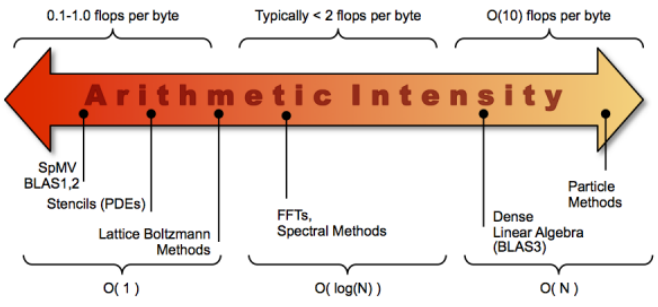
Multi-Platform Performance Portability for QCD Simulations

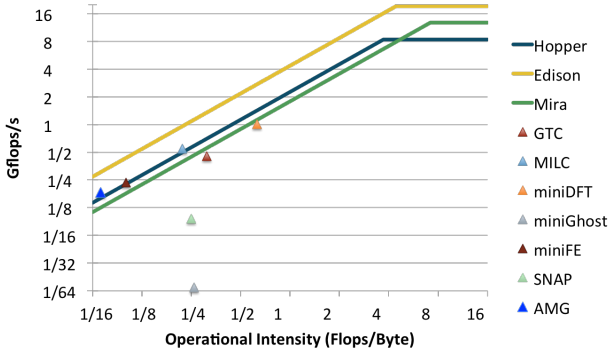
Peter Boyle
High Energy Theory
Brookhaven National Laboratory
and University of Edinburgh

- Will discuss:
  Extreme Scaling system for QCD simulation (2018, upgrade 208)
  Portability to future hardware
  Convergence of HPC and AI needs

Data motion is the rate determining step

- Floating point is now free
- Data motion is now key
  - Quality petaflops, not Linpack petaflops!





- Berkely roofline model: Flops/Second = (Flops/ Byte)× (Bytes/Second)
  - One dimensional - *only* memory bandwidth is considered
  - Arithmetic intensity = (Flops/ Byte)
- With more care can categorise data references by origin
  - Cache, Memory, Network

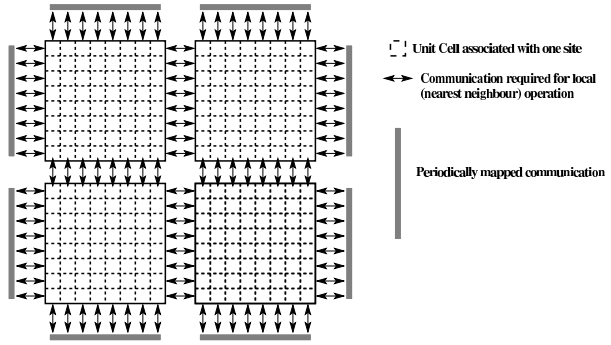
Requirements: scalable Quantum Chromodynamics

- Relativistic PDE for spin 1/2 fermion fields as engraved in St. Paul's crypt:

$$i\gamma \cdot \partial \psi = i\partial\!\!\!/ \psi = m\psi$$

QCD sparse matrix PDE solver communications

- L^4 local volume (space + time)
- finite difference operator 8 point stencil



- $\sim \frac{1}{L}$ of data references come from off node

Scaling QCD sparse matrix requires interconnect bandwidth for halo exchange

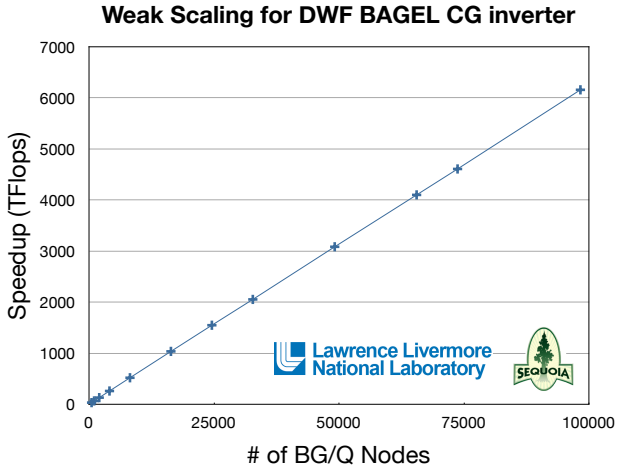
$$B_{network} \sim \frac{B_{memory}}{L} \times R$$

