IBM PSSC Montpellier Customer Center

BlueGene/P Architecture

Forschungszentrum Jülich

11, August 2009

Pascal Vezolle
HPC Performance Specialist
vezolle@fr.ibm.com

Content (2 days Blue Gene application user workshop)

- Blue Gene/P Architecture
  - Hardware Components
  - Networks
  - Partitioning
- Blue Gene/P Software Stack
  - I/O Node
  - Compute Node
  - HPC Software Stack
  - Software Locations
- Programming Models & Execution Modes
  - Programming Models
  - Execution Modes
  - Partition Types
- Blue Gene/P Compilation
  - Compilers
  - Mathematical Libraries
- Blue Gene/P Execution
  - MPIRUN / MPIEXEC Commands
  - Environment Variables
  - LoadLeveler
  - HTC
- Blue Gene/P Parallel Libraries
  - Shared Memory
  - Message Passing
- Blue Gene/P Advanced Topics
  - Blue Gene/P Memory
  - Advanced Compilation with IBM XL Compilers
  - SIMD Programming
  - Communications Framework
  - Checkpoint/Restart
  - I/O Optimization

Information Sources

- **IBM Redbooks for Blue Gene (http://www.redbooks.ibm.com/)**
  - Application Development Guide
  - Performance Tools
  - System Administration Guide
- **Open Source Communities (Argonne Web site, …)**
- **Doxygen Documentation (DCMF, SPI, …) on the system (/bgsys)**

Blue Gene Philosophy – Hardware Perspective with standard HPC programming models

- How to break through the power and cost issues limiting traditional supercomputer design

- **Optimize FLOPS per rack, Best FLOPS/Watt found with modest-frequency design point**
  - Relatively low architectural complexity
  - Choice: BG/L : IBM PowerPC 440 -- 700 MHz @1.6V
    BG/P: IBM PowerPC 450 -- 850 MHz @1.1V

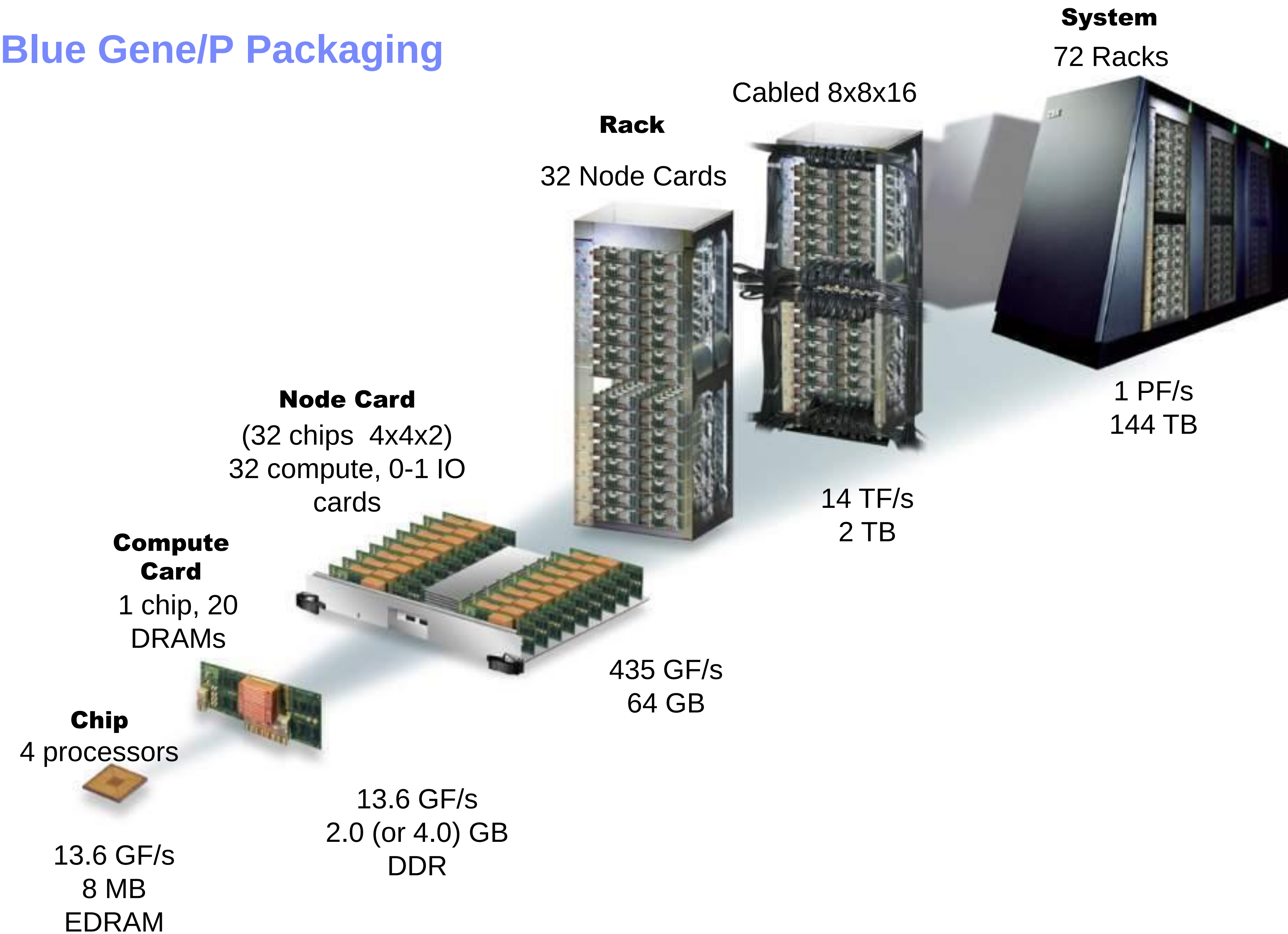
- **Scalable architecture and appropriate package can lead to a high density, low power, massively parallel system**

- **Some applications display very high levels of parallelism, and can be executed efficiently on tens of thousands of processors if:**
  - low-latency, high-bandwidth, integrated interconnect is present
  - machine is reliable enough
  - good compilers and programming models are available
  - machine is manageable (simple to build and operate)

- **Whatever the scalability it is compulsory to get the best from the Blue Gene processor and IO subsystem**

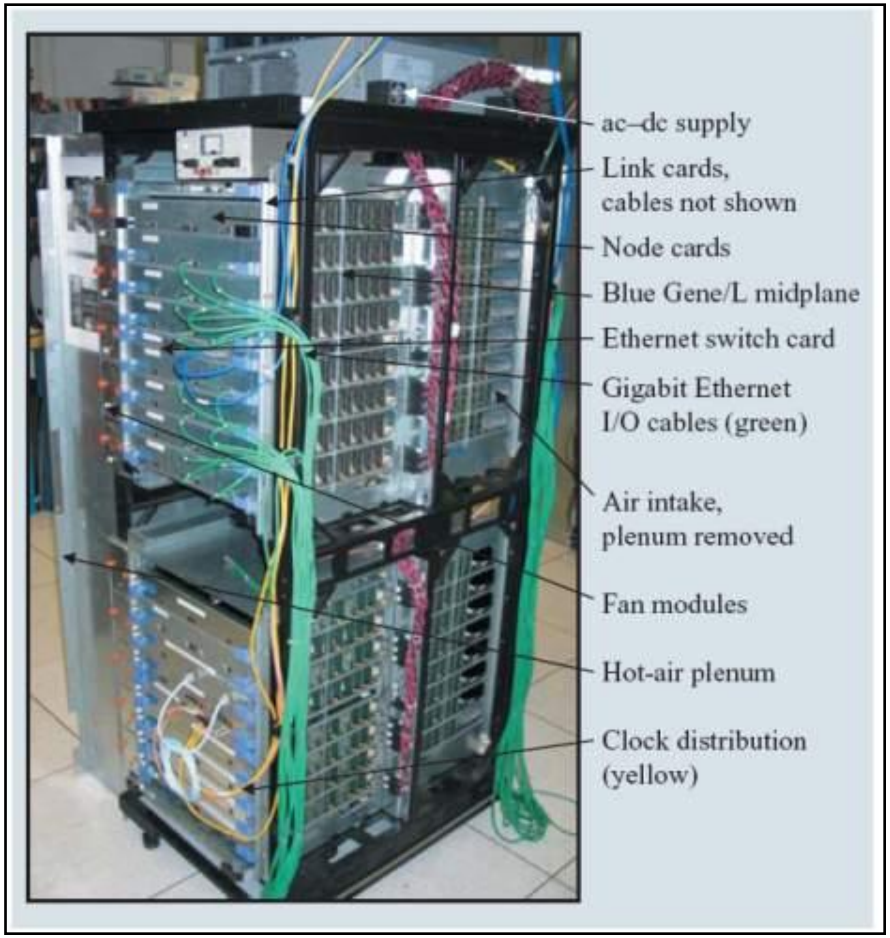
- **Embedded processor is based on IBM Power technology**

Blue Gene/P Packaging



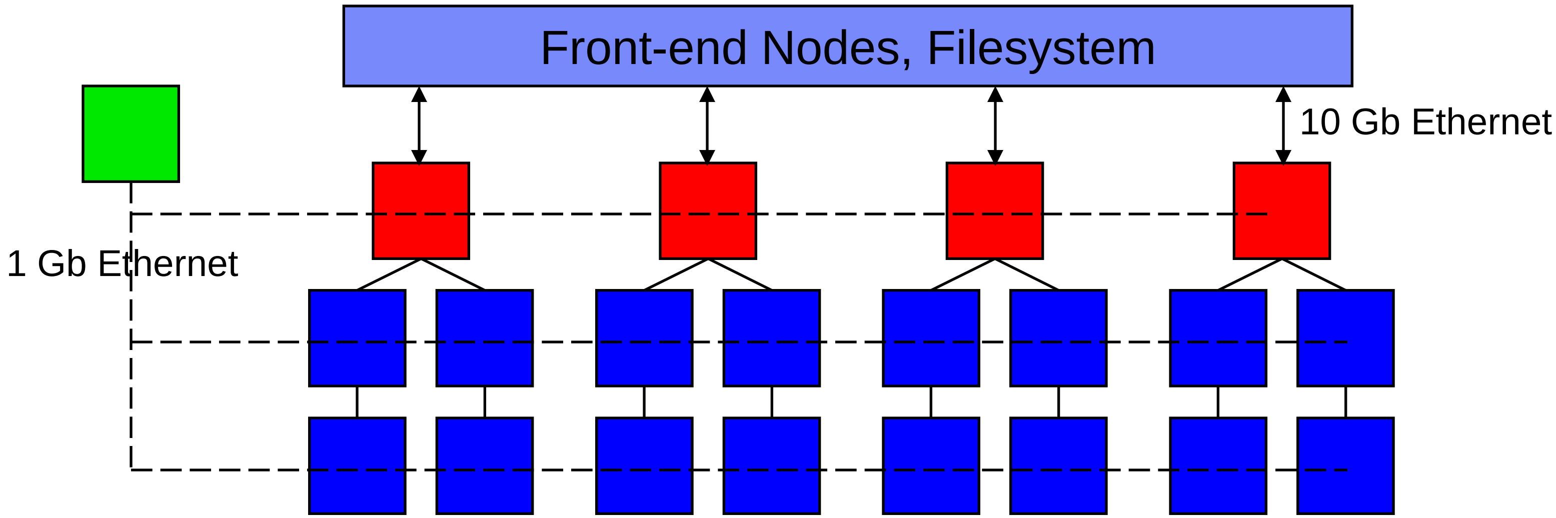
Blue Gene/P Rack Content

- **32 Compute Nodes populate a Node Card**
  - Node cards may be hot plugged for service
- **0-2 I/O Nodes populated on a Node Card**
  - Flexible ratio of compute to I/O nodes
  - I/O Nodes are identical to Compute Nodes other than placement in the Node Card which defines network connections
- **16 Node Cards form a Midplane**
- **2 Midplanes form a rack**
  - 1024 Compute Nodes per rack
  - 8 to 64 I/O nodes per rack: 80Gb to 640Gb Ethernet bw/rack

