



IBM

IBM Systems & Technology Group

Boston University App Porting Workshop Blue Gene/L Overview

Kirk E Jordan
Strategic Growth Business/Deep Computing
Systems & Technology Group

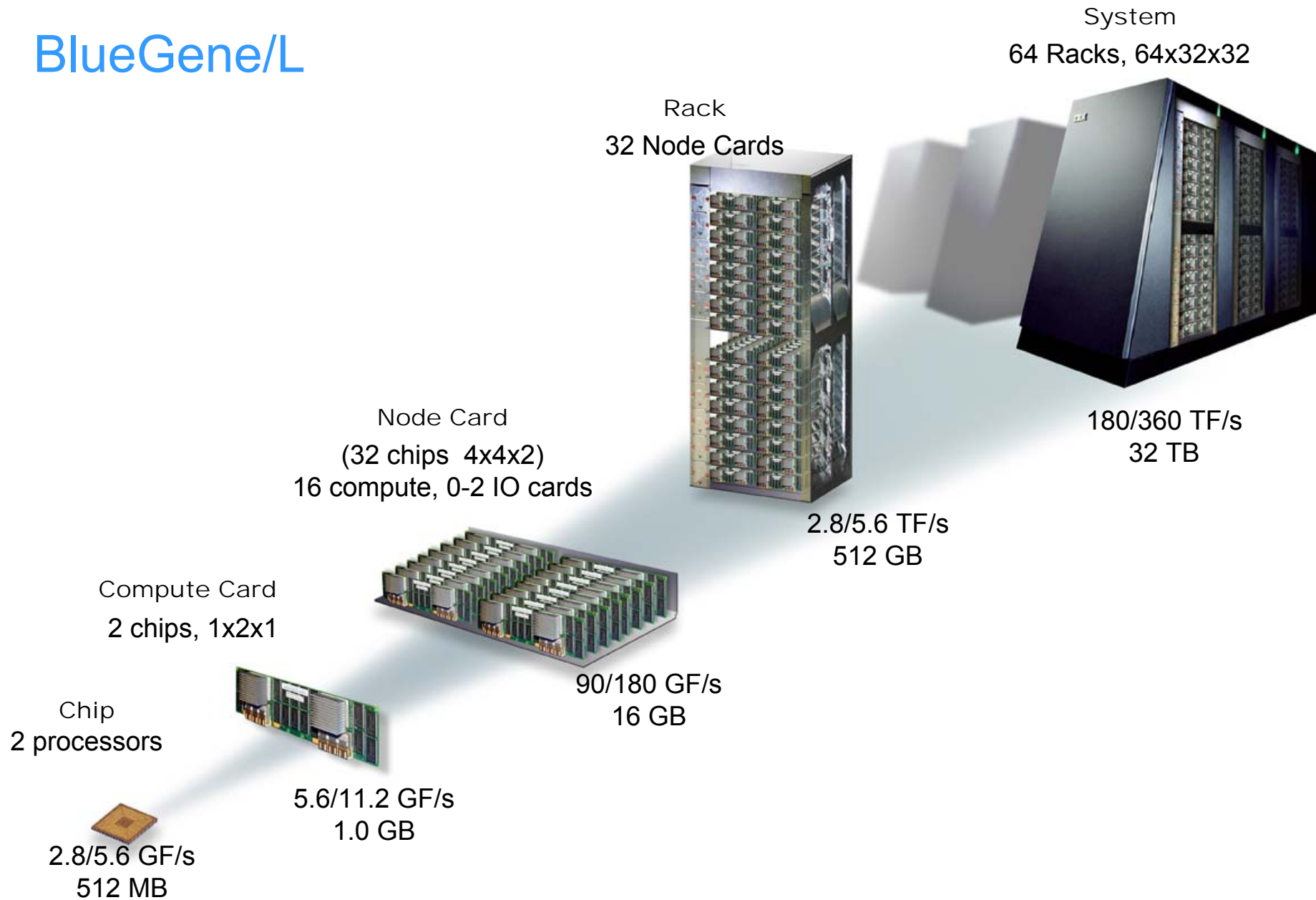
kjordan@us.ibm.com

Deep Computing

Outline

- What is Blue Gene (BG/L) – Architecture & Philosophy (SW)
- What is it like
- Some results on actual runs
- Remarks

BlueGene/L



System

64 Racks, 64x32x32

Rack

32 Node Cards

Node Card

(32 chips 4x4x2)

