AMS 250: An Introduction to High Performance Computing

Hitchhiker's Guide to the Hyades Cluster

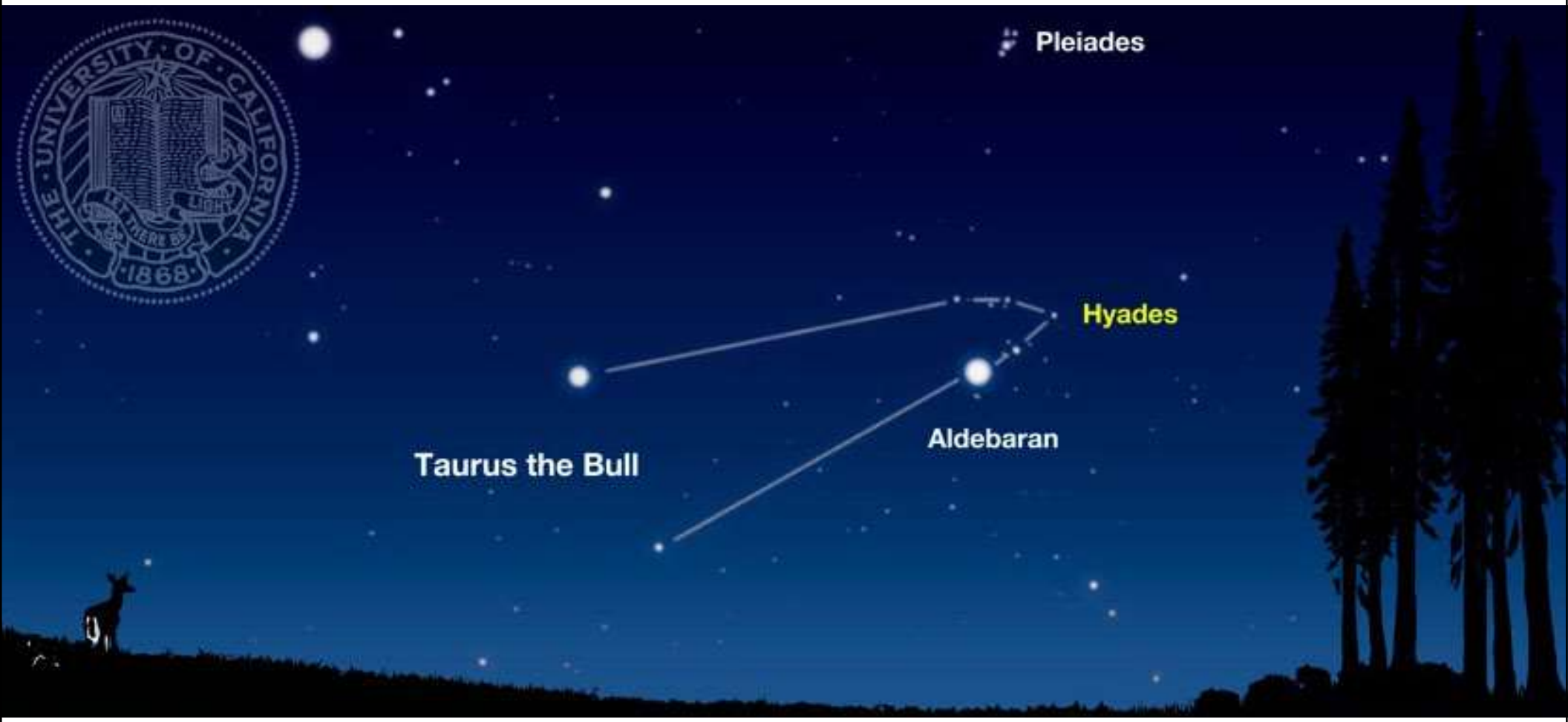
Shawfeng Dong

shaw@ucsc.edu

(831) 459-2725

Astronomy & Astrophysics

University of California, Santa Cruz


Pleiades
Hyades
Aldebaran
Taurus the Bull

Outline

- About Hyades
- Hardware Architecture
  - Nodes
  - Interconnects
  - Storage
- Accessing Hyades
- Computing Environment
- Compiling Codes
- Running Jobs
- Visualization and Analysis

About Hyades

- A cluster for Computational Astrophysics at UCSC
- Funded by a $1 million grant from the MRI (Major Research Instrumentation) program of NSF in 2012 (award # 1229745)
- Fully operational in January 2013
- 8 racks
- Power consumption is about 60 kW
- Hyades is the successor to Pleiades
  - Pleiades was funded by a $1 million grant from NSF-MRI in 2005 (award # AST-0521566)
- Pleiades was the successor to UpsAnd (Upsilon Andromedae), which was ranked as 163rd in Top500 (June 2002)

From Pleiades to Hyades

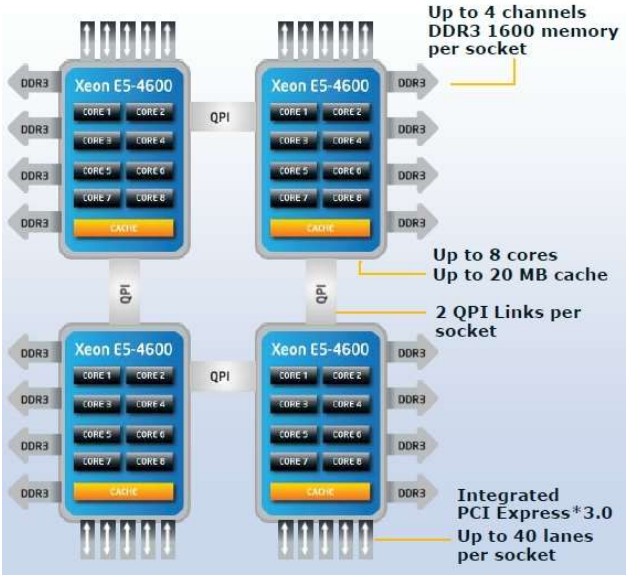
	PLEIADES	**HYADES**
Installation Year	2007	2013
# of Compute Nodes	207	188
# of Cores	828	3008
CPU microarchitecture	Intel Xeon *Woodcrest* (65nm) 2-core @ 2.33GHz	Intel Xeon *Sandy Bridge* (32nm) 8-core @ 2.0GHz
Total Memory	1.66 TB (2GB/core)	12TB (4GB/core)
Peak Performance	7.5 TFLOPS	48 TFLOPS, or 59 TFLOPS (w/ GPUs)
Scratch File System	Ibrix	Lustre
Total Scratch Space	55 TB	146 TB
Interconnects	1GbE Ethernet & 4x DDR InfiniBand	1GbE & 10GbE Ethernet & 4x QDR InfiniBand
	R_{max}=5.937 TFLOPS 255^{th} in Top500 (June 2007)	8 GPU nodes with Nvidia K20, 1 node with 2 Xeon Phi 5110P

Hardware Summary

Component	QTY	Description
Master Node	1	Dell PowerEdge R820, 4x 8-core Intel Xeon E5-4620, 128GB of memory, 8x 1TB HDD
Analysis Node	1	Dell PowerEdge R820, 4x 8-core Intel Xeon E5-4640, 512GB of memory, 2x 600GB SSD
Type I Compute Nodes	180	Dell PowerEdge R620, 2x 8-core Intel Xeon E5-2650, 64GB of memory, 1TB HDD
Type IIa Compute Nodes	8	Dell PowerEdge C8220x, 2x 8-core Intel Xeon E5-2650, 64GB of memory, 2x 500GB HDD, Nvidia K20 GPU
Type IIb Compute Nodes	1	Dell PowerEdge R720, 2x 6-core Intel Xeon E5-2630L, 64GB of memory, 500GB HDD, Xeon Phi 5110P
Lustre Scratch Storage	1	Dell/Intel solution, 146TB of usable storage.
Huawei UDS Storage	1	1PB private cloud storage
InfiniBand		Mellanox 4x QDR (40 Gb/s) non-blocking InfiniBand fabric

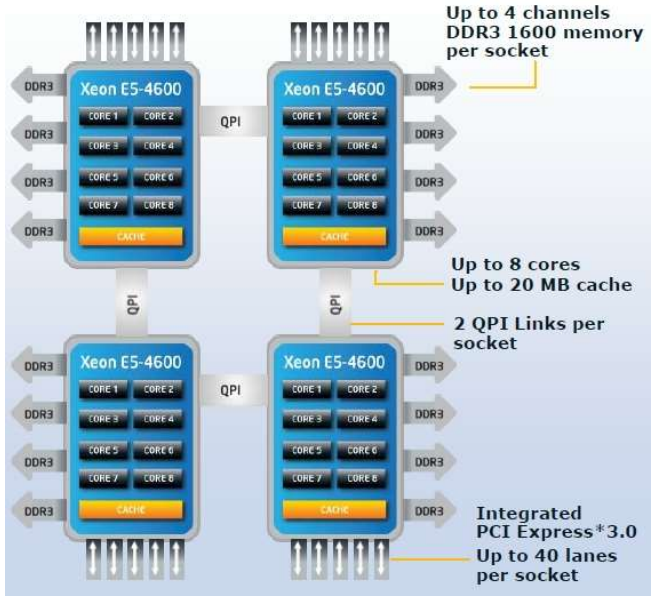
Master Node

- Master Node, aka Frontend or Login Node
  - provides plumbing for the cluster
- Dell PowerEdge R820 server, with
  - 4x Intel Xeon E5-4620 (8-core, 2.2GHz)
  - 128 GB memory
  - 8x 1TB HDD in RAID-6
  - Mellanox ConnectX-2 VPI QDR InfiniBand HCA
  - 2x Broadcom 10GbE interfaces
  - 2x Broadcom 1GbE interfaces
- Intended tasks:
  - compiling codes
  - debugging codes
  - submitting and monitoring jobs
  - light test runs
- Public hostname: hyades.ucsc.edu