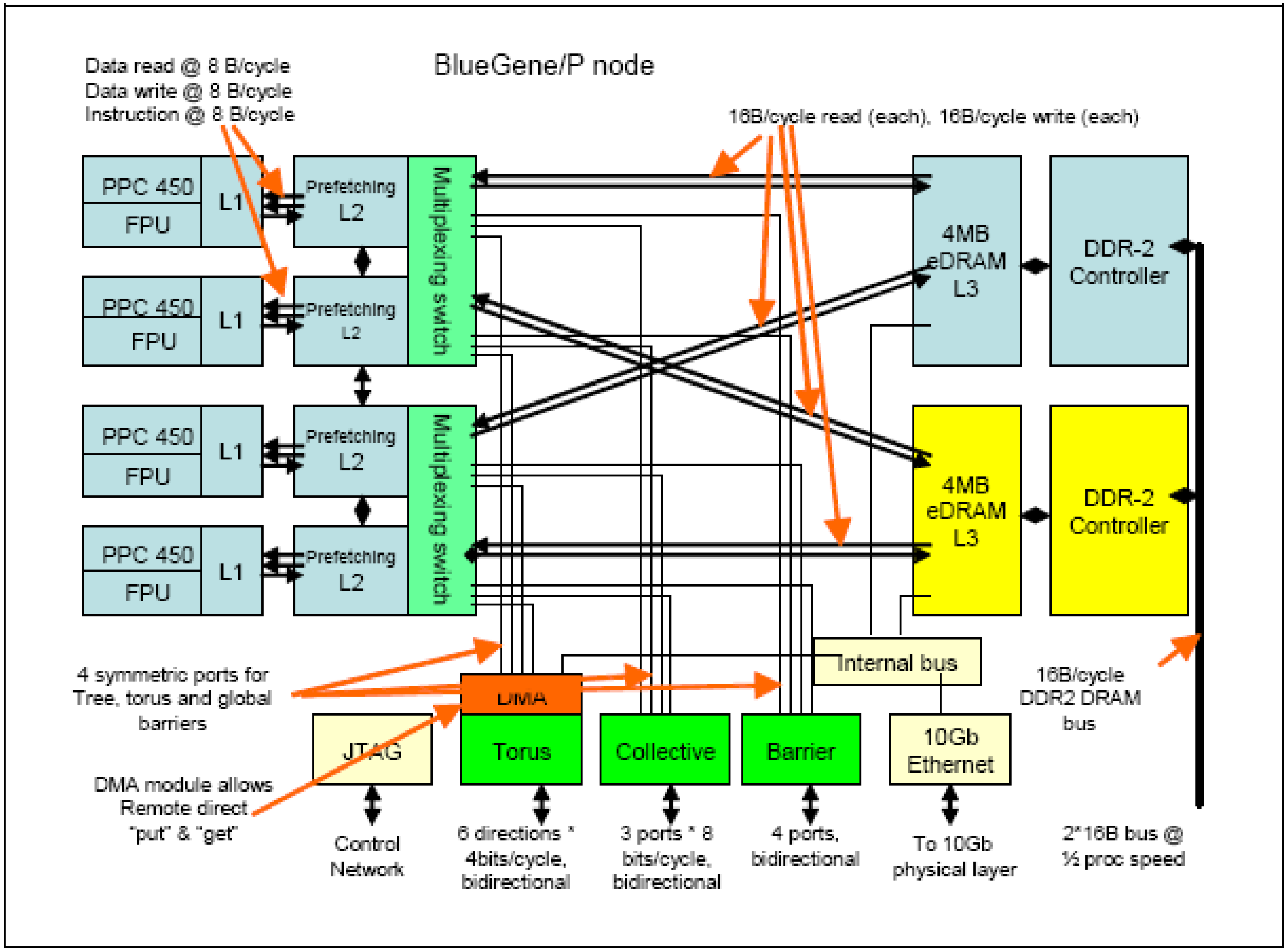


Blue Gene Blocks Hierarchical Organization

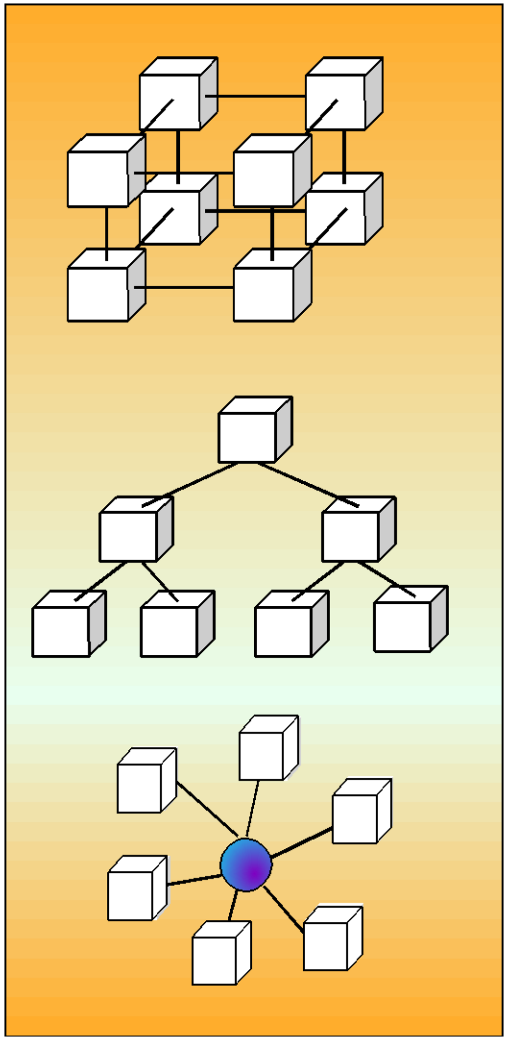
- **Compute Nodes dedicated to running user application, and almost nothing else - simple compute node kernel (CNK)**
- **I/O Nodes run Linux and provide a more complete range of OS services – files, sockets, process launch, signaling, debugging, and termination**
- **Service Node performs system management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



Blue Gene/P ASIC



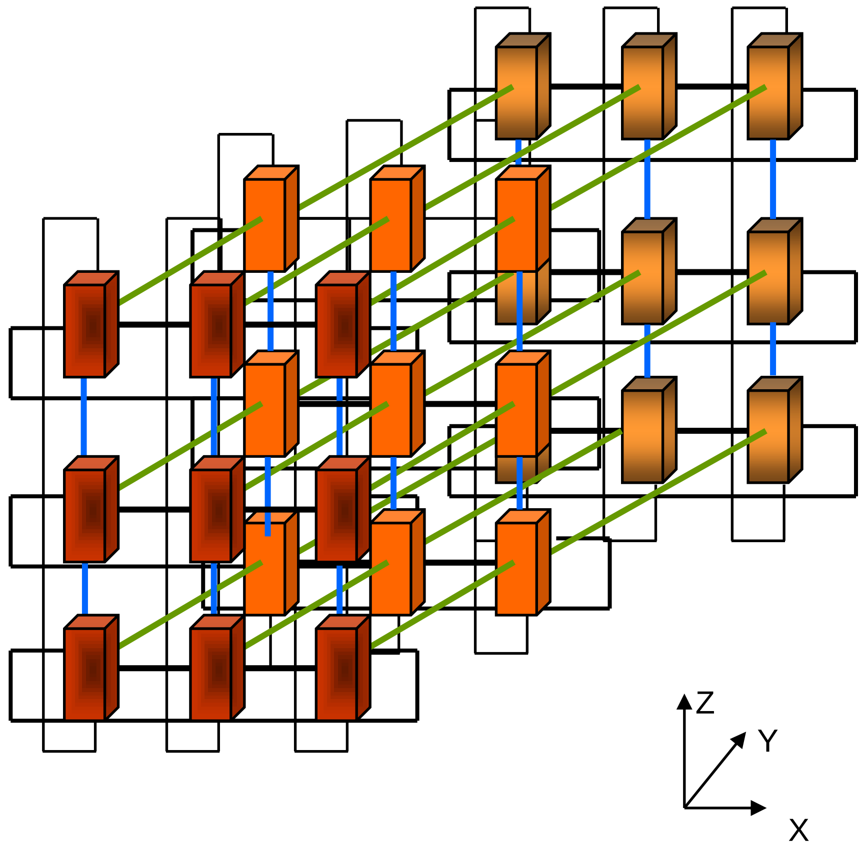
Blue Gene/P Interconnection Networks



- **3-Dimension Torus**
  - Interconnects all compute nodes
  - Virtual cut-through hardware routing
  - 3.4 Gb/s on all 12 node links (5.1 GB/s per node)
  - 0.5 µs latency between nearest neighbors, 5 µs to the farthest
  - MPI: 3 µs latency for one hop, 10 µs to the farthest
  - Communications backbone for computations
  - 1.7/3.9 TB/s bisection bandwidth, 188TB/s total bandwidth
- **Collective Network**
  - One-to-all broadcast functionality
  - Reduction operations functionality
  - 6.8 Gb/s of bandwidth per link
  - Latency of one way tree traversal 1.3 µs, MPI 5 µs
  - ~62 TB/s total binary tree bandwidth (72k machine)
  - Interconnects all compute and I/O nodes (1152)
- **Low Latency Global Barrier and Interrupt**
  - Latency of one way to reach all 72K nodes 0.65 µs (MPI 1.6 µs)
- **Other networks**
  - Functional Network
    - 10 Gb/s Ethernet
    - Linking I/O Nodes to shared filesystem (GPFS)
  - 1Gb Private Control Ethernet
    - Provides JTAG access to hardware. Accessible only from Service Node system

Torus VS Mesh

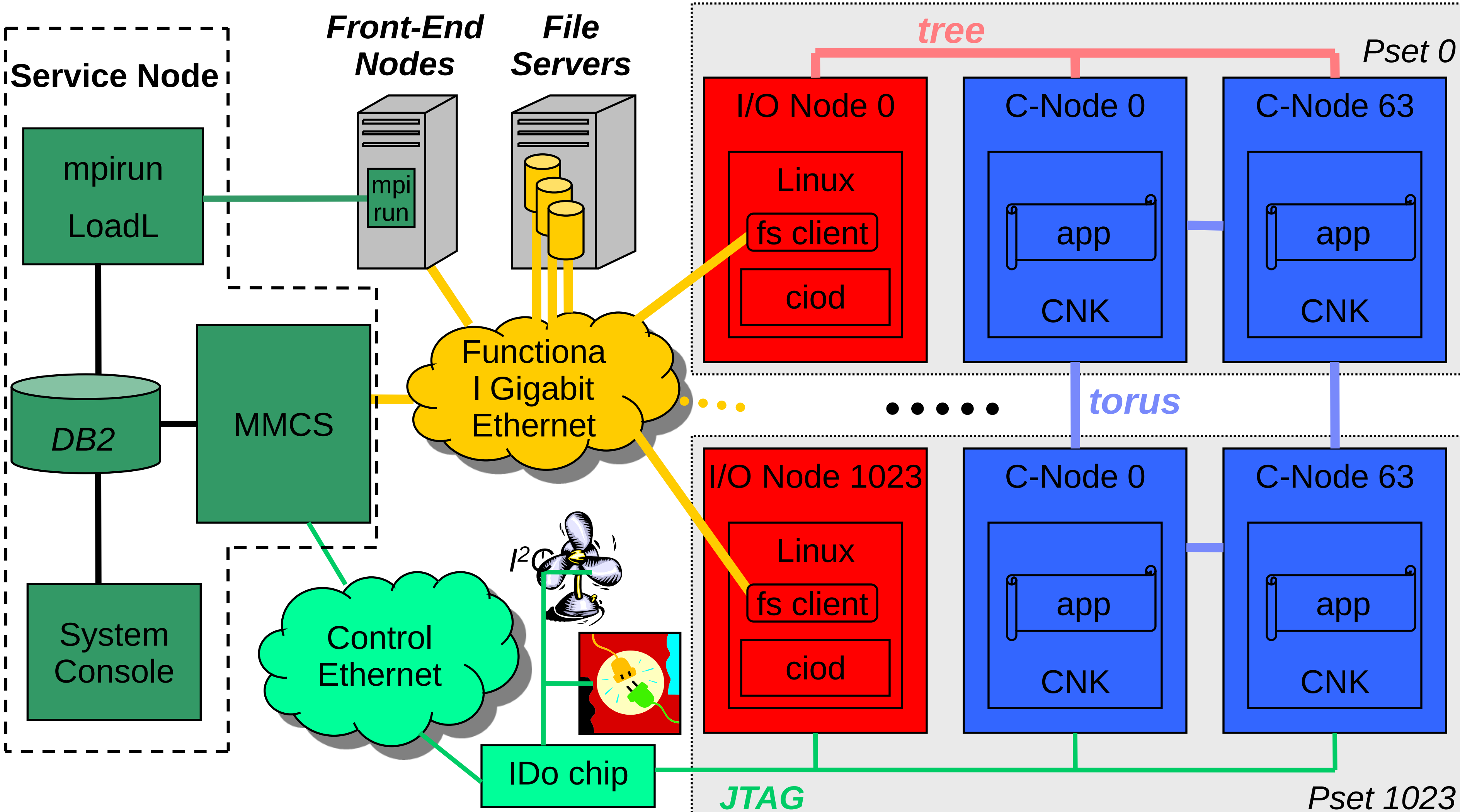
- **The basic block is the midplan, shape 8x8x8= 512 Computes Nodes (2048 cores)**
- **Only multiple midplans partition is a Torus; i.e. each CNode has 6 nearest-neighbours**
- **All the other partition is a mesh**
- **Capability from LoadLeveler to request a Torus or a Mesh with the field:**
  - # @ bg_connection= torus/mesh
- **The default is a mesh**



