ Alternating re/im parts layout (RIRI)
  - ▷ Hardware support for processing complex numbers
- Hand-optimized Wilson Dslash and Domain Wall kernels using ACLE and RIRI layout
- Available in upstream Grid develop branch (`configure --enable-simd=A64FX`)

Grid benchmark: SU(3) matrix multiplication

- Independent SU(3) matrix multiplication $z = x \times y$ on each lattice site
- Well suited for compiler testing
- GCC 10.1: up to $\sim$ 300 GFlop/s DP (800 GB/s)
  - ▷ GCC 10.2 and 11.1 achieve same performance
- clang/LLVM compilers underperform (incl. clang/LLVM 12.0)

Wilson Dslash and Domain Wall kernels

- Performance-relevant part of Domain Wall Dirac operator, same source code as Wilson Dslash ($L_s = 1$)

$$\psi'(n,s)_{\alpha a} = (D\psi)(n,s)_{\alpha a} = \sum_{\mu=1}^{4}\sum_{\beta=1}^{4}\sum_{b=1}^{3}\{U_\mu(n)_{ab}(1+\gamma_\mu)_{\alpha\beta}\psi(n+\hat{\mu},s)_{\beta b} + U_\mu^\dagger(n-\hat{\mu})_{ab}(1-\gamma_\mu)_{\alpha\beta}\psi(n-\hat{\mu},s)_{\beta b}\}$$

- Specialization for A64FX (`--dslash-asm`): manual instruction scheduling and software prefetching using ACLE
- GCC 10.1 and 10.2 achieve best performance, outperforming other compilers (incl. clang/LLVM 12.0 and GCC 11.1)

Alternative data layout of complex numbers

- Collaboration with Erlangen University (FAU), Germany
  - ▷ We extended GridBench [3] to support the A64FX [4]
  - ▷ Performance study of Domain Wall kernel on Fugaku (single node): alternative data layout separating re/im parts (RRII) outperforms RIRI
  - ▷ Details will be published in [5], preprint available
- QPACE 4 single node: RRII outperforms RIRI up to $\sim$ 20%
- RRII currently not supported by Grid (future work)

Domain Wall kernel: RRII vs. RIRI on QPACE 4 (GCC 10.1)
RRII
RIRI
GFlop/s DP
Volume
RIRI DP: memory re im re im re im re im
vector register
RRII DP: memory re re re re re re re re im im im im im im im im
vector register
vector register

Summary and outlook

- Fujitsu A64FX achieves outstanding performance for Lattice QCD applications
- GCC 10.1 and 10.2 achieve best overall performance
- Other compilers underperform (incl. clang/LLVM 12.0 and GCC 11.1)
- Alternative layout of complex numbers is beneficial on A64FX, but yet to be integrated into Grid (future work)

References

[1] P. Boyle et al., *Proceedings of LATTICE 15* (2016) 023 [arxiv:1512.03487]

[2] C. Lehner et al.,"Grid Python Toolkit (GPT)." `https://github.com/lehner/gpt`

[3] P. Boyle et al.,"GridBench – Single CPU benchmarks cutting down Grid." `https://github.com/paboyle/GridBench`

[4] N. Meyer et al.,"GridBench – AVX512 and A64FX extensions." `https://github.com/nmeyer-ur/GridBench/tree/intrinsics`

[5] C. Alappat et al., *Concurrency and Computation: Practice and Experience (PMBS special edition)* (2021) [arxiv:2103.03013]

Supported by the German Research Foundation (DFG) in the framework of SFB/TRR-55

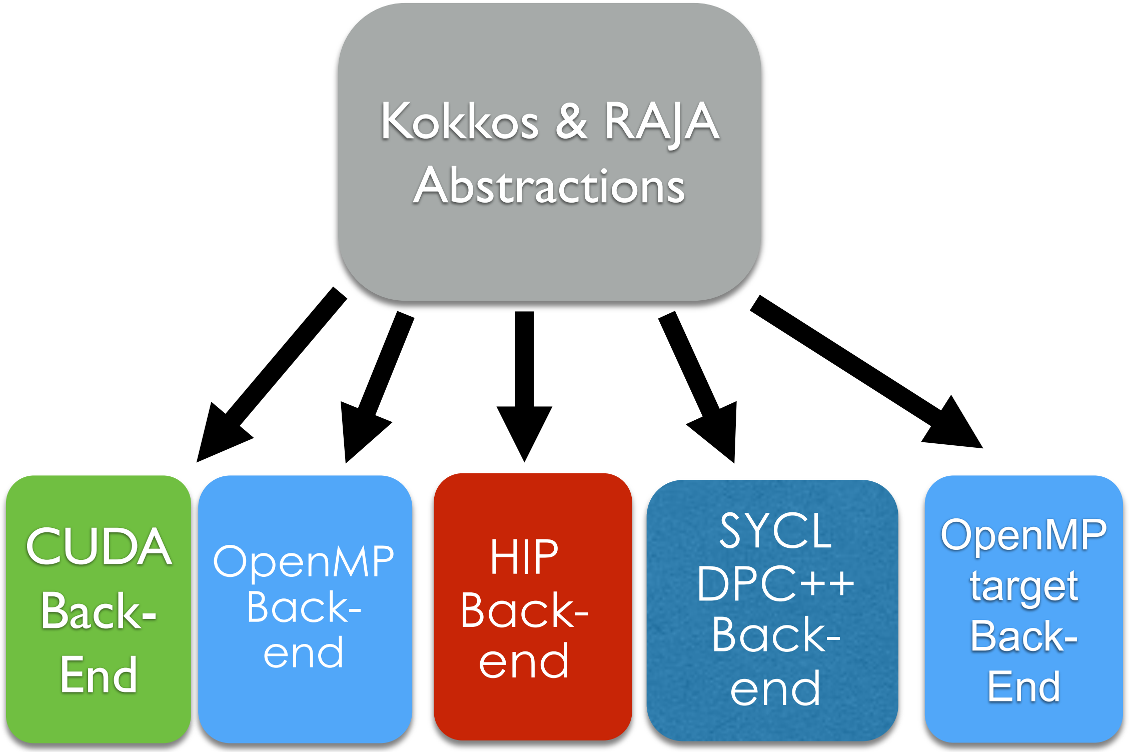
Software

Node Programming Models: Pick your standard

- Vendor options
  - NVIDIA CUDA: the CUDA you know, evolved..
    - Support for Tensor cores in cuBLAS, cuDNN and CUDA (wmma:: namespace)
    - ecosystem support: cuBLAS, cuFFT, cuRAND etc.
    - C++ Standard Library (cuda::std::)
    - Strong push to implement **ISO C++ standard** features in full.
  - **AMD HIP**
    - very similar to NVIDIA's CUDA core library (replace 'cuda' with 'hip')
    - **open source implementations for ROCm, also for CPUs** (HIP CPU)'
    - LLVM/Clang based compiler
    - ecosystem support: hipBLAS, hipFFT/rocFFT, hipRAND/rocRAND, etc
    - access to CDNA features through compiler
  - Intel oneAPI
    - based on **SYCL Standard from the Khronos Group, with OpenCL heritage**
    - Compiler in oneAPI SDK and in Intel's contributions to LLVM
    - ecosystem support: oneMKL, oneDNN, ...
    - oneAPI Tutorial at this conference by P. Steinbrecher
- Directive based approaches: **OpenMP-offload and OpenACC**
  - OpenMP: `#pragma omp teams distribute parallel for simd`
  - all vendors committed to supporting OpenMP-5.0 offload
  - OpenACC supported in NVIDIA HPC-SDK and GCC (work by Mentor Graphics)
  - OpenMP is likely the best option for Fortran (although nvfortran can offload DO CONCURRENT)
- **OpenCL** is available through all vendors as well
  - but beware vendor specificity in the implementations...

