21st Summer School of PARALLEL COMPUTING

July 2 - 13, 2012 (Italian)
September 3 - 14, 2012 (English)

BLUE GENE/Q @ CINECA

Mirko Cestari – m.cestari@cineca.it
Gruppo Supercalcolo - HPCD

CINECA

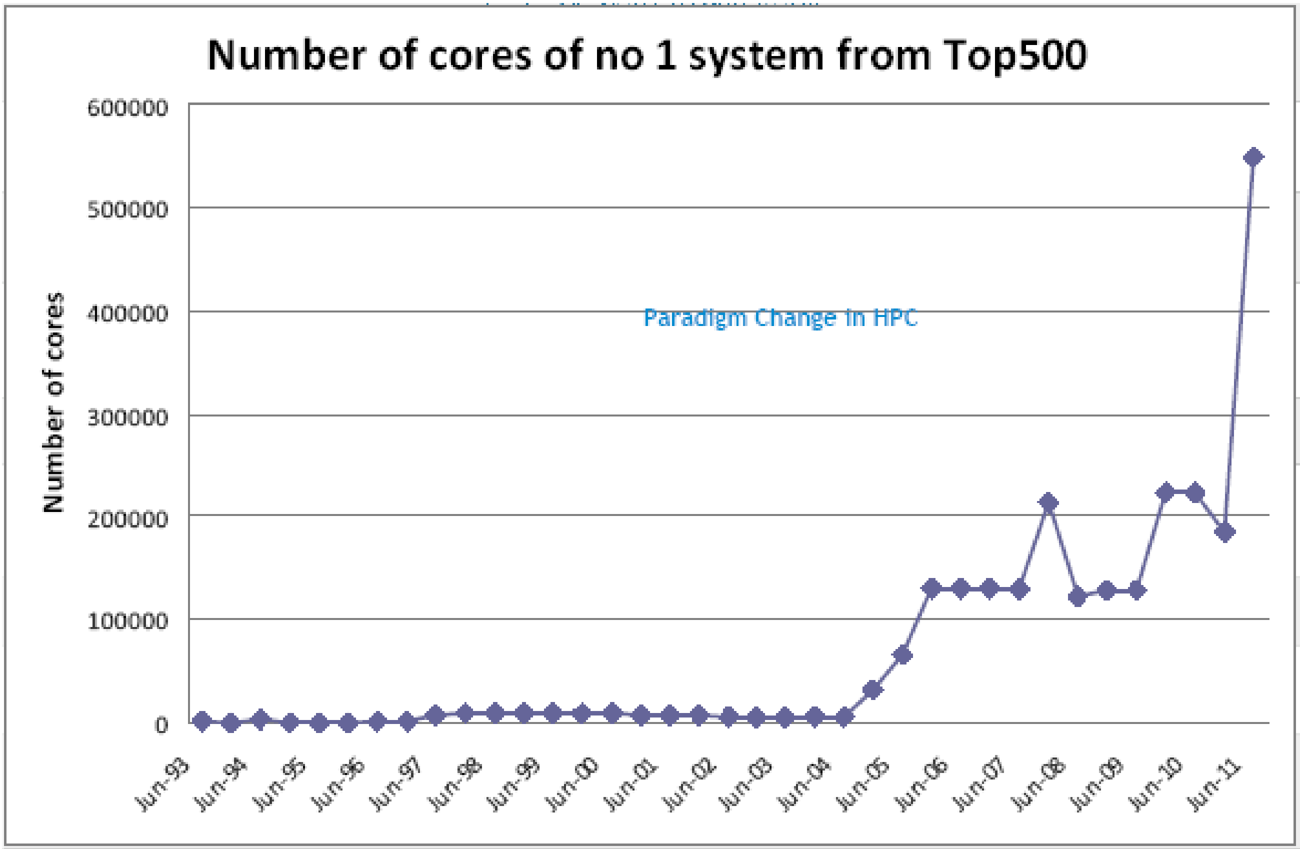
BlueGene: Outline

- **What is BG**
- **Overview of Blue Gene Q architecture and software**
  - Hardware components, networks and partitioning
  - Types of nodes and software stack
- **Developing applications for BGQ**
  - porting applications on BGQ
  - programming environment
- **Running and monitoring jobs on BGQ**
  - runjob command
  - environment variables
  - job script examples
  - LL commands
  - Available Debugging and Profiling Tools

Supercomputers:

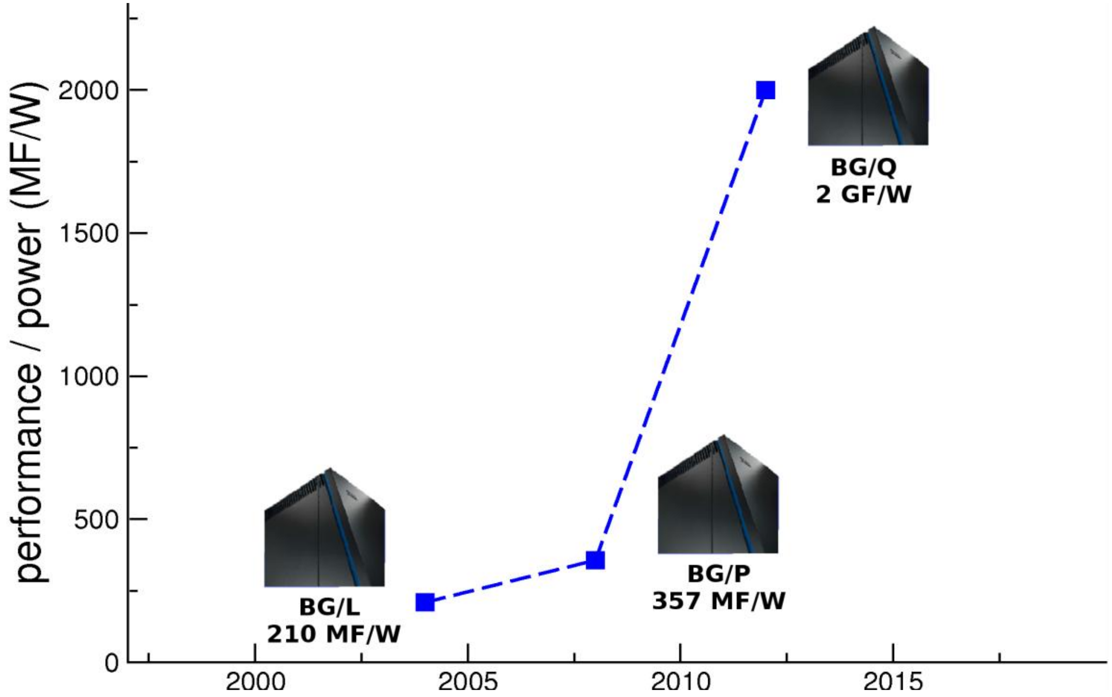
- **What are they**
  - (it’s like a pc… but faster! Well not so much…)
- **What are they used for**
  - you already know or can easily guess
- **What do you expect from them**
  - being powerful (fast?)
  - being reliable (a.k.a. oh no! all my simulations, oh no! all my data)
  - being easy to use and to exploit