16 compute, 0-2 IO cards

Compute Card

2 chips, 1x2x1

Chip

2 processors

2.8/5.6 GF/s

512 MB

5.6/11.2 GF/s

1.0 GB

90/180 GF/s

16 GB

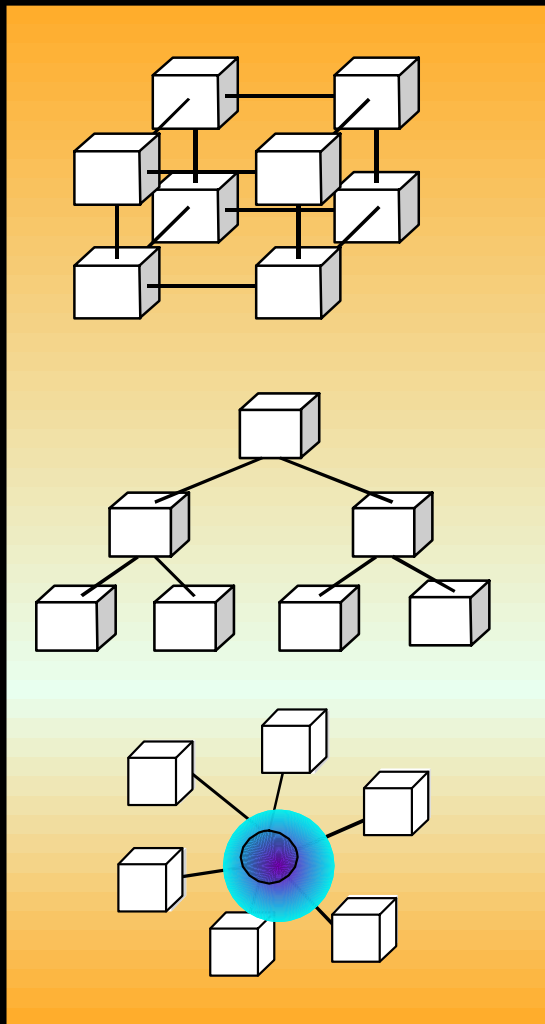
2.8/5.6 TF/s

512 GB

180/360 TF/s

32 TB

BlueGene/L Interconnection Networks



3 Dimensional Torus

- ▶ Interconnects all compute nodes (65,536)
- ▶ Virtual cut-through hardware routing
- ▶ 1.4Gb/s on all 12 node links (2.1 GB/s per node)
- ▶ Communications backbone for computations
- ▶ 0.7/1.4 TB/s bisection bandwidth, 67TB/s total bandwidth

Global Tree

- ▶ One-to-all broadcast functionality
- ▶ Reduction operations functionality
- ▶ 2.8 Gb/s of bandwidth per link
- ▶ Latency of tree traversal 2.5 μ s
- ▶ ~23TB/s total binary tree bandwidth (64k machine)
- ▶ Interconnects all compute and I/O nodes (1024)

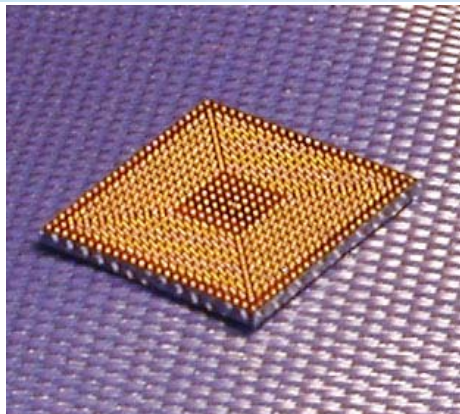
Ethernet

- ▶ Incorporated into every node ASIC
- ▶ Active in the I/O nodes (1:64)
- ▶ All external comm. (file I/O, control, user interaction, etc.)