Performance Portability via Kokkos and RAJA

- Kokkos and RAJA provide portability to C++ programs via ‘back ends’
  - Kokkos now has a Fortran interface project
- Both prog. models provide basic parallel patterns
  - forall, reduction, scan
- Kokkos provides ‘View’ datatype
  - multi-dimensional array, a precursor to std::mdspan
  - RAJA uses other packages (e.g. Umpire) for memory management
- Views and execution patterns are bound to memory and execution spaces via policies.
  - Policies have default values, which can be set in some architecture specific header.

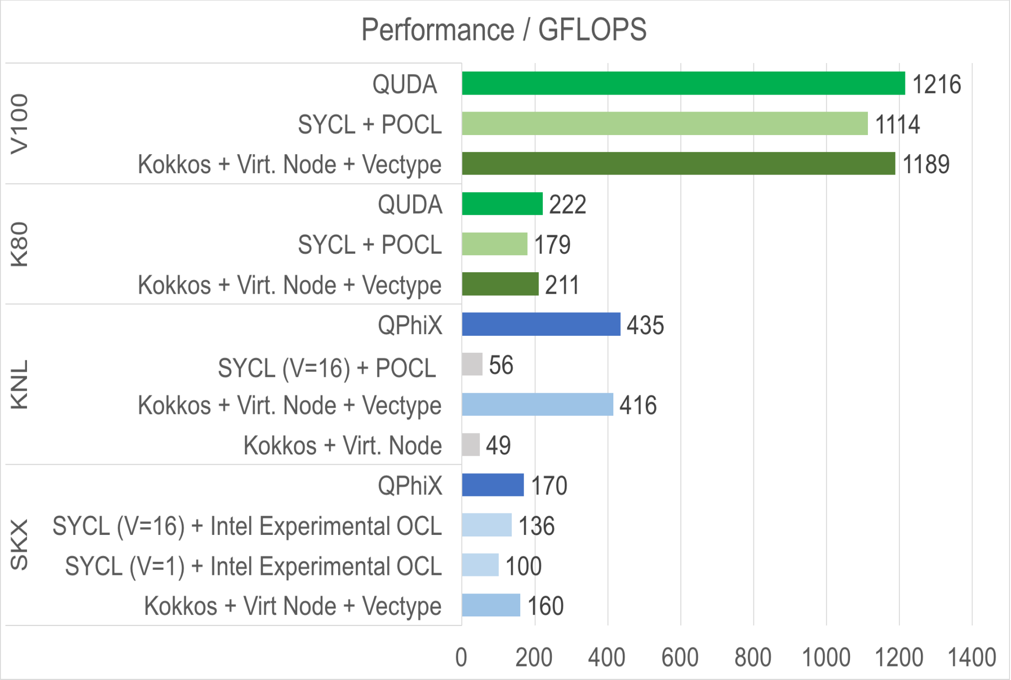


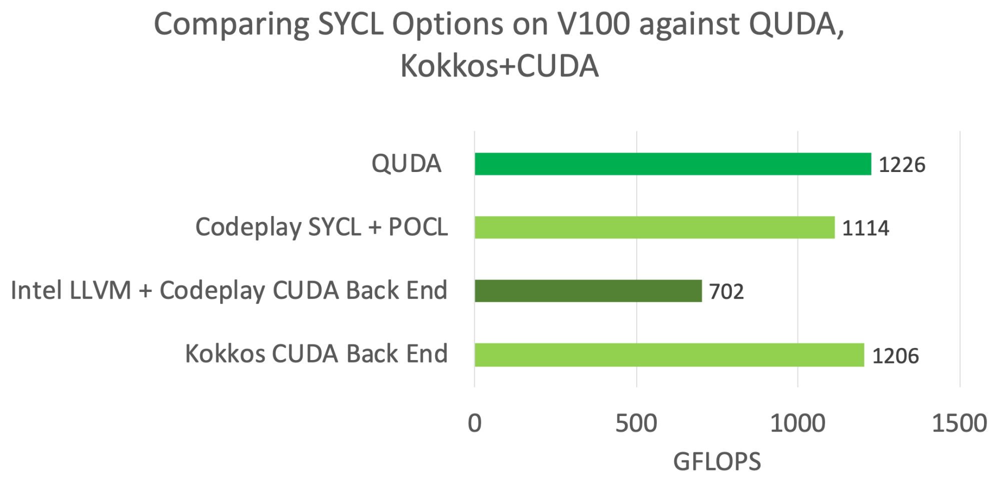
```
Kokkos::View<float[1000]> y(“y”), x(“x”);
// Fill x and y somehow – not shown

float alpha=0.5;
Kokkos::parallel_for(1000,
                     KOKKOS_LAMBDA(int i) {
         y(i) += alpha * x(i);
});
```

Portability of the programming Models

- Directive based approaches: should target all main systems
  - require strong compiler support, and vendors to agree on interpretation of the standard
- Intel Data Parallel C++: main target: Intel GPUs/CPUs/FPGAs etc.
  - Aurora Proxies: Linux laptops with Intel Graphics (Gen 9 or later), Intel Dev Cloud, JLSE Early access systems
  - NVIDIA GPUs: Intel LLVM compiler has CUDA back end from Codeplay
  - AMD GPUS: hipSYCL Project, a pilot AMD Back End for Intel LLVM is being developed by Codeplay
- HIP: main target AMD GPUs
  - Frontier Proxies: NVIDIA GPUs systems (Summit), Early Access Systems (Spock)
  - NVIDIA GPUs: Native support up to a certain CUDA level (substitutions: 'hip→'cuda')
  - Intel GPUs: ECP HIP-LZ (HIP on Level Zero) project
  - nascent HIP-CPU project to support CPUs (as a development aid)
- CUDA: main target NVIDIA GPUs
  - Perlmutter Proxies: Summit, JUWELS-Booster, Desktop GPU systems,
  - AMD: HIP includes 'hipify' conversion tool for 'developer assisted' conversion to HIP.
  - Intel GPUs: Intel provides 'developer assisted' conversion to Data Parallel C++
- Kokkos & RAJA: pick most performant back end (most likely the native one)





figures from B. Joo P3HPC Forum 2020 virtual meeting. Courtesty of B. Joo and collaborators.

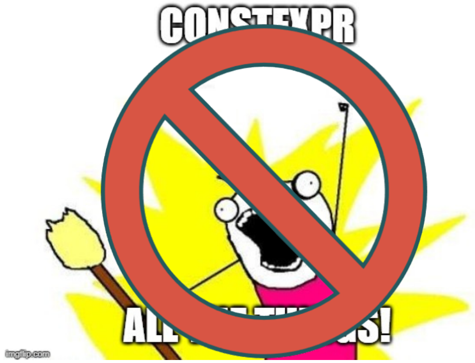
Code porting/development efforts: QUDA

- QUDA is a software library for QCD components
  - Advanced Optimized solvers (e.g. Aggregation based MG preconditioner)
  - Key optimized routines: GaugeForce (MILC), Gauge Fixing,
  - Multiple precision use since Day-1
  - Built in Autotuning
    - adjusting grid, block and shared memory sizes,
    - choosing communications policy e.g. P2P, GDR, MPI
- See ”Towards QUDA 2.0” talk by K. Clark (Wed, 1pm) for refactoring details
  - plus several other QUDA related talks at this conference (M. Wagner, E. Weinberg, J. Tu, ...)
- QUDA grew a ‘performance portability’ subgroup of developers:
  - K. Clark (NVIDIA) QUDA Lead developer, C++, std::par, pSTL, NVC++
  - B. Joo (OLCF) HIP Backend
  - D. Howarth (LLNL) HIP Backend
  - J. Osborn (ALCF) DPC++ Backend
  - X-Y. Jin (ALCF) OpenMP-offload Backend
  - A. Strelchenko (FNAL) – Intel CUDA-Conversion tool exploration
  - D. McDougall, C. Robeck (AMD) – HIP consultation
  - P. Steinbrecher (Intel) – Intel compiler consultation (DPC++ and OpenMP)
  - weekly meetings + notes of varying level of depth in our Slack #portability channel

