Introduction to Comet and Gordon

Mahidhar Tatineni

UC Irvine, Dec 3, 2015

NSF

UC San Diego

NVIDIA

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

DELL

intel

INDIANA UNIVERSITY

AEON COMPUTING

Mellanox
TECHNOLOGIES

This work supported by the National Science Foundation, award ACI-1341698.

Outline

- **System specifications and features**
- **Special features**
  - Virtualization
  - Gateway hosting
- **Comet usage modes (with hands on)**
  - General compute nodes (Intel Haswell Processors), exclusive, shared, MPI, OpenMP, Hybrid (ibrun)
  - Local scratch (SSDs)
  - GPU nodes (NVIDIA K-80s) : CUDA, OpenACC, mvapich2-gdr
- **Summary**

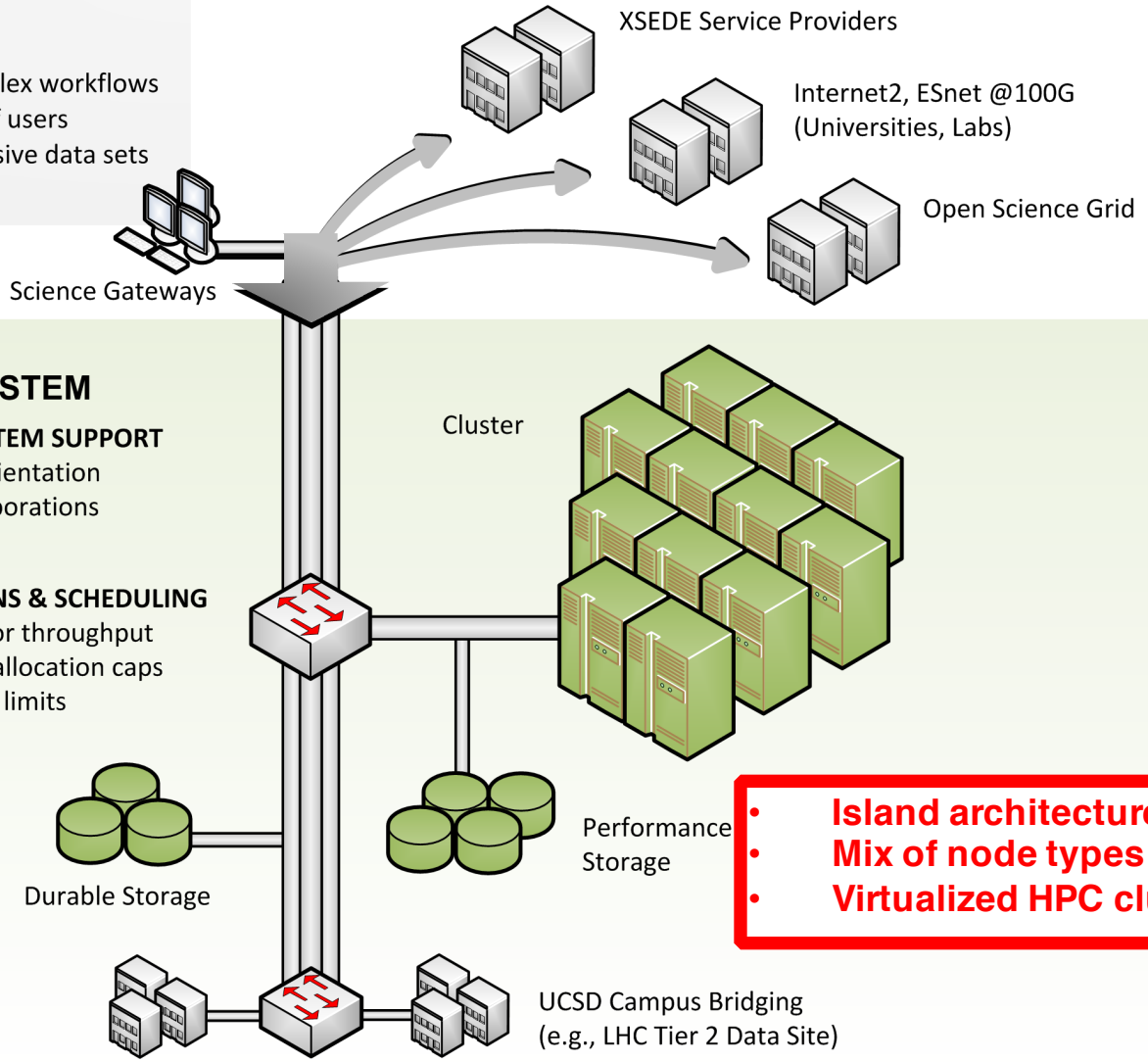
Comet
“HPC for the long tail of science”

iPhone panorama photograph of 1 of 2 server rows

Comet Built to Serve the 99%

CHALLENGES OUR PROPOSAL ADDRESSES

- ✓ Attract new users and communities
- ✓ Support diverse applications with complex workflows
- ✓ Ensure responsiveness for thousands of users
- ✓ Transfer, store, analyze, and share massive data sets
- ✓ Integrate with XSEDE



COMET COMPUTE SYSTEM

Cluster architecture
Fast standard nodes
Large-memory nodes
GPU-accelerated nodes
FDR InfiniBand

Storage architecture
Performance Storage
Durable Storage

Software
Science Gateways
Rich base of installed apps
Virtualization

USER & SYSTEM SUPPORT
New user orientation
XSEDE collaborations
FutureGrid

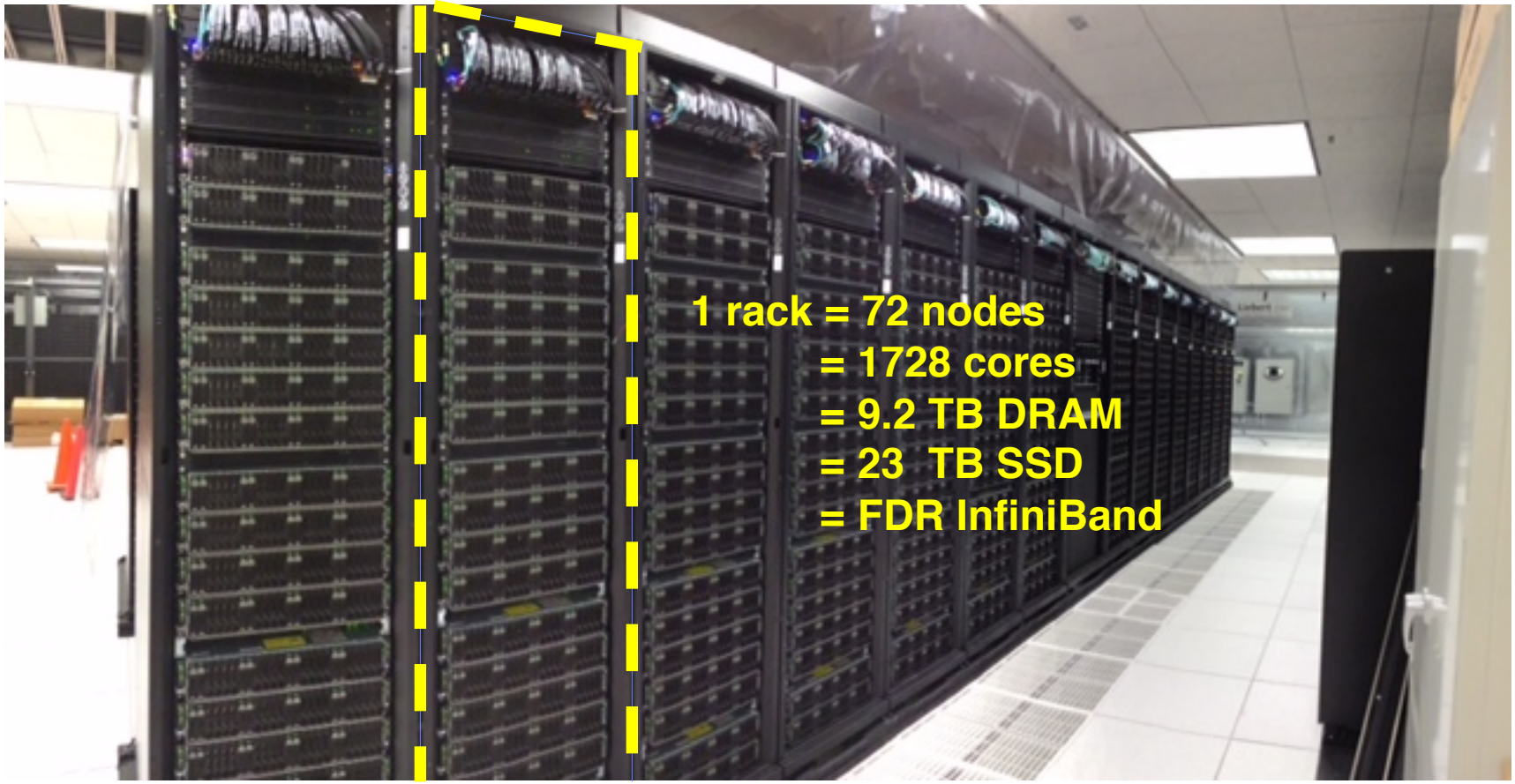
ALLOCATIONS & SCHEDULING
Optimized for throughput
Per-project allocation caps
Per-job core limits