where R is the *reuse* factor obtained for the stencil in caches

- Aim: Distribute 100^4 datapoints over 10^4 nodes

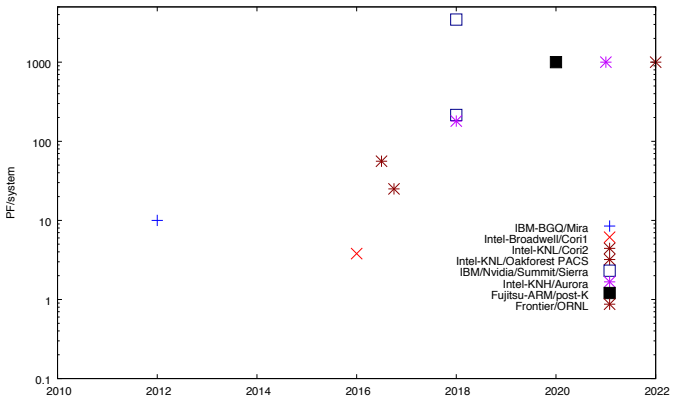
QCD on DiRAC BlueGene/Q (2012-2018)

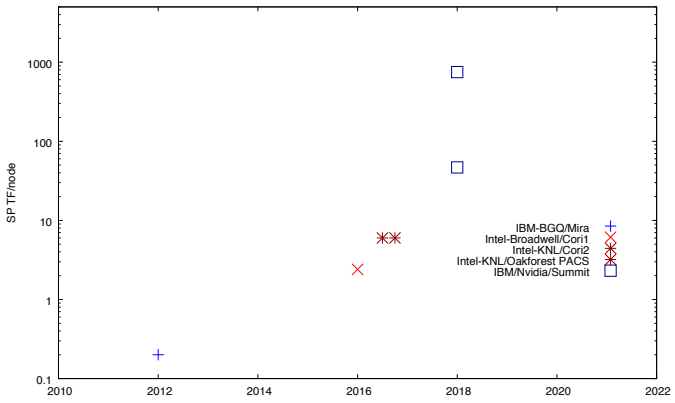
Tesseract Extreme Scaling (2018,2019) is a replacement for:

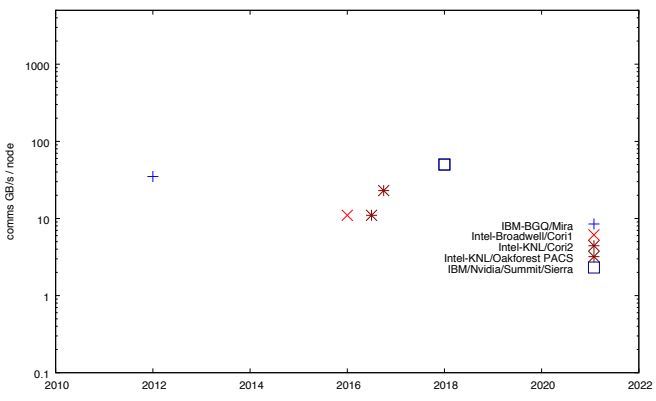


Sustained 7.2 Pflop/s on 1.6 Million cores (Gordon Bell finalist SC 2013)
Edinburgh system 98,304 cores (installed 2012)

Immediate Big HPC roadmap







System	Processor	Accelerator	Network
Summit	Power9	V100 x6	Mellanox EDR
Sierra	Power9	V100 x4	Mellanox EDR
Perlmutter	AMD	Nvidia	Cray Slingshot
Aurora	Intel Xeon	Intel Xe	Cray Slingshot
Frontier	AMD	AMD Radeon	Cray Slingshot
Fugaku	Fujitsu/ARM	-	Fujitsu Tofu

- 400x+ increase in SP node performance accompanied by NO increase in interconnect
  - *FP16 gain is 6x more again!*
- US exascale systems planned in 2021, 2022 (Aurora/Argonne, Frontier/ORNL)
- Many *different* accelerated compute nodes, but multicore remains important

Grid QCD code

Design considerations

- Performance portable across multi and many core CPU's

$$\mathrm{SIMD} \otimes \mathrm{OpenMP} \otimes \mathrm{MPI}$$

- Performance portable to GPU's

$$\mathrm{SIMT} \otimes \mathrm{offload} \otimes \mathrm{MPI}$$

- N-dimensional cartesian arrays
- Multiple grids
- Data parallel C++ layer : Connection Machine inspired