Refactoring for Perf Portability (led by K. Clark)

- Launch Abstractions
  - Kernel1D, Kernel2D, Kernel3D, Reduction, TransformReduce,
  - most portability frameworks do something like this: Kokkos, Raja, and also Grid
- API Elements
  - memory allocation/copy (quda_malloc() ....)
  - Streams (fixed number and use index as 'stream')
  - Shared memory: sharedMemory manager class
  - Constant memory – for large list of arguments (e.g. small matrices)
  - API functions error check internally (architecture dependent status codes are isolated/contained)
- Device constraints & Macro Reduction/Localization
  - constexpr functions
  - SFINAE, and template-template classes.
- Libraries
  - CUB/hipCUB, => namespace QudaCub aliases
  - cu/hipRand, cu/hipFFT => wrapped into 'shim headers'
  - certain math functions (sincos and rsqrt and half prec implementations) implemented on host/device
  - straightforward for HIP, since many libraries corresponding to CUDA exist, may be tricker for e.g. Intel
- Miscellany
  - vector types (e.g. float4, int4) – HIP returned proxy objects / accessors to some structure elements
  - IPC for Peer2Peer seems easy to port between CUDA and HIP

CppCon17 talk by Ben Deane and Jason Turner

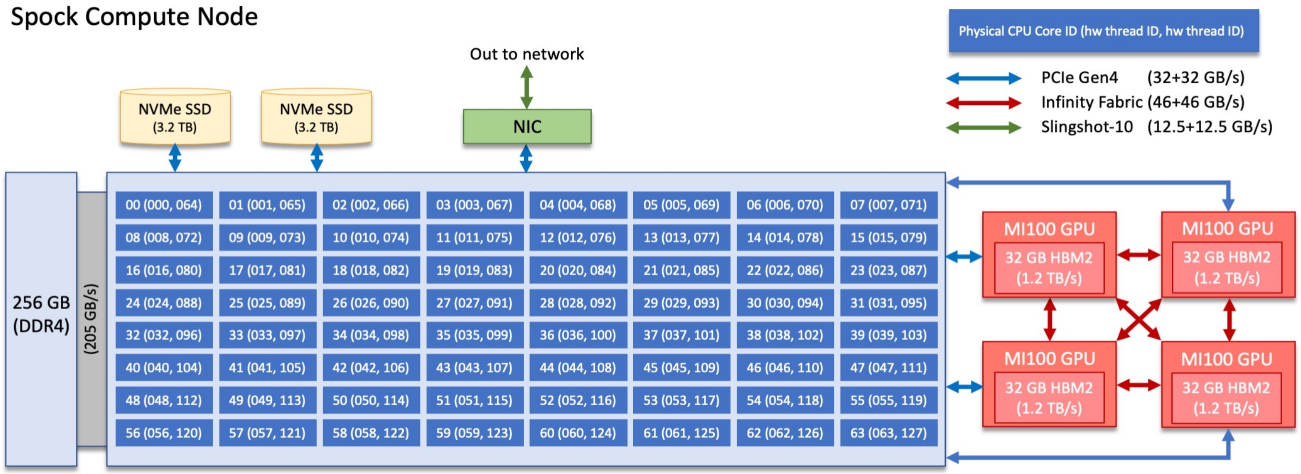


Also "Don't constexpr all the things" – David Sankel, CppNow 2021

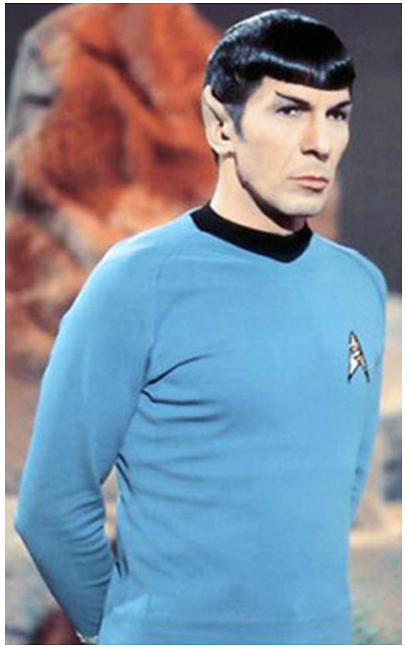
(actually this talk is about Circle)

Spock: OLCF’s Frontier Early Access System (EAS)

- 3 Cabinets w. 12 nodes/cabinet => 36 nodes, 144 GPUs total
- Each node has
  - 1x 64 core AMD EPYC 7662 “Rome” CPU (4 NUMA domains) + 256 GB DDR4 memory (205 GB/s)
  - 4x AMD Radeon Instinct MI100 GPUs (gfx908), 32 GB HBM2 @ (1.2 TB/sec)
    - GPU-GPU: All-to-all Infinity Fabric Interconnect, Host-GPU: PCie Gen4: 32+32 GB/sec
  - 2 x NVMe SSDs (3.2 TB each): 6800 MB/sec read, 4200 MB.sec write
- Slingshot 10 Interconnect: 12.5 + 12.5 GB/sec
- Spock Training Workshop materials: https://www.olcf.ornl.gov/spock-training
- Documentation: https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html


Spock Compute Node
Out to network
NVMe SSD (3.2 TB)
NVMe SSD (3.2 TB)
NIC
256 GB (DDR4)
(205 GB/s)
Physical CPU Core ID (hw thread ID, hw thread ID)
PCIe Gen4 (32+32 GB/s)
Infinity Fabric (46+46 GB/s)
Slingshot-10 (12.5+12.5 GB/s)
MI100 GPU 32 GB HBM2 (1.2 TB/s)
MI100 GPU 32 GB HBM2 (1.2 TB/s)
MI100 GPU 32 GB HBM2 (1.2 TB/s)
MI100 GPU 32 GB HBM2 (1.2 TB/s)

00 (000, 064)	01 (001, 065)	02 (002, 066)	03 (003, 067)	04 (004, 068)	05 (005, 069)	06 (006, 070)	07 (007, 071)
08 (008, 072)	09 (009, 073)	10 (010, 074)	11 (011, 075)	12 (012, 076)	13 (013, 077)	14 (014, 078)	15 (015, 079)
16 (016, 080)	17 (017, 081)	18 (018, 082)	19 (019, 083)	20 (020, 084)	21 (021, 085)	22 (022, 086)	23 (023, 087)
24 (024, 088)	25 (025, 089)	26 (026, 090)	27 (027, 091)	28 (028, 092)	29 (029, 093)	30 (030, 094)	31 (031, 095)
32 (032, 096)	33 (033, 097)	34 (034, 098)	35 (035, 099)	36 (036, 100)	37 (037, 101)	38 (038, 102)	39 (039, 103)
40 (040, 104)	41 (041, 105)	42 (042, 106)	43 (043, 107)	44 (044, 108)	45 (045, 109)	46 (046, 110)	47 (047, 111)
48 (048, 112)	49 (049, 113)	50 (050, 114)	51 (051, 115)	52 (052, 116)	53 (053, 117)	54 (054, 118)	55 (055, 119)
56 (056, 120)	57 (057, 121)	58 (058, 122)	59 (059, 123)	60 (060, 124)	61 (061, 125)	62 (062, 126)	63 (063, 127)