Intel Xeon Processor E5-4620

- **Microarchitecture**: Sandy Bridge
- 8 cores / 16 threads
- **Process**: 32 nm
- **Frequency**: 2.2GHz / 2.6GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 16 MB *shared*
- AVX (Advanced Vector Extensions)
  256-bit, 8 double-precision FLOP per cycle (why?)
- 2 QPI links @ 7.2 GT/s (3.6GHz)
  7.2 x 2 x 20 x 64/80 / 8 = 28.8 GB/s
- 42.656 GB/s memory bandwidth @DDR3-1333
  1.333 (GT/s) x 8 (Byte) x 4 (channels) = 42.656 GB/s

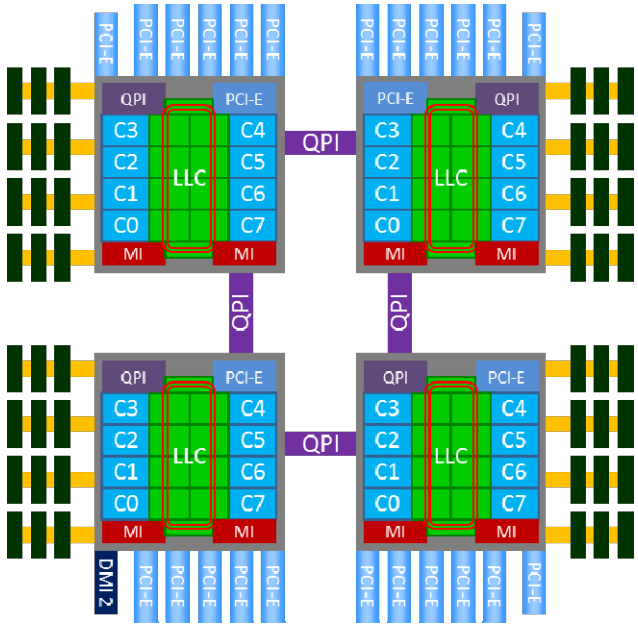


http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-4620.html

4-way Sandy Bridge System

- Intel QuickPath Interconnect (QPI)
  - point-to-point processor interconnect
  - two 20-lane links, one in each direction
  - unit of transfer is 80-bit “flit”, with 64 bits for data
  - bandwidth (Xeon E5-4620) =
    7.2 GT/s * 2 (links) * 20 (lanes) * 64/80 / 8 (bits/byte)
- NUMA (non-uniform memory access)
  - Local memory bandwidth: 42.7 GB/s (Xeon E5-4620)
  - Remote memory bandwidth: 28.8 GB/s (Xeon E5-4620)
- Cache Coherence
  - 256-bit/cycle ring bus interconnect between on-die cores
  - QPI between processors
  - MESIF protocol

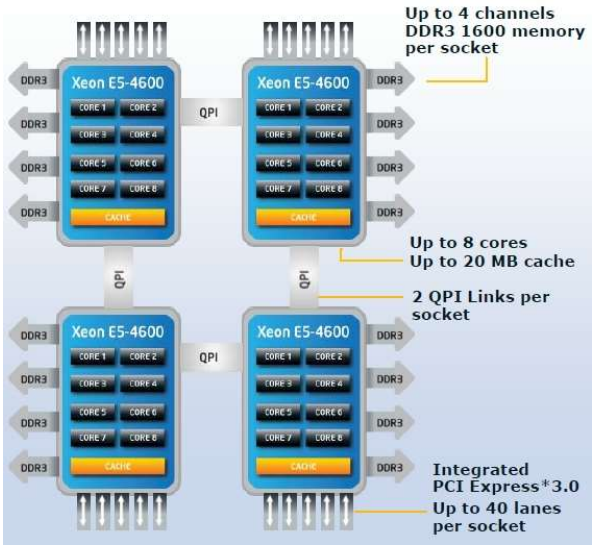
    5 states: Modified (M), Exclusive (E), Shared (S), Invalid (I) and Forward (F)



http://www.qdpma.com/systemarchitecture/systemarchitecture_sandybridge.html

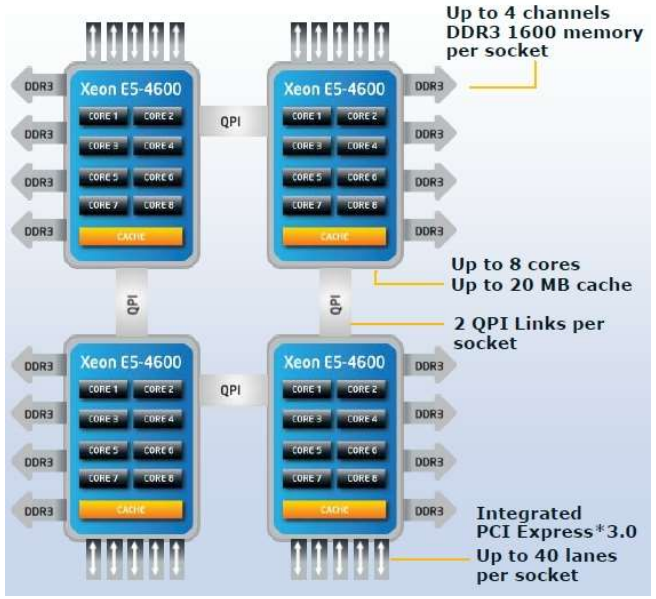
Analysis Node

- Dell PowerEdge R820 server, with
  - 4x Intel Xeon E5-4640 (8-core, 2.4GHz)
  - 512 GB memory (16GB/core)!
  - 2x 600GB SSD in RAID-0
  - Mellanox ConnectX-2 VPI QDR InfiniBand HCA
  - 2x Broadcom 10GbE interfaces
  - 2x Broadcom 1GbE interfaces
- 4-way ccNUMA system (4 NUMA nodes)
- Intended tasks:
  - visualization
  - data analysis
  - big memory jobs
- Public hostname: eudora.ucsc.edu



Intel Xeon Processor E5-4640

- **Microarchitecture**: Sandy Bridge
- 8 cores / 16 threads
- **Process**: 32 nm
- **Frequency**: 2.4GHz / 2.8GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 20 MB *shared*
- AVX (Advanced Vector Extensions)
  - 256-bit, 8 double-precision FLOP per cycle
- 2 QPI links @ 8.0 GT/s (4.0 GHz)
  - 8.0 x 2 x 20 x 64/80 / 8 = 32 GB/s
- 51.2 GB/s memory bandwidth @DDR3-1600
  - 1.6 (GT/s) x 8 (Byte) x 4 (channels) = 51.2 GB/s



http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-4640.html

Type I Compute Nodes

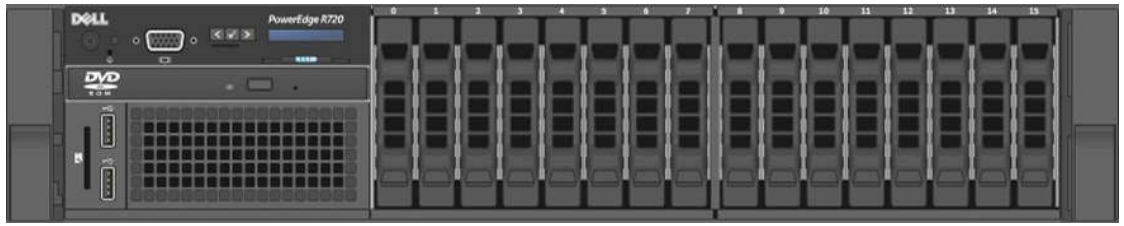
- 180 Dell PowerEdge R620 servers, each with
  - 2x Intel Xeon E5-2650 (8-core, 2.0GHz)
  - 64GB memory (4GB/core, 1.6GHz RDIMMs)
  - 1TB HDD
  - Mellanox ConnectX-2 VPI QDR (40 Gb/s) InfiniBand HCA
  - 2x Broadcom 1GbE interfaces

Type IIa Compute Nodes

- 8 Dell PowerEdge C8220x servers, each with
  - 2x Intel Xeon E5-2650 (8-core, 2.0GHz)
  - 64GB memory (4GB/core, 1.6GHz RDIMMs)
  - 2x 500GB HDD
  - Mellanox ConnectX-2 VPI QDR (40 Gb/s) InfiniBand HCA
  - 2x Intel 1GbE interfaces
  - Nvidia Tesla K20 GPU accelerator

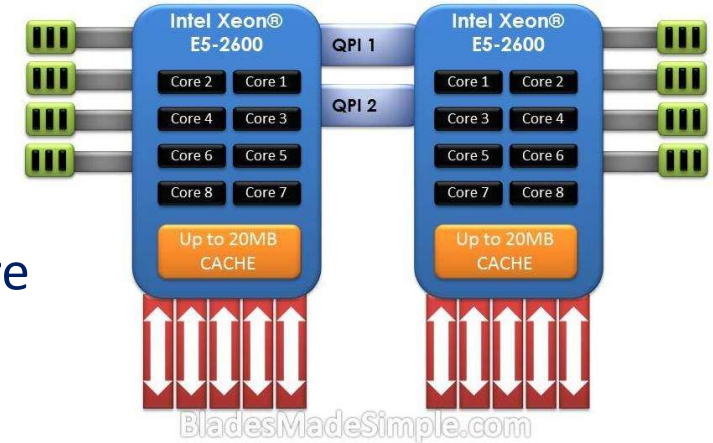
Type IIb Compute Node

- One Dell PowerEdge R720 server, with
  - 2x Intel Xeon E5-2630L (6-core, 2.0GHz)
  - 64GB memory (4GB/core, 1.6GHz RDIMMs)
  - 500GB HDD
  - Mellanox ConnectX-2 VPI QDR InfiniBand HCA
  - 2x Broadcom 10GbE interfaces
  - 2x Broadcom 1GbE interfaces
  - 2x Intel Xeon Phi 5110P coprocessors