Started in 2014 as Intel IPCC project
GPU portability studies with USQCD
Adopted as POR by USQCD in DOE Exascale Computing Project

Consistent emerging solution : advanced C++

Granularity exposed through ISA/Performance
$\Rightarrow$ data structures must change with each architecture

OpenMP, OpenAcc do not address data layout

Several packages arriving at similar conclusions:

- Kokkos (Sandia)
- RAJA (Livermore)
- Grid (Edinburgh + DOE ECP)

Use advanced C++11 features, inline header template library, auto, decltype etc..

- Discipline and coding standards are required. C++ can be inefficient otherwise
- Hide data layout in opaque container class
- Device lambda capture key enabling feature (CUDA, SyCL), or OpenMP 5.0

```
accelerator_for(iterator, range, {
   body;
});
```

GRID data parallel template library

- www.github.com/paboyle/Grid, arXiv:1512.03487, arxiv:1711.04883 [1]

Ordering	Layout	Vectorisation	Data Reuse
Microprocessor	Array-of-Structs (AoS)	Hard	Maximised
Vector	Struct-of-Array (SoA)	Easy	Minimised
Grid	Array-of-structs-of-short-vectors (AoSoSV)	Easy	Maximised

- Automatically transform layout of arrays of mathematical objects using vSIMD template parameter
- Conformable array operations are data parallel on the *same* Grid layout
- Internal type can be SIMD vectors *or* scalars

```
LatticeColourMatrix A(Grid);
LatticeColourMatrix B(Grid);
LatticeColourMatrix C(Grid);
LatticeColourMatrix dC_dy(Grid);

C = A*B;

const int Ydim = 1;

dC_dy = 0.5*Cshift(C,Ydim, 1 )
      - 0.5*Cshift(C,Ydim,-1 );
```

- *High-level* data parallel code gets 65% of peak on AVX2
- Single data parallelism model targets BOTH SIMD and threads efficiently.
- Expression template engine is only 400 lines (but pretty dense code!)
  - USQCD QDP++/PETE C++98 code is less flexible, over 10^5 lines of code

[1] Also: good, flexible C++ object serialisation using variadic macros. IDL's not required.

Capturing SIMT and SIMD under a single Kernel

The struct-of-array (SoA) portability problem:

- Scalar code: CPU needs struct memory accesses struct calculation
- SIMD vectorisation: CPU needs SoA memory accesses and SoA calculation
- SIMT coalesced reading: GPU needs SoA memory accesses struct calculation
- GPU data structures in memory and data structures in thread local calculations *differ*

Model	Memory	Thread
Scalar	Complex Spinor[4][3]	Complex Spinor[4][3]
SIMD	Complex Spinor[4][3][N]	Complex Spinor[4][3][N]
SIMT	Complex Spinor[4][3][N]	Complex Spinor[4][3]
Hybrid?	Complex Spinor[4][3][Nm][Nt]	Complex Spinor[4][3][Nt]

How to program portably?

- Use operator() to transform memory layout to per-thread layout.
- Two ways to access for read
- operator[] returns whole vector
  - operator() returns SIMD lane threadIdx.y in GPU code
  - operator() is a trivial identity map in CPU code
- Use coalescedWrite to insert thread data in lane threadIdx.y of memory layout.

Single coding style for the above

WAS

```
LatticeFermionView a,b,c;
accelerator_for(ss,volume, {
   a[ss] = b[ss] + c[ss] ;
});
```

NOW

```
LatticeFermionView a,b,c;
accelerator_for(ss,volume,Spinor::Nsimd(), {
   coalescedWrite(a[ss], b(ss) + c(ss) );
});
```

On GPU accelerator for sets up a volume $\times$ Nsimd thread grid.

- Each thread is responsible for one SIMD lane

On CPU accelerator for sets up a volume OpenMP loop.

- Each thread is responsible for Nsimd() SIMD lanes

Per-thread datatypes inside these loops cannot be hardwired.

C++ auto and decltype use the return type of operator () to work out computation variables in architecture dependent way.