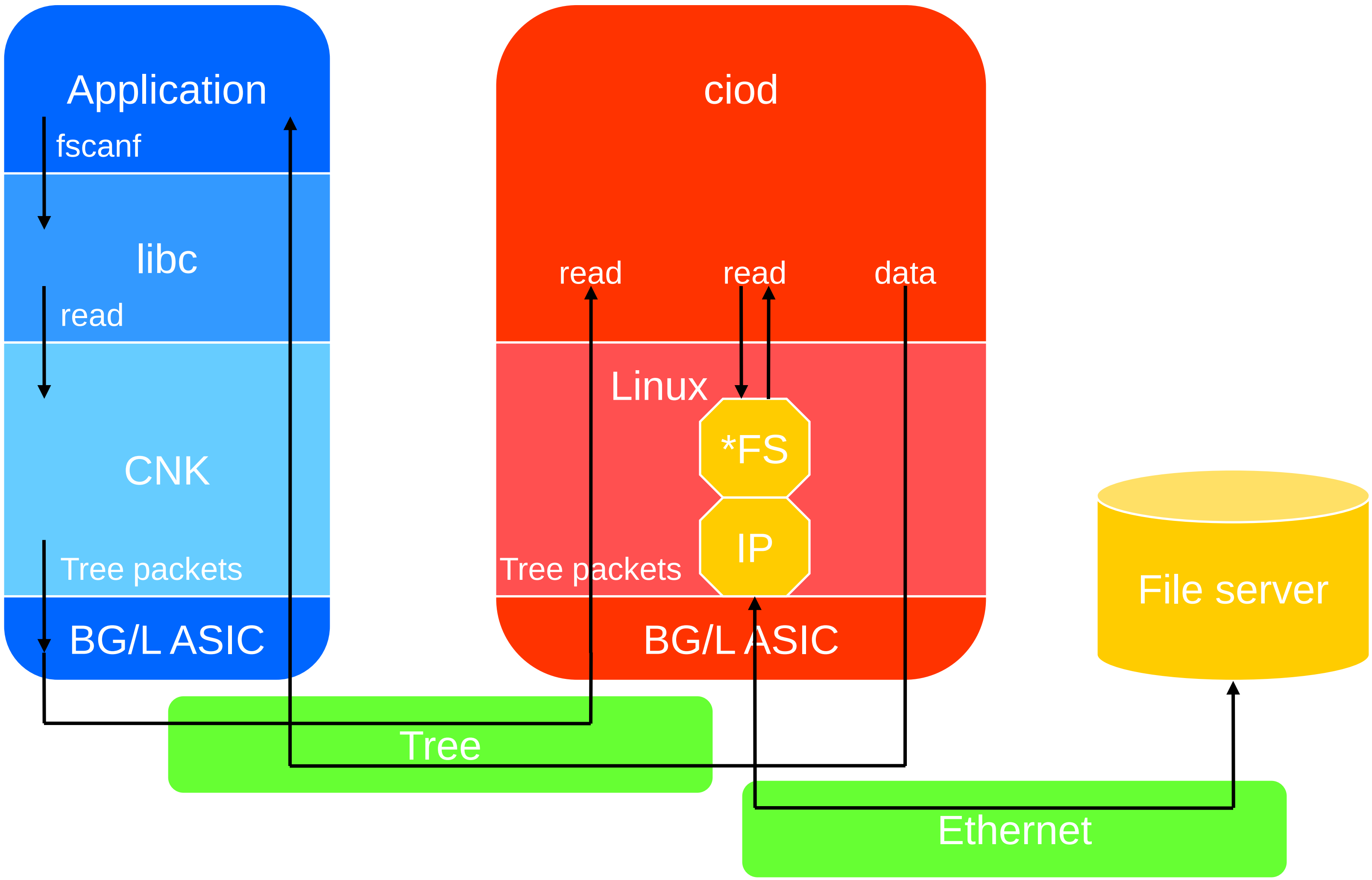
Partitioning

- **Partition = Subdivision of a single Blue Gene system**
- **Partitions are software defined**
- **Torus, Collective and Barrier networks are completely isolated from traffic from other partitions**
- **A single job runs on a partition**
  - Jobs never share resources or interfere with each other
- **Custom kernels may be booted in a partition**

Blue Gene System Architecture



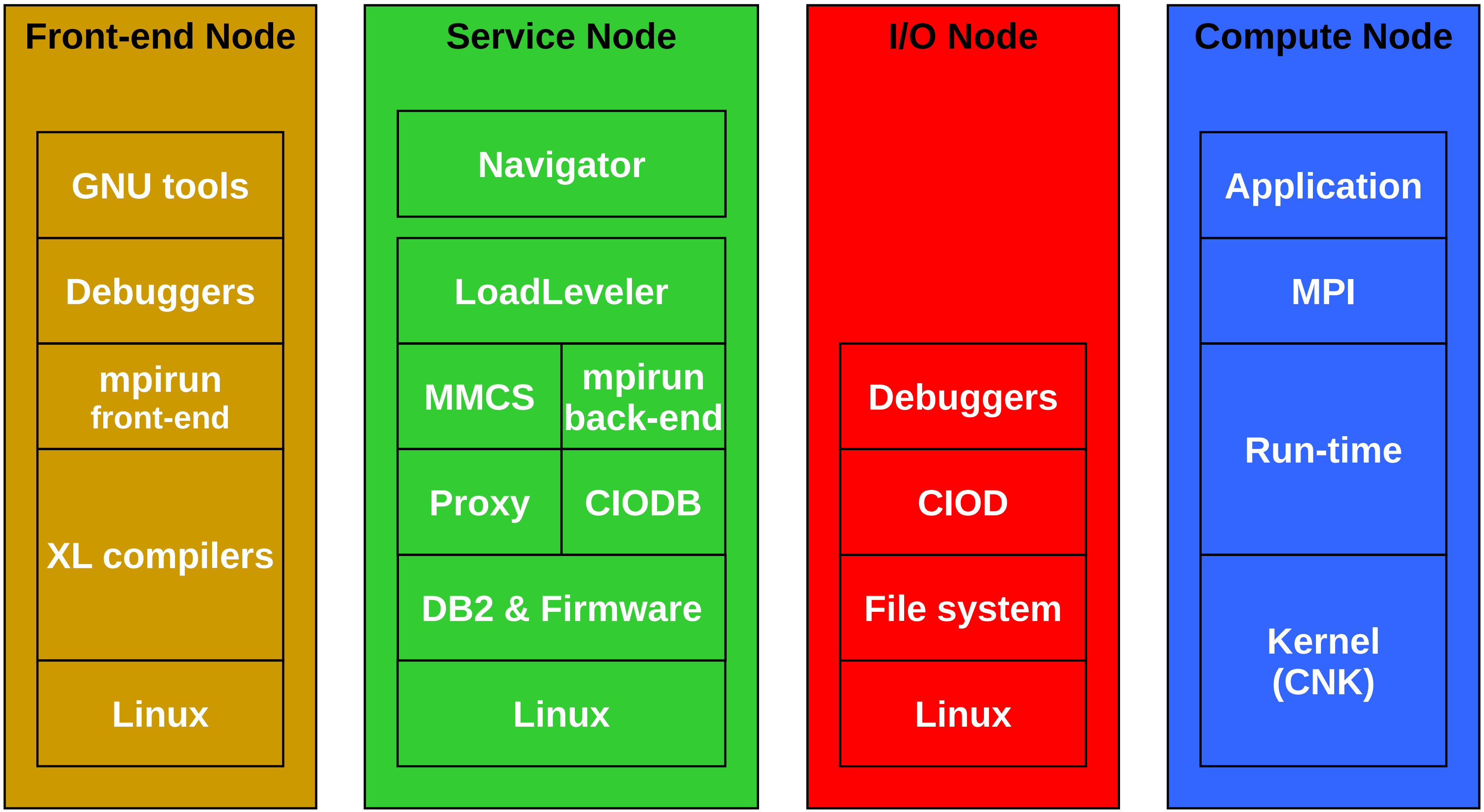
How File I/O works in Blue Gene applications



What's new with BG/P …

- **Torus DMA and numerous communication library optimizations**
- **pthreads and OpenMP support**
- **CNK application compatibility with Linux**
  - Dynamic linking
  - Use of mmap for shared memory
  - Protected readonly data and application code
  - Protection for stack overflow
  - Full socket support (client and server)
  - Better Linux compatibility in ciod on the I/O node
- **MPMD**
  - mpiexec supports multiple executables
  - Some restrictions: executable specified per pset
- **Numerous control system enhancements**