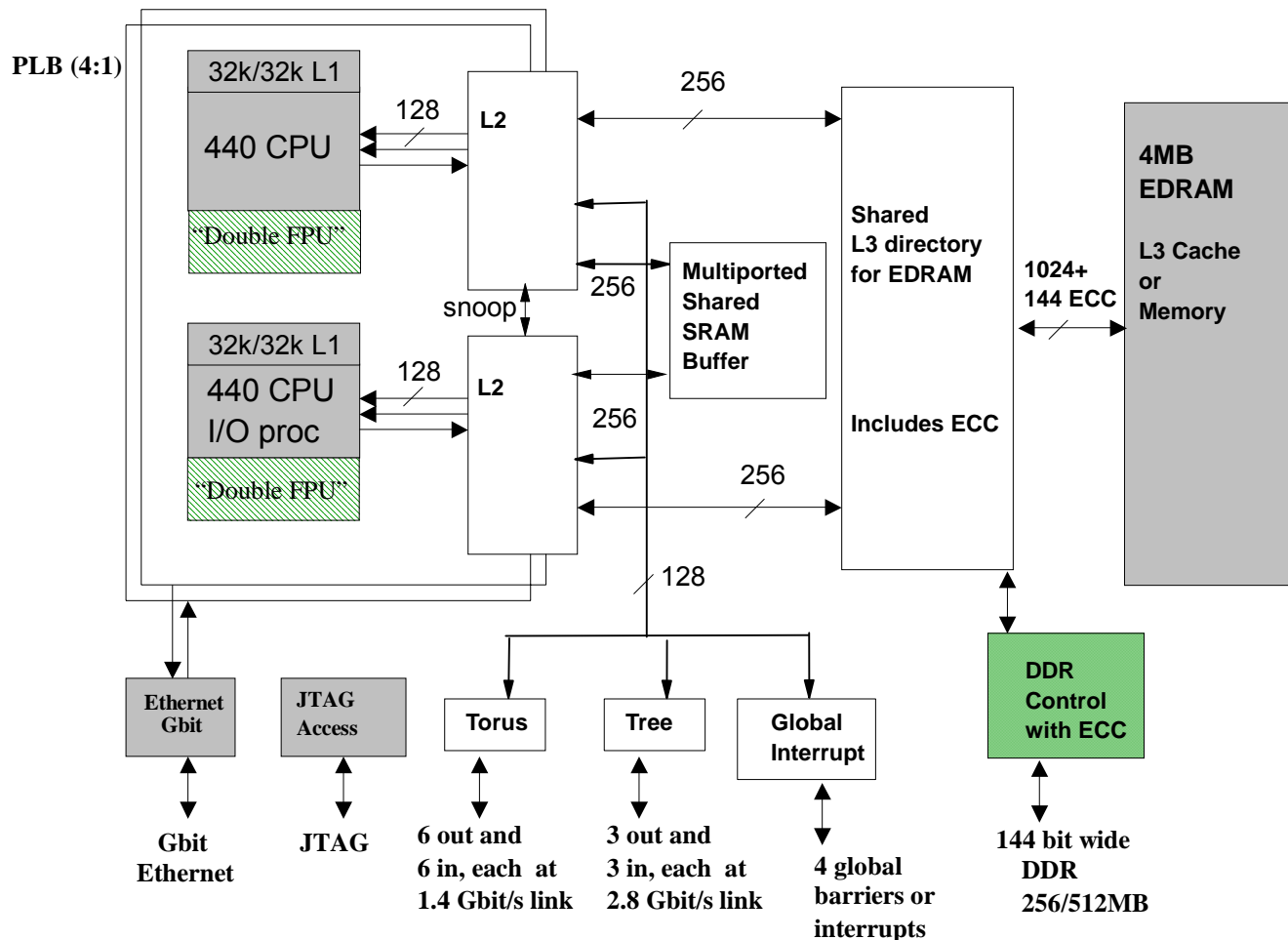
Low Latency Global Barrier and Interrupt

Control Network

BlueGene/L Compute ASIC



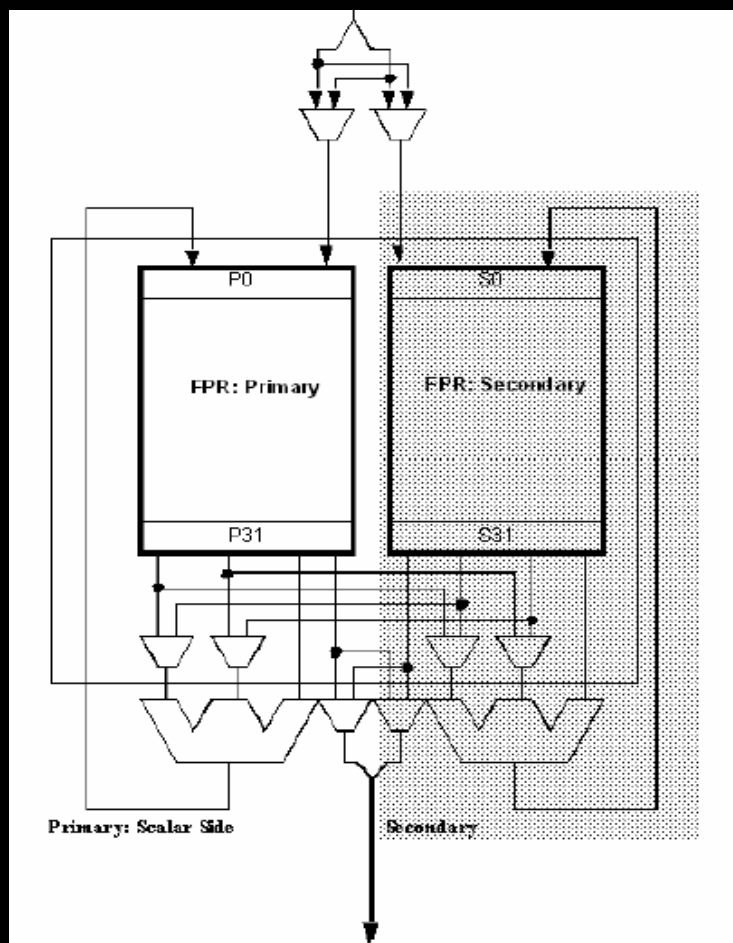
- IBM CU-11, 0.13 μm
- 11 x 11 mm die size
 - 25 x 32 mm CBGA
 - 474 pins, 328 signal
 - 1.5/2.5 Volt