Single coding style for the above

Introduce for GPU

- `vobj::scalar_object coalescedRead (vobj) ;`
- `vobj::scalar_object coalescedReadPermute (vobj,int ptype) ;`
- `coalescedWrite(vobj &,vobj::scalar_object &) ;`

Under the hood operator [] and () behave differently:

```
vobj & Lattice<vobj>::operator[] (Integer site) return odata[site]
#ifdef GRID_SIMT
const vobj::scalar_object Lattice<vobj>::operator() (Integer site) {
  return extractLane(odata[site],threadIdx.y);
}
#else
const vobj & Lattice<vobj>::operator() (Integer site) {
  return odata[site];
}
#endif
```

- C++11 auto and decltype used to *infer* the internal thread datum
- Sequence transforms with the architecture:

```
  accelerator_for(sss,nloop,nsimd,{
    uint64_t ss= sss*Ls;
    typedef decltype(coalescedRead(psi[0])) spinor; // <- type is arch dependent
    spinor tmp1, tmp2;                              // <- these live in stack on GPU
    for(int s=0;s<Ls;s++){
      uint64_t idx_u = ss+((s+1)%Ls);
      uint64_t idx_l = ss+((s+Ls-1)%Ls);
      spProj5m(tmp1,psi(idx_u));                    // psi() accesses coalesce
      spProj5p(tmp2,psi(idx_l));
      coalescedWrite(chi[ss+s],diag[s]*phi(ss+s)+upper[s]*tmp1+lower[s]*tmp2);
    }
  });
```

Grid single node performance

Architecture	Cores	GF/s (Ls x Dw)	peak
Intel Knight's Landing 7250	68	770	6100
Intel Knight's Corner	60	270	2400
Intel Skylakex2	48	1200	9200
Intel Broadwellx2	36	800	2700
Intel Haswellx2	32	640	2400
Intel Ivybridgex2	24	270	920
AMD EPYCx2	64	590	3276
AMD Interlagosx4	32 (16)	80	628
Nvidia Volta	84 SMs	1500	15700

- Dropped to inline assembly for key kernel in KNL and BlueGene/Q
- EPYC is MCM; ran 4 MPI ranks per socket, one rank per die
- Also: ARM Neon and ARM SVE port

Common source accelerator port.

- Assumed Unified Virtual Memory (not restriction to Nvidia as Intel and AMD GPU's support under OpenCL/Linux)
- CUDA; considering OpenMP 5.0 *and* SyCL for AMD & Intel accelerator portability

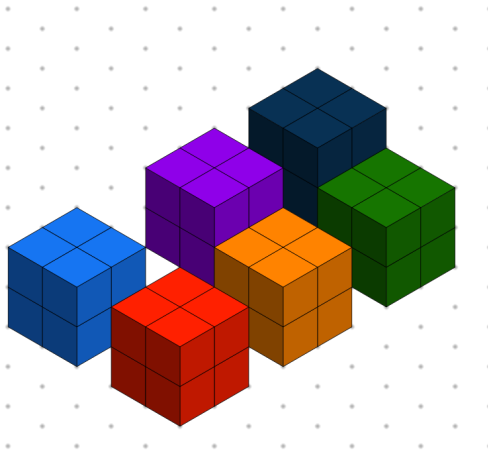
Exploiting locality: multiple chips/GPUs per node

Multi-socket servers: NUMA aware code

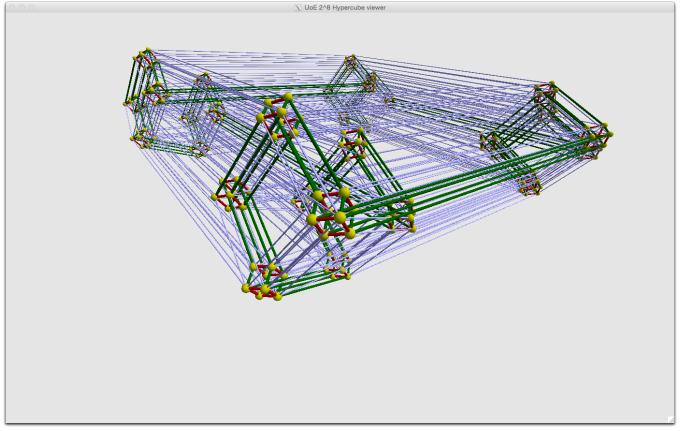
- Hybrid OpenMP + MPI
- Use 1:1 mapping between MPI ranks and sockets
- Unix shared memory between sockets/NUMA domains (over UPI)
- Reserve MPI transfers for inter-node