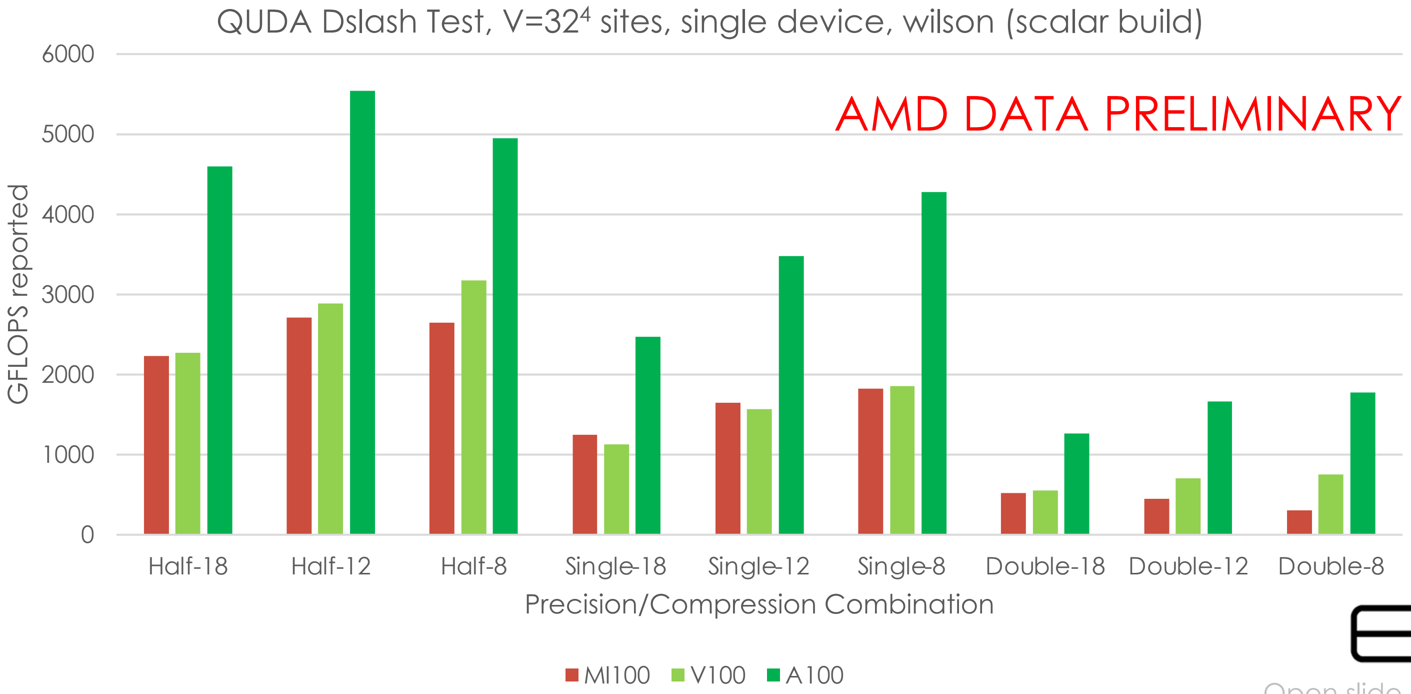


Original Spock Compute Node

QUDA Porting Status

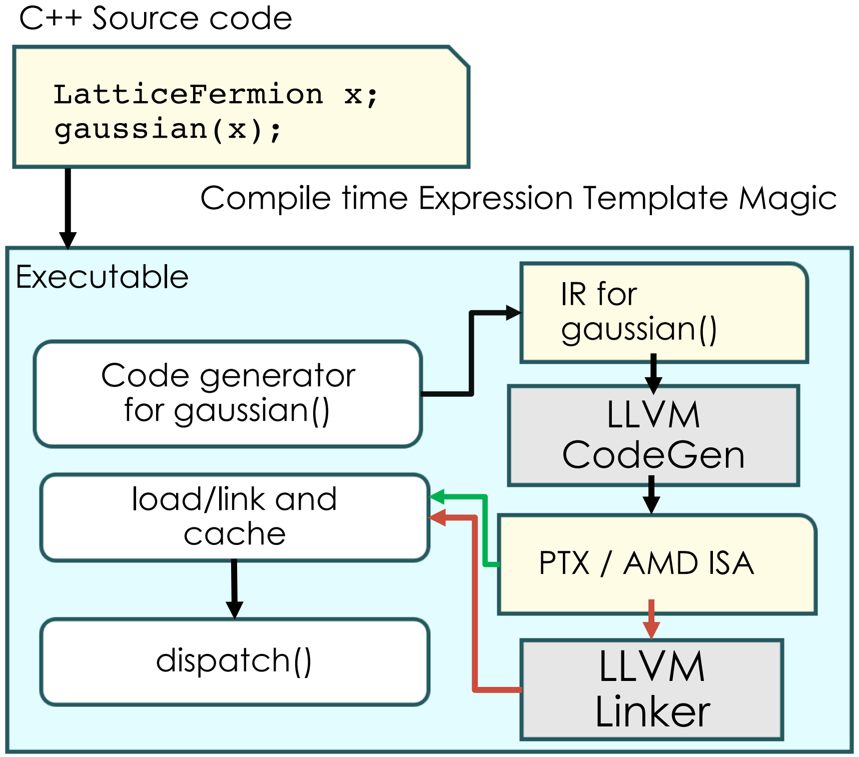
- (All of) the library builds with ROCm 4.2
  - feature/hip-compile-fixes branch
  - P2P & MPI Comms built. Currently undergoing testing and debugging
- All ‘Ctest’ pass on single AMD GPU
- Intel GPU porting in progress
  - James Osborn is still developing and testing the SYCL port and Xioyong-Jin is working on OpenMP-offload
  - focus is on making sure the refactorings which we discussed allow efficient porting to both programming models (access to shared memories, reductions, etc)

- *A100 data courtesy of E. Weinberg, NVIDIA. Data from* ***A100 80 GB*** *part*
  - ***~2.0TB/s*** *peak mem. B/W*
  - *Perlmutter has 1.56 TB/sec, 40GB parts*
- *V100 data from Summit*
  - ***~900*** *GB/sec peak mem B/W*
- *MI100 data from Spock*
  - ***~1.2TB****/sec peak mem B/W*