Intel Xeon Processor E5-2650

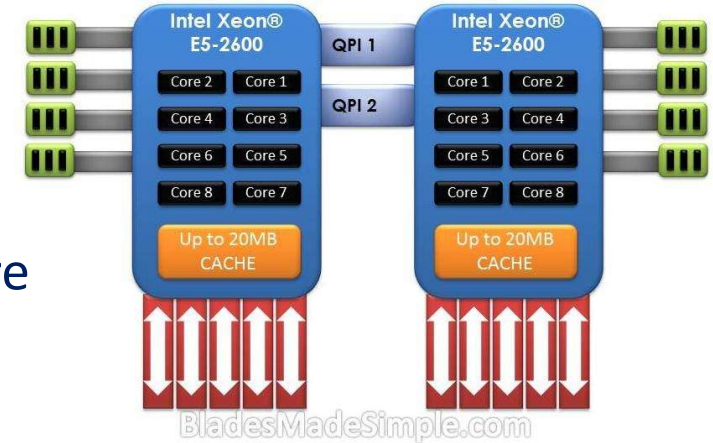
- **Microarchitecture**: Sandy Bridge
- 8 cores / 16 threads
- **Process**: 32 nm
- **Frequency**: 2.0GHz / 2.8GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 20 MB *shared*
- AVX (Advanced Vector Extensions)
  256-bit, **8** double-precision FLOP per cycle
- 2 QPI links @ 8.0 GT/s (4.0 GHz)
  8.0 x 2 x 20 x 64/80 / 8 = 32 GB/s
- 51.2 GB/s memory bandwidth @DDR3-1600
  1.6 (GT/s) x 8 (Byte) x 4 (channels) = 51.2 GB/s



http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-2650.html

Intel Xeon Processor E5-2630L

- **Microarchitecture**: Sandy Bridge
- 6 cores / 12 threads
- **Process**: 32 nm
- **Frequency**: 2.0GHz / 2.5GHz max Turbo
- **L1 cache**: 32 KB (code) + 32 KB (data) per core
- **L2 cache**: 256 KB per core
- **L3 cache**: 15 MB *shared*
- AVX (Advanced Vector Extensions)
  256-bit, **8** double-precision FLOP per cycle
- 2 QPI links @ 7.2 GT/s (3.6 GHz)
  7.2 x 2 x 20 x 64/80 / 8 = 28.8 GB/s
- 42.656 GB/s memory bandwidth @DDR3-1333
  1.333 (GT/s) x 8 (Byte) x 4 (channels) = 42.656 GB/s

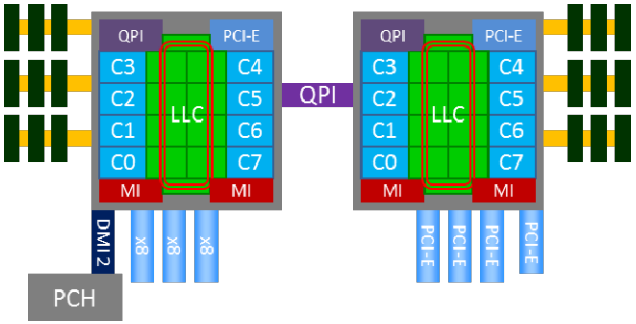


http://www.cpu-upgrade.com/CPUs/Intel/Xeon/E5-2630L.html

2-way Sandy Bridge System

- Intel QuickPath Interconnect (QPI)
  - point-to-point processor interconnect
  - two 20-lane links, one in each direction
  - unit of transfer is 80-bit “flit”, with 64 bits for data
  - bandwidth (Xeon E5-2650) =
    8.0 GT/s * 2 (links) * 20 (lanes) * 64/80 / 8 (bits/byte)
- NUMA (non-uniform memory access)
  - Local memory bandwidth: 51.2 GB/s (Xeon E5-2650)
  - Remote memory bandwidth: 32.0 GB/s (Xeon E5-2650)
- Cache Coherence
  - 256-bit/cycle ring bus interconnect between on-die cores
  - QPI between processors
  - MESIF protocol

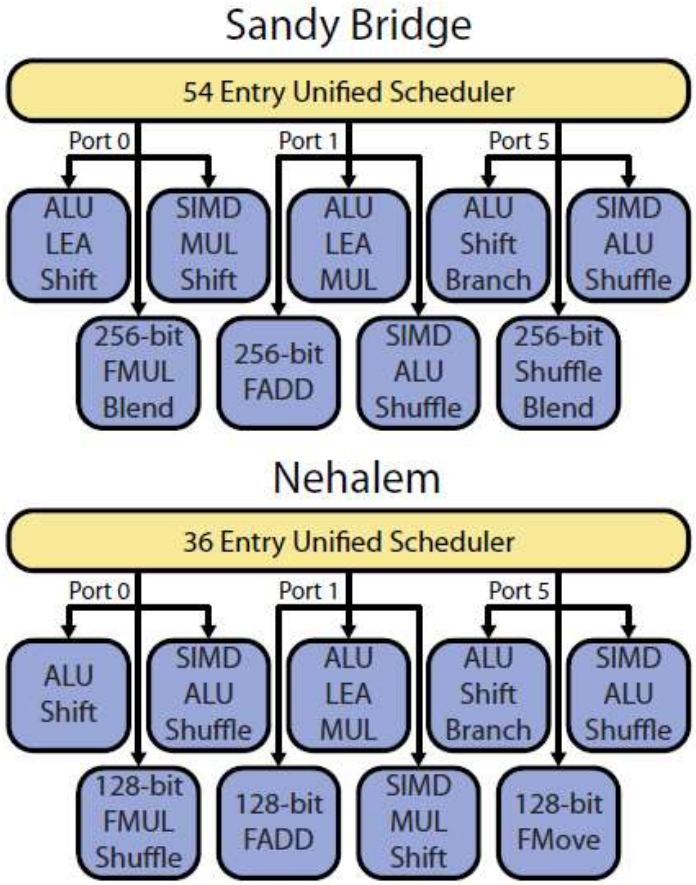
    5 states: Modified (M), Exclusive (E), Shared (S), Invalid (I) and Forward (F)



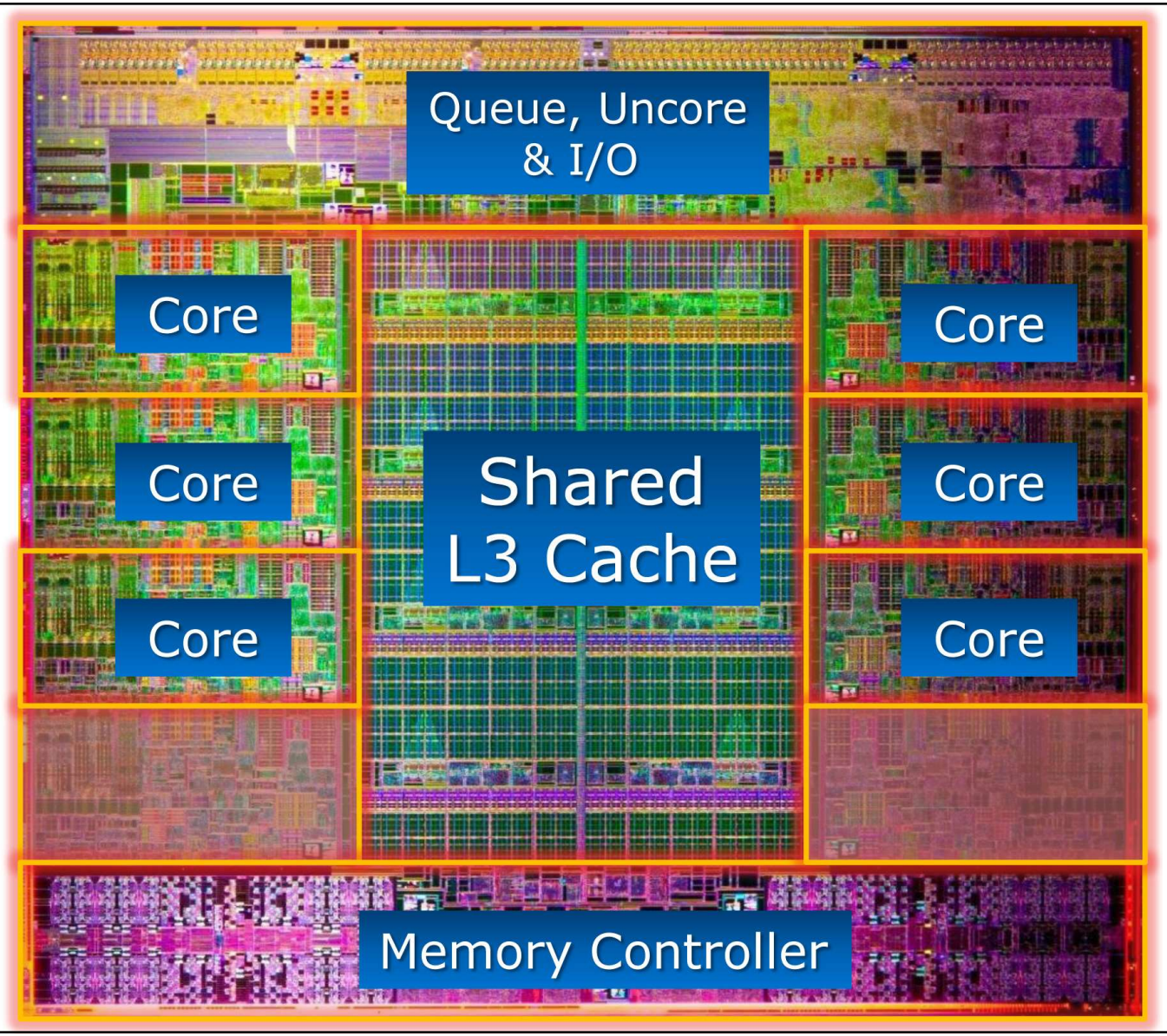
http://www.qdpma.com/systemarchitecture/systemarchitecture_sandybridge.html