- **Island architecture**
- **Mix of node types**
- **Virtualized HPC clusters**

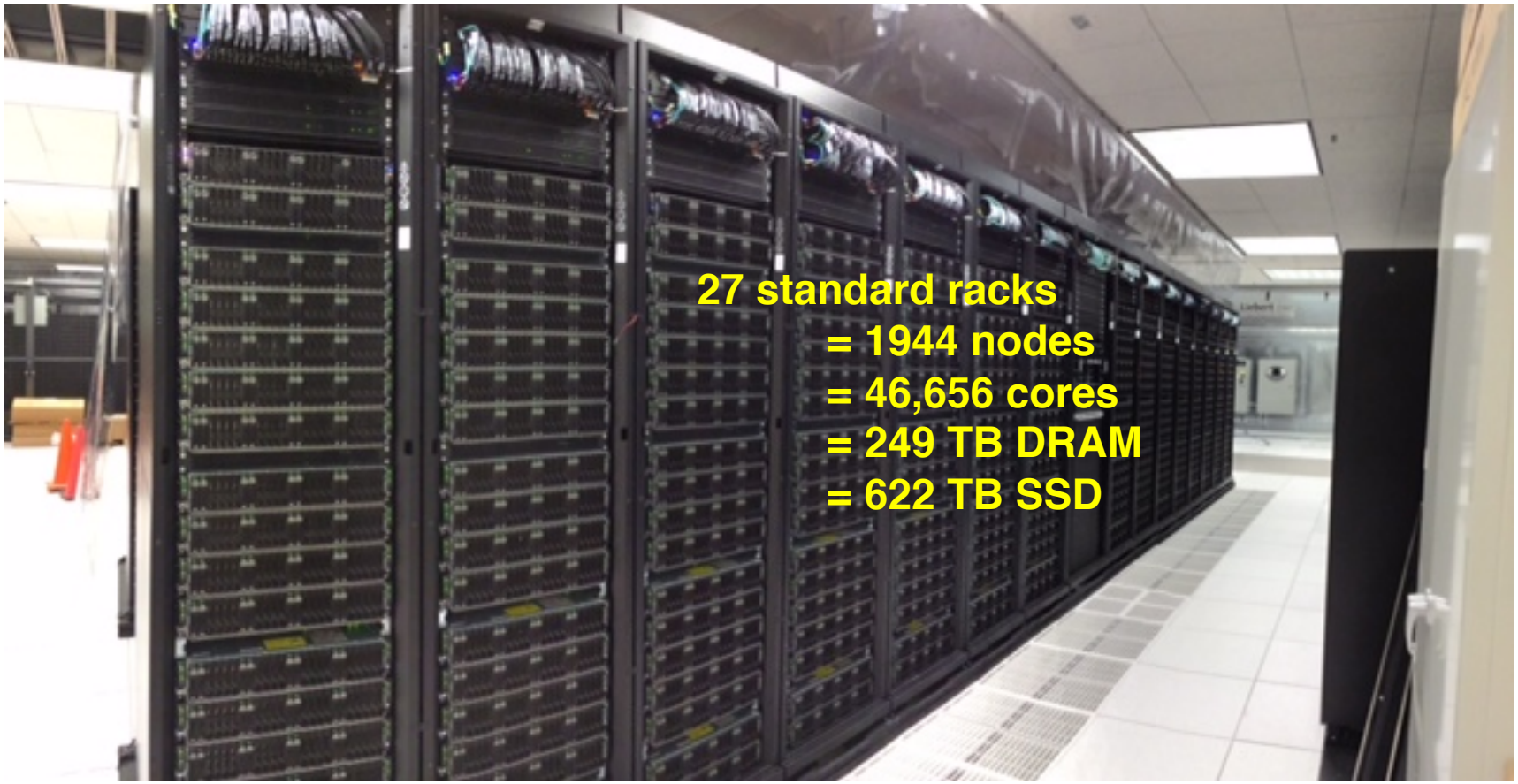
Comet: System Characteristics

- **Total peak flops ~2.1 PF**
- **Dell primary integrator**
  - Intel Haswell processors w/ AVX2
  - Mellanox FDR InfiniBand
- **1,944 standard compute nodes (46,656 cores)**
  - Dual CPUs, each 12-core, 2.5 GHz
  - 128 GB DDR4 2133 MHz DRAM
  - 2*160GB GB SSDs (local disk)
- **36 GPU nodes**
  - Same as standard nodes *plus*
  - Two NVIDIA K80 cards, each with dual Kepler3 GPUs
- **4 large-memory nodes**
  - 1.5 TB DDR4 1866 MHz DRAM
  - Four Haswell processors/node
  - 64 cores/node
- **Hybrid fat-tree topology**
  - FDR (56 Gbps) InfiniBand
  - Rack-level (72 nodes, 1,728 cores) full bisection bandwidth
  - 4:1 oversubscription cross-rack
- **Performance Storage (Aeon)**
  - 7.6 PB, 200 GB/s; Lustre
  - Scratch & Persistent Storage segments
- **Durable Storage (Aeon)**
  - 6 PB, 100 GB/s; Lustre
  - Automatic backups of critical data
- **Home directory storage**
- **Gateway hosting nodes**
- **Virtual image repository**
- **100 Gbps external connectivity to Internet2 & ESNet**