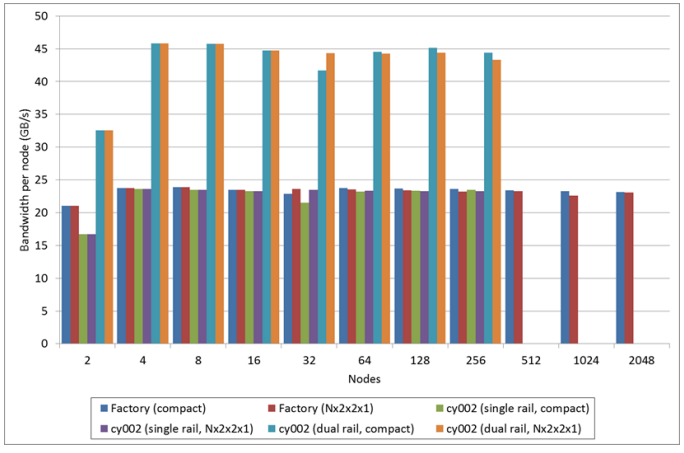
Multi-GPU servers: NVlink aware code

- Use 1:1 mapping between MPI ranks and GPU's
- Peer2peer used between GPUs (over NVlink)
- Reserve MPI transfers for inter-node, direct to GPU if possible



Exploiting locality: hypercube network



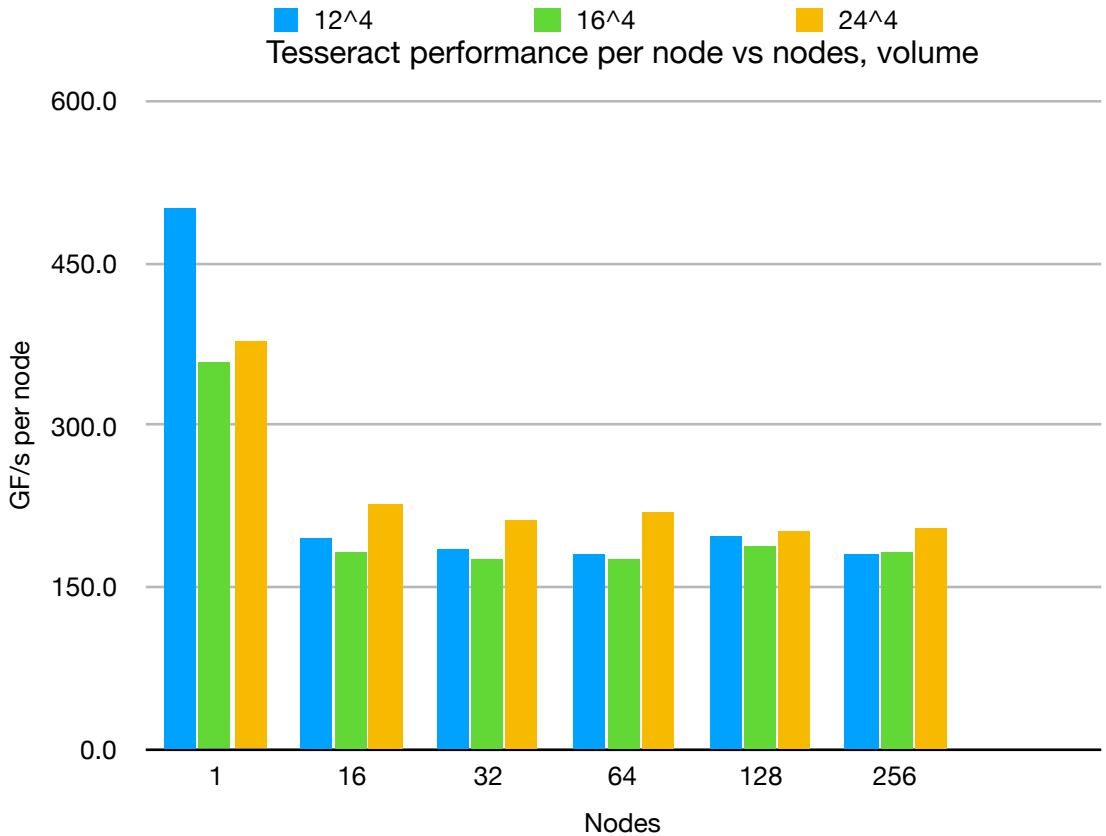


Improvement over Default Process Placement		
Nodes	**Decomp**	**Bandwidth**
2	2x1x1x1	0.98
4	2x2x1x1	1.00
8	2x2x2x1	1.00
16	4x2x2x1	1.27
32	4x4x2x1	1.70
64	4x4x4x1	2.00
128	8x4x4x1	2.09
256	8x8x4x1	2.60
512	8x8x8x1	3.30
1024	16x8x8x1	3.51
2048	16x16x8x1	3.84