Supercomputers: recent trends



http://top500.org/lists/2012/06

- BG is a massively parallel supercomputer
- It has different types of nodes and network interconnection types
- It is designed to have high energy-efficiency (performance/power)



BLUE GENE EVOLUTION

	Total		Biggest Config	Per rack		
	Performance [PF]	Efficiency [MF/W]	Max # of racks	Performance [TF]	Efficiency	# of cores
BG/L	0.596	210	104	5.7	2.02	2048
BG/P	1	357	72	13.9	4.96	4096
BG/Q	20	2000	96	209	20.83	16384

Towards higher and higher:

- Performance
- Efficiency
- Density of cores per rack

BGQ: among the most POWERFUL architectures

Rank	Site	Computer	Performance [PFlop/s]
1	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	16.32
2	RIKEN (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	10.51
3	DOE/SC/ANL United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	8.16
4	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	2.9
5	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	2.57

http://top500.org/lists/2012/06

BGQ: among the most POWERFUL architectures

Rank	Site	Computer	Performance
6	DOE/SC/Oak Ridge National Laboratory United States	**Jaguar** - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA 2090 Cray Inc.	1.94
7	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	1.72
8	Forschungszentrum Juelich (FZJ) Germany	**JuQUEEN** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	1.38
9	CEA/TGCC-GENCI France	**Curie thin nodes** - Bullx B510, Xeon E5-2680 8C 2.700GHz, Infiniband QDR Bull	1.36
10	National Supercomputing Centre in Shenzhen (NSCS) China	**Nebulae** - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 Dawning	1.27

http://top500.org/lists/2012/06

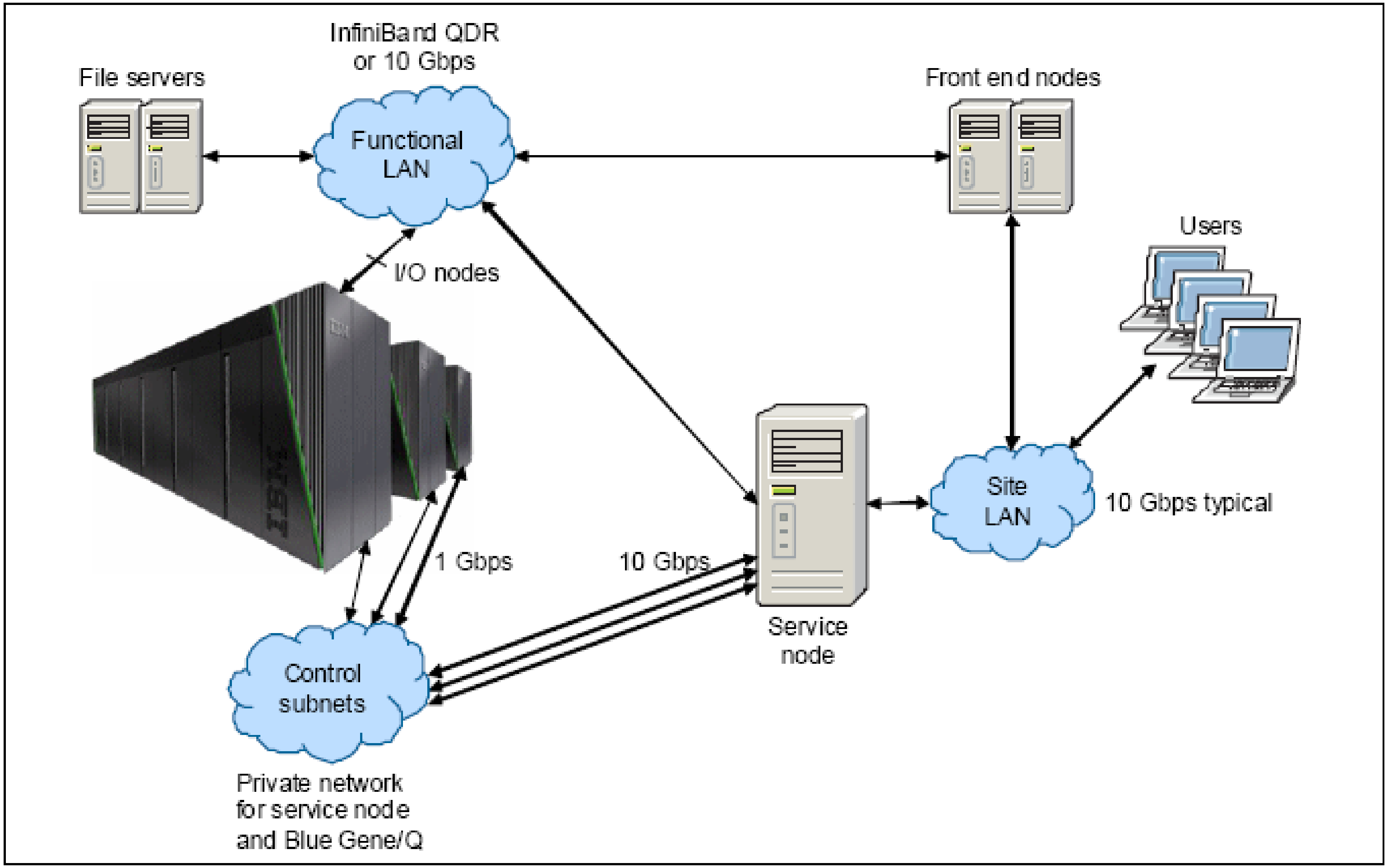
BGQ: currently the most EFFICIENT architecture

green500 "… to ensure that supercomputers are only simulating climate change and not creating climate change. "