Blue Gene Software Stack Review



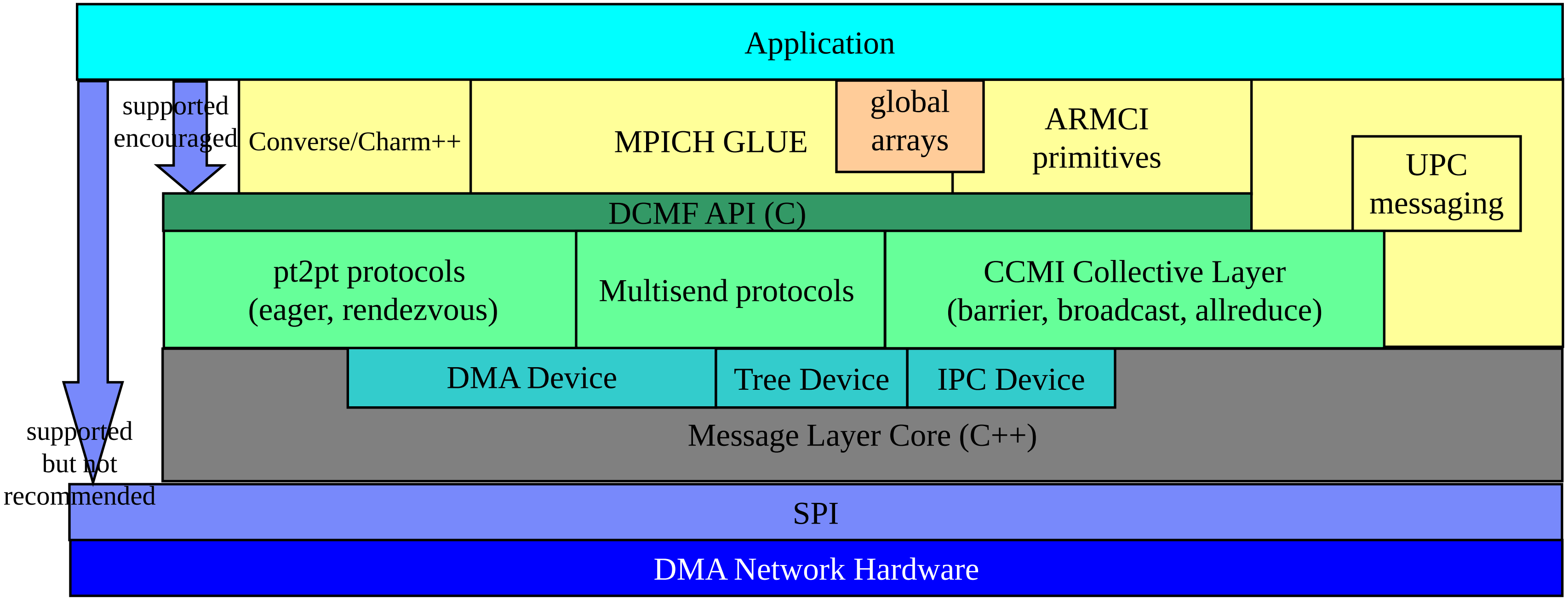
CNK System Calls

- **Direct Implementation**
  – exit, time, getpid, getuid, alarm, kill, times, brk, getgid, geteuid, getegid, getppid, sigaction, setrlimit, getrlimit, getrusage, gettimeofday, setitimer, getitimer, sigreturn, uname, sigprocmask, sched_yield, nanosleep, set_tid_address, exit_group
- **Implementation through forward to I/O Node**
  – open, close, read, write, link, unlink, chdir, chmod, lchown, lseek, utime, access, rename, mkdir, rmdir, dup, fcntl, umask, dup2, symlink, readlink, truncate, ftruncate, fchmod, fchown, statfs, fstatfs, socketcall, stat, lstat, fstat, fsync, llseek, getdents, readv, writev, sysctl, chown, getcwd, truncate64, ftruncate64, stat64, lstat64, fstat64, getdents64, fcntl64
- **Restricted Implementation**
  – mmap, munmap, clone, mutex

Programming Models & 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 front-end nodes running Linux – compilation, job submission, debugging
    - Compute Node Kernel provides look and feel of a Linux environment
      - POSIX system calls (with some restrictions)
      - BG/P adds pthread support, additional socket support,
    - Tools – support for debuggers, MPI tracer, profiler, hardware performance monitors, visualizer (HPC Toolkit), PAPI
    - Dynamic libraries
    - Python 2.5
- **Aggregate Remote Memory Copy (ARMCI), Global Arrays (GA), UPC, …**
- **Restrictions (lead to significant scalability benefits)**
  - 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, but on-demand linking
  - MPMD model limitations

Blue Gene/P Messaging Framework



Multiple programming paradigms supported

MPI, Charm++, ARMCI, GA, UPC (as a research initiative)

SPI : Low level systems programming interface

DCMF : Portable active-message API

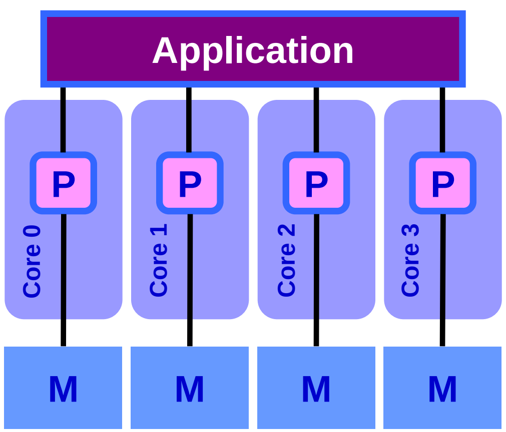
Programming Models & Execution Modes

- **Programming Models**
- **3 Types of Partition**
  - HPC
  - HTC (High Throughput Computing) with CNK – no MPI
  - HTC with CNL Kernel (Compute Node Linux) , IP address/compute Node, no MPI
- **Execution Modes**
  - SMP, DUAL, VN

Blue Gene/P Execution Modes (HPC and CNK HTC)

Quad Mode

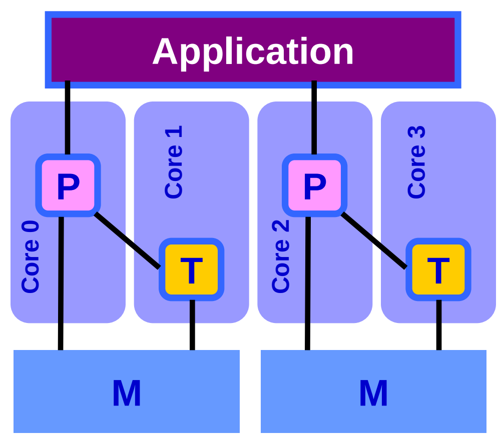
- Previously called Virtual Node Mode
- All four cores run one MPI process each
- No threading
- Memory / MPI process = ¼ node memory
- MPI programming model



Memory address space

Dual Mode

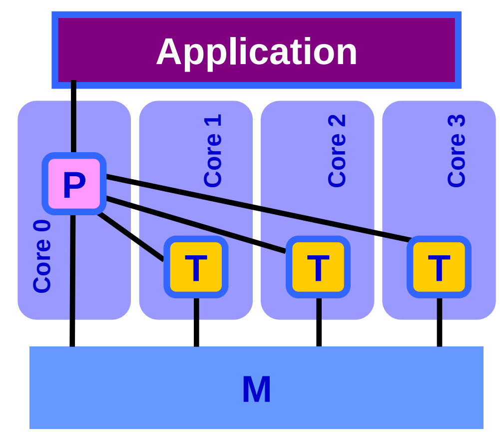
- Two cores run one MPI process each
- Each process may spawn one thread on core not used by other process
- Memory / MPI process = ½ node memory
- Hybrid MPI/OpenMP programming model



Memory address space

SMP Mode

- One core runs one MPI process
- Process may spawn threads on each of the other cores
- Memory / MPI process = full node memory
- Hybrid MPI/OpenMP programming model



Memory address space

High Throughput Computing (HTC) Mode

- **Many applications that run on Blue Gene today are "embarrassingly (pleasantly) parallel"**
  - They do not fully exploit the torus for MPI communication, since that is not needed for their problem
  - They just want a very large number of small tasks, with a coordinator of results
- **High Throughput Computing Mode on Blue Gene**
  - Enables a new class of workloads that use many single-node jobs
  - Leverages the low-cost, low-energy, small footprint of a rack of 1,024 compute nodes
    - Capacity machine ("cluster buster"): run 4,096 jobs on a single rack in virtual node mode (VN)
- **New HTC CNL mode with full Linux kernel on each Compute Node (from BG/P driver V1R3)**
    - Compute Node is seen as a regular Linux SMP system
    - Number of Processes and/or Threads is under user control
    - SSH session on Compute Node becomes possible

Blue Gene/P Execution

- **UNICORE, LoadLeveler & Environment Variables**

- **HPC Partition**
  - MPIRUN, MPIEXEC Commands
  - `mpirun -cwd ~/hello_world -exe hello_world -mode SMP -np 128 -env " OMP_NUM_THREADS=4 XLSMPOPTS=spins=0:yields=0:stack=64000000"`

- **HTC Partition**
  - SUBMIT Command
  - IBM Simple Scheduler

STDIN / STDOUT / STDERR Support

- **STDIN, STDOUT, and STDERR work as expected. You can pipe or redirect files into mpirun and pipe or redirect output from mpirun, just as you would on any command line utility. STDIN may also come from the keyboard interactively.**
- **Any compute node may send STDOUT or STDERR data. Only MPI rank 0 may read STDIN data.**
- **Mpirun always tells the control system and the C runtime on the compute nodes that it is writing to TTY devices. This is because logically MPIRUN looks like a pipe; it can not do seeks on STDIN, STDOUT, and STDERR even if they are coming from files.**
- **As always, STDIN, STDOUT and STDERR are the slowest ways to get input and output from a supercomputer. Use them sparingly.**
- **STDOUT is not buffered and can generate a huge overhead for some applications. It advices for such applications to buffer the stdout with**

  -enable_tty_reporting

IBM Scheduler for HTC Glide-In to LoadLeveler

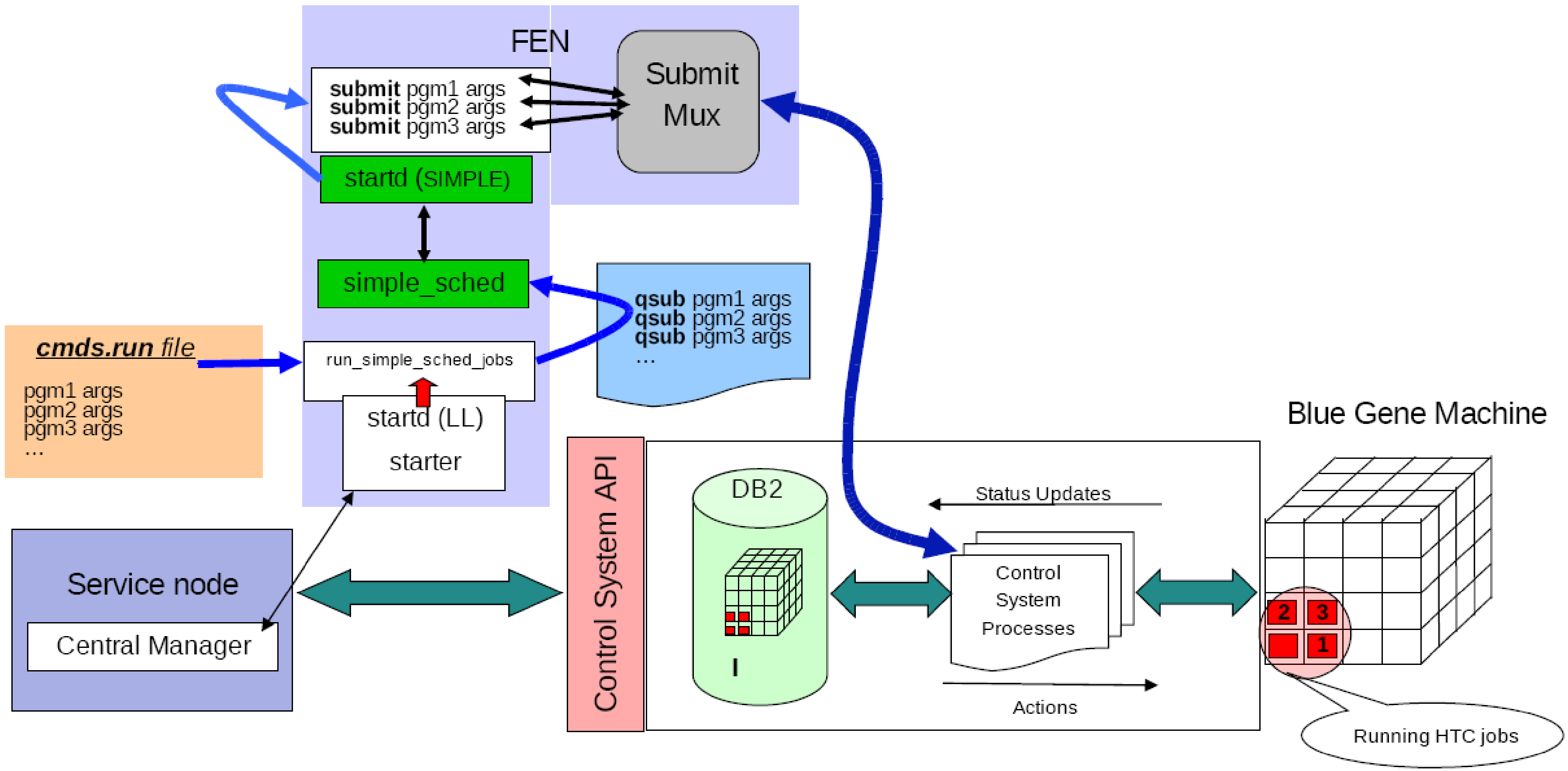


Figure 2: HTC Scheduler as LoadLeveler glide-in

XL Runtime Environment variables

- **Number of threads per MPI tasks**
  - OMP_NUM_THREADS
- **OMP and SMP runtime options**
  - XLSMPOPTS=option1=XXX:option2=YYY: ….
    - schedule= static[=n]:dynamic [=n]:guided [=n]:affinity [=n]
    - parthds= number of threads (with –qsmp in the compilation, should be set for esslsmp)
    - stack= amount of space in Bytes for the all thread stack (defauklt 4MB)
    - For performance
    - spins= number of spins before a yield
    - yields= number of yields before a sleep
      - On BGP spins=0:yields=0
- **512 mpi tasks, 4 OpenMP threads with 64MBytes stack per thread**
  - mpirun –mode SMP –np …. –env ‘’ OMP_NUM_THREADS=4 XLSMPOPTS=spins:0:yields=0:stack=67108864’’
- **A lot of other potential XL runtime variables: c.f. XL documentation**