Powerpc-440 Processor

- 32-bit architecture at 700 MHz
- single integer unit (fxu)
- single load/store unit
- special double floating-point unit (dfpu)
- L1 Data cache : 32 KB total size, 32-Byte line size,
- 64-way associative, round-robin replacement
- L2 Data cache : prefetch buffer, holds 16 128-byte lines
- L3 Data cache : 4 MB, ~35 cycles latency, on-chip
- Memory : 512 MB DDR at 350 MHz, ~85 cycles latency
- Double FPU has 32 primary floating-point registers, 32 secondary floating-point registers, and supports :
 - ▶ standard powerpc instructions, which execute on fpu0 (fadd, fmadd, fadds, fdiv, ...), and
 - ▶ SIMD instructions for 64-bit floating-point numbers (fpadd, fpmadd, fpre, ...)
- Floating-point pipeline : 5 cycles
- Floating-point load-to-use latency : 4 cycles

Dual FPU Architecture



- Two 64 bit floating point units
- Designed with input from compiler and library developers
- SIMD instructions over both register files
 - ▶ FMA operations over double precision data
 - ▶ More general operations available with cross and replicated operands
 - Useful for complex arithmetic, matrix multiply, FFT
- Parallel (quadword) loads/stores
 - ▶ Fastest way to transfer data between processors and memory
 - ▶ Data needs to be 16-byte aligned
 - ▶ Load/store with swap order available
 - Useful for matrix transpose

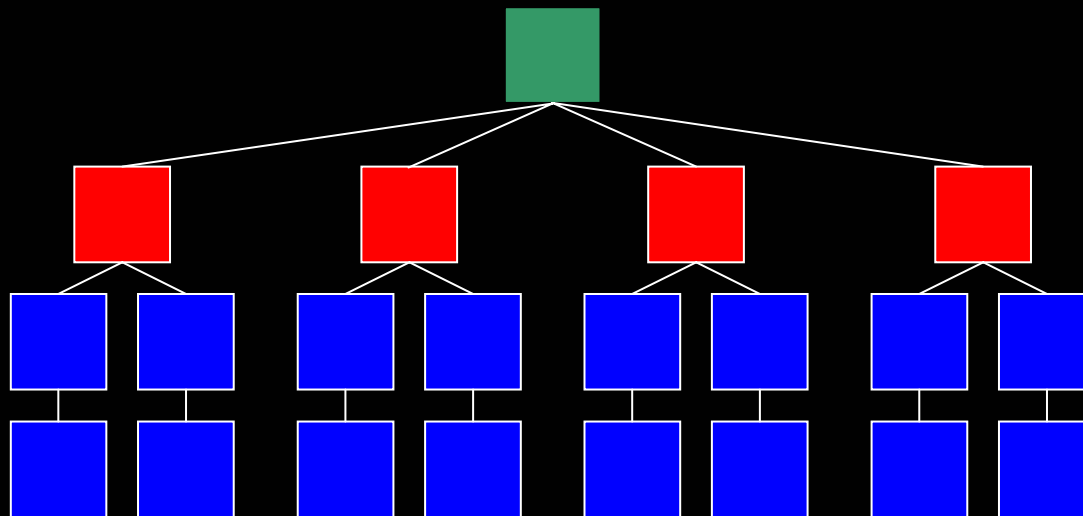
The Software Solution Philosophy

- **Simplicity**
 - ▶ Avoid features not absolutely necessary for high performance computing
 - ▶ Using simplicity to achieve both efficiency and reliability