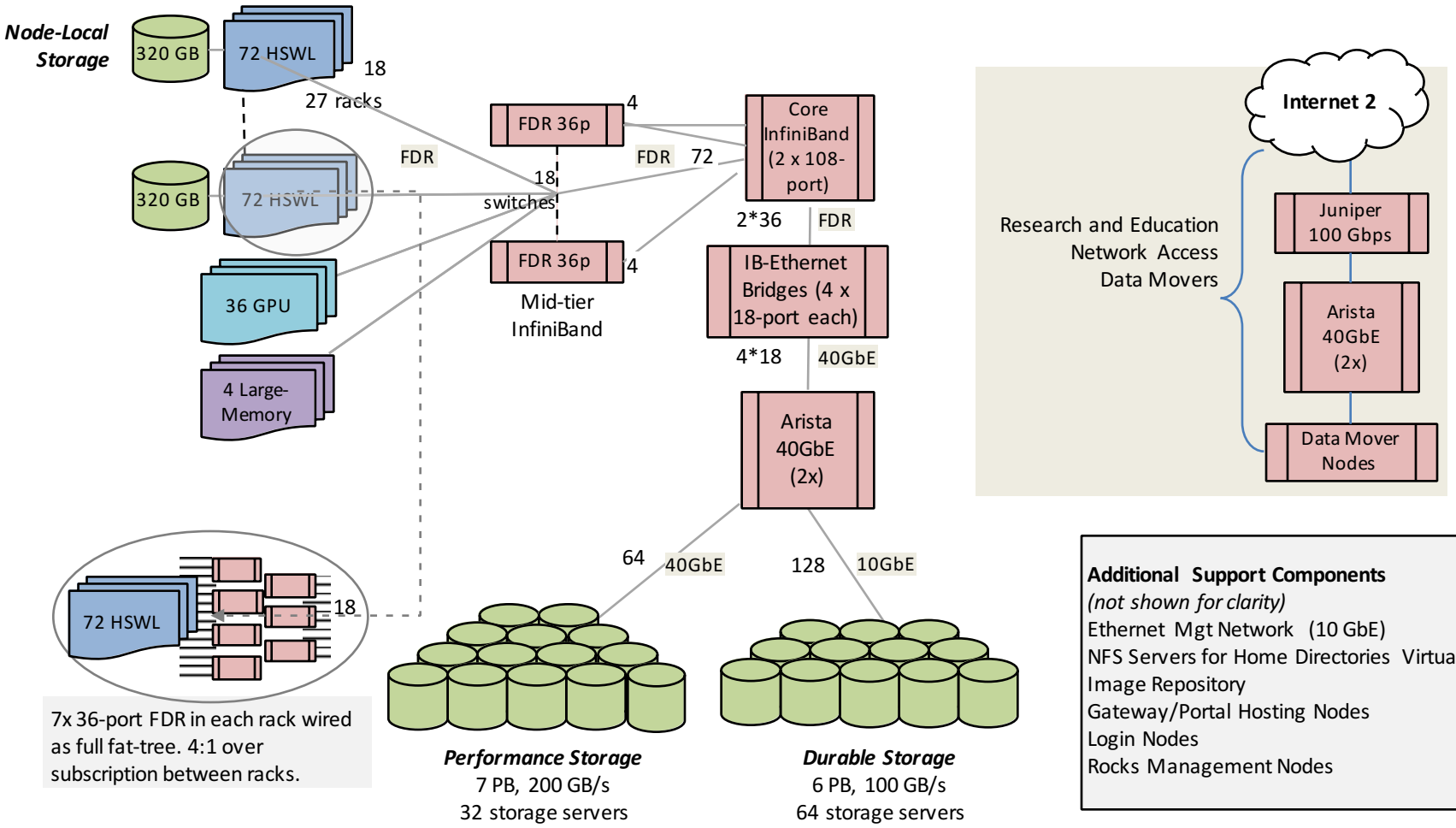
~67 TF supercomputer in a rack



And 27 single-rack supercomputers



Comet Network Architecture
InfiniBand compute, Ethernet Storage



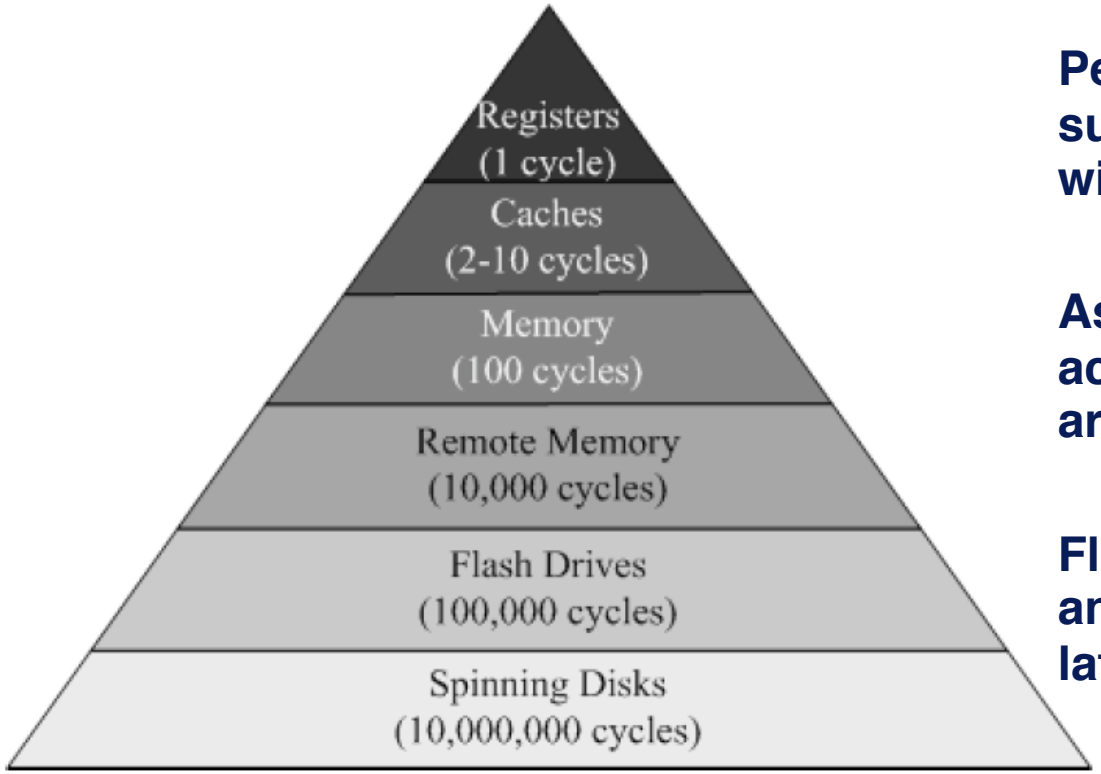
Additional Support Components
(not shown for clarity)
Ethernet Mgt Network (10 GbE)
NFS Servers for Home Directories Virtual Image Repository
Gateway/Portal Hosting Nodes
Login Nodes
Rocks Management Nodes

Gordon – A Data Intensive Supercomputer

Gordon – A Data Intensive Supercomputer

- Designed to accelerate access to massive amounts of data in areas of genomics, earth science, engineering, medicine, and others
- Emphasizes memory and IO over FLOPS.
- Appro integrated 1,024 node Sandy Bridge cluster
- 300 TB of high performance Intel flash
- Large memory supernodes via vSMP Foundation from ScaleMP
- 3D torus interconnect from Mellanox
- In production operation since February 2012
- Funded by the NSF and available through the NSF Extreme Science and Engineering Discovery Environment program (XSEDE)

For data intensive applications, the main advantage of flash is the low latency



Performance of the memory subsystem has not kept up with gains in processor speed

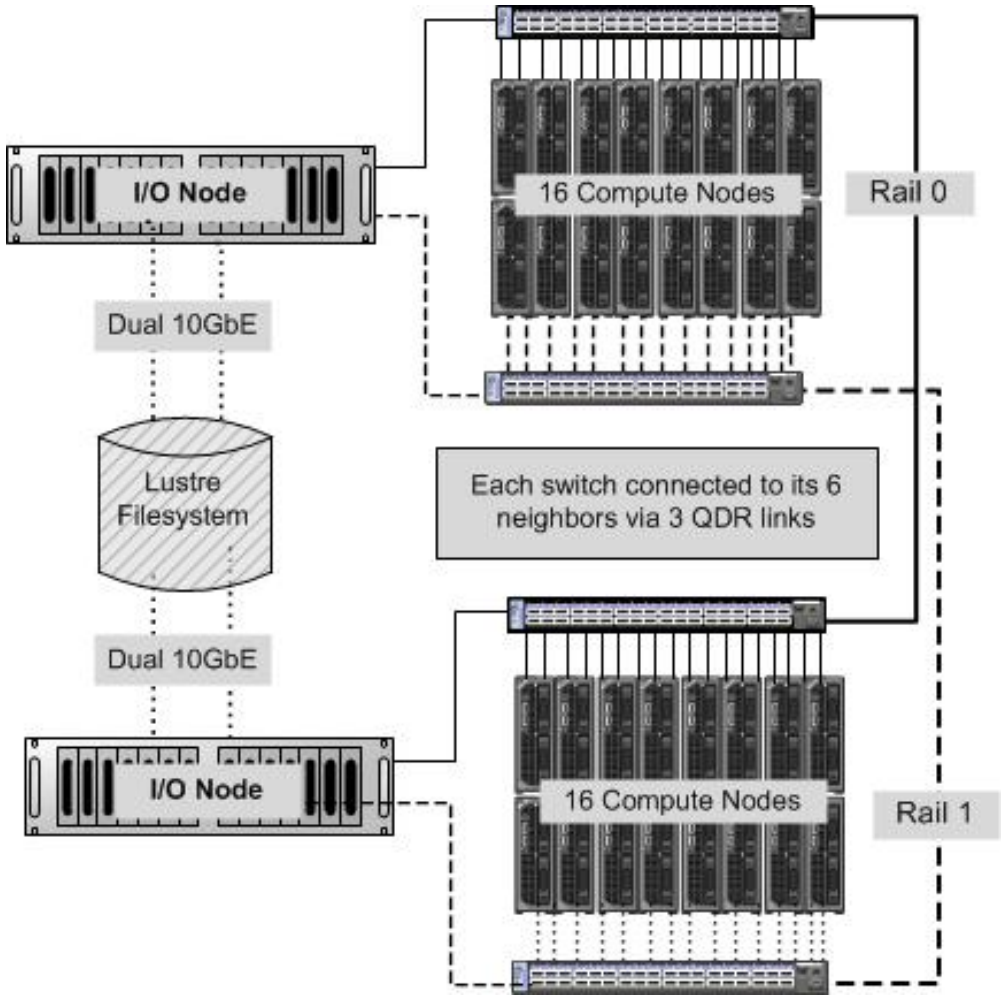
As a result, latencies to access data from hard disk are O(10,000,000) cycles