Rank	MFLOPS/W	Site	Computer	Total Power (kW)
1	2100.88	LLNL	BG/Q	41.10
2	2100.88	IBM Watson	BG/Q	41.10
3	2100.86	ANL	BG/Q	82.20
4	2100.86	ANL	BG/Q	82.20
5	2100.86	RPI	BG/Q	82.20
6	2100.86	Un. Rochester	BG/Q	82.20
7	2100.86	IBM Watson	BG/Q	82.20
8	2099.56	Un. of Edinburgh	BG/Q	493.10
9	2099.50	Daresbury Lab.	BG/Q	575.30
10	2099.46	Julich	BG/Q	657.50
11	2099.39	CINECA	BG/Q	821.90

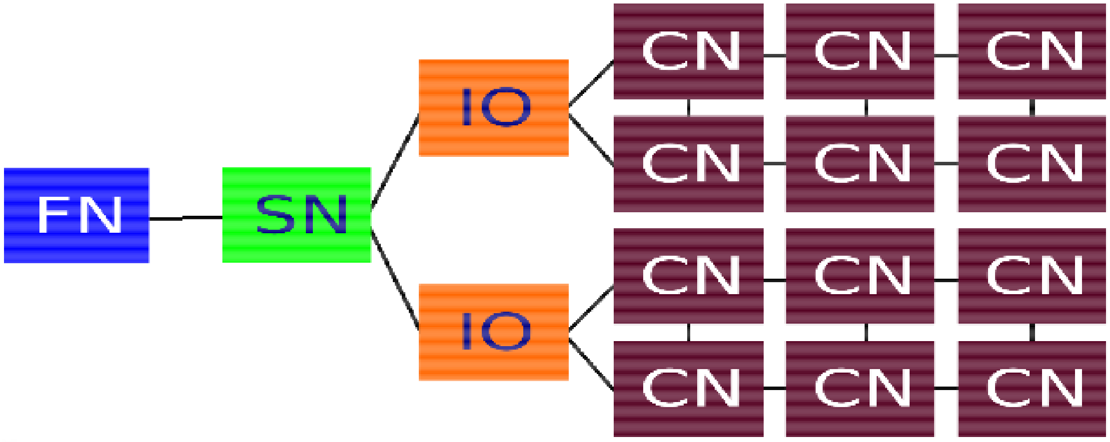
21st Summer School of PARALLEL COMPUTING

BGQ System architecture

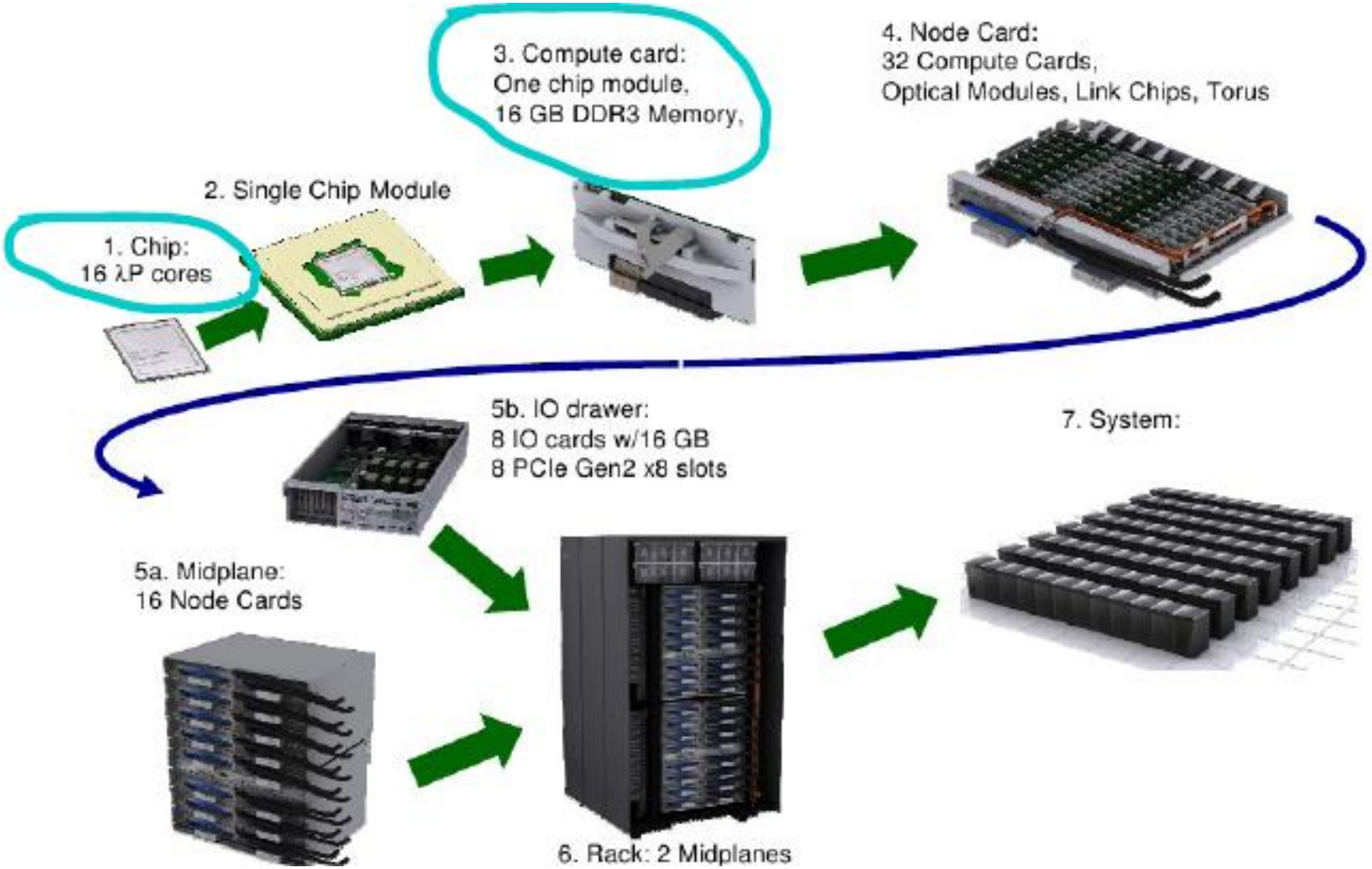


Blue Gene Blocks Hierarchical Organization

- **Front-end nodes (FN)**, dedicated for user's to login, compile programs, submit jobs, query job status, debug applications
- **Service nodes (SN)**, perform system management services, create and monitoring processes, initialize and monitor hardware, configure partitions, control jobs, store statistics
- **I/O nodes (IO)**, provide a number of OS services, such as files, sockets, process management, debugging
- **Compute nodes (CN)**, run user application, limited OS services