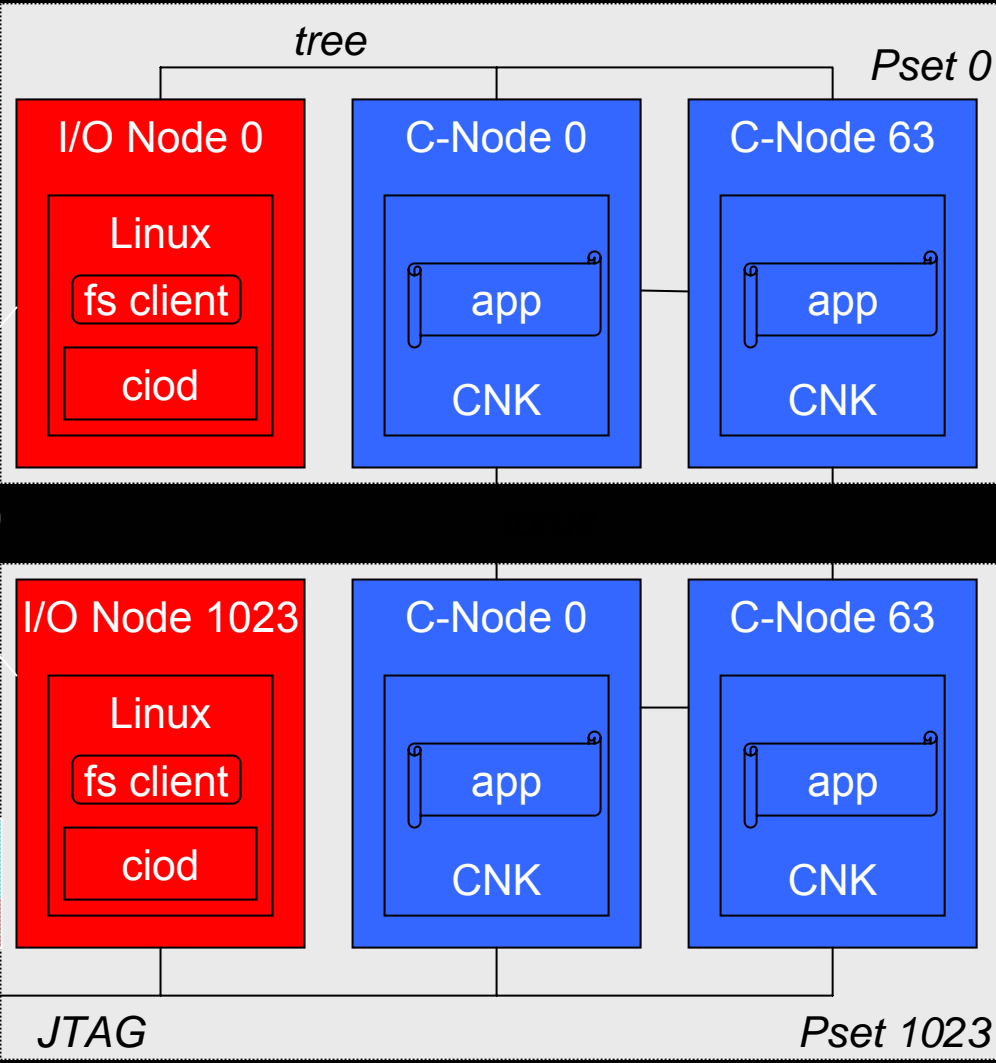
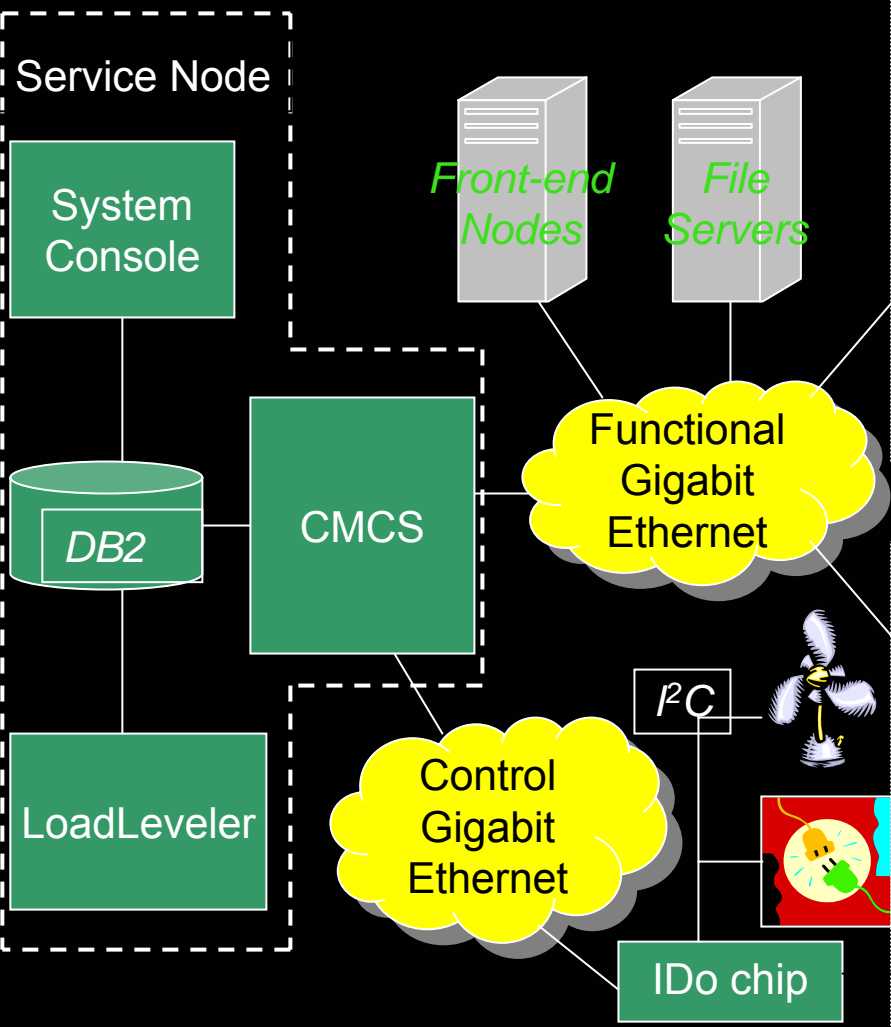
- **New organization of familiar functionality**
 - ▶ Same interface, new implementation
 - ▶ Hierarchical organization
 - ▶ Message passing provides foundation
 - Research on higher level programming models using that base