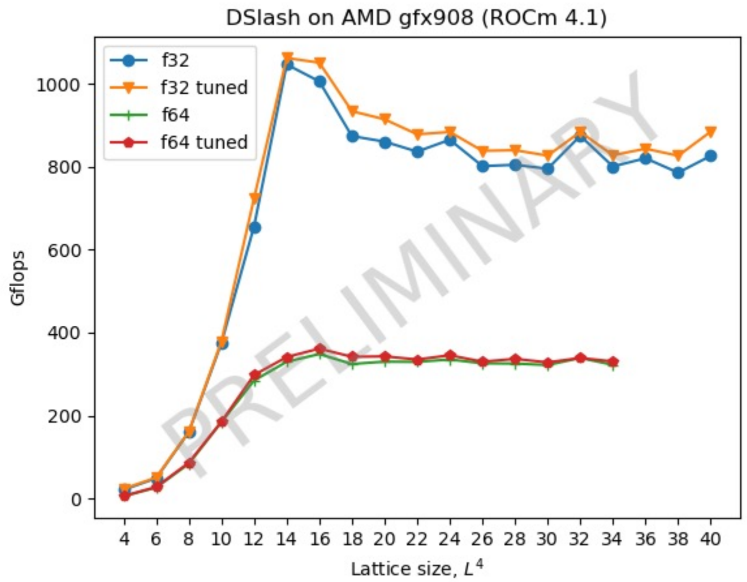
Progress with QDP-JIT

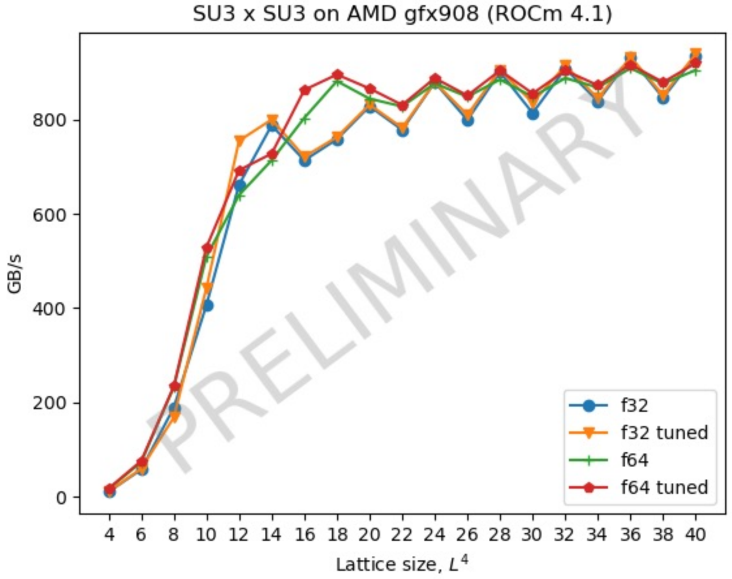
- Frank Winter ported QDP-JIT to AMD GPUs: Presented this conference Wed 7/28
- QDP-JIT can run a full Chroma HMC trajectory on AMD GPU
  - 800+ GFLOPS from Native QDP-JIT Dslash implementation, 800+ GB/sec on SU(3)xSU(3) kernels
  - MI100 has ~1200 GB/sec maximum HBM bandwidth
- AMDGPU backend for LLVM generates ISA rather than an intermediate form like PTX
  - Needs additional linking step
- Intel GPUs next…


C++ Source code

```
LatticeFermion x;
gaussian(x);
```

Compile time Expression Template Magic


DSlash on AMD gfx908 (ROCm 4.1)

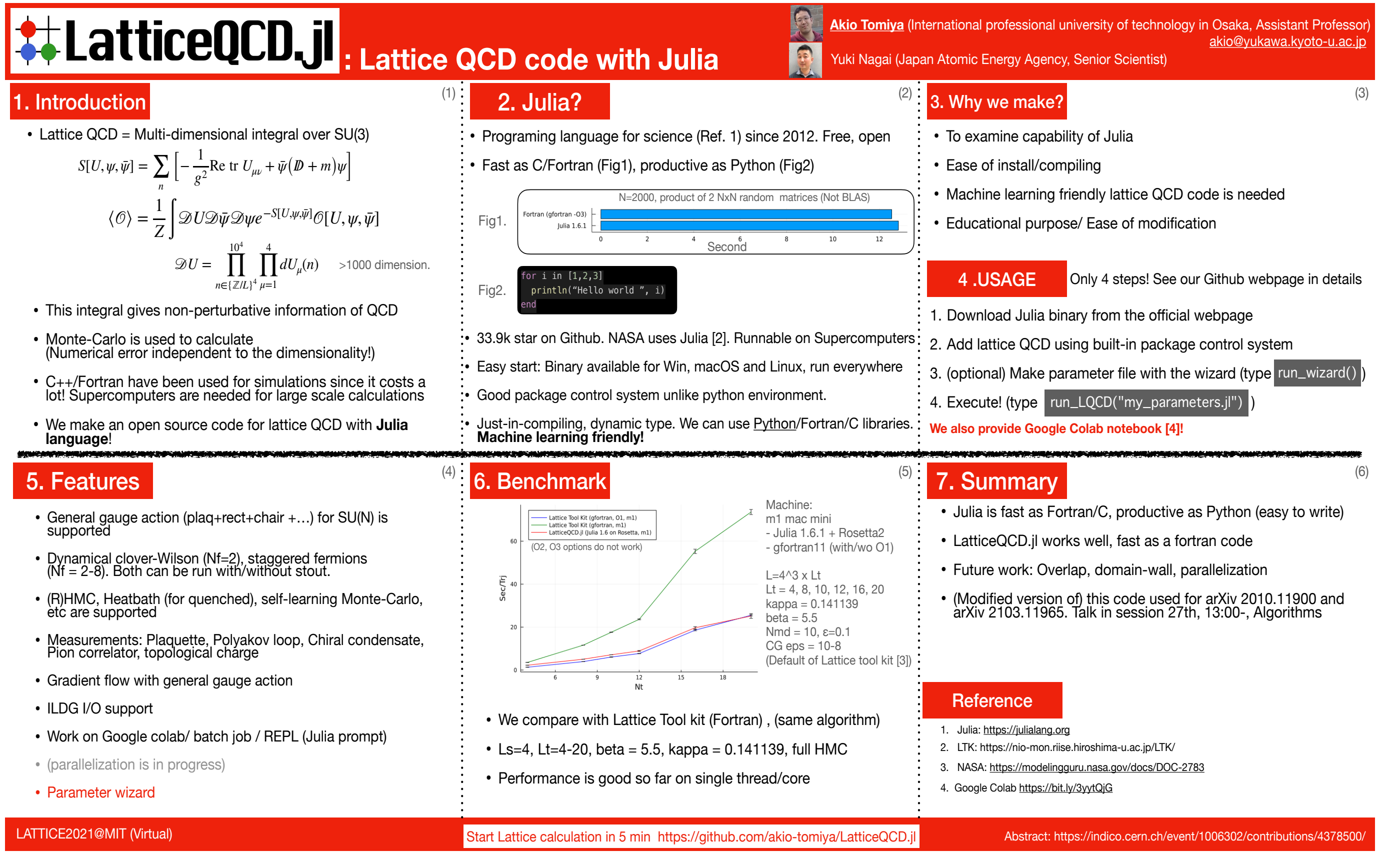

SU3 x SU3 on AMD gfx908 (ROCm 4.1)

Preliminary MI100 results from the ECP All Hands Meeting poster by F. Winter. Courtesy of Frank Winter.

This work utilized the early-access system Tulip hosted by HPE and supported by the HPE and AMD staff of the Frontier Center of Excellence.

Julia For LQCD

- Interesting work from Akio Tomiya -- poster at this conference.
- Julia uses JIT compilation via LLVM.
- “Very high level” with good C/C++/Python interoperability – ML friendly
- Support for Wilson-Clover & Staggered Fermions, Stout Smearing, RHMC, Heat-Bath, Self Learning Monte Carlo,
- Similar performance to Fortran code
- Related:
  - work by A. Strelchenko (2019 USQCD All Hands Meeting) for algorithimic epxloration


N=2000, product of 2 NxN random matrices (Not BLAS)
Fortran (gfortran -O3)
Julia 1.6.1
0
2
4
6
8
10
12
Second
Lattice Tool Kit (gfortran, O1, m1)
Lattice Tool Kit (gfortran, m1)
LatticeQCD.jl (Julia 1.6 on Rosetta, m1)
(O2, O3 options do not work)
Sec/Trj
0
20
40
60
6
9
12
15
18
Nt

LatticeQCD.jl : Lattice QCD code with Julia

Akio Tomiya (International professional university of technology in Osaka, Assistant Professor) akio@yukawa.kyoto-u.ac.jp

Yuki Nagai (Japan Atomic Energy Agency, Senior Scientist)

1. Introduction

(1)

- Lattice QCD = Multi-dimensional integral over SU(3)

$$S[U,\psi,\bar{\psi}] = \sum_n \left[-\frac{1}{g^2} \mathrm{Re\ tr}\ U_{\mu\nu} + \bar{\psi}(\not{D} + m)\psi \right]$$

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}U \mathcal{D}\bar{\psi} \mathcal{D}\psi e^{-S[U,\psi,\bar{\psi}]} \mathcal{O}[U,\psi,\bar{\psi}]$$

$$\mathcal{D}U = \prod_{n\in\{\mathbb{Z}/L\}^4}^{10^4} \prod_{\mu=1}^{4} dU_\mu(n)$$ >1000 dimension.