Flash memory fills this gap and provides O(100) lower latency

Gordon System Specification

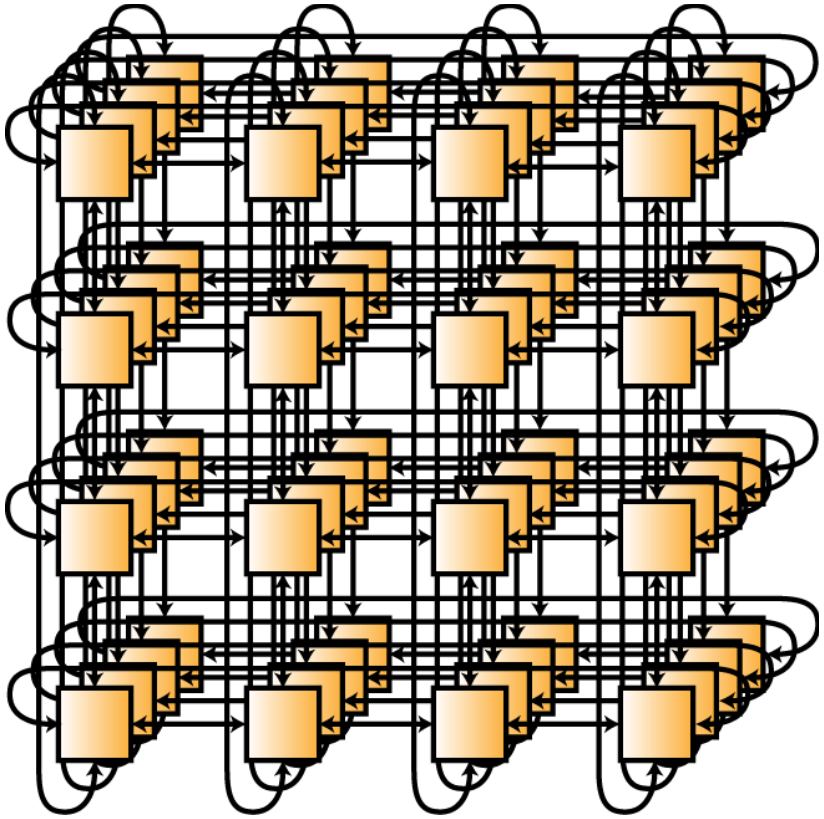
Intel Sandy Bridge Compute node	
Sockets	2
Cores	16
Clock speed	2.6
DIMM slots per socket	4
DRAM capacity	64 GB
Intel Flash I/O Node	
NAND flash SSD drives	16
SSD capacity per drive/Capacity per node/total	300 GB / 4.8 TB / 300 TB
Flash bandwidth per drive (read/write) Flash bandwidth per node (write/read)	270 MB/s / 210 MB/s 4.3 /3.3 GB/s
SMP Super-Node	
Compute nodes	32
I/O nodes	2
Addressable DRAM	2 TB
Addressable memory including flash	12TB
Gordon	
Compute Nodes	1,024
Total compute cores	16,384
Peak performance	341TF
Aggregate memory	64 TB
InfiniBand Interconnect	
Aggregate torus BW	9.2 TB/s
Type	Dual-Rail QDR InfiniBand
Link Bandwidth	8 GB/s (bidirectional)
Latency (min-max)	1.25 μs – 2.5 μs
Disk I/O Subsystem	
Total storage	/oasis/scratch (1.6 PB), /oasis/projects/nsf(1.5PB)
I/O bandwidth	100 GB/s
File system	Lustre

Subrack Level Architecture



16 compute nodes per subrack

3D Torus of Switches



- Linearly expandable
- Simple wiring pattern
- Short Cables- Fiber Optic cables generally not required
- Lower Cost :40% as many switches, 25% to 50% fewer cables
- Works well for localized communication
- Fault Tolerant within the mesh with 2QoS Alternate Routing
- Fault Tolerant with Dual-Rails for all routing algorithms

3rd dimension wrap-around not shown for clarity

Application Performance on Comet compared to Gordon

Code	Cores	Speedup compared to Gordon
NEURON 7.3	*48* *768*	*1.18* *1.24*
OpenFOAM 2.2.0	*96* *384*	*1.18* *1.37*
Quantum ESPRESSO 5.0.2: PWscf	*96* *768*	*1.35* *1.41*
RAxML 7.4.4	*16* *48*	*1.26* *1.23*
WRF 3.0	*96* *768*	*1.20* *1.32*

Hardware speedup varies from 1.18 to 1.41; XRAC conversion will be 1.25

Software – Applications, Libraries

- Users can manage environment via modules.
- Applications packaged into “Rocks Rolls” that can built and deployed on any of the SDSC systems. Benefits wider community deploying software on their Rocks clusters.
- Efficient system administration pooling software install/testing efforts from different projects/machines – Comet benefits from work done for Trestles, Gordon, and Triton Shared Computing Cluster (TSCC).
- Users benefit from a familiar applications environment across SDSC systems => can easily transition from Trestles to Comet.
- Rolls available for all installed applications on Trestles, Gordon. Updated recently for newer compiler, application versions. Listed on next slide.

List of Compilers, Applications, Libraries

Category	List of Software/Libraries
Compilers	*Intel, PGI, GNU, MVAPICH2,OPENMPI*
Bioinformatics	*BamTools, BEAGLE, BEAST, BEAST 2, bedtools, Bismark, BLAST, BLAT, Bowtie, Bowtie 2, BWA, Cufflinks, DPPDiv, Edena, FastQC, FastTree, FASTX-Toolkit, FSA, GARLI, GATK, GMAP-GSNAP, IDBA-UD, MAFFT, MrBayes, PhyloBayes, Picard, PLINK, QIIME, RAxML, SAMtools, SOAPdenovo2, SOAPsnp, SPAdes, TopHat, Trimmomatic, Trinity, Velvet*
Chemistry	*CPMD, CP2K, GAMESS, Gaussian, MOPAC, NWChem, Q-Chem, VASP*
Molecular Dynamics	*AMBER, Gromacs, LAMMPS, NAMD*
Engineering	*ABAQUS*
Data Analysis/Analytics	*Hadoop 1, Hadoop 2 (with YARN), Spark, R, Weka, KNIME*
Visualization	*VisIt, IDL*
Numerical libraries	*ATLAS, FFTW, GSL, LAPACK, MKL, ParMETIS, PETSc, ScaLAPACK, SPRNG, Sundials, SuperLU, Trilinios*
Debugging/Profiling	*DDT, PAPI, TAU, mpiP, Valgrind*

Data Intensive Computing & Viz Stack

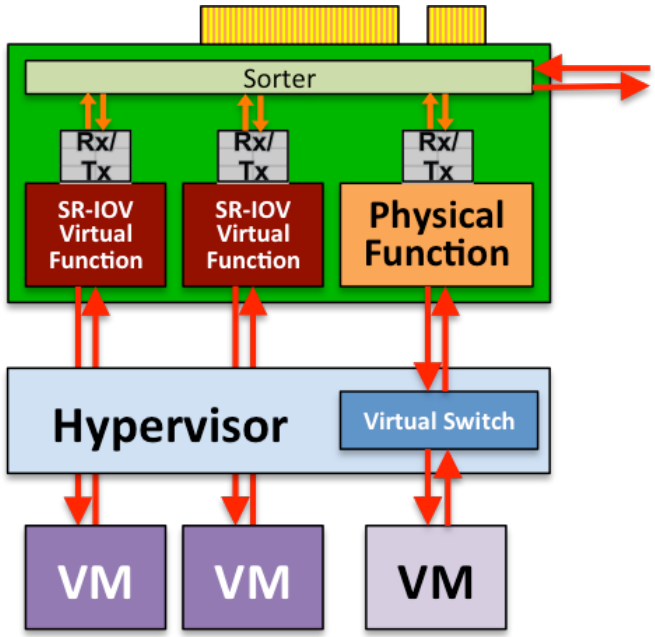
- **Comet retains a lot of Gordon's data intensive computing software and visualization stack.**
- **Comet compute nodes, just like Gordon, have access to the high speed Lustre fileystems and SSD based filesystems.**
- **Several libraries and packages have been installed to enable data intensive computing and visualization:**
  - Hadoop: Version 1.2.1 and 2.6.0 are available
  - Spark : Versions 1.2.0, 1.5.2 available.
  - R – Software environment for statistical computing and graphics.
  - Weka – Tools for data analysis and predictive modeling
  - RapidMiner – Environment for machine learning, data mining, text mining, and predictive analytics
  - Octave
  - Matlab Distributed Computing Server software (limited licenses).
  - VisIt
- **The myHadoop infrastructure was developed to enable use Hadoop for distributed data intensive analysis.**

Comet Special Features

(1) Single Root IO Virtualization (SR-IOV)
(2) Gateway hosting
(3) VM Repository

Single Root I/O Virtualization in HPC

- **Problem**: Virtualization generally has resulted in significant I/O performance degradation (e.g., excessive DMA interrupts)
- **Solution**: SR-IOV and Mellanox ConnectX-3 InfiniBand host channel adapters
  - One physical function → multiple virtual functions, each light weight but with its own DMA streams, memory space, interrupts
  - Allows DMA to bypass hypervisor to VMs
- ***SRIOV enables virtual HPC cluster w/ near-native InfiniBand latency/bandwidth and minimal overhead***



Mellanox TECHNOLOGIES

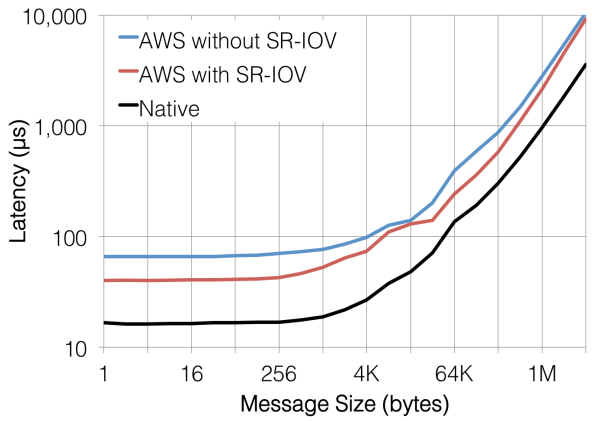
Benchmarks

- Fundamental performance characteristics of interconnect evaluated using OSU Micro-Benchmarks – latency, unidirectional and bidirectional bandwidth tests.