Blue Gene/P Compilation

- **Compilers**
- **Mathematical Libraries**

GNU Tools and Libraries

- **GCC 4.1.1**
  - Standard System Locations
    - /bgsys/drivers/ppcfloor/gnu-linux/
      - powerpc-bgp-linux-gcc
  - No support for OMP in this version
  - Specificities
    - gfortran replaces the older g77
      - -std=legacy emulates previous behavior
- **GLIBC 2.4**
  - Thread support enabled
    - Link Option: –lpthread
  - Standard System Location
    - /bgsys/drivers/ppcfloor/gnu-linux/bin
      - powerpc-bgp-linux-addr2line
        - Decode more BG/P syscalls
      - gdb / gdbserver
      - python2.5

IBM XL Compilers for Blue Gene

- **XL Fortran 11.1**
- **XL C/C++ 9.0**
- **Standard System Locations**
  - /opt/ibmcmp/xlf/bg/11.1/
  - /opt/ibmcmp/vacpp/bg/9.0/
- **Specificities**
  - Fortran 2003 standard supported
    - xlf2003
  - Blue Gene/P Optimization Options
    - -qarch=450[d] –qtune=450

IBM XL Compilers for Blue Gene / MPI Wrappers

- **Included in the BG/P driver**
- **Standard System Location**
  - /bgsys/drivers/ppcfloor/comm/bin
    - mpicc / mpicxx / mpixlc / mpixlcxx / mpixlc_r / mpixlcxx_r
    - mpixlf2003 / mpixlf77 / mpixlf90 / mpixlf95 / mpif77 / mpixlf2003_r / mpixlf77_r / mpixlf90_r / mpixlf95_r
  - /bgsys/drivers/ppcfloor/comm/bin/fast
    - Fast versions
      - The 'fast' scripts use libraries that are built with assertions turned off and MPICH debug turned off
    - Recommendations
      - Build and test with original wrappers (/comm/bin/mpi*)
      - Make sure you have successful runs of application before switching
    - Using these shaves roughly a microsecond off of most communications calls (which can be 25% improvement)

IBM Compilers Key Options

- **-qarch=440, 450**
  - Generates only instructions for one floating point (option minimal option with blrts_)
- **-qarch=440d, 450d**
  - Generates only instructions for 2 floating point pipes
- **-qtune=450/440**
- **-O3 (-qstrict)**
  - Minimal level for SIMDization
- **-O3 -qhot (=simd)**
- **-O4 (-qnoipa)**
- **-O5**
- **-qdebug=diagnostic**
  - Provides details about SIMDization, only with –qhot
- **-qreport –qlist –qsource**
  - Provides pseudo-assembler code in .lst generated file
- **-qsmp (-qsmp=omp + -qdirectives=…) for OpenMP**

  - Recommended: mpiXXX**_r -g -qarch=450d –qtune450 –qmaxmem=-1 –O3 [-qhot]**

Compiler Options

- **-C, -qcheck**
  - Checks each reference to an array element, array section, or character substring to ensure the reference stays within the defined bounds of the entity.
- **-g, -qdbg**
  - Generates debug information for use by a symbolic debugger.
- **-qdpcl**
  - Generates symbols that tools based on the IBM Dynamic Probe Class Library (DPCL) can use to see the structure of an executable file.
- **-qextchk**
  - Generates information to perform type-checking at both compile and link time to ensure consistency between declarations and references.
- **-qflttrap**
  - Determines what types of floating-point exception conditions to detect at run time.
- **-qformat (XLC)**
  - Warns of possible problems with string input and output format specifications.
- **-qinitauto**
  - Initializes uninitialized automatic variables to a specific value, for debugging purposes.
- **-qkeepparm**
  - When used with -O2 or higher optimization, specifies whether function parameters are stored on the stack.
- **-qobject**
  - Specifies whether to produce an object file or to stop immediately after checking the syntax of the source files.
- **-qoptdebug**
  - When used with high levels of optimization, produces files containing optimized pseudocode that can be read by a debugger.
- **-qxflag=dvz**
  - Causes the compiler to generate code to detect floating-point divide-by-zero operations

MASS Library

- **MASS = Mathematical Acceleration Subsystem**
- **Collection of tuned mathematical intrinsic functions**
- **Components**
  - MASS
    - Scalar functions
  - MASSV
    - Vector functions (Single & Double precision)
- **Standard System Location**
  - /opt/ibmcmp/xlmass/bg/4.4/bglib
    - libmass.a
    - libmassv.a
  - /opt/ibmcmp/xlmass/bg/4.4/include
- **Link Syntax**
  - -L/opt/ibmcmp/xlmass/bg/4.4/bglib –lmass -l massv

ESSL Library

- **ESSL = Engineering and Scientific Subroutine**
- **Optimization library and intrinsics for better application performance**
  - Serial Static Library supporting 32-bit applications
  - Callable from C/C++ and Fortran
  - PowerPC 450 optimized
- **Components**
  - ESSL
  - ESSL SMP
    - SMP Support
  - Parallel ESSL
- **Standard System Location**
  - /opt/ibmmath/essl/4.4
    - ./lib/libesslbg.a
    - ./lib/libesslsmpbg.a
- **Link Syntax**
  - Fortran Linker
    - -L/opt/ibmmath/essl/4.3/lib –lesslbg [–lesslsmpbg]
  - C/C++ Linker
    - -L/opt/ibmmath/essl/4.3/lib –lesslbg [–lesslsmpbg] -L/opt/ibmcmp/xlf/bg/11.1/lib -lxlf90_r -lxlopt -lxl -lxlfmath

A word on FFTs ...

- Use ESSL or FFTW on BlueGene
  - Only FFTW-2.1.5 and ESSL are optimized for “double hummer”
  - Must compile code with -qarch=450d (to get alignment right)
- 3D volumetric FFTs
  - Easy out-of-the-box solution
    - Free P3DFFT package from San Diego Supercomputing Center (SDSC)
      - Uses a 2d 'pencil' decomposition on top of FFTW or ESSL
      - IBM PESSL or FFTW(MPI) only support 1d slab decomposition
      - Proven scalability up to 32k MPI tasks for up to 4096**3 FFT sizes
  - Roll your own 'pencil' decomposition
    - Beneficial for very large FFTs (1024**3 and larger) if transpositions are blocked for L3 cache and MPI Datatypes are used
      - Must interleave FFT transpositions with programs fourier space code
      - 50% speedup over P3DFFT package possible
    - Sample code should be available on NIC website next week

Available Tools Summary

- **Integrated Tools**
  - Personality
  - Compiler Options
  - Kernel Functions
  - Core Processor
  - Core Files + addr2line
  - GDB
  - Hardware counters
- **Supported Commercial Software**
  - TotalView Debugger
  - Allinea DDT
  - Juelich’s tools, +++

Personality / Provided Information

- **personality.Network_Config.[X|Y|Z]nodes**
  - Nb X / Y / Z Nodes in Torus
- **personality.Network_Config.[X|Y|Z]coord**
  - X / Y / Z Node Coordinates in Torus
- **Kernel_PhysicalProcessorID()**
  - Core ID on Compute Node (0, 1, 2, 3)
- **BGP_Personality_getLocationString(&personality, location)**
  - Location string
    - Rxx-Mx-Nxx-Jxx

- **Two Include Files**
  - #include <common/bgp_personality.h>
  - #include <common/bgp_personality_inlines.h>
    - In Directory: /bgsys/drivers/ppcfloor/arch/include
- **Structure**
  - _BGP_Personality_t personality;
- **Query Function**
  - Kernel_GetPersonality(&personality, sizeof(personality));

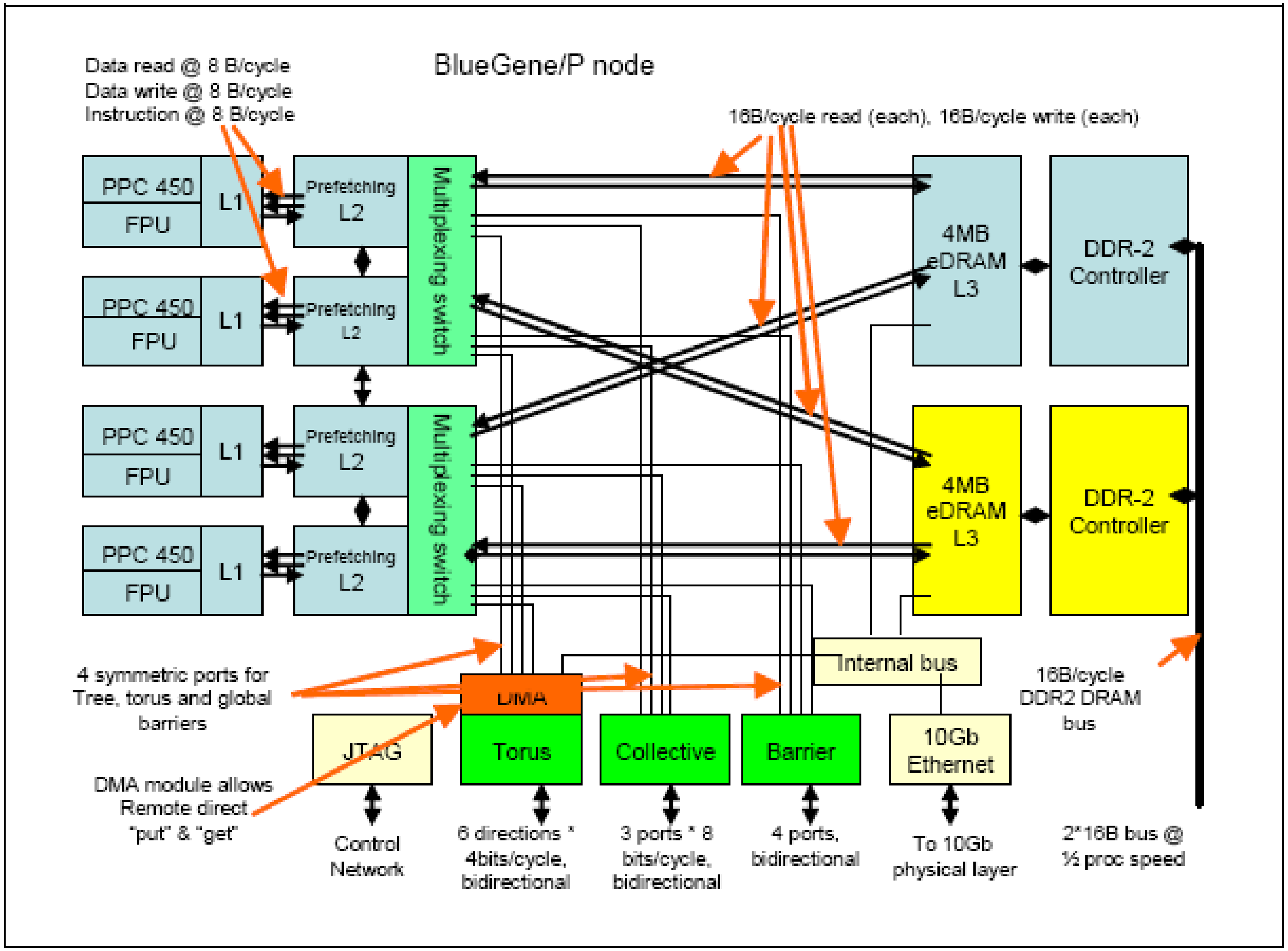
Personality / Example

```
#include <spi/kernel_interface.h>
#include <common/bgp_personality.h>
#include <common/bgp_personality_inlines.h>

int main(int argc, char * argv[]) {
   int taskid, ntasks;
   int memory_size_MBytes;
   _BGP_Personality_t personality;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &taskid);

   Kernel_GetPersonality(&personality,
   sizeof(personality));
   memory_size_MBytes =
   personality.DDR_Config.DDRSizeMB;
   printf("Memory size = %d MBytes\n",
   memory_size_MBytes);
   node_config =
   personality.Kernel_Config.ProcessConfig;
   if (node_config == _BGP_PERS_PROCESSCONFIG_SMP)
   printf("SMP mode\n");
   else if (node_config ==
   _BGP_PERS_PROCESSCONFIG_VNM) printf("Virtual-node
   mode\n");
   else if (node_config ==
   _BGP_PERS_PROCESSCONFIG_2x2) printf("Dual
   mode\n");
   else printf("Unknown mode\n");

   xcoord =
   personality.Network_Config.Xcoord;
   ycoord =
   personality.Network_Config.Ycoord;
   zcoord =
   personality.Network_Config.Zcoord;
   xsize =
   personality.Network_Config.Xnodes;
   ysize =
   personality.Network_Config.Ynodes;
   zsize =
   personality.Network_Config.Znodes;
   pset_num  =
   personality.Network_Config.PSetNum;
   pset_size =
   personality.Network_Config.PSetSize;
   pset_rank =
   personality.Network_Config.RankInPSet;
   BGP_Personality_getLocationString(&person
   ality, location);
   procid = Kernel_PhysicalProcessorID();
}
```