- This integral gives non-perturbative information of QCD
- Monte-Carlo is used to calculate (Numerical error independent to the dimensionality!)
- C++/Fortran have been used for simulations since it costs a lot! Supercomputers are needed for large scale calculations
- We make an open source code for lattice QCD with **Julia language**!

2. Julia?

(2)

- Programing language for science (Ref. 1) since 2012. Free, open
- Fast as C/Fortran (Fig1), productive as Python (Fig2)

Fig1.

Fig2.

```
for i in [1,2,3]
  println(“Hello world ”, i)
end
```

- 33.9k star on Github. NASA uses Julia [2]. Runnable on Supercomputers
- Easy start: Binary available for Win, macOS and Linux, run everywhere
- Good package control system unlike python environment.
- Just-in-compiling, dynamic type. We can use Python/Fortran/C libraries. **Machine learning friendly!**

3. Why we make?

(3)

- To examine capability of Julia
- Ease of install/compiling
- Machine learning friendly lattice QCD code is needed
- Educational purpose/ Ease of modification

4 .USAGE

Only 4 steps! See our Github webpage in details

1. Download Julia binary from the official webpage
2. Add lattice QCD using built-in package control system
3. (optional) Make parameter file with the wizard (type `run_wizard()`)
4. Execute! (type `run_LQCD("my_parameters.jl")`)

We also provide Google Colab notebook [4]!

5. Features

(4)

- General gauge action (plaq+rect+chair +…) for SU(N) is supported
- Dynamical clover-Wilson (Nf=2), staggered fermions (Nf = 2-8). Both can be run with/without stout.
- (R)HMC, Heatbath (for quenched), self-learning Monte-Carlo, etc are supported
- Measurements: Plaquette, Polyakov loop, Chiral condensate, Pion correlator, topological charge
- Gradient flow with general gauge action
- ILDG I/O support
- Work on Google colab/ batch job / REPL (Julia prompt)
- (parallelization is in progress)
- Parameter wizard

6. Benchmark

(5)

Machine:
m1 mac mini
- Julia 1.6.1 + Rosetta2
- gfortran11 (with/wo O1)

L=4^3 x Lt
Lt = 4, 8, 10, 12, 16, 20
kappa = 0.141139
beta = 5.5
Nmd = 10, ε=0.1
CG eps = 10-8
(Default of Lattice tool kit [3])

- We compare with Lattice Tool kit (Fortran) , (same algorithm)
- Ls=4, Lt=4-20, beta = 5.5, kappa = 0.141139, full HMC
- Performance is good so far on single thread/core

7. Summary

(6)

- Julia is fast as Fortran/C, productive as Python (easy to write)
- LatticeQCD.jl works well, fast as a fortran code
- Future work: Overlap, domain-wall, parallelization
- (Modified version of) this code used for arXiv 2010.11900 and arXiv 2103.11965. Talk in session 27th, 13:00-, Algorithms

Reference

1. Julia: https://julialang.org
2. LTK: https://nio-mon.riise.hiroshima-u.ac.jp/LTK/
3. NASA: https://modelingguru.nasa.gov/docs/DOC-2783
4. Google Colab https://bit.ly/3yytQjG

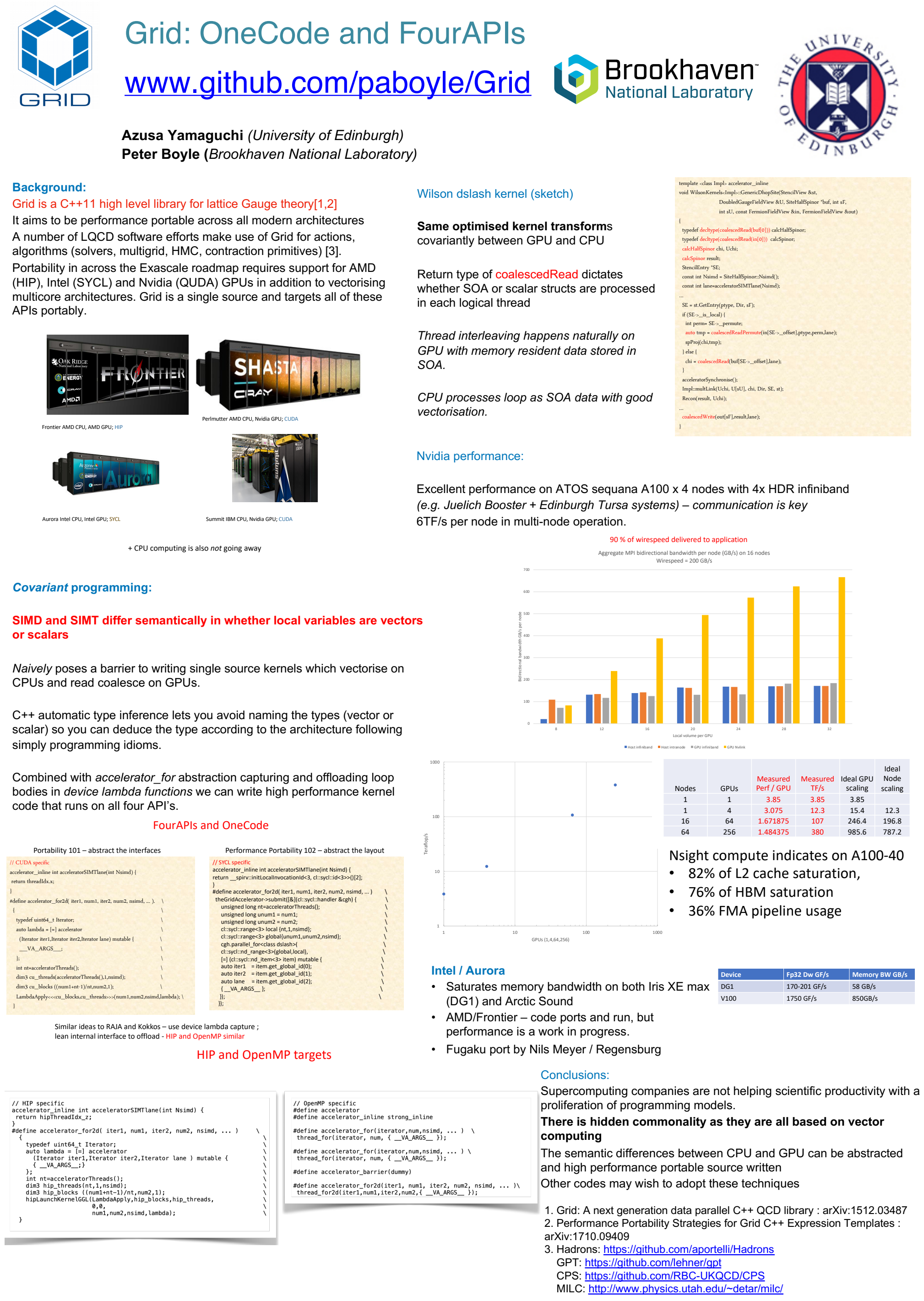
LATTICE2021@MIT (Virtual)