Sandy Bridge Microarchitecture

- Instruction-Level Parallelism (ILP)
  - 3 integer ALU, 2 vector ALU and 2 AGU per core
  - 2 load/store operations per CPU cycle for each memory channel
  - 4 branch predictors
  - hyper-threading (2 logical core per physical core)
  - decoded micro-operation cache (uop cache)
  - 14- to 19- stage instruction pipeline, depending on the uop cache hit or miss
- Data Parallelism
  - 256-bit AVX (Advanced Vector Extensions)
  - can execute a 256-bit FP multiply, a 256-bit FP add and a 256-bit shuffle every cycle



http://www.realworldtech.com/sandy-bridge/6/



- 64-byte cache line size
- 32KB, 8-way associative L1 data cache per core
- 32KB, 4-way associative L1 instruction cache /core
- 256KB, 8-way associative L2 cache per core
- 20MB shared L3 cache

Memory Hierarchy

Xeon E5-2650	Latency (clocks)	Bandwidth	Size (bytes)
Registers	1	8192 GB/s	~100
L1 Cache	4	2048 GB/s	32 K (data) + 32K (instruction) per core
L2 Cache	11	744.7 GB/s	256 K per core
L3 Cache	25	327.7 GB/s	20 M shared
Local Memory	160	51.2 GB/s	32 GB
QPI	256	32.0 GB/s	
PCIe 2.0 x16	1024	8 GB/s	
InfiniBand QDR	2048	4 GB/s	
SSD	16384	~500 MB/s	600 GB
HDD	81920	~100 MB/s	1 TB

Performance Tips

- Hide latency
- Avoid data movement

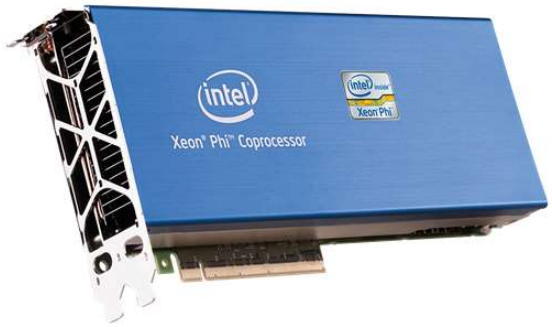
Nvdia Tesla K20 GPU

- **Microarchitecture**: Kepler
- 2496 CUDA cores / 832 DP units / 13 SMX
- Core speed: 705 MHz
- Double precision performance: 1.173 TFLOPS
  = 0.705 (GHz) x 832 (DP units) x 2 (FMA)
- Single precision performance: 4.577 TFLOPS
  = 0.705 (GHz) x 2496 (CUDA cores) x 2 (FMA)
- Memory: 5.2GHz, 320-bit wide, 5GB GDDR5
  5.2 (GHz) x 320 / 8 = 208 GB/s (51.2 GB/s for CPU)
- PCI express 2.0 x16
  500 (MB/s) x 8/10 x 16 = 8 GB/s (16 GB/s duplex)

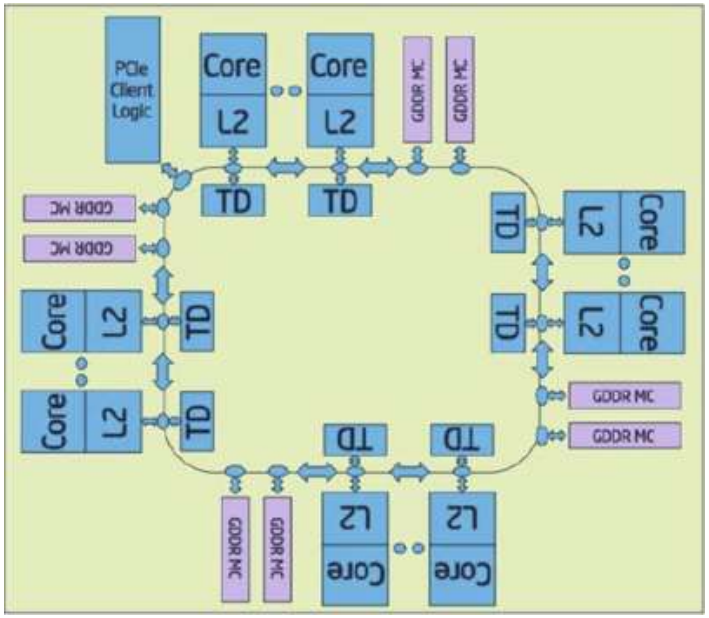

SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

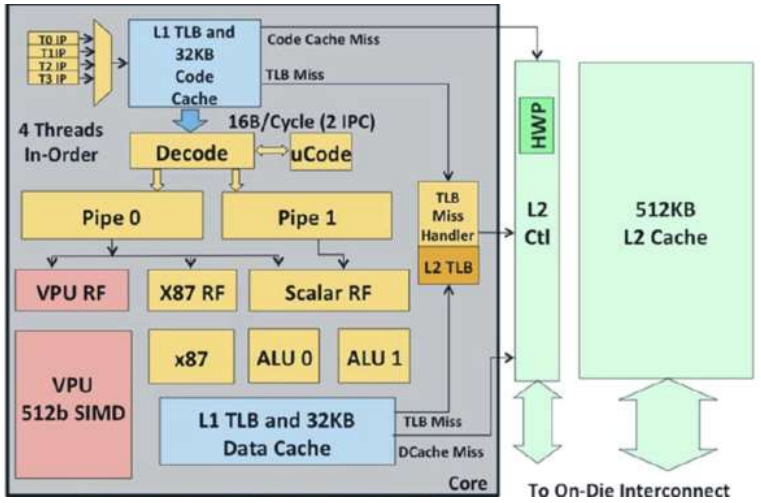
Intel Xeon Phi 5110P coprocessor

- **Microarchitecture**: Knights Corner
- 60 cores (in-order, dual-issue x86 design)
- 4 threads per core
- Core speed: 1.053 GHz
- 512-bit AVX
- Double precision performance: 1.01 TFLOPS
  = 1.053 (GHz) x 60 (cores) x 512/64 x 2
- Memory: 16GB GDDR5
  5 (GT/s) x 16 (channels) x 4 (B) = 320 GB/s
- PCI express 2.0 x16
  500 (MB/s) x 8/10 x 16 = 8 GB/s (16 GB/s duplex)



http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core





http://www.tomshardware.com/reviews/xeon-phi-larrabee-stampede-hpc,3342-3.html

Theoretical Peak Performance of Hyades

Peak performance of Intel Xeon E5-2650

= 8 (AVX) x 8 (cores) x 2.0 (GHz) = 128 GFLOPS

Peak Performance of all Intel Xeon E5-2650s

= 128 GFLOPS x 2 x 188 (nodes) = 48 TFLOPS

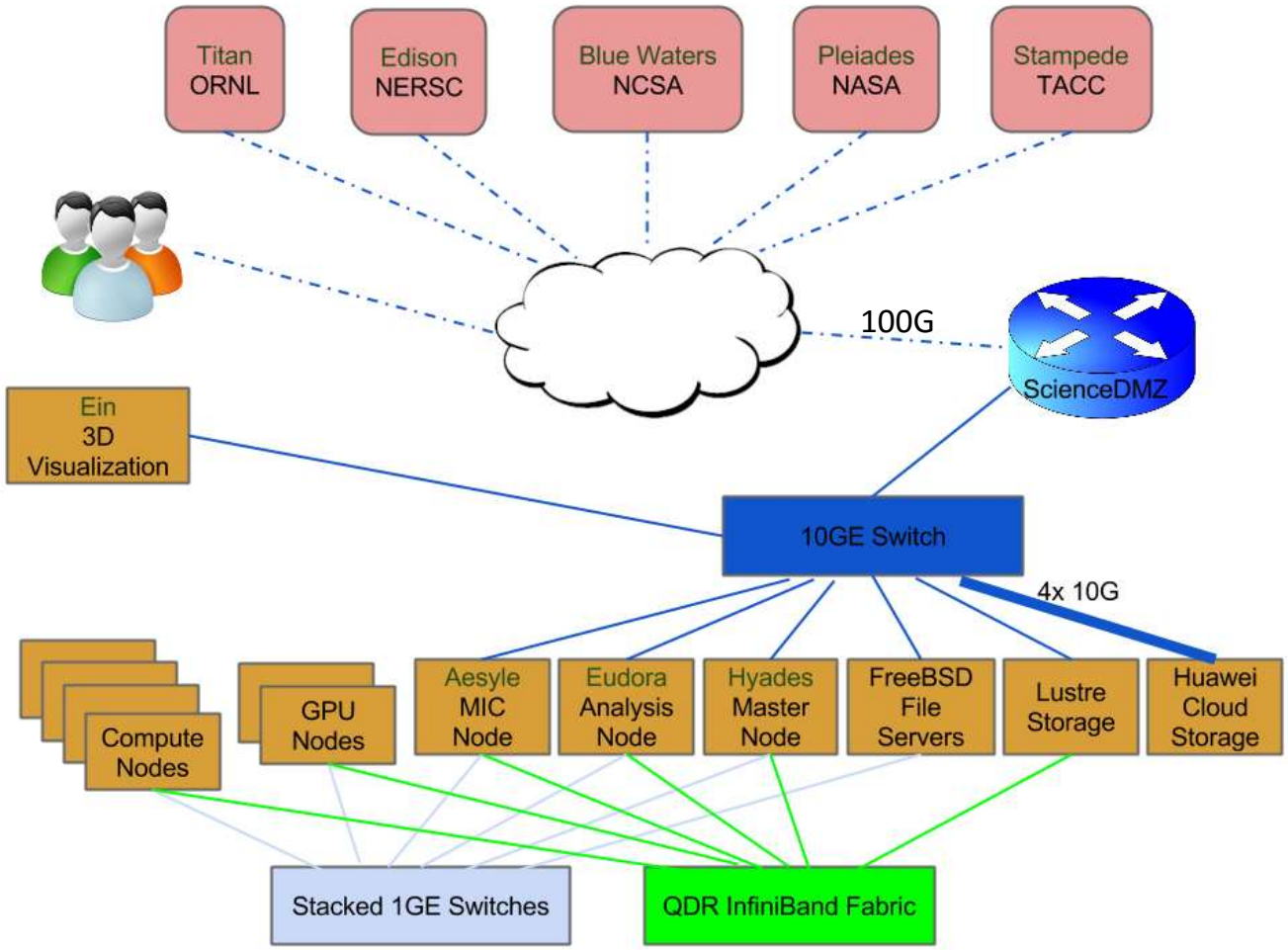
Peak Performance of Nvidia K20 = 1.17 TFLOPS