BGQ


1. Chip:
16 λP cores
2. Single Chip Module
3. Compute card:
One chip module,
16 GB DDR3 Memory,
4. Node Card:
32 Compute Cards,
Optical Modules, Link Chips, Torus
5b. IO drawer:
8 IO cards w/16 GB
8 PCIe Gen2 x8 slots
7. System:
5a. Midplane:
16 Node Cards
6. Rack: 2 Midplanes

Content of a BGQ Rack

- 2 **midplanes** form a rack
- Each midplane has 16 **Node Cards**
- Each Node Card has 32 **Compute Nodes**
  - 2x32x16=1024 compute nodes per rack
- Flexible I/O nodes / node cards ratio
  (at least 512 cores per I/O node)
  - In our configuration: 2 rack with 1024 core per I/O node
    8 rack with 2048 core per I/O node

BGQ system architecture

Why do I need to care?

- to exploit at best the computing resources
- to avoid costly mistakes
  - ➢Jobs crash (memory? wrong executable type?)
  - ➢Jobs don't start (wrong requests in the job script?)
  - ➢Jobs are slow (tasks overloading on a single node?)

• What do we need to care?

➔Front end (login) nodes **differ** from compute nodes

➔nodes are **diskless**; no local file system.

- ➢an I/O channel needs to be attached to the **partition** of the system where your job is running
- ➢**ratio** compute nodes / I/O links (64:1 and 128:1)
  - \- this set-up it's fixed and won't change

FERMI Configuration

10 racks BG/Q

- 10240 compute nodes
- 163840 **compute cores**

  Compute Card/Processor:
  - 1.6 GHz Power A2 (16 core) 4-way SMT
  - memory: 16 GB/core
- 96 **I/O nodes** (2x16+8x8)

8 **Front-end** (login) nodes

+ 1 **Service Node** (system monitoring, only for admin)

- File systems:

  1.7 PB on $CINECA_SCRATCH

  200 TB on $CINECA_DATA

  8.6 TB on $HOME

Per rack	
Peak performance	209 TF
Sustained *(Linpack)*	~170+ TF
Power efficiency*	~2.1 GF/W
*Without optical interconnect	

Access to Blue Gene Q

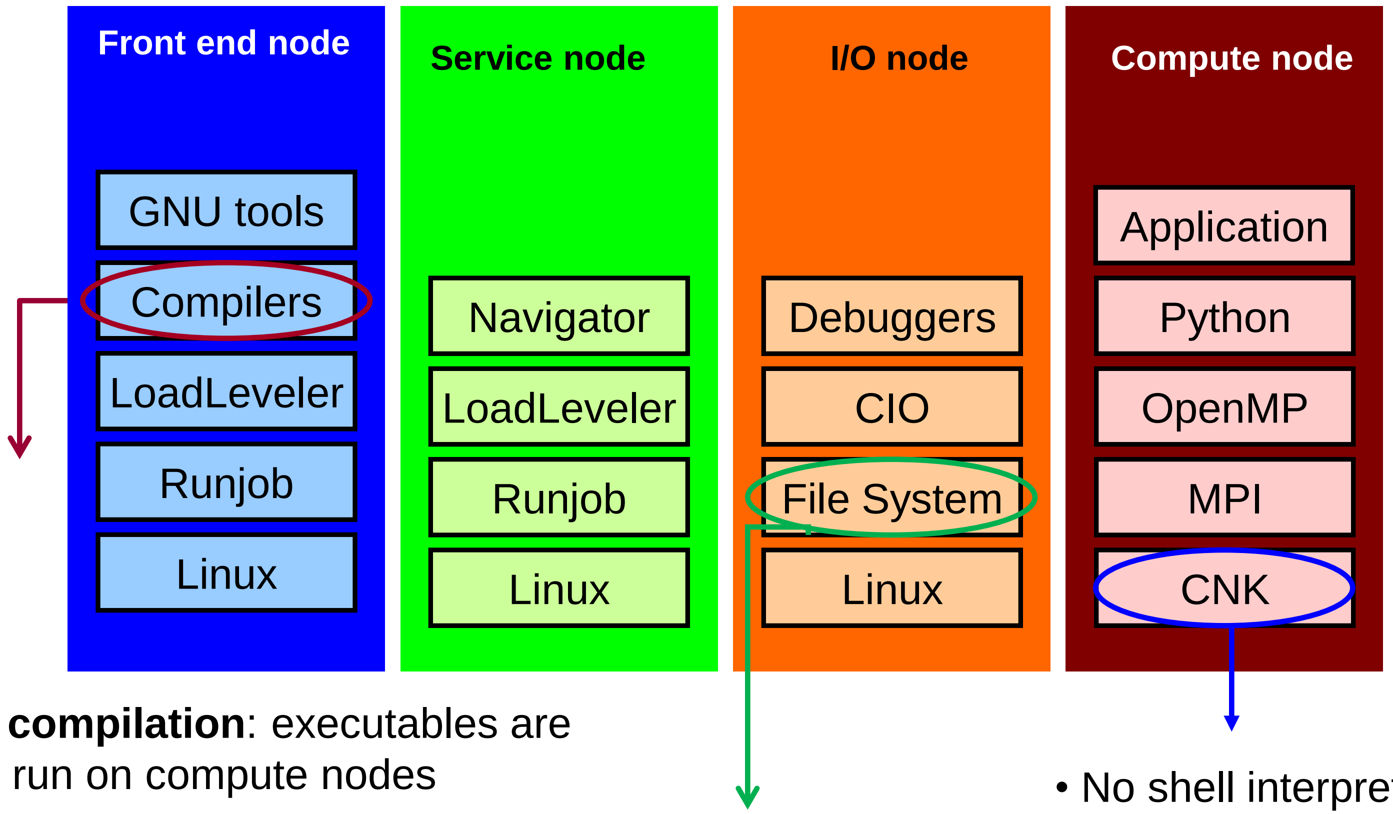
- Access through SSH connection with the hostname login.fermi.cineca.it

```
ssh <username>@login.fermi.cineca.it
```

Login on FEN nodes for:

- Compiling
- Job submission
- Debugging

Software stack: Overview



Cross compilation: executables are built to run on compute nodes

Not on compute nodes!