BlueGene/L Software Hierarchical Organization

- **Compute nodes** dedicated to running user application, and almost nothing else - simple compute node kernel (CNK)
- **I/O nodes** run Linux and provide a more complete range of OS services – files, sockets, process launch, signaling, debugging, and termination
- **Service node** performs system management services (e.g., heart beating, monitoring errors) - transparent to application software



BlueGene/L System Architecture



Programming Models and Development Environment

■ Familiar Aspects

▶ SPMD model - Fortran, C, C++ with MPI (MPI1 + subset of MPI2)

- Full language support
- Automatic SIMD FPU exploitation

▶ Linux development environment

- User interacts with system through FE nodes running Linux – compilation, job submission, debugging
- Compute Node Kernel provides look and feel of a Linux environment – POSIX system calls (with restrictions)

▶ Tools – support for debuggers (Aetnus TotalView), MPI tracer, profiler, hardware performance monitors, visualizer (HPC Toolkit, Paraver, Kojak)

■ Restrictions (lead to significant scalability benefits)

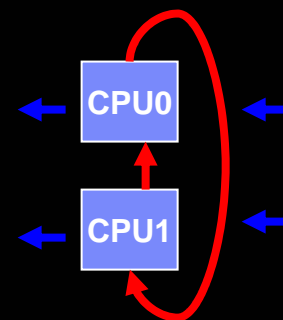
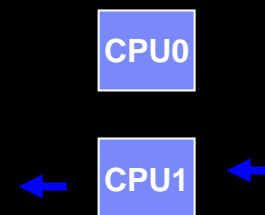
- Strictly space sharing - one parallel job (user) per partition of machine, one process per processor of compute node
- Virtual memory constrained to physical memory size
 - Implies no demand paging, only static linking

■ Other Issues: Mapping of applications to torus topology

- ▶ More important for larger systems (multi-rack systems)
- ▶ Working on techniques to provide transparent support

Execution Modes for Compute Node

- **Communication coprocessor mode:** CPU 0 executes user application while CPU 1 handles communications
 - ▶ Preferred mode of operation for communication-intensive and memory bandwidth intensive codes
 - ▶ Requires coordination between CPUs, which is handled in libraries
 - ▶ **Computation offload feature (optional):** CPU 1 also executes some parts of user application offloaded by CPU 0
 - Can be selectively used for compute-bound parallel regions
 - Asynchronous coroutine model (`co_start` / `co_join`)
 - Need careful sequence of cache line flush, invalidate, and copy operations to deal with lack of L1 cache coherence in hardware
- **Virtual node mode:** CPU0 and CPU1 handle both computation and communication
 - ▶ Two MPI processes on each node, one bound to each processor
 - ▶ Distributed memory semantics – lack of L1 coherence not a problem



Booting a BG/L partition with the database console

- Partitions or blocks of BG/L are typically booted using the database console
- From the FE invoke the bglconsole

```
/bgl/console/bin/bglconsole
```

```
mmcs$ allocate R00-M1 [virtual_node_mode]  
mmcs$ submitjob R00-M1 full_path_exe full_path_rundir [BGLMPI_SIZE=256] [BGLMPI_MAPPING=TXYZ] ...  
mmcs$ list bgljob jobid  
mmcs$ free R00-M1  
mmcs$ quit
```