- Small project with SGI/HPE on Mellanox EDR networks (James Southern)
- Embed 2^n QCD torus inside hypercube so that nearest neigbour comms travels single hop
  4x speed up over default MPI Cartesian communicators on large systems
  $\Rightarrow$ Customise HPE 8600 (SGI ICE-XA) to use $16 = 2^4$ nodes per leaf switch

DiRAC HPE ICE-XA hypercube network

- Edinburgh HPE 8600 system (Installed March 2018, 2 days ahead of schedule)
  - Low end Skylake Silver 4116, 12 core parts
  - Single rail Omnipath interconnect
  - Relatively cheap node: high node count and scalability
    - Improve price per core, bandwidth per core, reduce power



- 16 nodes (single switch) delivers bidirectional 25GB/s to every node (wirespeed)
- 512 nodes topology aware bidirectional 19GB/s
  - 76% wirespeed using every link in system concurrently

HPC and AI

What do fundamental physics and AI have in common?

- On the surface very little!!!
- **But** present similar computational problems and solutions
  - Both MCMC and AI Training are serial, tightly coupled problem
  - Data motion is key in a large distributed memory computer
  - Enormous floating point requirement, with mixed precision tolerance

AI is a nonlinear optimisation problem:

$$\mathrm{Cost}(\mathrm{Weights}) = \sum_i |\mathrm{Network}(\mathrm{Weights}, \mathrm{Sample}_i) - \mathrm{Reference}_i|^2$$

Most common approach, Stochastic Gradient Descent iterates:

1. Choose random B subset (batch) of samples
2. Evaluate $\mathrm{Gradient} = \nabla_W \sum_{i \in B} Cost_i(W)$
3. Update $W \rightarrow W - \alpha \mathrm{W}$

Different sample costs are independent and parallelisable; must reduce the gradient across machine

Desperately seeking Bandwidth

- Collaboration w. Intel, Brookhaven Lab: concurrency in Intel MPI and Omnipath software
- With thanks to Joe Curley, Larry Meadows & Michael Chuvalev
  - Reentrancy to MPI needed with hybrid threads + MPI when many HFI's
  - Avoid 4KB pages due to per page software overhead

https://arxiv.org/pdf/1711.04883.pdf

- EDI + BNL + CU + Intel paper

Accelerating HPC codes on Intel® Omni-Path Architecture networks: From particle physics to Machine Learning

Peter Boyle,[1] Michael Chuvelev,[2] Guido Cossu,[3] Christopher Kelly,[4] Christoph Lehner,[5] and Lawrence Meadows[2]

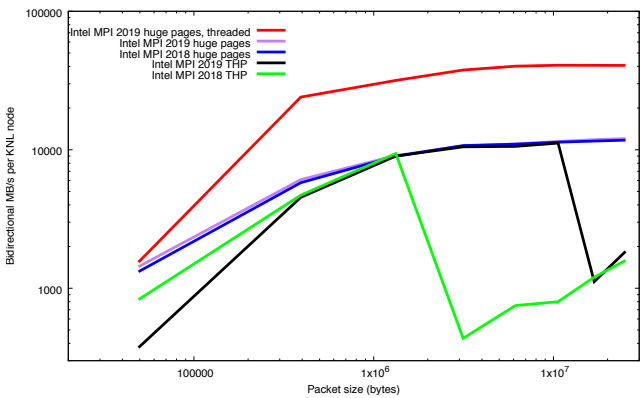
[1]*The University of Edinburgh and Alan Turing Institute*