Start Lattice calculation in 5 min https://github.com/akio-tomiya/LatticeQCD.jl

Abstract: https://indico.cern.ch/event/1006302/contributions/4378500/

Grid

- Grid is an LQCD framework headed up by P. Boyle and team
  - several Grid related talks/posters at this conference
  - Poster by P. Boyle, A. Yamaguchi, GPT Python bindings tutorial by C. Lehner, etc....
- C++ expression template based LQCD software system
- Portability to several “back-ends”
  - OpenMP
  - HIP
  - CUDA
  - SYCL
- coalescedRead()/Write() – abstraction
- accelerator_for() – abstraction (macro?)
  - wraps actual kernel launch via HIP/SYCL/CUDA
- for more details please contact the Authors
- excellent performance, including near wire-speed saturation of networks reported



Grid: OneCode and FourAPIs

www.github.com/paboyle/Grid

Azusa Yamaguchi *(University of Edinburgh)*
Peter Boyle (*Brookhaven National Laboratory)*

Background:

Grid is a C++11 high level library for lattice Gauge theory[1,2]

It aims to be performance portable across all modern architectures

A number of LQCD software efforts make use of Grid for actions, algorithms (solvers, multigrid, HMC, contraction primitives) [3].

Portability in across the Exascale roadmap requires support for AMD (HIP), Intel (SYCL) and Nvidia (QUDA) GPUs in addition to vectorising multicore architectures. Grid is a single source and targets all of these APIs portably.

Frontier AMD CPU, AMD GPU; HIP

Perlmutter AMD CPU, Nvidia GPU; CUDA

Aurora Intel CPU, Intel GPU; SYCL

Summit IBM CPU, Nvidia GPU; CUDA

+ CPU computing is also *not* going away

***Covariant* programming:**

SIMD and SIMT differ semantically in whether local variables are vectors or scalars

Naively poses a barrier to writing single source kernels which vectorise on CPUs and read coalesce on GPUs.

C++ automatic type inference lets you avoid naming the types (vector or scalar) so you can deduce the type according to the architecture following simply programming idioms.

Combined with *accelerator_for* abstraction capturing and offloading loop bodies in *device lambda functions* we can write high performance kernel code that runs on all four API’s.

FourAPIs and OneCode

Portability 101 – abstract the interfaces

```
// CUDA specific
accelerator_inline int acceleratorSIMTlane(int Nsimd) {
 return threadIdx.x;
}
#define accelerator_for2d( iter1, num1, iter2, num2, nsimd, ... ).   \
 {                                                                  \
   typedef uint64_t Iterator;                                       \
   auto lambda = [=] accelerator                                    \
    (Iterator iter1,Iterator iter2,Iterator lane) mutable {         \
     __VA_ARGS__;                                                   \
   };                                                               \
   int nt=acceleratorThreads();                                     \
   dim3 cu_threads(acceleratorThreads(),1,nsimd);                   \
   dim3 cu_blocks ((num1+nt-1)/nt,num2,1);                          \
   LambdaApply<<<cu_blocks,cu_threads>>>(num1,num2,nsimd,lambda); \
 }
```

Performance Portability 102 – abstract the layout

```
// SYCL specific
accelerator_inline int acceleratorSIMTlane(int Nsimd) {
return __spirv::initLocalInvocationId<3, cl::sycl::id<3>>()[2];
}
#define accelerator_for2d( iter1, num1, iter2, num2, nsimd, ... )     \
 theGridAccelerator->submit([&](cl::sycl::handler &cgh) {              \
   unsigned long nt=acceleratorThreads();                              \
   unsigned long unum1 = num1;                                         \
   unsigned long unum2 = num2;                                         \
   cl::sycl::range<3> local {nt,1,nsimd};                              \
   cl::sycl::range<3> global{unum1,unum2,nsimd};                       \
   cgh.parallel_for<class dslash>(                                     \
   cl::sycl::nd_range<3>(global,local),                                \
   [=] (cl::sycl::nd_item<3> item) mutable {                           \
   auto iter1   = item.get_global_id(0);                               \
   auto iter2   = item.get_global_id(1);                               \
   auto lane    = item.get_global_id(2);                               \
   { __VA_ARGS__ };                                                    \
  });                                                                  \
 });
```

Similar ideas to RAJA and Kokkos – use device lambda capture ;
lean internal interface to offload - HIP and OpenMP similar

HIP and OpenMP targets

```
// HIP specific
accelerator_inline int acceleratorSIMTlane(int Nsimd) {
 return hipThreadIdx_z;
}
#define accelerator_for2d( iter1, num1, iter2, num2, nsimd, ... )     \
  {                                                                    \
    typedef uint64_t Iterator;                                         \
    auto lambda = [=] accelerator                                      \
      (Iterator iter1,Iterator iter2,Iterator lane ) mutable {         \
      { __VA_ARGS__;}                                                  \
    };                                                                 \
    int nt=acceleratorThreads();                                       \
    dim3 hip_threads(nt,1,nsimd);                                      \
    dim3 hip_blocks ((num1+nt-1)/nt,num2,1);                           \
    hipLaunchKernelGGL(LambdaApply,hip_blocks,hip_threads,             \
                       0,0,                                            \
                       num1,num2,nsimd,lambda);                        \
  }
```

```
// OpenMP specific
#define accelerator
#define accelerator_inline strong_inline

#define accelerator_for(iterator,num,nsimd, ... )  \
 thread_for(iterator, num, { __VA_ARGS__ });

#define accelerator_for(iterator,num,nsimd, ... ) \
 thread_for(iterator, num, { __VA_ARGS__ });

#define accelerator_barrier(dummy)

#define accelerator_for2d(iter1, num1, iter2, num2, nsimd, ... )\
 thread_for2d(iter1,num1,iter2,num2,{ __VA_ARGS__ });
```

Wilson dslash kernel (sketch)

Same optimised kernel transforms covariantly between GPU and CPU

Return type of coalescedRead dictates whether SOA or scalar structs are processed in each logical thread

Thread interleaving happens naturally on GPU with memory resident data stored in SOA.

CPU processes loop as SOA data with good vectorisation.

```
template <class Impl> accelerator_inline
void WilsonKernels<Impl>::GenericDhopSite(StencilView &st,
             DoubledGaugeFieldView &U, SiteHalfSpinor *buf, int sF,
             int sU, const FermionFieldView &in, FermionFieldView &out)
{
 typedef decltype(coalescedRead(buf[0])) calcHalfSpinor;
 typedef decltype(coalescedRead(in[0]))  calcSpinor;
 calcHalfSpinor chi, Uchi;
 calcSpinor result;
 StencilEntry *SE;
 const int Nsimd = SiteHalfSpinor::Nsimd();
 const int lane=acceleratorSIMTlane(Nsimd);
…
 SE = st.GetEntry(ptype, Dir, sF);
 if (SE->_is_local) {
  int perm= SE->_permute;
  auto tmp = coalescedReadPermute(in[SE->_offset],ptype,perm,lane);
  spProj(chi,tmp);
 } else {
  chi = coalescedRead(buf[SE->_offset],lane);
 }
 acceleratorSynchronise();
 Impl::multLink(Uchi, U[sU], chi, Dir, SE, st);
 Recon(result, Uchi);
…
 coalescedWrite(out[sF],result,lane);
}
```

Nvidia performance:

Excellent performance on ATOS sequana A100 x 4 nodes with 4x HDR infiniband *(e.g. Juelich Booster + Edinburgh Tursa systems) – communication is key*
6TF/s per node in multi-node operation.