- Some code results
 - Sanity
 - Monte Carlo calculation of PI
 - Linpack Benchmark
 - NAS Parallel Benchmark MG

Blue Gene Check Systems

Sanity.rts

stdout[20]: MPI: 20/32, Pers: <0,1,1,0>/<4,4,2,1>, Torus? X0Y0Z0, VN? 0, Mem:
512MB(6), Loc: R00-M1-Nf-C:J14-U11

stdout[20]: MPI: 20/64, Pers: <0,1,1,0>/<4,4,2,2>, Torus? X0Y0Z0, VN? 1, Mem:
512MB(6), Loc: R00-M1-N2-C:J14-U11

Blue Gene PI – Monte Carlo

stdout[0]:	#cpus	#trials	pi(est)	err(est)	err(abs)	time(s)	Mtrials/s
stdout[0]:	32	256000000	3.14176	0.00022	0.00017	1.082	236.58
stdout[0]:	16	256000000	3.14164	0.00022	0.00004	2.164	118.29
stdout[0]:	8	256000000	3.14157	0.00022	0.00002	4.328	59.15
stdout[0]:	4	256000000	3.14160	0.00022	0.00000	8.656	29.57
stdout[0]:	2	256000000	3.14155	0.00022	0.00004	17.313	14.79
stdout[0]:	1	256000000	3.14145	0.00022	0.00014	34.625	7.39

LINPACK Benchmark Blue Gene 32 nodes

```

stdout[0]: =====
stdout[0]: BG/L High-Performance Linpack benchmark -- Sept 1, 2003
stdout[0]: Written by G. Almasi, V. Austel, S. Chatterjee, J. Gunnels,
stdout[0]: F. Gustavson, and J. Sexton,
stdout[0]: IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.
stdout[0]: =====
stdout[0]: Initial matrix taken from and code design based upon:
stdout[0]: Linpack 1.0 -- High-Performance Linpack benchmark -- September 27, 2000
stdout[0]: Written by A. Petitet and R. Clint Whaley, Innovative Computing Labs., UTK
stdout[0]: =====
stdout[0]: Final : N_SIZE = 40959 D_SIZE = 128 P_MESH = 4 Q_MESH = 8
stdout[0]: =====
stdout[0]: T/V          N  NB  P  Q          Time          Gflops
stdout[0]: -----
stdout[0]: WR21R2L2    40959 128  4  8          331.79          1.381e+02
stdout[0]: -----
stdout[0]: ||Ax-b||_oo / ( eps * ||A||_1 * N      ) =    0.0064729 ..... PASSED
stdout[0]: ||Ax-b||_oo / ( eps * ||A||_1 * ||x||_1 ) =    0.0025873 ..... PASSED
stdout[0]: ||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =    0.0004933 ..... PASSED
stdout[0]: =====
stdout[0]:
Finished      1 tests with the following results:
stdout[0]:          1 tests completed and passed residual checks,
stdout[0]:          0 tests completed and failed residual checks,
stdout[0]:          0 tests skipped because of illegal input values.
stdout[0]: -----
stdout[0]:
End of Tests.
stdout[0]: =====

```

NAS Parallel Benchmark MG

stdout[0]: **NAS Parallel Benchmarks 2.4 -- MG Benchmark**

stdout[0]:

stdout[0]: No input file. Using compiled defaults

stdout[0]: Size: 512x 512x 512 (class C)

stdout[0]: Iterations: 20

stdout[0]: **Number of processes: 32**

stdout[0]:

stdout[0]: Initialization time: 2.440 seconds

stdout[0]:

stdout[0]: Benchmark completed

stdout[0]: VERIFICATION SUCCESSFUL

stdout[0]: L2 Norm is .570673228574E-06

stdout[0]: Error is -.159772397417E-11

stdout[0]:

stdout[0]: MG Benchmark Completed.

stdout[0]: Class = C

stdout[0]: Size = 512x 512x 512

stdout[0]: Iterations = 20

stdout[0]: Time in seconds = 26.06

stdout[0]: Total processes = 32

stdout[0]: Compiled procs = 32

stdout[0]: **Mop/s total = 5975.36**

stdout[0]: Mop/s/process = 186.73

stdout[0]: Operation type = floating point

stdout[0]: Verification = SUCCESSFUL

stdout[0]: Version = 2.4

stdout[0]: Compile date = 31 Mar 2005

stdout[0]:

stdout[0]: Compile options:

stdout[0]: MPIF77 = blrts_xlf

stdout[0]: FLINK = blrts_xlf

stdout[0]: FMPI_LIB = -Impich.rts -lmsglayer.rts -lrts.rts -ldevi...