- No shell interpreter
- No fork/exec support

BGQ Networks

5D topology for point-to-point communication

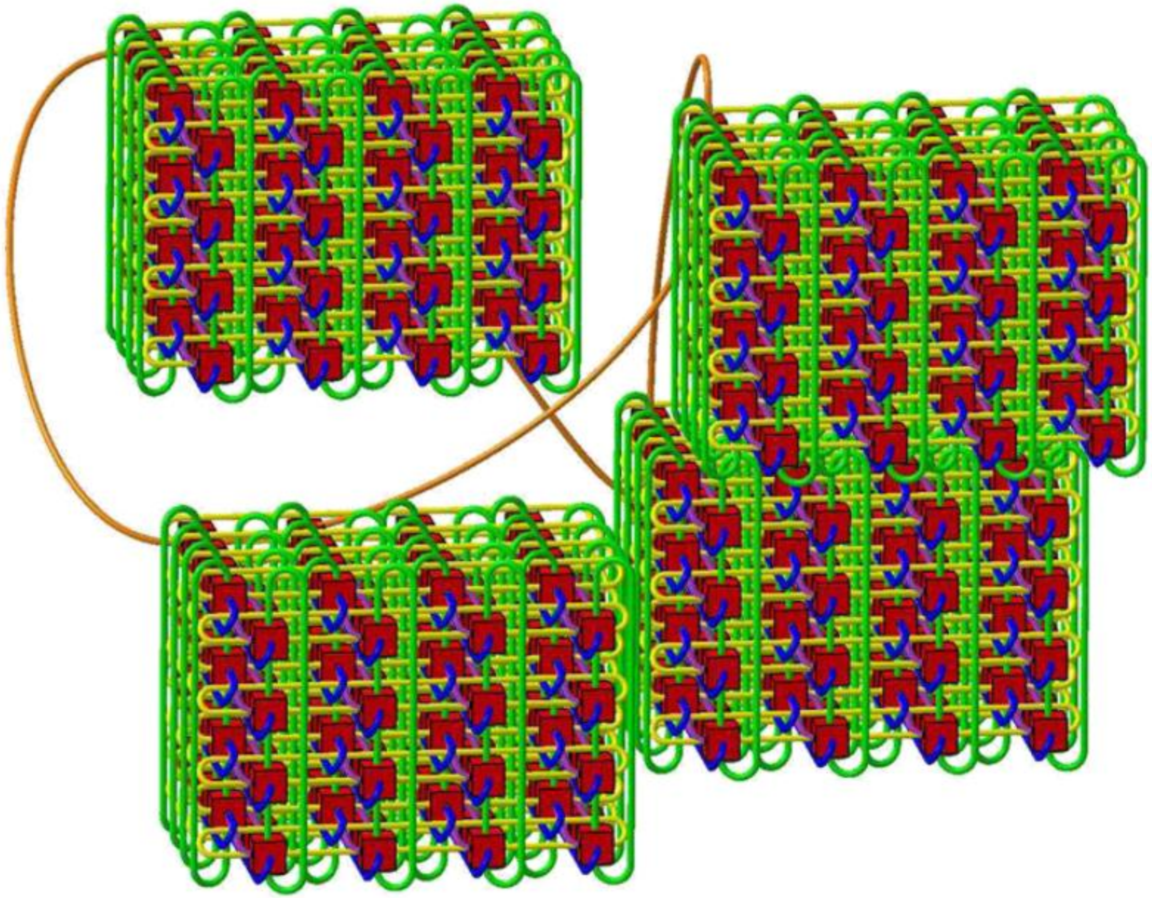
- 2 GB/s bidirectional bandwidth on all (10+1) links, 5D nearest neighbour exchange at 1.75 GB/s per link
- Collective and barrier networks embedded in 5-D torus network.
- 90+% of the peak performance

External, independent and dynamic I/O system

- I/O nodes in separate drawers/rack with private interconnections
- I/O network to/from Compute rack: 2 links (4 GB/s in 4 GB/s out)

21st Summer School of PARALLEL COMPUTING

5 D Torus Network



Further details

21st Summer School of PARALLEL COMPUTING

Outline

- What is BG
- Overview of Blue Gene Q architecture and software
  - Hardware components, networks and partitioning
  - Types of nodes and software stack
- Developing applications for BGQ
  - porting applications on BGQ
  - programming environment
- Running and monitoring jobs on BGQ
  - runjob command
  - environment variables
  - job script examples
  - LL commands
  - Available Debugging and Profiling Tools

Porting applications on BGQ

1. Evaluation of the effort required
2. Enabling of the code on the system

Is my application “suitable” to BGQ architecture?

a) Does the code use the Message Passing Interface **MPI**? (OpenMP is supported only on individual nodes).

b) Is the **memory requirement** per MPI task (less than a) 1 GB (pure MPI) or 16 GB (MPI+OpenMP)? Is it possible to exploit SMT technology?

c) Is the code **computationally intensive**? That is, is there a small amount of I/O compared to computation? Is the code **floating-point intensive**?

d) Does the algorithm allow for distributing the work on a large **number of cores**? How does it scale up to with thousands of threads/tasks?

If I need to change the code, is it a worth effort?