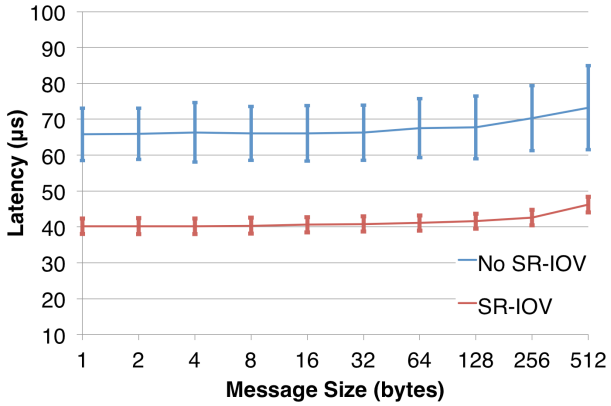
- WRF: widely used weather modeling application that is run in both research and operational forecasting. CONUS-12km benchmark used for performance evaluation.

- Quantum ESPRESSO: Application that performs density functional theory (DFT) calculations for condensed matter problems. DEISA AUSURF112 benchmark.

SR-IOV with 10GbE*: Latency Results



MPI point-to-point latency as measured by the `osu_latency` benchmark.



MPI point-to-point latency as measured by the `osu_latency` benchmark. Error bars are +/- three standard deviations from the mean.

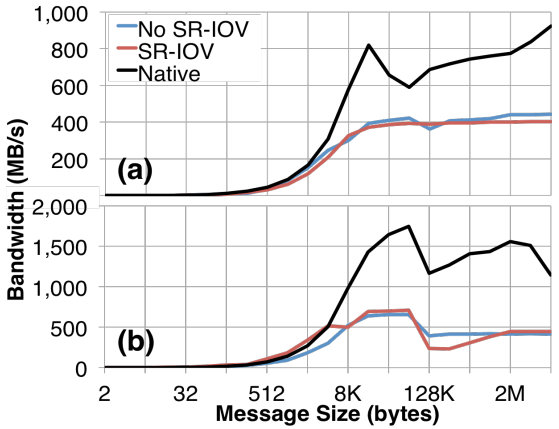
- 12-40% improvement under virtualized environment with SR-IOV .
- 2-2.5X slower than native case, even with SR-IOV.

* SR-IOV provided with Amazon's C3 instances

- SR-IOV provides 3× to 4× less variation in latency for small message sizes.

SR-IOV with 10GbE: Bandwidth Results

- Unidirectional messaging bandwidth never exceeds 500 MB/s (~40% of line speed).
- Native performance is 1.5-2X faster.
- Similar results for bidirectional bandwidth. SR-IOV has very little benefit in both cases.
- SR-IOV helps slightly (13% for random ring, 17% for natural ring) in collective bandwidth tests.
- Native total ring bandwidth was more than 2X faster than SR-IOV based virtualized results.



MPI (a) unidirectional bandwidth and (b) bidirectional bandwidth for 10GbE interconnect as measured by the `osu_bw` and `osu_bibw` benchmarks, respectively.

SR-IOV with InfiniBand: Latency

- **SR-IOV**
  - < 30% overhead for M < 128 bytes
  - < 10% overhead for eager send/recv
  - Overhead → 0% for bandwidth-limited regime
- **Amazon EC2**
  - > 5000% worse latency
  - Time dependent (noisy)

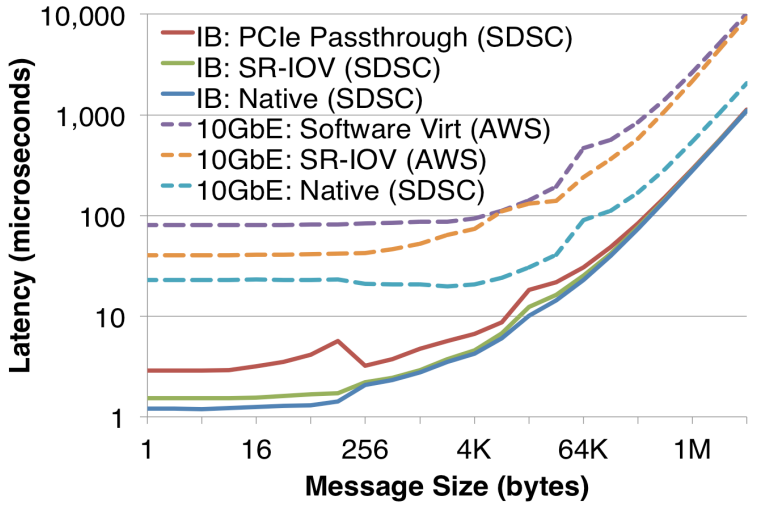


Figure 5. MPI point-to-point latency measured by osu_latency for QDR InfiniBand. Included for scale are the analogous 10GbE measurements from Amazon (AWS) and non-virtualized 10GbE.

50x less latency than Amazon EC2

OSU Microbenchmarks (3.9, osu_latency)

SR-IOV with InfiniBand: Bandwidth

- **SR-IOV**
  - < 2% bandwidth loss over entire range
  - > 95% peak bandwidth
- **Amazon EC2**
  - < 35% peak bandwidth
  - 900% to 2500% worse bandwidth than virtualized InfiniBand