stdout[0]: FMPI_INC = -I\$(BGLSYS)/include

stdout[0]: FFLAGS = -g -O3 #qdebug=function_trace

stdout[0]: FLINKFLAGS = -L\$(BGLUSR)/lib -L\$(BGLSYS)/lib

stdout[0]: RAND = randi8

The Real Question

- What can you do with 130K processors? (8K, 16K, 32K)
 - ▶ Really BIG problems – Maybe
 - ▶ Same problems but much finer resolution, refinements, larger searches in shorter time – Maybe
 - ▶ Explore parameters – large parameter space – Maybe
- BUT
 - ▶ Perhaps need to rethink the problem
 - ▶ Most parallel programs are Single Program Multiple Data
- **What if**
 - ▶ **Multiple Programs Multiple Data - - Systems of Complex Systems interacting?**

Remarks

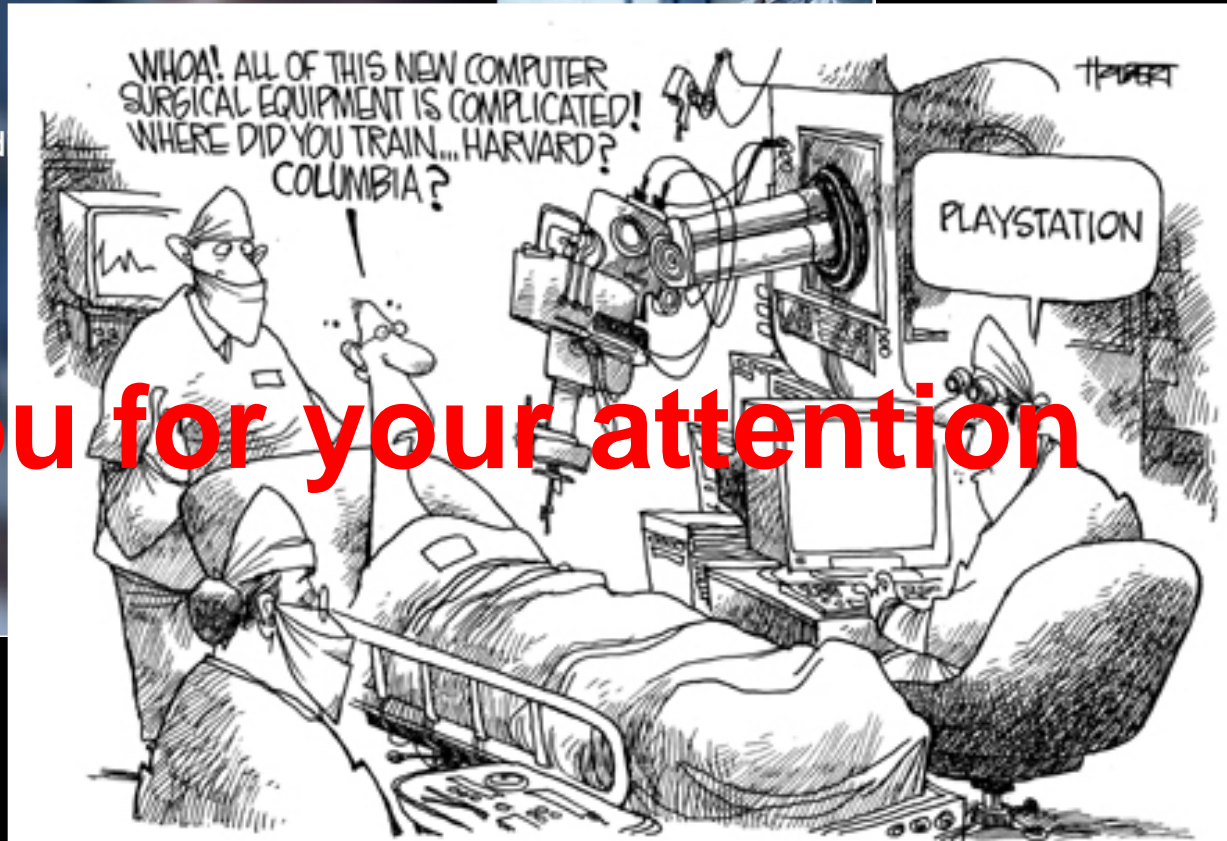
- Is Blue Gene a systems for computational science?
 - ▶ Does computational science problems need lots of cycles?
 - ▶ How to effectively use lots of processors
- Need to re-think how tackle problems - -
- Think of problems might tackle that until now would not dream of - -
- **The answers are left to the audience/the reader/the users - - you!!**
- **Lets get started**
<http://www.mcs.anl.gov/bgconsortium>





OR

A doctor saves a patient from heart d



Thank you for your attention