- future of HPC seems to involve systems with many many cores
  - BG/X
  - Clusters with GPUs
  - Clusters with MICs (Intel Multiprocessors card)

In order to exploit new-coming HPC systems:

- be prepared to handle hundreds of tasks or threads. Possibly both of them
  - hybrid parallelism
- be prepared to carefully manage RAM memory
  - Expensive resource in terms of power consumption
  - it won't increase easily

Programming Environment: Compilers

Two Different compilers family for both front-end and back-end nodes

- **IBM Compilers**
- **GNU Compilers**

	Back-end Compilers		Front-end Compilers	
	XL family	GNU family	XL family	GNU family
C	bgxlc, mpixlc_r	gcc, mpicc	xlc	gcc
C++	bgxlc++, mpixlcxx	g++. mpicxx	xlc++, xlC	g++
Fortran	bgxlf,bgxlf90,… mpixlf90,…	gfortran, mpif90	xlf, xlf90,…	gfortran

Cross compilation:
mpixlc –O3 **–qarch=qp –qtune=qp** myprog.c

Compilation:
xlc –q64 myprog.c

Available scripts

- Use available scripts to compile and link MPI programs
  - IBM XL compilers
    - default installation directory with PDF documentation: /opt/ibmcmp
    - wrappers are in /bgsys/drivers/ppcfloor/comm/xl/bin
    - Provide higher level optimization than GNU compilers
  - GNU compilers
    - default installation directory: /bgsys/drivers/ppcfloor/gnu-linux/bin
    - wrappers are in
    - /bgsys/drivers/ppcfloor/comm/gcc/bin

mpich2version Prints MPICH2 version information

Useful remarks

- ✓ “bg” compilers are for cross-compilation (**only IBM xl family**)
- ✓ “mpi” are wrappers for cross-compilation
- ✓ The _r-suffixed invocations allow for thread-safe compilation
  - use them if you want to create threaded applications
  - The -qsmp option must be used together with thread-safe compiler invocation modes

Some key options for IBM compilers

Option	Meaning
-qarch=qp	Produces object code for the BGQ platform and: Enables BGQ vector data type Sets the –qsimd=auto option
-qtune=qp	Default with –qarch=qp or without –qarch, -qtune options and bg-prefixed compilers. Also set if specifying –q64 or –O4,-O5
-q64	Sets 64-bit compiler mode
-qstaticlink	The compiler links only static libraries with the object file being produced. (Enabled by default; specify -qnostaticlink to dynamically link your programs).
-qtm	enables support for transactional memory Default is –qnotm To use with thread-safe compilation
-qsimd	-qsimd=auto enables automatic generation of QPX vector instructions. Enabled by default at all optimization levels. To disable automatic generation of QPX instructions, use -qsimd=noauto.

Programming/Production Environment: Software and libraries

Available:

- Mathematical Libraries (essl, blas, Lapack, Scalapack, blacs, PETSc, fftw, ...)
- I/O Libraries (HDF5, NetCDF)
- Namd
- GROMACS
- QE
- Pluto
- CPMD
- CP2K

Outline

- What is BG
- Overview of Blue Gene Q architecture and software
  - Hardware components, networks and partitioning
  - Types of nodes and software stack
- Developing applications for BGQ
  - Access to BGQ
  - Porting applications on BGQ
- Running and monitoring jobs on BGQ
  - runjob command
  - environment variables
  - job script examples
  - LL commands
  - Available Debugging and Profiling Tools

Running jobs: How to

1. Compile your application using the proper BGQ **cross compiler**

2. run a **batch job**
    1. Prepare a job script
    2. Submit your job script

Determining the number of processes

I. **Compute block size:**

Minimum size on Fermi: 64(128) compute nodes =1024 (2048) cores

Small blocks:

- consist of one or more node boards within a single midplane (**at least 2 or multiple on FERMI**)
- always multiple of 32 (**64 on FERMI**) compute nodes
- Not a torus in all five dimensions

Large blocks:

- consist of one or more complete midplanes
- always multiple of 512 nodes
- Can be a torus in all five dimensions

Determining the number of processes

II. **Processes per node:**

Best configuration depends on:

a. type of application (can it scale up?)

b. memory requirements (is it memory/task demanding)

c. implemented parallel paradigm (is it hybrid?)

Processes per node	Number of A2 cores per process	Maximum number of active hardware threads per process
1	16	64
2	8	32
4	4	16
8	2	8
16	1	4
32	2 processes per core	2
64	4 processes per core	1

Job script: general structure

```
#!/bin/bash
# @ job_name = bgsize.$(jobid)
# @ output = z.$(jobid).out
# @ error  = z.$(jobid).err
# @ shell = /bin/bash
# @ job_type = bluegene
# @ wall_clock_limit = 02:00:00
# @ notification = never
# @ bg_size = 256
# @ bg_connectivity = torus
# @ class = keyproject
# @ account_no = cinstaff
# @ restart = no
# @ queue

cd /gpfs/scratch/userinternal/cin0753a/mydir

runjob –ranks-per-node 64 ./program.exe
```

LL keywords

Application block

LL keywords: General Keywords (I)

Keyword	Meaning	Possible values	default
#@wall_clock_limit	elapsed time limit	GG+HH:MM:SS	
#@input	File used for standard input	<some-filename>	
#@output	File used for standard output	<some-filename>	
#@error	File used for standard error	<some-filename>	
#@initialdir	Initial working directory during job execution	<some-pathname>	Current working dir at the time the job was submitted

LL keywords: General Keywords (II)

Keyword	Meaning	Possible values	default
#@notification	when the system notifies the user about the job	start/complete/error/always/never	complete
#@notify_user	Address for email notification. Required if #@notification is not set to never	<valid-email-address>	
#@environment	Specifies initial environment variables set by LL when your job step starts	COPY_ALL=all variables are copied $var= variable to be copied	No default value is set
#@queue	Terminates LL directives		

21st Summer School of PARALLEL COMPUTING

LL keywords: BGQ Specific Keywords

Keyword	Meaning	Possible values	default
#@job_type	Specifies the type of job step to process. Set to **bluegene!**	serial/parallel/bluegene/ MPICH	serial
#@bg_size	Size of the job in number of compute nodes to be used	<integer>	128
#@bg_connection	Type of wiring requested for the partition	MESH/TORUS/ EITHER	MESH

runjob

Application has to be executed through **runjob** command to run on the compute nodes

- Syntax:
  - runjob [options]
  - runjob [options] binary [arg1 arg2 ... argn]
- Parameters can be set
  - by command-line options (higher priority!)
  - environment variables
- runjob –h for a complete list