Peak Performance of Xeon Phi 5110P = 1.01 TFLOPS

Total Peak Performance of Hyades = 48 + 8x1.17 + 2x1.01

= 59.4 TFLOPS

Hyades Network Topology



Hyades Interconnects

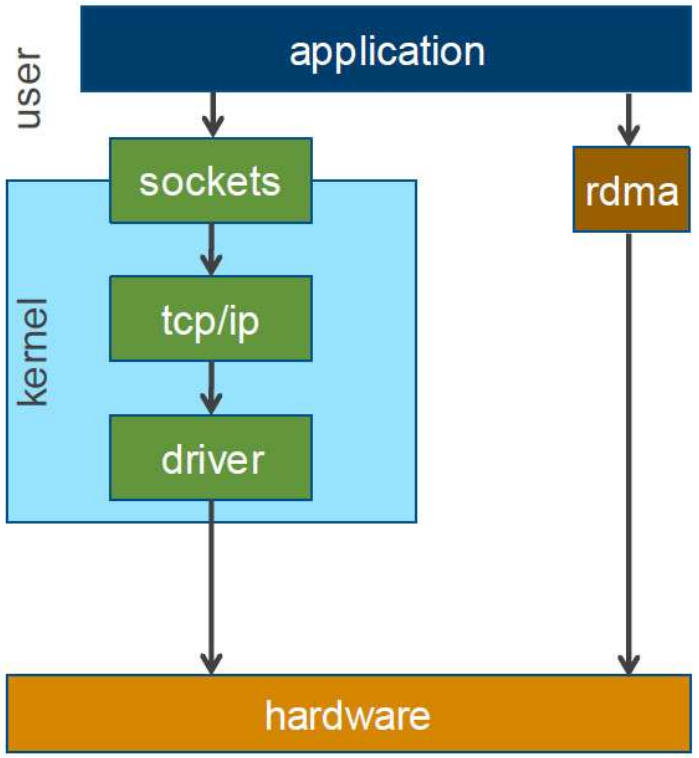
- Gigabit Ethernet Network
  - Backbone is made up of 7x Dell 6248 GbE switches, stacked in a Ring topology
  - All nodes are connected to the private GbE subnet (10.6.0.0/16)
  - Mostly used for management and NFS (Network File System) traffics
  - Not suitable for MPI communications

    Link bandwidth = 1 Gbps, Link latency ~ 0.15 millisecond
- 10-Gigabit Ethernet Network
  - a Dell 8132F 10GbE switch
  - Public 10GbE subnet (128.114.126.224/27)
  - Private 10GbE subnet (10.7.0.0/16), mostly serving NFS traffics

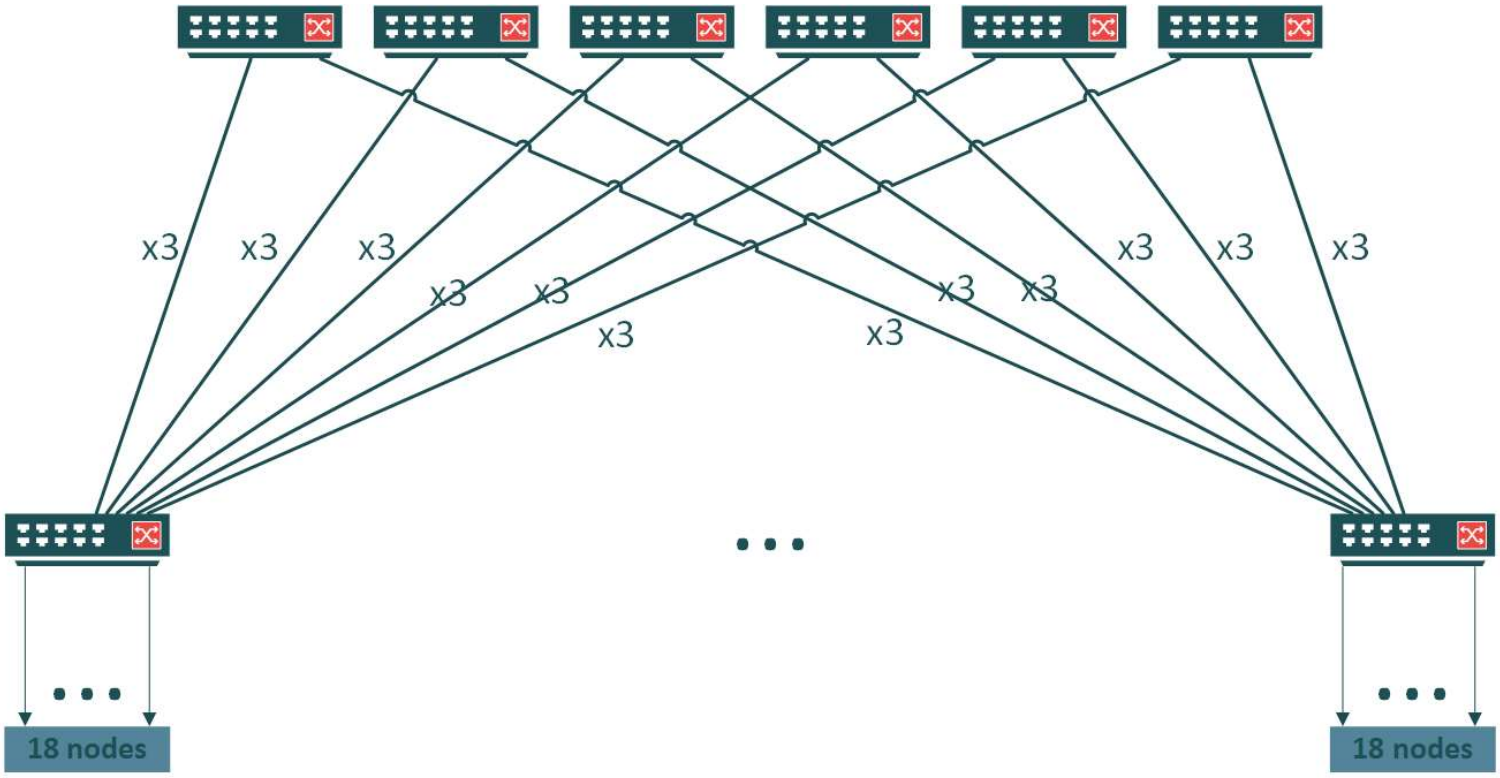
    Link bandwidth = 10 Gbps, Link latency ~ 0.1 millisecond
- InfiniBand fabric

InfiniBand (IB)

- RDMA – Remote Direct Memory Access
- Kernel bypass & low CPU overhead
- Mellanox ConnectX-2 VPI QDR 4x InfiniBand HCA (host channel adapter)
  - Link Bandwidth: 40Gbps signaling rate & 32Gbps data rate (8/10 encoding)
  - Link Latency: 1.3 microseconds
- Used for:
  - Message Passing
  - Lustre Network (LNET)
  - IPoIB (IP over InfiniBand) – subnet 10.8.0.0/16
- InfiniBand fabric for Hyades
  - 18x 36-port Mellanox IS5024 QDR IB switches
  - 1:1 non-blocking fat-tree topology