90 % of wirespeed delivered to application

Nodes	GPUs	Measured Perf / GPU	Measured TF/s	Ideal GPU scaling	Ideal Node scaling
1	1	3.85	3.85	3.85	
1	4	3.075	12.3	15.4	12.3
16	64	1.671875	107	246.4	196.8
64	256	1.484375	380	985.6	787.2

Nsight compute indicates on A100-40

- 82% of L2 cache saturation,
- 76% of HBM saturation
- 36% FMA pipeline usage

Intel / Aurora

- Saturates memory bandwidth on both Iris XE max (DG1) and Arctic Sound
- AMD/Frontier – code ports and run, but performance is a work in progress.
- Fugaku port by Nils Meyer / Regensburg

Device	Fp32 Dw GF/s	Memory BW GB/s
DG1	170-201 GF/s	58 GB/s
V100	1750 GF/s	850GB/s

Conclusions:

Supercomputing companies are not helping scientific productivity with a proliferation of programming models.

There is hidden commonality as they are all based on vector computing

The semantic differences between CPU and GPU can be abstracted and high performance portable source written

Other codes may wish to adopt these techniques

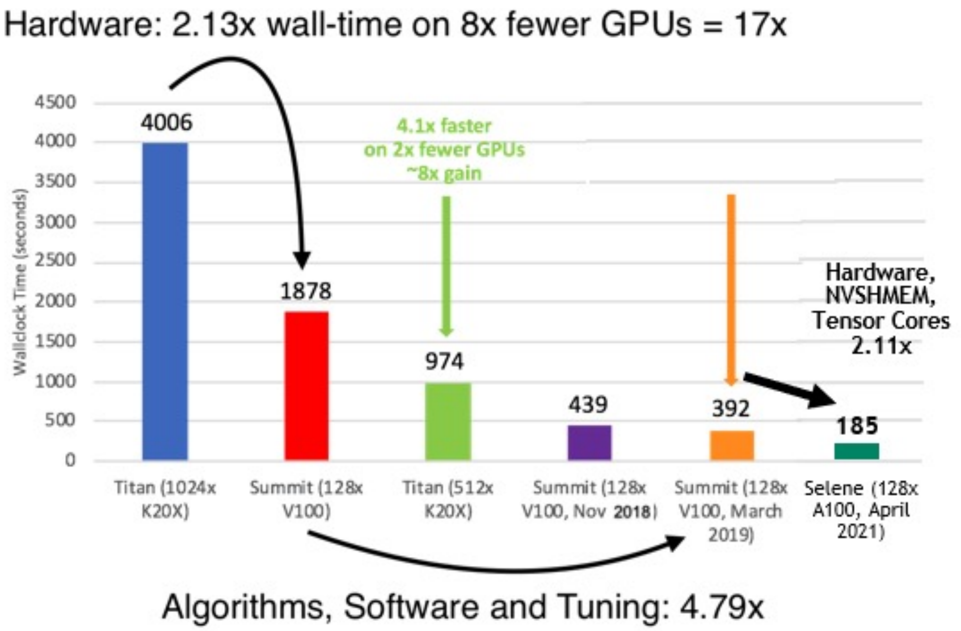
1. Grid: A next generation data parallel C++ QCD library : arXiv:1512.03487
2. Performance Portability Strategies for Grid C++ Expression Templates : arXiv:1710.09409
3. Hadrons: https://github.com/aportelli/Hadrons
   GPT: https://github.com/lehner/gpt
   CPS: https://github.com/RBC-UKQCD/CPS
   MILC: http://www.physics.utah.edu/~detar/milc/

Summary

- Very exciting time as code porting work is coming to fruition
  - Perlmutter is here, we are looking forward to hopping on...
  - Frontier will be starting to arrive very soon
  - Slightly stressful: tooling is still maturing in some cases, bugs to iron out
- Some interesting new machines on the horizon
  - Aurora at ALCF
  - Large A64FX based system by Atos with GENCI and CEA in France
  - A system based on the Grace CPU from NVIDIA at Los Alamos
- With so many architectures it is important to have a good performance portability strategy
- Algorithms also advancing – sadly not enough time to cover in this talk
  - Multigrid algorithms advances (E. Weinberg, P. Boyle, ...)
  - Advances in Domain Wall algorithms: Multi-Grid and DDHMC (talk by Peter Boyle, Chulwoo Jung and others)
  - Algorithms to combat critical slowing down and adding machine learning to accelerate Monte-Carlo methods. (talks by Sam Foreman & Xiao-Yong Jin)
  - and many more...

Departing Plot

- In 2018 we measured a baseline of our HMC gauge generation algorithm on 1024 nodes of Titan at OLCF
  - No multigrid yet, but GPU accelerated – K20X GPUs
- On this figure we can track improvement of algorithms, software and algorithm tuning
- At the current time what took 4006 sec. on 1024 Titan nodes, takes only 185 sec. on the NVIDIA Selene System (#6 on Top500 system)
  - 21.6x speedup on 8x fewer devices
  - 173x “integrated improvement”
- Looking forward to adding some Frontier data on this plot soon
- Where will new advances take us next?



Hardware: 2.13x wall-time on 8x fewer GPUs = 17x

4.1x faster on 2x fewer GPUs ~8x gain

Hardware, NVSHMEM, Tensor Cores 2.11x

Algorithms, Software and Tuning: 4.79x

Chroma w/ QDP-JIT and QUDA, ECP FOM data, V=64^3x128 sites, m_π ~172 MeV, (QDP-JIT by F. Winter, Jefferson Lab)

Original figure credit Balint Joo ~2 years ago, new numbers from M. Wagner

Contribution and Collaboration Acknowledgements

- I would like to express my thanks to everyone in the QUDA Portability working group: K. Clark, D. Howarth, X-Y Jin, D. McDougall, J. Osborn, C. Robeck, P. Steinbrecher, A. Strelchenko
- Collaborators in the previous Kokkos and SYCL work (C. Trott, D. Sunderland, D. Ibanez, N. Liber, J. DeSlippe, T. Kurth, J. Kim, P. Steinbrecher, D. Doerfler)
- K. Clark, E. Weinberg, J, Tu, M. Wagner, NVIDIA for long time collaboration and specifically providing figures and helpful comments
- Corbin Robeck, Damon McDougall, Nick Curtis, Rene van Oostrum from AMD at the Frontier Center Of Excellence, and Leopold Ginberg from the AMD Business Unit for detailed collaboration on porting and optimizing on AMD GPUs.
- F. Winter for use ot his QDP-JIT plot
- P. Boyle for their Grid Poster and details of Domain Wall algorithms,
- N. Meyer for information about QPACE 4, and the QPACE 4 Poster,
- Sam Foreman, and Xiao-Yong Jin for the l2hmc and ML Transformation based HMC algorithms
- Photographs of computers courtesy of their institutions and the internet.
- Use of computer systems for data generated here, including Summit and Spock at Oak Ridge Leadership Computing Facility (OLCF), the Tulip System operated by HPE/Cray,   Inte Gen-9 workstation at Jefferson Lab
- Philip Roth for suggesting the Hospital Pain Chart on Slide 8.

Funding Acknowledgements

- This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

- This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

- Portions of this research were supported by the Office of Nuclear Physics , the Office of High Energy Physics, and the Office of Advanced Scientific Computing Research; U.S. Department of Energy under the SciDAC-4 Program and carried out at Thomas Jefferson National Accelerator Facility, manged by Jefferson Science Associates, LLC under U.S. DOE Contract No. DE-AC05-06OR23177.