21st Summer School of PARALLEL COMPUTING

runjob options – I (job)

Command line option	Environment variable	Description
--exe executable	RUNJOB_EXE= executable	Specifies the full path to the executable (this argument can be also specified as the first argument afer ":") runjob --exe /home/user/a.out runjob : /home/user/a.out
--args prg_args	RUNJOB_ARGS="prg_args"	Passes "prg_args" to the launched application on the compute node runjob : a.out hello world runjob --args hello world --exe a.out
--envs ENVVAR=values	RUNJOB_ENVS= "ENVVAR=value"	Sets the environment variable ENVVAR=value in the job environment on the compute nodes
-exp_env ENVVAR	RUNJOB_EXP_ENV =ENVVAR	Exports the environment variable ENVVAR in the current environment to the job on the compute nodes

runjob options – II (resources)

Command line option	Environment variable	Description
--ranks-per-node		Specifies the number of processes per compute node. Valid values are:1,2,4,8,16,32,64
--np n	RUNJOB_NP=n	Number of processes in the job(<= block_size*ranks-per_node)

runjob options – III (misc)

Command line option	Environment variable	Description
--start_tool		Path to tool to start with the job
--tool_args		Arguments for the tool

Example I (pure MPI)

```
#!/bin/bash
# @ job_name = example_1
# @ comment = "BGQ Job by Size"
# @ error = $(job_name).$(jobid).out
# @ output = $(job_name).$(jobid).out
# @ environment = COPY_ALL
# @ wall_clock_limit = 1:00:00
# @ notification = error
# @ notify_user = my_address@my.institution
# @ job_type = bluegene
# @ bg_size = 1024
# @ bg_connection = TORUS
# @ queue

runjob –ranks-per-node 64   --exe $DIRmy_program
```

65536 MPI tasks
(-np flag not needed!)

Example II (MPI+OMP)

```
# @ job_name = example_2
# @ error = $(job_name).$(jobid).out
# @ output = $(job_name).$(jobid).out
# @ environment = COPY_ALL
# @ wall_clock_limit = 12:00:00
# @ notify_user = my_address@my.institution
# @ notification = complete
# @ job_type = bluegene
# @ bg_connection = TORUS
# @ bg_size = 1024
# @ queue

module load scalapack
runjob –ranks-per-node 1 --exe /path/myexe --envs
OMP_NUM_THREADS=16
```

- 1024 MPI tasks
- 16 threads per task

LoadLever COMMANDS

Command	Description
llq -H	Help for LL commands
llq -?	Short usage message
llq –b	Shows BlueGene specific info
llsubmit script	Submits the job described in the “script” file
llq –u <username>	Returns information about your jobs in the queues
llq –s <joblist>	Returns detailed information about why the job remains idle
llq –l <joblist>	Returns detailed information about the specific job
llcancel <joblist>	Cancels a job from the queues, either if it is waiting or running

Debugging & Profiling

- ❑ addr2line
- ❑ gdb
- ❑ totalview

Performance Analysis tools:

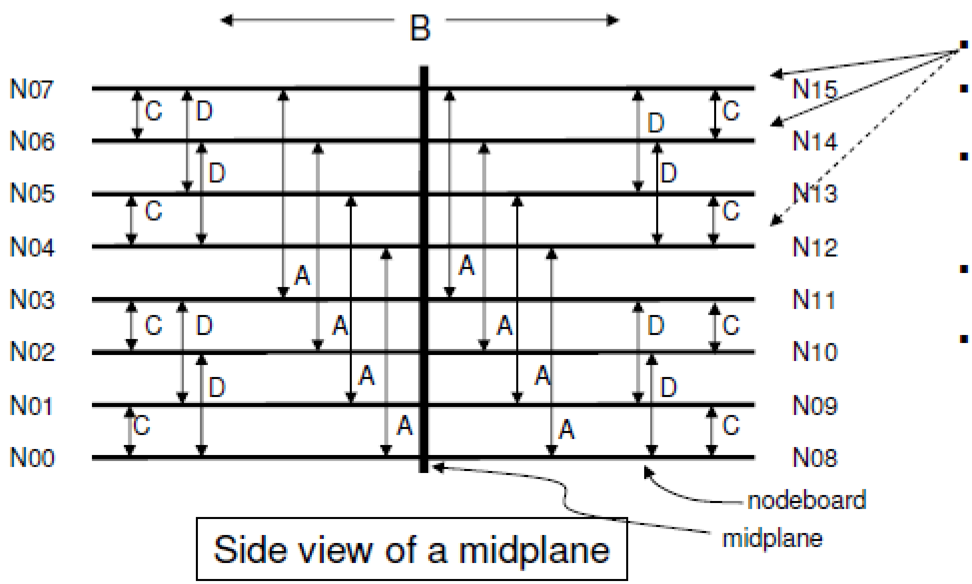
- ✓ Automatically Available Performance Counters
- ✓ SCALASCA
- ✓ High Performance Computing Toolkit
    - MPI Profiler and Tracer
    - CPU profiler Xprofiler
    - Hardware Performance Monitoring
    - I/O Performance

References

- IBM Redbook – Application Development
- IBM Redbook – System Administration
- IBM – Compilers:
  - xl c/c++ for IBM Blue Gene/Q, V12.1
  - xl Fortran for IBM Blue Gene/Q, V14.1
- FERMI Documentation/User Guide on hpc.cineca.it

5D Torus Wiring in a Midplane: (A,B,C,D,E)