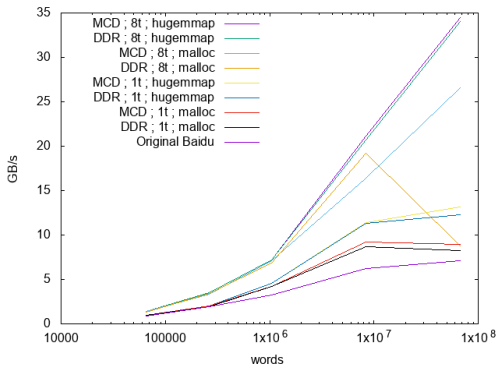
[2]*Intel*

[3]*The University of Edinburgh*

[4]*Columbia University*

[5]*Brookhaven National Laboratory*





- Brookhaven Lab dual rail KNL/OPA system
- 10x acceleration of Baidu "optimised reduction" code
- http://research.baidu.com/bringing-hpc-techniques-deep-learning/
- https://github.com/baidu-research/baidu-allreduce
- Thread scalable MPI reenter on multiple communicators.
- Who needs MPI endpoints?

https://www.nextplatform.com/2017/11/29/the-battle-of-the-infinibands/

Comparison: Summit

- ORNL, 4608 nodes, fastest in top500
  - 6 V100 GPU's, 90TF/s single precision, 750TF/s half precision for AI
  - 5000+ GB/s memory bandwidth
- Dual rail 50GB/s EDR exterior interconnect
  - 100:1 memory to network ratio ☹

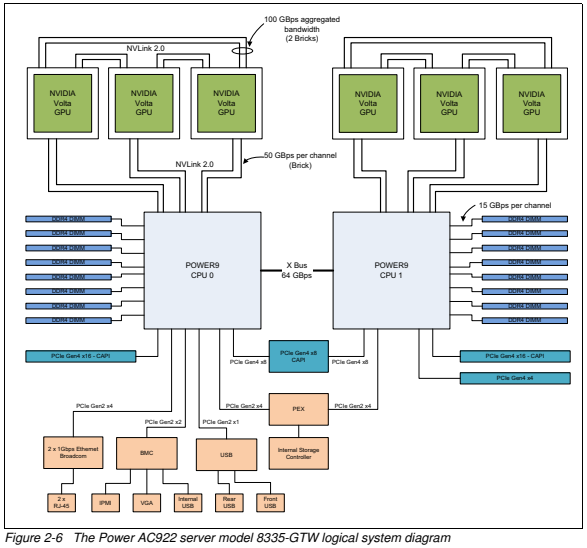
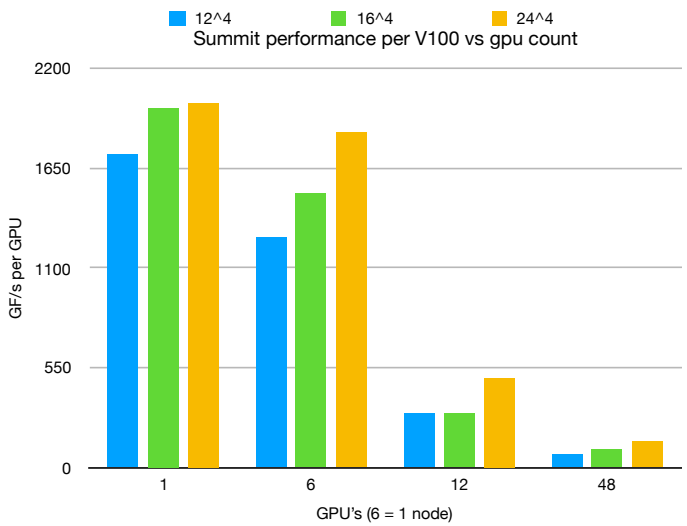


Figure 2-6 The Power AC922 server model 8335-GTW logical system diagram

Same tightly coupled problem on Summit

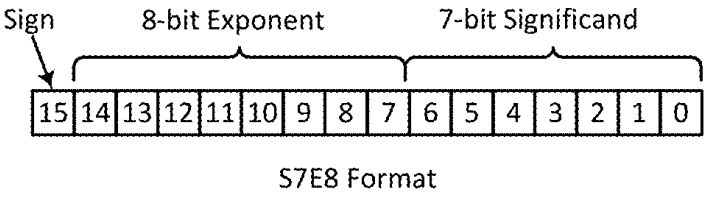
Use Nvidia QUDA code: This is a bad (apples to oranges) comparison at present for three reasons