Blue Gene/P Advanced Topics

- **Blue Gene/P Memory**
- **Advanced Compilation with IBM XL Compilers**
- **SIMD Programming**
- **Communications Frameworks**
- **Checkpoint/Restart**
- **I/O Optimization**

Blue Gene/P ASIC



Memory Cache Levels

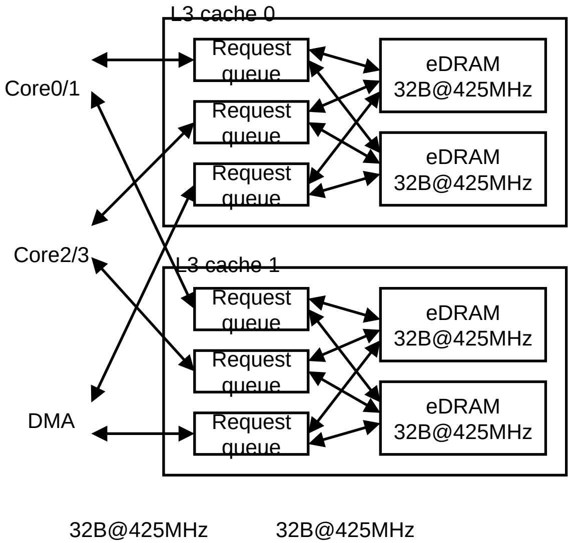
Cache	Total per node	Size	Replacement Policy	Associativity
L1 Instruction	4	32 KB	Round-Robin	64-way set-associative 16 sets 32B line size
L1 Data	4	32 KB	Round-Robin	64-way set-associative 16 sets 32B line size
L2 PreFetch	4	14x256 B	Round-Robin	Fully associative (15-way)128 B Line size
L3	2	2x4 MB	Least Recently Used	8way associative 2 Bank Interleaved 128 B Line

L1-2-3 Caches

- **L1 Cache**
  - Avoid instructions prefetching in L1. Reduce the number of prefetch streams below 3
  - Programmer can use the XL compiler directives or assembler instruction (dcbt)
  - Without an intensive reused of the L1 and register, memory subsystem is not allowed to feed the double FPU

- **L2 Cache**
  - 128B line Prefetch engine, up to 7 streams
  - L2 boosts the overall performance and does not require any special software provisions.

- **L3 Cache**
  - **Request queue per port**
    - 8 read requests
    - 4 write requests
  - **4 eDRAM banks per chip, each containing independent**
    - directory
    - 15 entry 128B-wide write combining buffer
  - **Hit under Miss resolution**
    - Limit defined by request queues and write buffer
      - Up to 8 read misses per port
      - Up to 15 write misses per write combining buffer
  - **Limitation: banking conflict (possibility to configure dedicated L3/core – need IBM Lab support)**



Bottlenecks

- **L2 – L3 switch**
  - Not a full core to L3 bank crossbar
  - Request rate and bandwidth limited if two cores of one dual processor group access the same L3 cache
- **4 memory module-internal banks (4x512 MB)**
  - 4 banks on 512Mb DDR modules
  - Burst-8 transfer (128B): 16 cycles
  - Page open, access, precharge: 64 cycles
  - Peak bandwidth only achievable if accessing 3 other banks before accessing the same bank again

Main Memory Banking Optimization Example

- **For sequential access, two arrays used in a single operation must not be aligned on the same bank**

```
parameter (n=12800000)
real(8) x(n), y(n), w(n)
…..
do j=1,n
x(j) = x(j) + y(j)*w()
Enddo
…
```

BG memory tuning →

```
parameter (n=12800000, offset=16)
real(8) x(n+offset), y(n+2*offset), w(n)
…..
do j=1,n
x(j) = x(j) + y(j)*w()
Enddo
…
```


CNK / Shared Memory Support

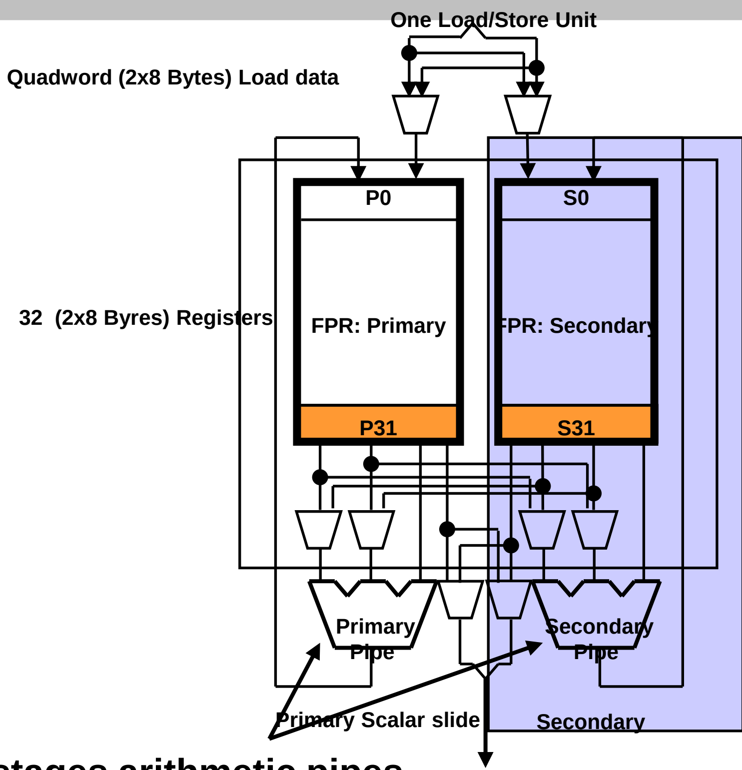
- **Shared Memory is supported in Virtual and Dual mode**
- **Normal theme: do it the Linux way…**
- **shm_open() standard interface**
  - Allocate:
    - fd = shm_open(SHM_FILE, O_RDWR, 0600);
    - ftruncate(fds[0], MAX_SHARED_SIZE);
    - shmptr1 = mmap(NULL, MAX_SHARED_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
  - Deallocate:
    - munmap(shmptrl, MAX_SHARED_SIZE);
    - close(fd)
    - shm_unlink(SHM_FILE);

CNK / Persistent Memory

- **Persistent memory is process memory that retains its contents from job to job.**
- **To allocate persistent memory, the environment variable BG_PERSISTMEMSIZE=X must be specified,**
  - where X is the number of 1024*1024 bytes to be allocated for use as persistent memory.
- **In order for the persistent memory to be maintained across jobs, all job submissions must specify the same value for BG_PERSISTMEMSIZE.**
- **The contents of persistent memory can be re-initialized during job startup by either changing the value of BG_PERSISTMEMSIZE or by specifying the environment variable BG_PERSISTMEMRESET=1.**
- **The following new kernel function was added to support persistent memory:**
  - persist_open()

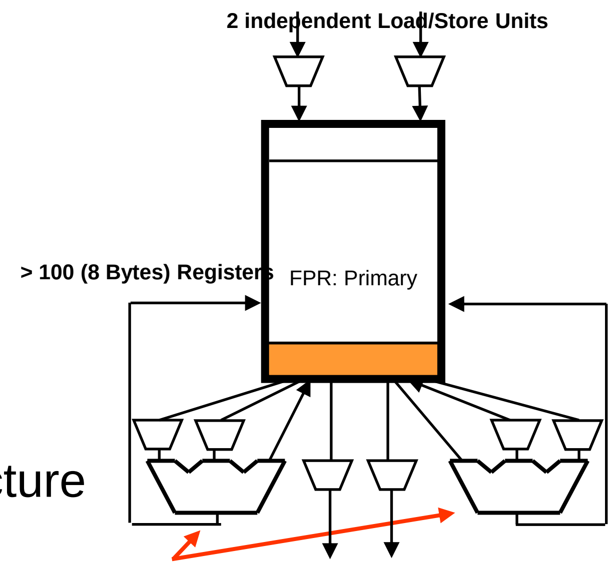
Dual FPU Architecture

- 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



5 stages arithmetic pipes
Running the same operation

Power 5/6 Architecture



5 stages arithmetic pipes
Running different operations

SIMD Implementation Control

- **-qarch=440d, 450d generates instructions for 2 floating point pipes**
- **Obtaining and understanding an object code listing**
  - qdebug=diagnostic
    - Provides details about SIMDization, only with –qhot
  - qreport –qlist –qsource
    - Provides pseudo-assembler code
    - .lst file
- **BG/P Hardware counters**
- **or a profiling tool**

Program to Disable Simdization

- **Turn off the right optimizations**
  - do not invoke TPO
    - compile with –O3 without “-qhot” or “-qipa”
    - add –qhot=nosimd at –O4, -O5
  - disable simdization (with at least –O3 –qhot=simd)
    - for a loop
      - #pragma nosimd
      - !IBM* NOSIMD
    - completely
      - -qhot=nosimd / -qdebug=nosimd

SIMD Instructions for Double FPU

- **Main constraint: 16 Bytes data alignment**
- **Help the compiler with directives**

Fortran

```
call alignx(16,x(1))
call alignx(16,y(1))
!ibm* unroll(10)
do i = 1, n
y(i) = a*x(i) + y(i)
end do
```

C :

```
double * x, * y;  double x[256] __attribute__((aligned(16));
#pragma disjoint (*x, *y)
__alignx(16,x);
__alignx(16,y);
#pragma unroll(10)
for (i=0; i<n; i++) y[i] = a*x[i] + y[i];
```

How to Enable SIMD Instructions

- **Use Libraries (ESSL)**
  - Surely the most efficient
- **Use compiler options**
  - -qarch=450d -qhot=simd
- **Align data on 16 Bytes and add compiler directives**
  - alignx, #pragma disjoint
  - Versioning with alignement testing
- **Implement SIMD instrinsics (versioning)**

```
void reciprocal_roots (double *x, double *f, int n)
{
/* are both x & f 16 byte aligned? */
if ( ((((int) x) | ((int) f)) & 0xf) == 0) /* This could also be done as:
if (((int) x % 16 == 0) && ((int) f % 16) == 0) */
    aligned_ten_reciprocal_roots (x, f, n);
else
     ten_reciprocal_roots (x, f, n);
}
```


Division on BG (similar for Square Root)

```
Do i=1,N
   Y(I) = 1./X(I)
Enddo
```