1:1 non-blocking fat-tree topology



6 core switches

12 core switches

216 nodes

bisection bandwidth = 108 x 40 Gb/s = 4320 Gb/s

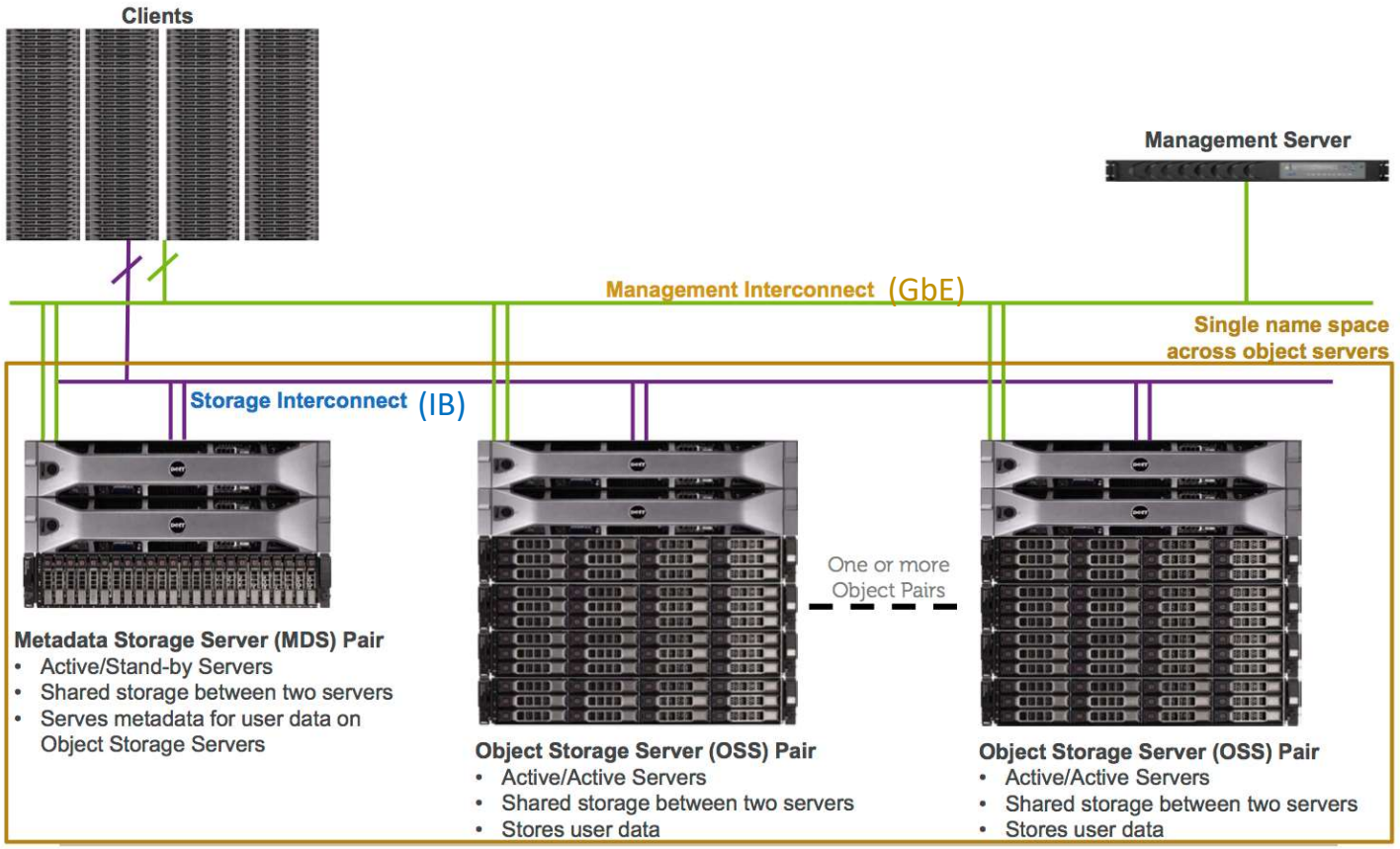
Storage

- Lustre Storage
  - Lustre is a high-performance parallel file system, used for large-scale cluster computing
  - 146 TB usable space, mounted at /pfs on all nodes
  - Intended as scratch space for running parallel simulations
- NFS Storage
  - Served by 2 SuperMicro storage nodes, running FreeBSD and ZFS
  - 36 TB usable space, mounted at /home, for storage of scripts and source codes
  - 78 TB usable space, mounted at /trove, for medium-term data storage
  - 188 TB usable space, mounted at /zang, for medium-term data storage
- Private Cloud Storage
  - 1PB Huawei Object Storage system, utilizing the Amazon S3 protocol
  - For long-term data archival and sharing
  - https://pleiades.ucsc.edu/hyades/S3

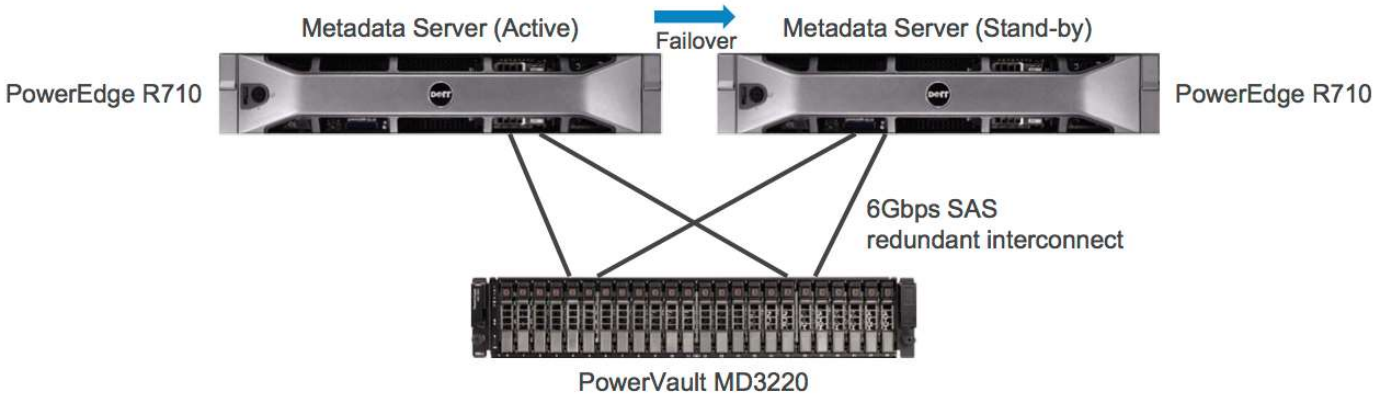
Lustre Storage

- 2 **Metadata Servers** (MDS), forming an active/passive highly available (HA) pair
  - The MDS pair shares a Dell MD3220 Storage Array as the **Metadata Target** (MDT);
  - MDT stores namespace metadata, such as filenames, directories, access permissions, and file layout.
- 4 **Object Storage Servers** (OSS), forming 2 active/active HA pairs
  - Each OSS pair shares 2x MD3220 Storage Arrays and 2x MD1220 Storage Arrays, divided into 8x **Object Storage Targets** (OSTs);
  - OSTs store file data;
  - The capacity of a Lustre file system is the total capacity provided by the OSTs.
- Lustre presents clients with a unified namespace, using standard POSIX semantics; and allows concurrent and coherent read and write access to the files in the filesystem.
- **Lustre Network** (LNET) layer can use several types of network interconnects, including native InfiniBand verbs, TCP/IP on Ethernet, etc.

Lustre Storage Cluster



Metadata Storage Servers (MDS)



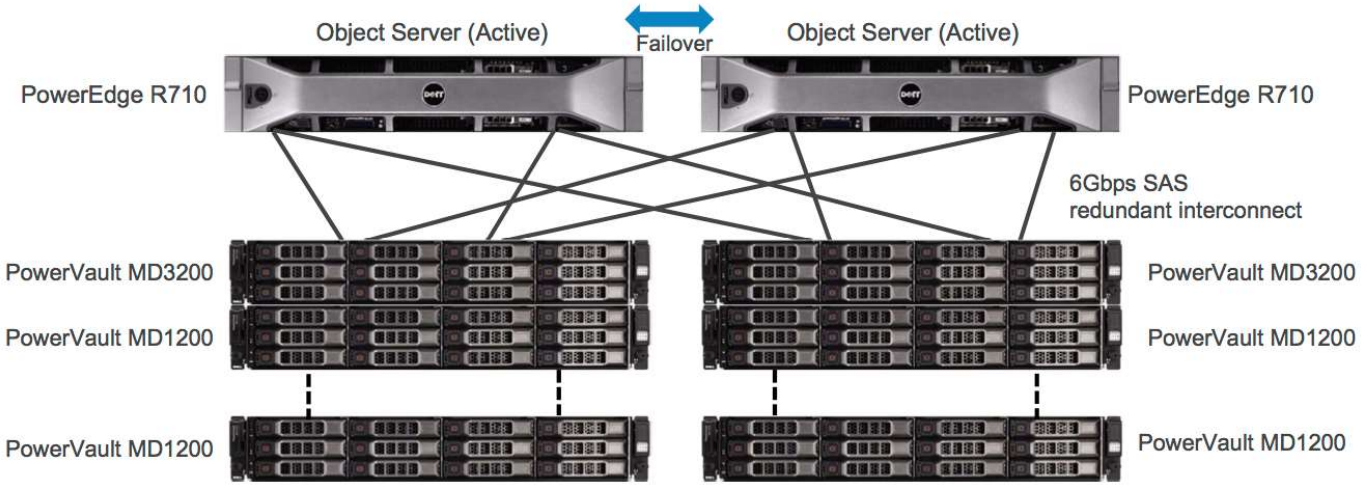
Metadata Storage Servers

- Two PowerEdge R710 servers
- Dual Intel Xeon 5620 Processors
- 48GB Memory
- Two 250GB 7.2K SATA boot drives configured in RAID 1
- 1 Dual port Dell 6Gbps SAS controller
- 1 Mellanox QDR InfiniBand HCA
- Dell iDRAC Enterprise

Metadata Storage

- One PowerVault MD3220
- 24 500GB 2.5" NL SAS Disks configured in RAID 10
- Dual active/active RAID controllers
- 2GB battery backed cache per controller
- High Performance Tier
- Redundant Management

Object Storage Servers (OSS)



Object Storage Servers

- Two PowerEdge R710 servers
- One Intel Xeon 5620 Processor
- 24GB Memory
- Two 250GB 7.2K SATA boot drives configured in RAID 1
- 2 Dual port Dell 6Gbps SAS controller
- 1 Mellanox QDR InfiniBand HCA
- Dell iDRAC Enterprise

Object Storage

- Two PowerVault MD3200s
- 12 2TB 3.5" NL SAS Disks configured in RAID 5 on each MD3200. Capacity expandable using MD1200s
- 48TB to 384TB per OSS pair
- Dual active/active RAID controllers
- 2GB battery backed cache per controller
- High Performance Tier
- Redundant Management

Accessing Hyades

Only SSH access is allowed!