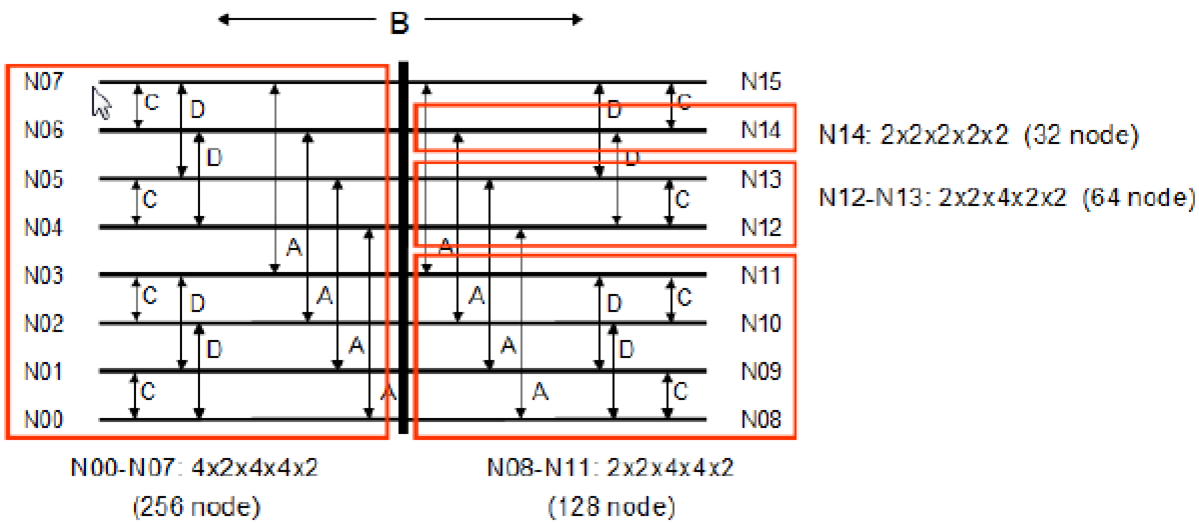
The 5 dimensions are denoted by the letters A, B, C, D, and E. The latest dimension E is always 2, and is contained entirely within a midplane.



- Each nodeboard is 2x2x2x2x2
- Arrows show how dimensions A,B,C,D span across nodeboards
- Dimension E does not extend across nodeboards
- The nodeboards combine to form a 4x4x4x4x2 torus
- Note that nodeboards are paired in dimensions A,B,C and D as indicated by the arrows

IBM

Partitioning a Midplane into Blocks



- Partitioning a midplane is the same concept as in BG/P

- 5th dimension always wrapped
- Other dimensions wrapped whenever they are 4 nodes wide

Partition : mesh versus Torus

Previously on BG/P we had torus vs. mesh for an entire block, now will have torus vs. mesh on a per-dimension basis, partial torus for sub-midplane blocks, partial torus for multi-midplane blocks.

# Node Cards	# Nodes	Dimensions	Torus (ABCDE)
1	32	2x2x2x2x2	00001
2 (adjacent pairs)	64	2x2x4x2x2	00101
4 (quadrants)	128	2x2x4x4x2	00111
8 (halves)	256	4x2x4x4x2	10111

IBM

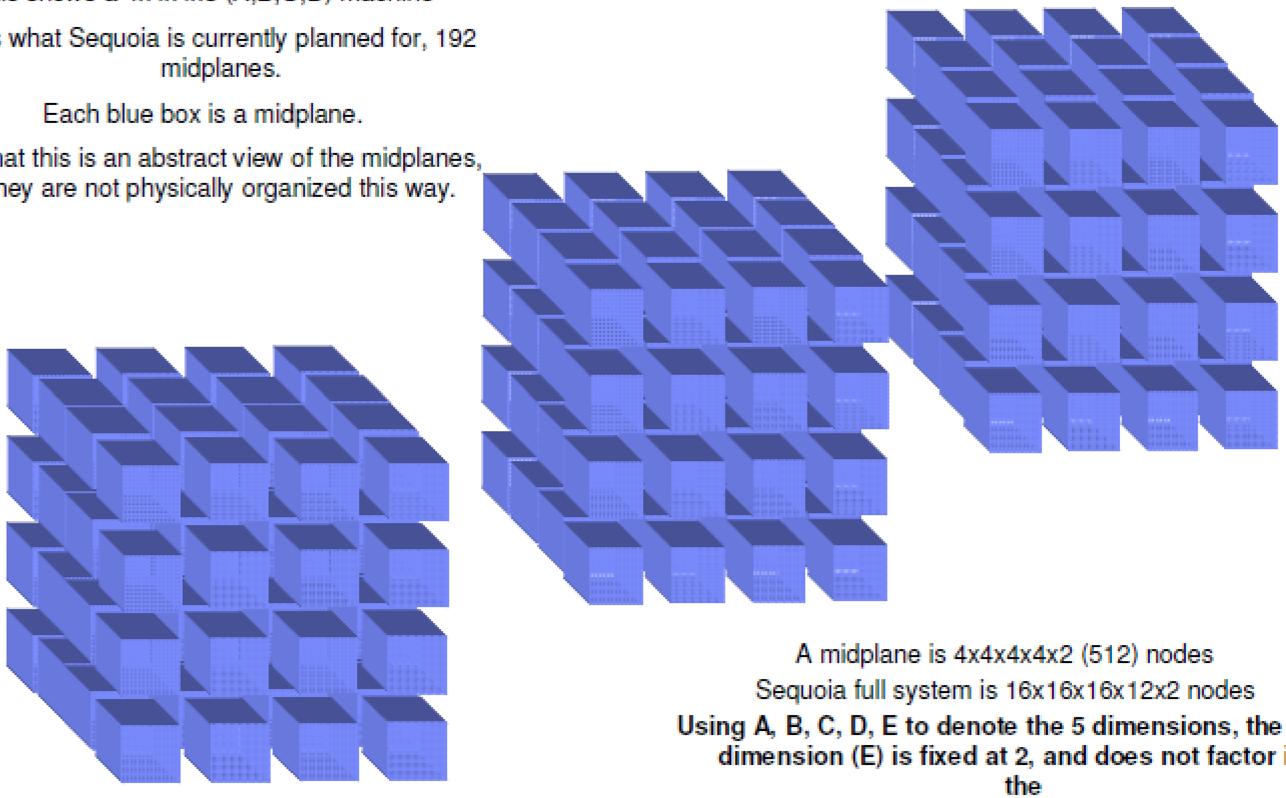
Compute Partitions - Large Blocks

This shows a 4x4x4x3 (A,B,C,D) machine

This is what Sequoia is currently planned for, 192 midplanes.

Each blue box is a midplane.

Note that this is an abstract view of the midplanes, they are not physically organized this way.



A midplane is 4x4x4x4x2 (512) nodes

Sequoia full system is 16x16x16x12x2 nodes

Using A, B, C, D, E to denote the 5 dimensions, the last dimension (E) is fixed at 2, and does not factor into the

Partitioning details