Normal computation ~30 cycles

1. Use compiler option –qhot –O3, check implementation with –qsource –qlist
2. Use libmass: vsrec function

3. Use Hardware approximation

Power hardware able to provide an estimation 10-4 in 1 cycles
using FRE or FRSQRT
Blue Gene Double able to provide 2 estimation per cycles (FPRE, FPRSQRT)

```
Do i=1,N
   e = FPE(X(I))
   T1 = e + e*(1-X(I)*e)   for single precision
   Y(I) = T1 + e*(1-X(I)*T1) for double precision
Enddo
```

4. Use SWDIV_NOCHECK function

```
Do i=1,N
   Y(I) = swdib_nocheck(1.,X(I)
Enddo
```

5. Use SIMD intrinsic functions (LOADFP, FPRE, FPNMSUB, FPMUL, FPMADD, STOREFP, …)

Checkpoint/Restore Library / 1

- **Checkpoint/Restore implemented as an application library that saves state in a file per node**
- **General use by modifying the application code**
  - At the beginning of the code
    - BGCheckpointInit("/path/for/checkpoint/files")
  - At any point in the code
    - <barrier>
      BGCheckpoint()
      <barrier>
- **The checkpoint itself saves state in a unique file per node with a sequence number**

Checkpoint/Restore Library / 2

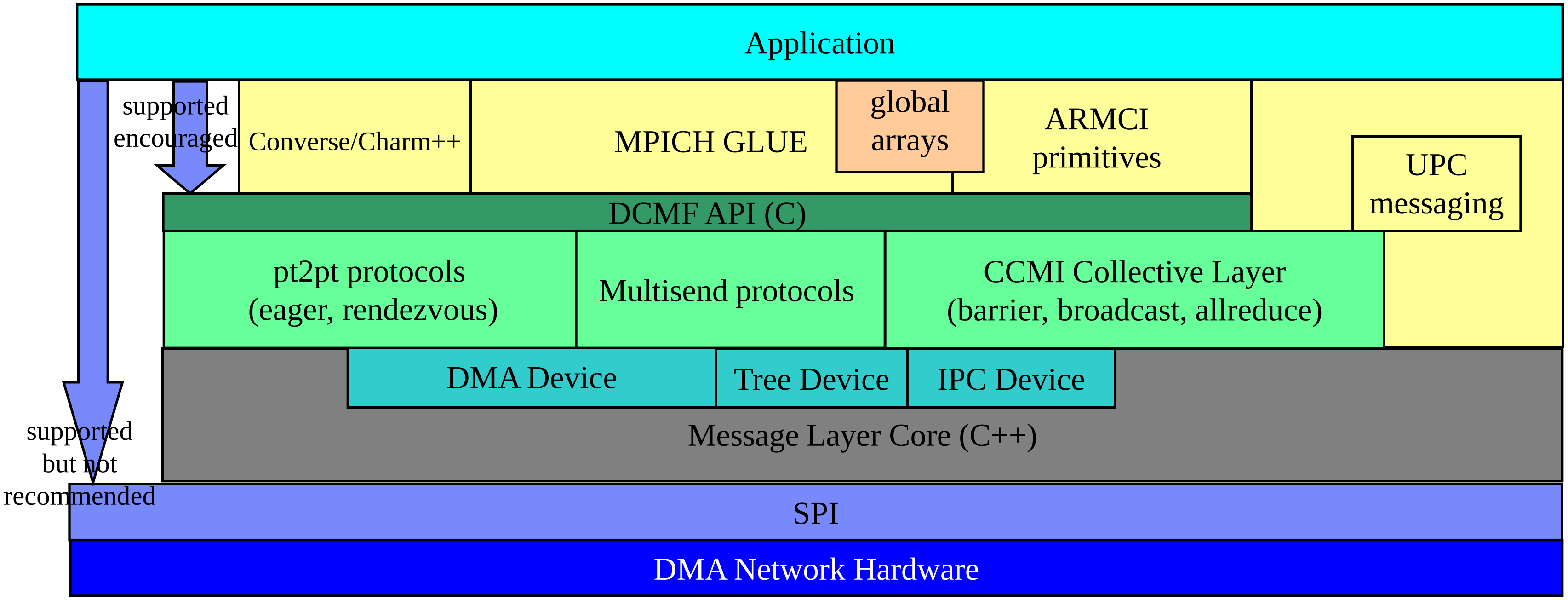
- **For restart, the same job is launched again with the environment variables**
- **BG_CHKPTRESTARTSEQNO and BG_CHKPTDIRPATH set to the appropriate values. The BGCheckpointInit() function checks for these environment variables and, if specified, restarts the application from the desired checkpoint.**

BG_CHKPT_ENABLED	Is set (to 1) if checkpoints are desired; otherwise it is not specified.
BG_CHKPTDIRPATH	Default path to keep checkpoint files.
BG_CHKPTRESTARTSEQNO	Set to a desired checkpoint sequence number from where a user wants the application to restart. If set to zero, each process restarts from its individual latest consistent checkpoint. This option must not be specified, if no restart is desired.

Blue Gene/P Parallel Libraries

- **Shared Memory**
- **Message Passing**

Blue Gene/P Messaging Framework



Multiple programming paradigms supported

MPI, Charm++, ARMCI, GA, UPC (as a research initiative)

SPI : Low level systems programming interface

DCMF : Portable active-message API

MPI on BGP vs BGL

- **MPI 2 Standard compliance**
  - SMP with thread mode multiple
    - Thread mode multiple is default
    - Simpler thread modes need to be initialized with MPI_Thread_init
  - Dual and quad mode also supported
- **DMA engine to optimize communication**
  - Improved progress semantics over Blue Gene/L
  - DMA makes sends and receives packets while the processor is busy computing
  - Drive network harder: can keep ~5 links busy for near neighbor traffic in *both* directions
  - Allows overlap of computation and communication
- **Comm-thread which is enabled on packet arrival to make BGP MPI fully progress compliant**
  - Allow tag matching of Rzv messages in comm thread
  - Enabled through environment variable DCMF_INTERRUPTS=1
  - SMP mode has 1 commthread, dual has two and quad (VN) mode has four comm threads
  - Commthreads are scheduled by interrupts
- **Built on top of the DCMF messaging API**
  - 3+ us latency and 370 MB/s bandwidth per link

Communication Libraries

- **MPI**
  - MPICH2 1.0.4p2
  - Optimized collectives built on DCMF
  - Redbook Application development
- **DCMF (Deep Computing Message Framework)**
- **SPI (System Programming Interfaces)**

MPI Point-to-Point Routing

- **The route from a sender to a receiver on a torus network has two possible paths:**
  - Deterministic routing
  - Adaptive routing
- **Selecting deterministic or adaptive routing depends on the protocol that is used for the communication.**
- **The Blue Gene/P MPI implementation supports three different protocols:**
  - **MPI short protocol**
    - The MPI short protocol is used for short messages (less than 224 bytes), which consist of a single packet. These messages are always deterministically routed. The latency for eager messages is around 3.3 μs.
  - **MPI eager protocol**
    - The MPI eager protocol is used for medium-sized messages. It sends a message to the receiver without negotiating with the receiving side that the other end is ready to receive the message. This protocol also uses deterministic routes for its packets.
  - **MPI rendezvous protocol**
    - Large (greater than 1200 bytes) messages are sent using the MPI rendezvous protocol. In this case, an initial connection between the two partners is established. Only after that will the receiver use direct memory access (DMA) to obtain the data from the sender. This protocol uses adapStive routing and is optimized for maximum bandwidth. Naturally, the initial rendezvous handshake increases the latency.
- DCMF_EAGER **variable**

Optimized Collectives

Collective (All are non-blocking at the DCMF API level)	**Network**		
	Torus via DMA	**Collective**	**Barrier**
Barrier	Binomial algorithm	N/A	Uses Global Interrupt wires to determine when nodes have entered the barrier.
Sync Broadcast (BG/L style broadcast where all nodes need to reach the broadcast call before data is transmitted)	Rectangular algorithm Binomial algorithm	Uses a Collective Broadcast via spanning class route. To prevent unexpected packets, broadcast is executed via global BOR.	N/A
All-to-All(v)	Each node sends messages in randomized permutations to keep the bisection busy.	N/A	N/A
Reduce	Rectangular algorithm Binomial algorithm	Same as Collective All-reduce, but with no store on non-root nodes.	N/A
All-reduce	Rectangular algorithm Binomial algorithm	Uses a Collective Broadcast via spanning class route. Native tree operations, single and double pass double precision floating point operations.	N/A
All-gather(v)	Broadcast, reduce, and all-to-all based algorithms. Algorithm used depends on geometry, partition size, and message size.		N/A

MPI Environment Variables / MPICH2

- **DCMF_EAGER**
  - This value, passed through atoi(), is the smallest message that will be sent using the Rendezvous protocol. This is also one greater than the largest message sent using the Eager protocol.
  - (Synonyms: DCMF_RVZ, DCMF_RZV)
- **DCMF_COLLECTIVES**
  - When set to "0", this will disable the optimized collectives. When set to "1", this will enable the optimized collectives. Otherwise, this is left at the default.
  - (Synonyms: DCMF_COLLECTIVE)
- **DCMF_TOPOLOGY**
  - When set to "0", this will disable the optimized topology routines. When set to "1", this will enable the optimized topology routines. Otherwise, this is left at the default.
- **DCMF_ALLREDUCE**
  - Possible options: MPICH, BINOMIAL, RECTANGLE, TREE
- **DCMF_INTERRUPTS**
  - When set to "0", interrupts are disabled. Message progress occurs via polling. When set to "1", interrupts are enabled. Message progress occurs in the background via interrupts, and/or by polling. Default is "0".
  - (Synonyms: DCMF_INTERRUPT)

Alltoall

- **DMA based alltoall**
- **Uses adaptive routing on the network**
  - Optimized for latency and bandwidth (latency 0.9 us/destination)
  - 96% of peak throughput on a midplane
- **Alltoall performance for large messages optimized by the all-to-all mode in the DMA device**
  - DCMF_FIFOMODE=ALLTOALL (20% more)

Mapping Tasks to physical Nodes or Cores

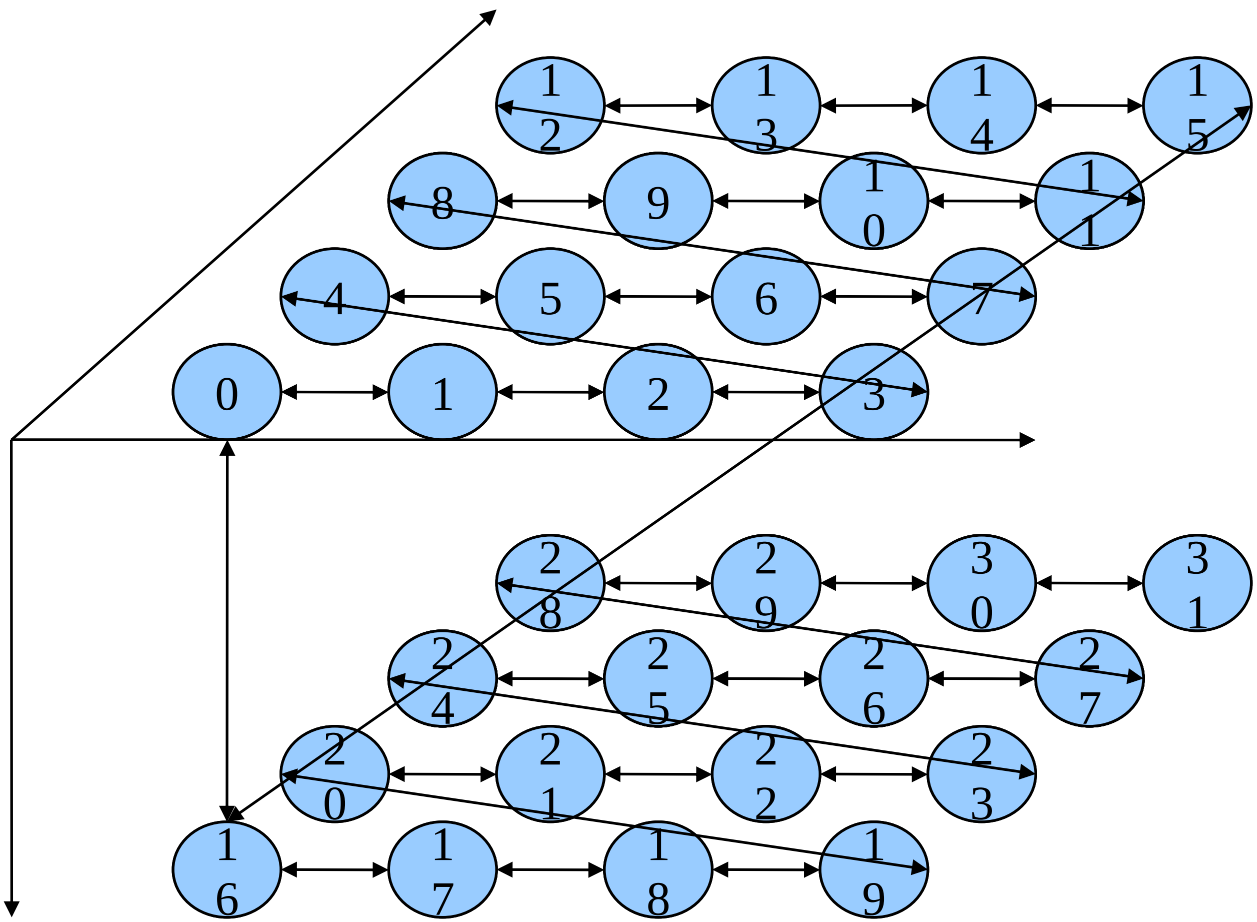
- **Three Options**
- BG_MAPPING environment variable
  - **Equivalent to BGLMPI_MAPPING on BGL. Allows user to specify mapping as an environment variable. Options are: TXYZ, TXZY, TYXZ, TYZX, TZXY, TZYX, XYZT, XZYT, YXZT, YZXT, ZXYT, ZYXT or a path to a mapping file**
    - Rotations and point-mirroring operators (XYZT is default)
- -mapfile option of mpirun
  - <CR> separated list of physical core coordinates per task
    - **x0 y0 z0 t0**
    - **X1 y1 z1 t1**
    - **...**
    - **XN yN zN tN**
- Use cartesian communicators and let BG MPI **reorder** the tasks