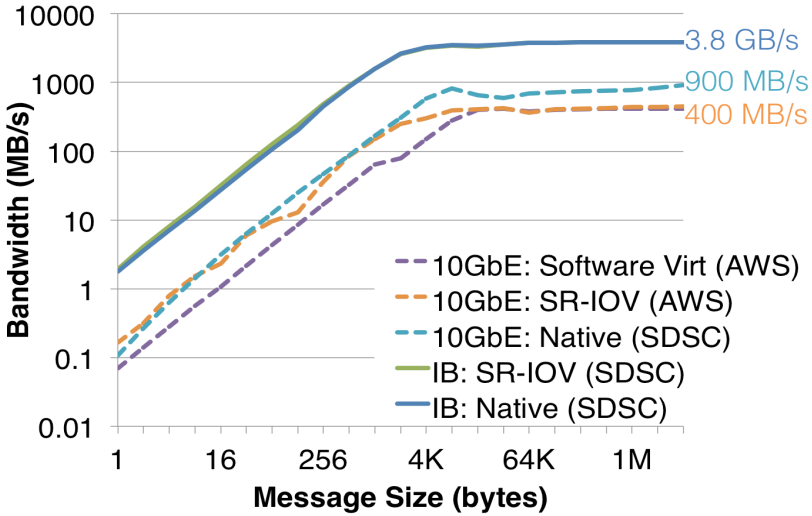
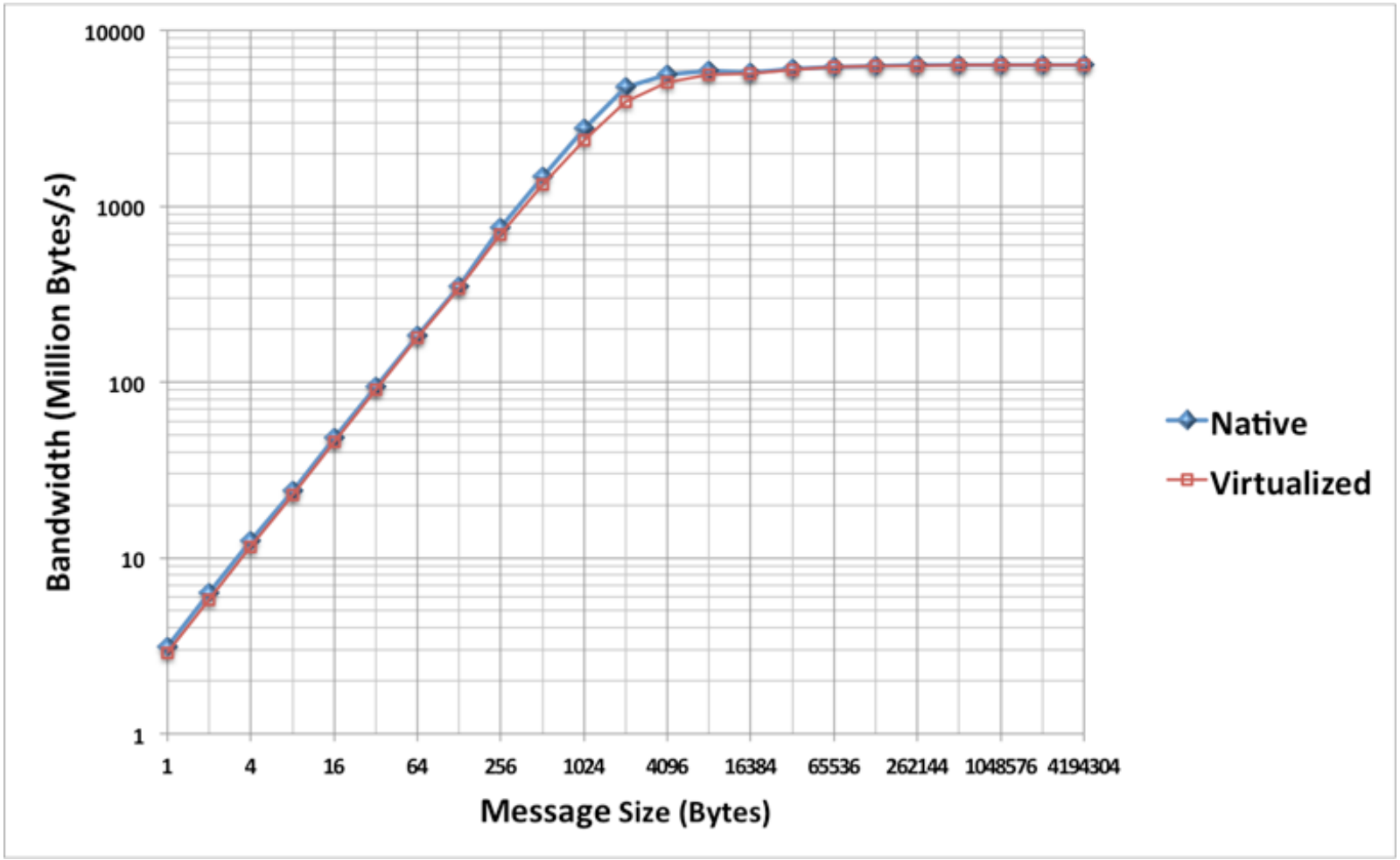


Figure 6. MPI point-to-point bandwidth measured by `osu_bw` for QDR InfiniBand. Included for scale are the analogous 10GbE measurements from Amazon (AWS) and non-virtualized 10GbE.

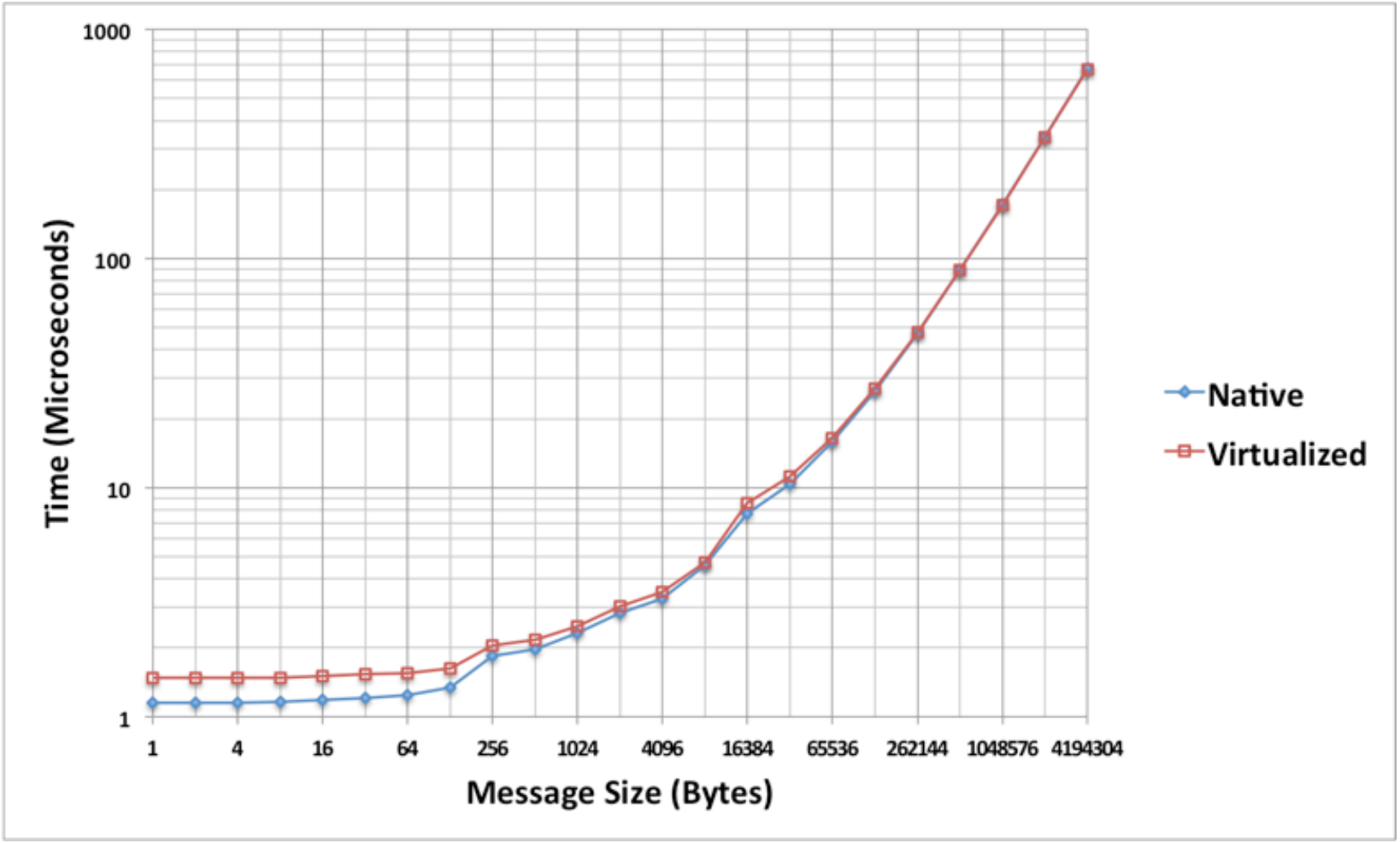
10x more bandwidth than Amazon EC2

OSU Microbenchmarks (3.9, osu_bw)

Comet: MPI bandwidth slowdown from SR-IOV is at most 1.21 for medium-sized messages & negligible for small & large ones



Comet: MPI latency slowdown from SR-IOV is at most 1.32 for small messages & negligible for large ones

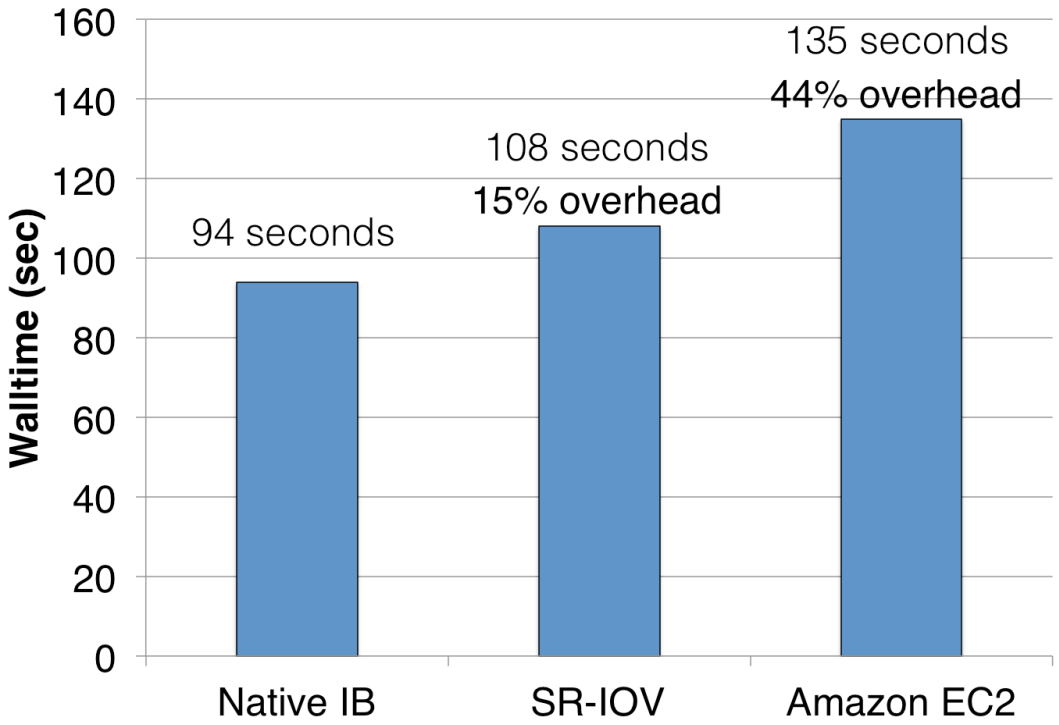


Application Benchmarks: WRF

- WRF CONUS-12km benchmark. The domain is 12 KM horizontal resolution on a 425 by 300 grid with 35 vertical levels, with a time step of 72 seconds.
- Run using six nodes (96 cores) over QDR4X InfiniBand virtualized with SR-IOV.
- SR-IOV test cluster has 2.2 GHz Intel(R) Xeon E5-2660 processors.
- Amazon instances were using 2.6 GHz Intel(R) Xeon E5-2670 processors.