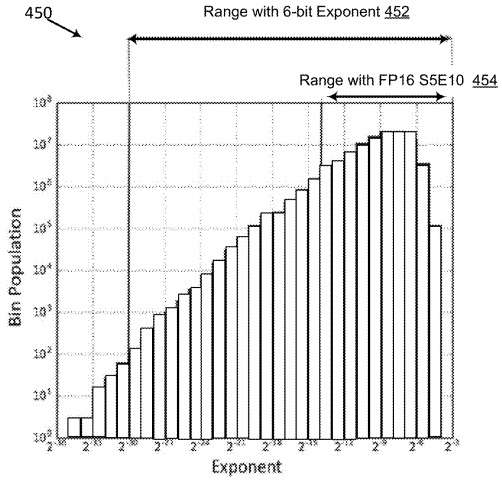
1. Tesseract node is perhaps 1/2 price of a Volta GPU, currently 2x performance
   - But: GPU has 2x better price/performance for communication light code
2. Summit does not yet have Gpu Direct RDMA (GDR) enabling MPI from device memory
   - Anticipate a gain when GDR is enabled on Summit
   - *Even if break even on price/performance for this (interconnect heavy) code, programming model simplicity swings it for Tesseract*
   - Partially address by using half precision preconditioner
   - Partially address by using domain decomposition preconditioner (2x wall clock gain, 10x flop/s increase)
     - STFC wouldn't let me blow up the power budget this way for the 2x !
3. Many problems, even in QCD, are *not* so communication heavy (e.g. multigrid Wilson)

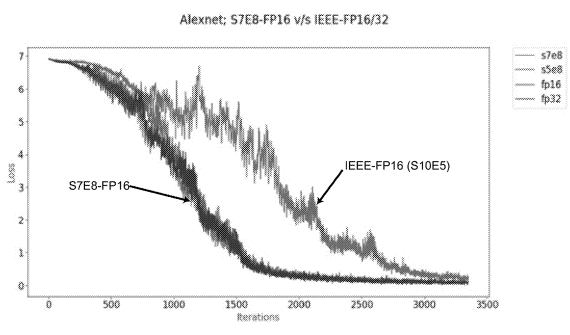


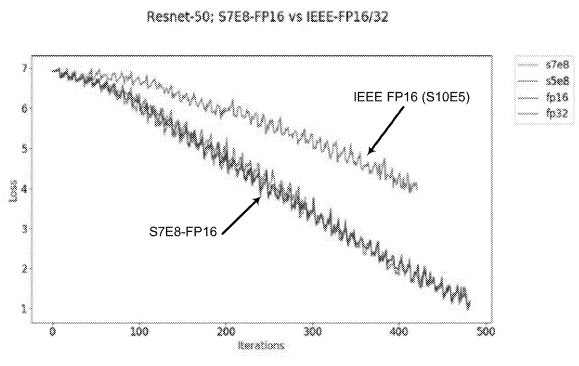
Low precision AI work

- Intel's US Patent Application 20190042544
  (FP16-S7E8 MIXED PRECISION FOR DEEP LEARNING AND OTHER ALGORITHMS)
- Authors: Boyle (ATI, Edinburgh), Sid Kashyap, Angus Lepper (Intel, former DiRAC RSE's)









- Systematic study using Gnu Multi Precision library.
- BFP16 displays greater numerical stability for machine learning training.
- Understanding: histogramme of multiply results during SGD gradient calculations
- Patent application full text searchable on uspto.gov

Lessons learned & Dream machine

Cross platform, single source, performance portability is achievable.

1. Advanced C++ can give faster than FORTRAN performance if used judiciously
2. Use macros and _Pragma to mark up loops flexibly
3. Capture parallel loop bodies in a macro
4. Store memory arrays in an opaque template container
   if you can hide the layout, you can change the layout with architecture
5. A per thread accessor () can hide the difference between SIMT and SIMD

Dream machine

- Aim to have accelerated nodes
- With 1:1 ratio between accelerators and 100Gbit/s HFI
- All Cray Slingshot systems planned for US look promising
  - Avoid the Summit interconnect cliff
  - Avoid lock in to any one accelerator vendor
  - Wrapping acceleration primitives as described is key
- OpenMP 5.0 acceleration critical for general science community.
- OpenMP 5.0 does not address data layout; users must still think about code.

Ideal QCD machine and ideal AI training machine have similar requirements.