Master Node (hyades.ucsc.edu):

ssh -l username hyades.ucsc.edu

ssh username@hyades.ucsc.edu

Analysis Node (eudora.ucsc.edu):

ssh -l username eudora.ucsc.edu

ssh username@eudora.ucsc.edu

SSH clients

- On Unix-like system, including OS X and Linux, OpenSSH server and client are commonly present
- On Windows
  - You can install a Unix-like environment, such as
    - Cygwin (https://www.cygwin.com/)
    - MinGW/MSYS (http://www.mingw.org/)
    - Git for Windows (https://git-for-windows.github.io/)
  - Or native SSH clients, such as
    - PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)
    - WinSCP (https://winscp.net/eng/download.php)

Using SSH Key for Authentication

Generate a pair of SSH keys on your client computer (Unix-like):

```
cd $HOME/.ssh
ssh-keygen -b 2048 -t rsa -f hyades
```

Once the public key (*hyades.pub*) is uploaded to Hyades (*~/.ssh/authorized_keys*), you can log in with your private key (*hyades*):

```
ssh -i ~/.ssh/hyades -l username hyades.ucsc.edu
```

Add the following to *$HOME/.ssh/config*

```
host h
  HostName hyades.ucsc.edu
  User username
  IdentityFile ~/.ssh/hyades
```

Then a lot of keystrokes will be saved:

```
ssh h
```

https://pleiades.ucsc.edu/hyades/SSH

Computing Environment

- User home directories
  - /home/*username* ($HOME)
  - 36TB total capacity
  - NFS mounted, intended for storing scripts and source codes
- User work directories
  - /pfs/username & symbolic link $HOME/pfs
  - 146TB total capacity
  - Lustre file system, intending as a scratch space for running simulation
- NFS storage
  - /trove, 78TB total capacity
  - /zang, 188 TB total capacity
  - For medium-term data storage

Modules Software Environment

We use the *Modules* utility to manage nearly all software on Hyades.

Advantages:

1. We can provide many different software and many different versions of the same software (including a default version as well as several older and newer versions)
2. Users can easily switch to different software or version without having to explicitly specify different paths.

The Modules utility consists of 2 parts:

1. The *module* command interface
2. The *modulefiles* on which *module* operates

Modulefile

- Written in TCL (Tool Command Language)
- Begins with the magic cookie *#%Module*
- Sets or adds environmental variables, like PATH, LD_LIBRARY_PATH, etc.
- Hides the notion of different type of shells (sh, bash, csh, tcsh, ksh, zsh, etc.)
- You can write your own, private modulefiles
- http://modules.sourceforge.net/man/modulefile.html

A Sample Modulefile

```
#%Module1.0

proc ModulesHelp { } {
    global version
    puts stderr "\n\tp_HDF5_impi_intel Version $version"
}

module-whatis   "Set up environment for Parallel HDF5 compiled
with Intel MPI"

# for Tcl script use only
set  version  1.8.10-patch1
set  root     /pfs/sw/parallel/impi_intel/hdf5-${version}

prepend-path  PATH          $root/bin
prepend-path  LIBRARY_PATH  $root/lib
prepend-path  INCLUDE       $root/include

setenv  I_MPI_EXTRA_FILESYSTEM       on
setenv  I_MPI_EXTRA_FILESYSTEM_LIST  lustre

conflict hdf5
```

Module Command Interface

To learn the usage of each sub-command:

```
module help
```

To list the loaded modules:

```
module list
```

To list all available modulefiles in the current MODULEPSATH:

```
module avail
```

To list all available modulefiles whose names start with, e.g., *python*:

```
module avail python
```

```
---------------------- /pfs/sw/modulefiles -----------------------
python/2.6.6           python/3.2.1          python/Anaconda3-2.1.0
python/2.7.3           python/3.4.1          python/pypy-2.4.0
python/2.7.8(default)  python/Anaconda-2.1.0 python/pypy3-2.3.1
```

Module Command Interface (cont'd)

To print the module-specific help information for, e.g., the module python/3.4.1:

```
module help python/3.4.1
```

To display the information about a module (full path of the modulefile and environment changes)

```
module show python/3.4.1
module display python/3.4.1
```

To load, e.g., the module python/2.7.8 into the shell environment:

```
module load python/2.7.8
```

Or simply (since it is the default):

```
module load python
```

To switch the loaded, e.g., module python/2.7.8 with, e.g., module python/Anaconda-2.1.0:

```
module swap python/2.7.8 python/Anaconda-2.1.0
module switch python/2.7.8 python/Anaconda-2.1.0
```

Module Command Interface (cont'd)

To remove a loaded module from the shell environment:

```
module rm python/Anaconda-2.1.0
module unload python/Anaconda-2.1.0
```

What the heck do you intend to do with this command?

```
module swap python python
```

To load you own private module (with absolute path):

```
module load $HOME/modulefiles/python-3.5.1
```

Further reading:
http://modules.sourceforge.net/man/module.html

Default Environment

- Two modules are loaded by default on Hyades:
  - intel_compilers/14.0.1
  - intel_mpi/4.1.3
- You can modify your environment so that certain modules are loaded whenever you log in. Put your changes in one of the following files, depending on your shell:
  - .cshrc or .tcshrc
  - .bashrc

Compilers

- Intel Compilers (default and recommended)
  - User and Reference Guide for the Intel C++ Compiler 15.0: https://software.intel.com/en-us/compiler_15.0_ug_c
  - User and Reference Guide for the Intel Fortran Compiler 15.0: https://software.intel.com/en-us/compiler_15.0_ug_f
- PGI Compilers
  - PGI documentation: https://www.pgroup.com/resources/docs.htm
  - PGI Compiler User's Guide: https://www.pgroup.com/doc/pgiug.pdf
  - PGI Compiler Reference Manual: https://www.pgroup.com/doc/pgiref.pdf
- GCC (GNU Compiler Collection)
  - GCC documentation: https://gcc.gnu.org/onlinedocs/
  - GNU Fortran Compiler: https://gcc.gnu.org/onlinedocs/gfortran/

Compilers (cont'd)

	Intel	**PGI**	**GNU**
C Compiler	icc	pgcc	gcc
C++ Compiler	icpc	pgCC pgcpp pgc++ (GNU-compatible)	g++
Fortran Compiler	ifort	pgfortran pgf77 (Fortan 77) pgf90 (Fortan 90/95) pgf95 (Fortan 90/95)	gfortran

1. The Intel and PGI C compilers are mostly object compatible (ABI compatible) with gcc, except for some OpenMP contructs;
2. C++ compilers are generally not ABI compatible; but Intel C++ is mostly compatible with g++;
3. Fortran compilers are generally not ABI compatible. Even different versions of gfortran may not be ABI compatible.

Stages of Compilation

We've all done this before:

```
$ gcc hello.c
$ ls
a.out  hello.c
$ ./a.out
Hello, world!
```

```c
#include <stdio.h>

int main()
{
  printf("Hello, world!\n");
  return 0;
}
```

What happens behind the scene:

```
$ gcc -save-temps hello.c
$ ls
a.out  hello.c hello.i  hello.o  hello.s
```

4 stages of compilation:

preprocessing -> compilation -> assembling -> linking

http://althing.cs.dartmouth.edu/local/www.acm.uiuc.edu/sigmil/RevEng/ch02.html

4 Stages of Compilation

1. **Preprocessing**, performing the following tasks:
   - Macro substitution
   - Stripping comments off
   - Expansion of the included files

   ```
   $ gcc -E hello.c -o hello.i
   ```

   or:

   ```
   $ cpp hello.c -o hello.i
   ```

2. **Compilation**, or *Parsing and Translation* stage, turning source code into assembly (*hello.s*):

   ```
   $ gcc -S hello.c
   ```

4 Stages of Compilation (cont'd)

3. **Assembling**, translating assembly code to object file (*hello.o*):

```
$ gcc -c hello.s
```

or:

```
$ as hello.s
```

We often combine stages:

```
$ gcc -c hello.c
$ gcc -c hello.i
```

4. **Linking**, which links against libraries and generates an executable file (*a.out* by default):

```
$ gcc hello.o
```

What it does behind the scene:

```
$ gcc -v hello.o
...
 /usr/libexec/gcc/x86_64-redhat-linux/4.4.7/collect2 ...
```

Quiz: which stages do those options belong to?

This is typically how we compile an MPI program:

```
icc –DNDEBUG -O2 -mp1 mpi_hostname.c -o mpi_hostname.x
-I/opt/intel/impi/4.1.3.045/intel64/include
-L/opt/intel/impi/4.1.3.045/intel64/lib
-Xlinker --enable-new-dtags -Xlinker -rpath
-Xlinker /opt/intel/impi/4.1.3.045/intel64/lib
-Xlinker -rpath -Xlinker /opt/intel/mpi-rt/4.1 -lmpigf
-lmpi -lmpigi -ldl -lrt -lpthread
```

Which stages do those options belong to?

- `–DNDEBUG -I/opt/intel/impi/4.1.3.045/intel64/include:` preprocessing
- `-O2 -mp1:` compilation
- `-L/opt/intel/impi/4.1.3.045/intel64/lib –Xlinker ...:` linking

Basic Levels of Optimization for Intel Compilers

Level	Description
-O0	Fast compilation, full debugging support; equivalent to -g
-O1 -O2	Low to moderate optimization, partial debugging support: • instruction rescheduling • copy propagation • software pipelining • common subexpression elimination • prefetching, loop transformations
-O3	Aggressive optimization - compile time/space intensive and/or marginal effectiveness; may change code semantics and results (sometimes even breaks code!): • enables -O2 • more aggressive prefetching, loop transformations

Some Important Options for Intel Compilers

Option	Description
-c	For compilation of source file only.
-O3	Aggressive optimization (-O2 is *default*).
-xHost	Generates instructions for the high-est instruction set available on the compilation host processor.
-xAVX	Optimizes for Intel processors that support AVX (Advanced Vector Extensions) instructions.
-g	Debugging information, generates symbol table.
-mp	Maintain floating point precision (disables some optimizations).
-mp1	Improve floating-point precision (speed impact is less than -mp).
-ip	Enable single-file interprocedural (IP) optimizations (within files).
-ip0	Enable multi-file IP optimizations (between files).
-prefetch	Enables data prefetching (requires -O3).
-openmp	Enable the parallelizer to generate multi-threaded code based on the OpenMP directives.