Weather Modeling (WRF)

- 96-core (6-node) calculation
- Nearest-neighbor communication
- Scalable algorithms
- SR-IOV incurs modest (15%) performance hit
- …but still still 20% faster*** than Amazon
- Comet (new results): The overhead for this case is 2%!

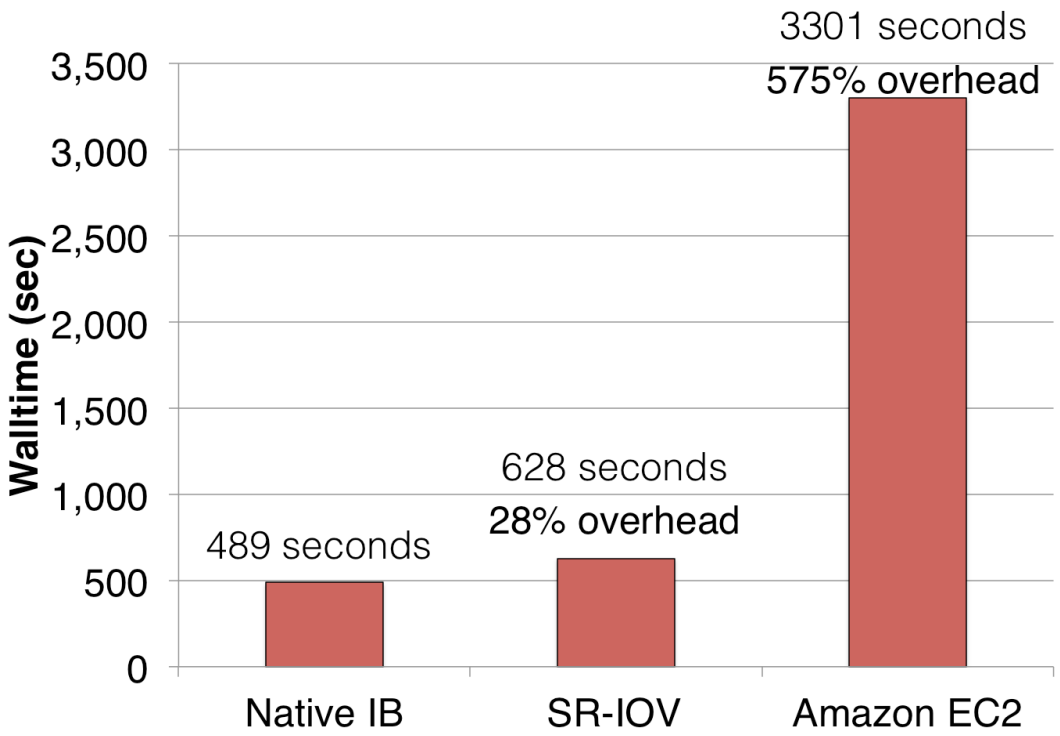


WRF 3.4.1 – 3hr forecast

*** 20% faster despite SR-IOV cluster having 20% slower CPUs

Quantum ESPRESSO: 5x Faster than EC2

- 48-core, 3 node calc
- CG matrix inversion (irregular comm.)
- 3D FFT matrix transposes (All-to-all communication)
- 28% slower w/ SR-IOV
- SR-IOV still > 500% faster*** than EC2
- Comet (new results): SR-IOV is only 8% slower than the native result!



Quantum Espresso 5.0.2 – DEISA AUSURF112 benchmark
*** 20% faster despite SR-IOV cluster having 20% slower CPUs

SR-IOV

- SR-IOV: huge step forward in high-performance virtualization
  - Shows **substantial improvement in latency** over Amazon EC2, and it has **negligible bandwidth overhead**
  - Benchmark application performance confirms this: significant improvement over EC2
- SR-IOV: lowers performance barrier to virtualizing the interconnect and **makes fully virtualized HPC clusters viable**
- Comet will deliver virtualized HPC to new/non-traditional communities that need flexibility without major loss of performance

Virtualization Support for Science Gateways

- **Comet has local VM hosting capability for gateways**
- **HPC Virtual Cluster environment**
  - Safely supports gateways such as the Neuroscience gateway where “compile your own code” is required
  - Supports gateways with their own complex software stacks

Getting Started

- **System Access – Logging in**
  - Linux/Mac – Use available ssh clients.
  - ssh clients for windows – Putty, Cygwin
    - http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - Login hosts for the SDSC Comet:
    - comet.sdsc.edu
- **For NSF Resources – Users can login via the XSEDE user portal:**
  - https://portal.xsede.org/

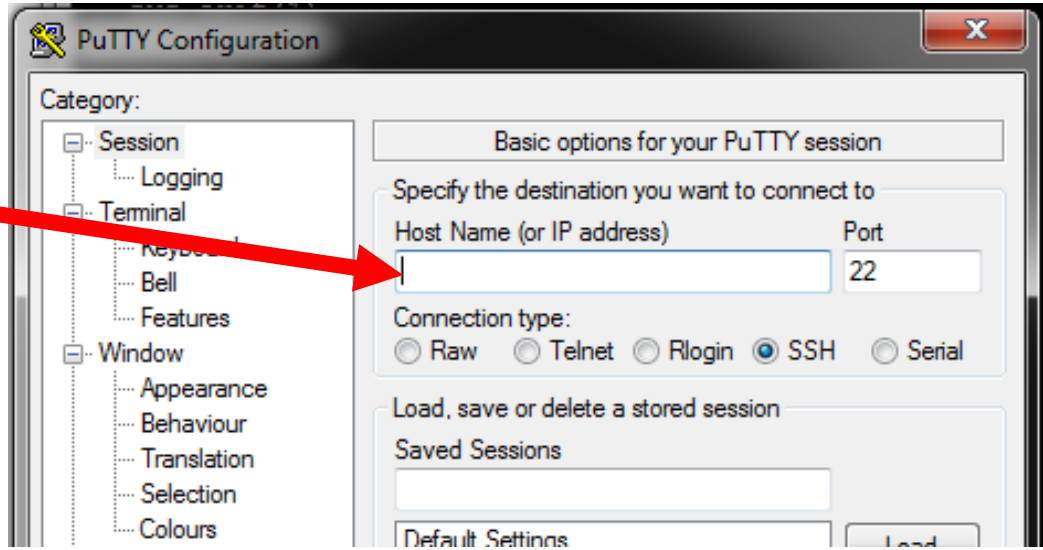
Logging into Comet

Mac/Linux:

```
ssh etrainXY@comet.sdsc.edu
```

Windows (PuTTY):

comet.sdsc.edu



Access Via Science Gateways (XSEDE)

- Community-developed set of tools, applications, and data that are integrated via a portal.
- Enables researchers of particular communities to use HPC resources through portals without the complication of getting familiar with the hardware and software details. Allows them to focus on the scientific goals.
- CIPRES gateway hosted by SDSC PIs enables large scale phylogenetic reconstructions using applications such as MrBayes, Raxml, and Garli. Enabled ~320 publications in 2013 and accounts for a significant fraction of the XSEDE users.
- NSG portal hosted by SDSC PIs enables HPC jobs for neuroscientists.

- Next: Hands On work with Comet

Example Locations

- **Today's examples are located in the training account home directories:**
  - /share/apps/examples/UCI2015

- **Following subdirectories should be visible:**

  ls /share/apps/examples/UCI2015

  CUDA HADOOP HYBRID LOCALSCRATCH MPI OpenACC OPENMP RDMA-Hadoop

Comet Compute Nodes

2-Socket (Total 24 cores) Intel Haswell Processors

Hands On Examples using:

(1) MPI

(2) OpenMP

(3) HYBRID

(4) Local Scratch (SSDs)

Comet – Compiling/Running Jobs

- **Copy and change to workshop directory:**

  cp -r /share/apps/examples/UCI2015 /home/$USER

  cd /home/$USER/UCI2015/MPI

- **Verify modules loaded:**

  module list

  Currently Loaded Modulefiles:

  1) gnutools/2.69 2) intel/2013_sp1.2.144 3) mvapich2_ib/2.1

- **Compile the MPI hello world code:**

  mpif90 -o hello_mpi hello_mpi.f90

- **Verify executable has been created:**

  ls -lt hello_mpi

  -rwxr-xr-x 1 mahidhar sdsc 721912 Mar 25 14:53 **hello_mpi**

Comet: Hello World on compute nodes

The submit script is hellompi-slurm.sb:

```
#!/bin/bash
#SBATCH --job-name="hellompi"
#SBATCH --output="hellompi.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.
#ibrun in verbose mode will give binding detail

ibrun -v ./hello_mpi
```

Comet: Hello World on compute nodes

IBRUN: Command is ../hello_mpi
IBRUN: Command is /share/apps/examples/MPI/hello_mpi
…
…
IBRUN: MPI binding policy: compact/core for 1 threads per rank (12 cores per socket)
IBRUN: Adding MV2_CPU_BINDING_LEVEL=core to the environment
IBRUN: Adding MV2_ENABLE_AFFINITY=1 to the environment
IBRUN: Adding MV2_DEFAULT_TIME_OUT=23 to the environment
IBRUN: Adding MV2_CPU_BINDING_POLICY=bunch to the environment
…
…
IBRUN: Added 8 new environment variables to the execution environment
IBRUN: Command string is [mpirun_rsh -np 48 -hostfile /tmp/rssSvauaJA -export /share/apps/examples/MPI/hello_mpi]
node 18 : Hello world
node 13 : Hello world
node 2 : Hello world
node 10 : Hello world

Compiling OpenMP Example

- **Change to the examples directory:**
  cd /home/$USER/UCI2015/OPENMP

- **Compile using –openmp flag:**
  ifort -o hello_openmp -openmp hello_openmp.f90

- **Verify executable was created:**

  [mahidhar@comet-08-11 OPENMP]$ **ls -lt hello_openmp**
  -rwxr-xr-x 1 mahidhar sdsc 750648 Mar 25 15:00 **hello_openmp**

OpenMP job script

```
#!/bin/bash
#SBATCH --job-name="hell_openmp"
#SBATCH --output="hello_openmp.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#SET the number of openmp threads
export OMP_NUM_THREADS=24

#Run the job using mpirun_rsh
./hello_openmp
```

Output from OpenMP Job

```
$ more hello_openmp.out
 HELLO FROM THREAD NUMBER =      7
 HELLO FROM THREAD NUMBER =      6
 HELLO FROM THREAD NUMBER =      9
 HELLO FROM THREAD NUMBER =      8
 HELLO FROM THREAD NUMBER =      5
 HELLO FROM THREAD NUMBER =      4
 HELLO FROM THREAD NUMBER =      0
 HELLO FROM THREAD NUMBER =     12
 HELLO FROM THREAD NUMBER =     14
 HELLO FROM THREAD NUMBER =      3
 HELLO FROM THREAD NUMBER =     13
 HELLO FROM THREAD NUMBER =     10
 HELLO FROM THREAD NUMBER =     11
 HELLO FROM THREAD NUMBER =      2
 HELLO FROM THREAD NUMBER =      1
 HELLO FROM THREAD NUMBER =     15
```

Running Hybrid (MPI + OpenMP) Jobs

- Several HPC codes use a hybrid MPI, OpenMP approach.
- "ibrun" wrapper developed to handle such hybrid use cases. Automatically senses the MPI build (mvapich2, openmpi) and binds tasks correctly.
- "ibrun –help" gives detailed usage info.
- hello_hybrid.c is a sample code, and hello_hybrid.cmd shows "ibrun" usage.

hello_hybrid.cmd

```
#!/bin/bash
#SBATCH --job-name="hellohybrid"
#SBATCH --output="hellohybrid.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

#This job runs with 2 nodes, 24 cores per node for a total of 48 cores.

# We use 8 MPI tasks and 6 OpenMP threads per MPI task
export OMP_NUM_THREADS=6
ibrun --npernode 4 ./hello_hybrid
```

Hybrid Code Output

```
[user@comet-08-11 HYBRID]$ more hellohybrid.435461.comet-17-41.out
Hello from thread 0 out of 6 from process 2 out of 8 on comet-17-41.local
Hello from thread 3 out of 6 from process 2 out of 8 on comet-17-41.local
Hello from thread 4 out of 6 from process 2 out of 8 on comet-17-41.local
Hello from thread 5 out of 6 from process 2 out of 8 on comet-17-41.local
Hello from thread 0 out of 6 from process 3 out of 8 on comet-17-41.local
Hello from thread 2 out of 6 from process 2 out of 8 on comet-17-41.local
Hello from thread 1 out of 6 from process 3 out of 8 on comet-17-41.local
Hello from thread 2 out of 6 from process 3 out of 8 on comet-17-41.local
…
…
Hello from thread 4 out of 6 from process 7 out of 8 on comet-17-42.local
Hello from thread 2 out of 6 from process 7 out of 8 on comet-17-42.local
Hello from thread 3 out of 6 from process 7 out of 8 on comet-17-42.local
Hello from thread 5 out of 6 from process 7 out of 8 on comet-17-42.local
Hello from thread 1 out of 6 from process 6 out of 8 on comet-17-42.local
```

Using SSD Scratch

```
#!/bin/bash
#SBATCH --job-name="localscratch"
#SBATCH --output="localscratch.%j.%N.out"
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH -t 01:30:00

export WKDIR=`pwd`
#Change to local scratch (SSD) and run IOR benchmark
cd /scratch/$USER/$SLURM_JOBID

#Run IO benchmark
ibrun -np 4 $WKDIR/IOR.exe -F -t 1m -b 4g -v –v > IOR_native_scratch.log
cp IOR_native_scratch.log $WKDIR/
```

Using SSD Scratch

- **Snapshot on the node during the run:**

```
$ pwd
/scratch/mahidhar/435463
$ ls -lt
total 22548292
-rw-r--r--  1 mahidhar hpss 5429526528 May 15 23:48 testFile.00000001
-rw-r--r--  1 mahidhar hpss 6330253312 May 15 23:48 testFile.00000003
-rw-r--r--  1 mahidhar hpss 5532286976 May 15 23:48 testFile.00000000
-rw-r--r--  1 mahidhar hpss 5794430976 May 15 23:48 testFile.00000002
-rw-r--r--  1 mahidhar hpss       1101 May 15 23:48 IOR_native_scratch.log
```

- **Performance from single node (in log file copied back):**
  - Max Write: 250.52 MiB/sec (262.69 MB/sec)
  - Max Read: 181.92 MiB/sec (190.76 MB/sec)

Comet GPU Nodes

2 NVIDIA K-80 Cards (4 GPUs total) per node.

Hands On Examples using:

(1) CUDA

(2) OpenACC

GPU Nodes – CUDA Example

```
#!/bin/bash
#SBATCH --job-name="CUDA"
#SBATCH --output="CUDA.%j.%N.out"
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=24
#SBATCH --export=ALL
#SBATCH --gres=gpu:4
#SBATCH -t 01:00:00

### Setup the environment:
module load cuda
#Run the job

export CUDA_VISIBLE_DEVICES=0,1
./simpleP2P
```

GPU Node – OpenACC Example

- **Setup PGI compiler:**

  module purge

  module load pgi

- **Compiling:**

```
pgcc -acc -Minfo=accel laplace2d.c
laplace:
     58, Generating copy(A[:][:])
         Generating create(Anew[:][:])
     63, Generating Tesla code
     64, Loop is parallelizable
     66, Loop is parallelizable
         Accelerator kernel generated
         64, #pragma acc loop gang /* blockIdx.y */
         66, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
         70, Max reduction generated for error
     74, Generating Tesla code
     75, Loop is parallelizable
     77, Loop is parallelizable
         Accelerator kernel generated
         75, #pragma acc loop gang /* blockIdx.y */
         77, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

GPU Nodes – OpenACC script

```
#!/bin/bash
#SBATCH --job-name="openacc"
#SBATCH --output="openacc.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --export=ALL
#SBATCH --gres=gpu:1
#SBATCH -t 01:00:00

./laplace_openacc
```

GPU Nodes – OpenACC Example

```
main()
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 15.914405 s
```

Thanks!
Questions: Email mahidhar@sdsc.edu