Running Jobs on Hyades

- **Torque** + **Maui** are the batch scheduler on Hyades.
- **Torque**, base on *PBS*, is the resource manager.
- Torque utilities
  - **qsub**, submitting job
  - **qstat**, monitoring status of jobs
  - **qdel**, terminating jobs prior to completion
- **Maui** is the job scheduler.
- Maui utilities
  - **showq**, listing both active and idle jobs
  - **showbf**, showing what resources are immediately available (backfill)
  - **checkjob**, viewing details of a job in the queue

Queues on Hyades

Queue	Total # of nodes	Resource per node	Max Walltime	qsub options
normal	150	16 cores (hyper-threading disabled)	2 days	-l nodes=*n*:ppn=16 -q normal
hyper	30	32 cores (hyper-threading enabled)	4 days	-l nodes=*n*:ppn=32 -q hyper
gpu	8	16 cores and 1 GPU	10 days	-l nodes=*n*:ppn=16 -q gpu

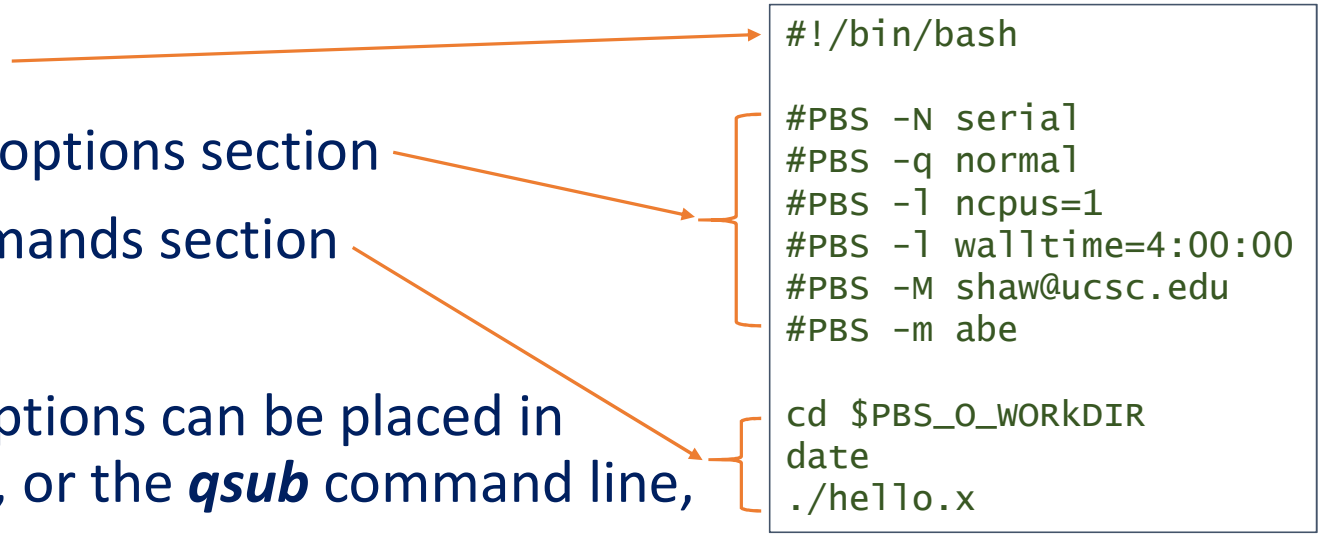
Notes:

1. *n* is the number of nodes you request for your job.
2. The default queue is **normal**. Your job will be submitted to the **normal** queue if no queue name is specified.
3. Run `showbf -p all` to find out what resources are immediately available.

Batch Scripts

Batch scripts, or job submission scripts, are the mechanism by which a user submits and configures a job for eventual execution. A batch script is simply a shell script which contains:

- The interpreter line
- The batch scheduler options section
- The executable commands section



The batch scheduler options can be placed in either the batch script, or the ***qsub*** command line, or both.

Run "`man qsub`" to learn the options.

Interactive Batch Job

You can start an interactive batch job by calling **qsub** with the "-I" option:

```
[dong@hyades dong]$ pwd
/pfs/dong
[dong@hyades dong]$ qsub -q gpu -I -l nodes=2:ppn=16,walltime=1:00:00
qsub: waiting for job 37955.hyades.ucsc.edu to start
qsub: job 37955.hyades.ucsc.edu ready

[dong@gpu-7 ~]$ pwd
/home/dong
[dong@gpu-7 ~]$ printenv | grep PBS
[dong@gpu-7 ~]$ echo $PBS_O_WORKDIR
/pfs/dong
[dong@gpu-7 ~]$ cat $PBS_NODEFILE
```

Sample Batch Script for Serial Jobs

Batch script *serial.pbs:*

```
#!/bin/bash

#PBS -N serial
#PBS -q normal
#PBS -l ncpus=1
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

cd $PBS_O_WORkDIR
./hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request only 1 core
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends

### go to the directory where you submit the job
### run your serial executable
```

To submit the job:

```
qsub serial.pbs
```

serial2.pbs:

```
#!/bin/bash

cd $PBS_O_WORkDIR
./hello.x
```

Which is equivalent to:

```
Qsub -N serial -q normal -l ncpus=1 \
-l walltime=4:00:00 -M shaw@ucsc.edu \
-m abe serial2.pbs
```

Sample script for embarrassingly parallel jobs

For example, there is a serial program (*jobarray_hello.x*) that takes an integer argument:

```
./jobarray_hello.x 23
Hello master, I am slave no. 23 running on hyades.ucsc.edu!
```

Let's assume that we need to run the following instances:

```
./jobarray_hello.x 101
./jobarray_hello.x 102
...
./jobarray_hello.x 164
```

Instead of submitting 64 serial jobs, we can submit only one job array:

```
qsub jobarray.pbs
```

Batch script *jobarray.pbs:*

```
#!/bin/bash

#PBS -N jobarray
#PBS -q normal
#PBS -l ncpus=1
#PBS -l walltime=4:00:00
#PBS -t 101-164
#PBS -M shaw@ucsc.edu
#PBS -m abe

cd $PBS_O_WORkDIR
./jobarray_hello.x $PBS_ARRAYID
```

Sample script for embarrassingly parallel jobs

Batch script *omp.pbs:*

```
#!/bin/bash

#PBS -N omp
#PBS -q normal
#PBS -l nodes=1:ppn=16
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

export OMP_NUM_THREADS=16
cd $PBS_O_WORkDIR
./omp_hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request 1 node (16 cores)
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends

### set the maximum no. of OpenMP threads to 16
### go to the directory where you submit the job
### run your OpenMP executable
```

To submit the job:

```
qsub omp.pbs
```

Sample Batch Script for OpenMP Jobs

Batch script *omp.pbs:*

```
#!/bin/bash

#PBS -N omp
#PBS -q normal
#PBS -l nodes=1:ppn=16
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe

export OMP_NUM_THREADS=16
cd $PBS_O_WORkDIR
./omp_hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request 1 node (16 cores)
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends

### set the maximum no. of OpenMP threads to 16
### go to the directory where you submit the job
### run your OpenMP executable
```

To submit the job:

```
qsub omp.pbs
```

Sample Batch Script for MPI Jobs

Batch script *impi.pbs:*

```
#!/bin/bash

#PBS -N impi
#PBS -q normal
#PBS -l nodes=4:ppn=16
#PBS -l walltime=4:00:00
#PBS -M shaw@ucsc.edu
#PBS -m abe
#PBS -j oe

cd $PBS_O_WORkDIR
mpirun -genv I_MPI_FABRICS shm:ofa -n 64 ./mpi_hello.x
```

Comments:

```
### your favorite shell

### job name
### job queue
### request 4 node (16 cores per node)
### and 4 hours walltime
### ask Torque to send emails
### when jobs aborts, starts and ends
### merge standard error with standard output

### go to the directory where you submit the job
### run your MPI executable
```

To submit the job:

```
qsub impi.pbs
```

Visualization & Analysis

Eudora is the dedicated Visualization & Analysis node.

You can use the same SSH private key (*hyades*) to log into *eudora*:

```
ssh -Y -i ~/.ssh/hyades -l username eudora.ucsc.edu
```

Add the following to *$HOME/.ssh/config*

```
host e
  HostName eudora.ucsc.edu
  User username
  IdentityFile ~/.ssh/hyades
  ForwardX11 yes
  ForwardX11Trusted yes
```

Then a lot of keystrokes will be saved:

```
ssh e
```

Visualization & Analysis Tools

- VisIt

```
module load visIt
```

- Yt

```
module load yt
```

- IDL

```
module load IDL
```

- Python (matplotlib, NumPy, SciPy, mpi4py, etc.)

```
module load python
```

Further Readings

1. Intel 64 and IA-32 Architectures Software Developer Manuals: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
2. Introduction to InfiniBand for End Users: http://www.mellanox.com/pdf/whitepapers/Intro_to_IB_for_End_Users.pdf
3. RDMA Aware Networks Programming User Manual: http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf
4. Quick Reference Guide to Optimization with Intel C++ and Fortran Compilers: https://software.intel.com/sites/default/files/managed/12/f1/Quick-Reference-Card-Intel-Compilers-v16.pdf