Mapping for Nearest-Neighbor-Communication

- **powers of 2 on BlueGene in partitioning are king !**
- 1d -> 4d (AxBxCxD) physical (smallest partition is 4x4x2x(4))
  - Use 'slithering snake' mapping on small partitions (no torus)
  - TXYZ mapping on torus partitions probably fine
- 2d (NxM) -> 4d physical (AxBxCxD)
  - Try to decouple problem into two 1d -> 2d mappings,
    then use 'slithering snake'
- 3d (LxNxM) -> 4d physical (AxBxCxD)
  - Try to map one L,N or M to a product of two physical dimensions
    then map the remaining dimensions one-to-one
  - What, if that's not possible
    - Try to split the D (intra-node dimension) into 2x2 and see if that works out
  - 4d -> 4d
    - Really only works well if all dimensions can be mapped one-to-one,
      then use the MAPPING variable

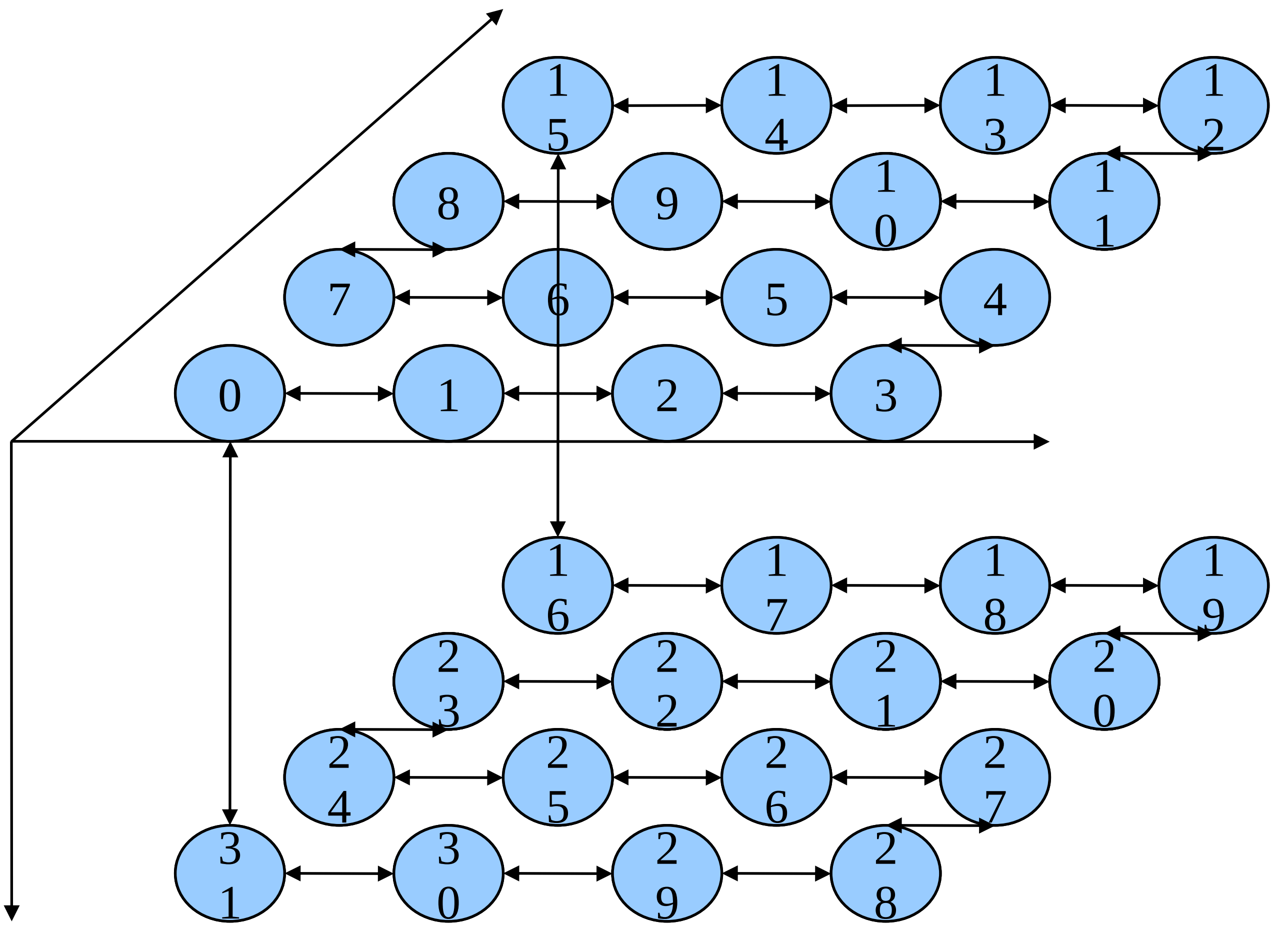
Mapping Examples (1a)

- 1d ring communicator mapping (default XYZT): worst case is 7 hops



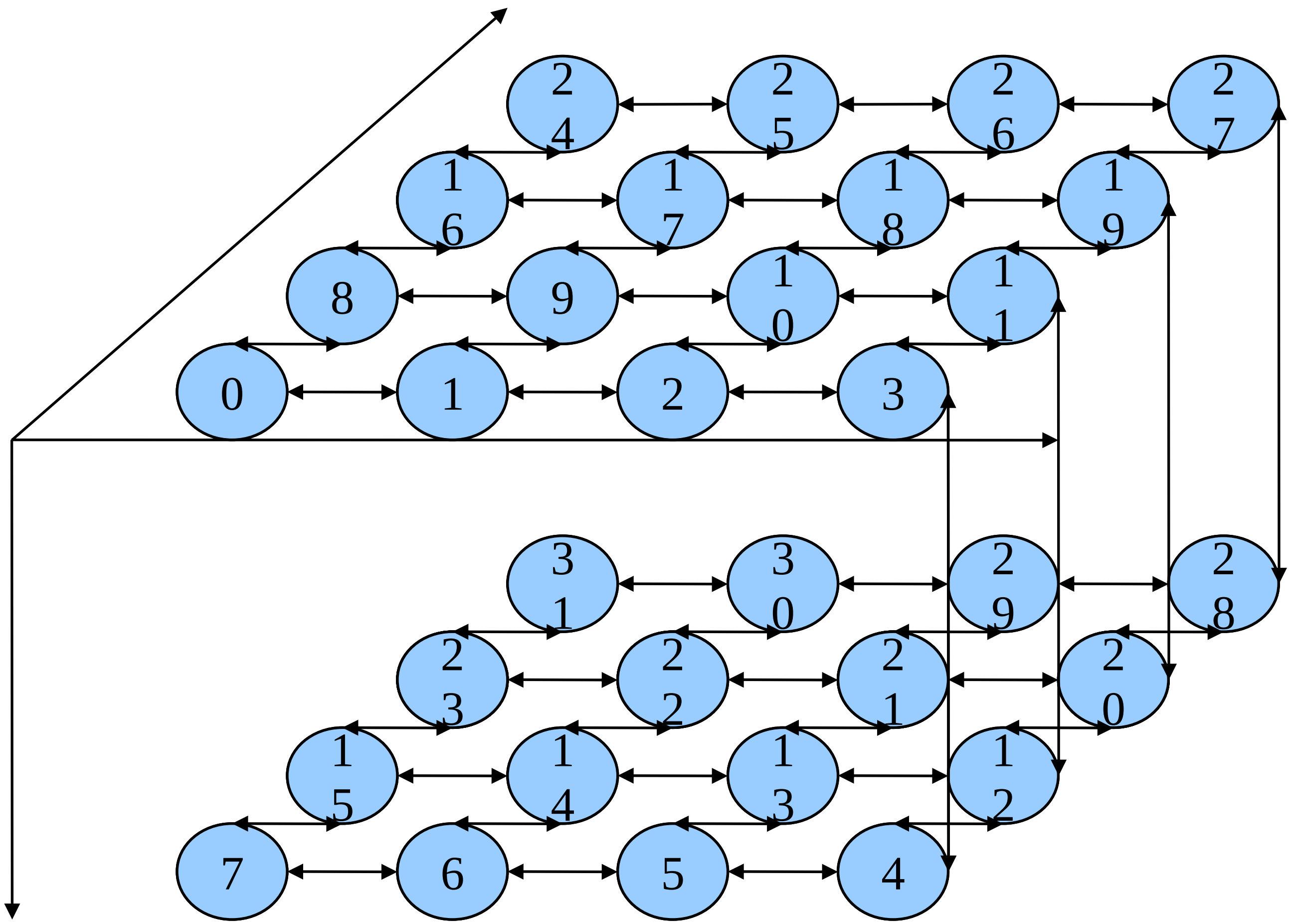
Mapping Examples (1b)

- 1d ring communicator mapping (slithering snake) -> single hop between neighbors



Mapping Examples (2)

- 8x4 2d communicator mapping (folded paper) -> single hop between neighbors



MPI topologies

- **Defined in ch06 of the MPI 1.1 standard**
- **Idea: attach knowledge of application's inherent topology (2D grid, etc.) to an MPI *communicator***
  - *inside* the program, not *external* mapping like --mapfile
- **Create a new communicator based on**
  - input communicator (e.g. MPI_COMM_WORLD)
  - description of the app's topology (shape, periodicity)
  - programmer may allow the runtime to reorder, or not
    - advice is to ALLOW reordering to optimize placement of tasks onto the torus network
- **Then, use new communicator in your MPI calls, instead of the usual MPI_COMM_WORLD**

Using cartesian communicators

- **Simplest case: just rely on the re-ordered rank**
  - if your program's coord/rank calculation is "natural", this is often good enough: MPI runtime has done the placement
- **Use MPI topology coords/rank transformations:**
  - MPI_Cart_rank() and MPI_Cart_coords()
    - mainly convenience, makes program easier to read
- **Express neighborhood in app's coords not rank:**
  - MPI_Cart_shift() – again mainly convenience
- **Use collectives on cartesian sub-communicators:**
  - use MPI_Cart_sub() to create row or column sub-communicators (call similar to MPI_Comm_split)
  - then use these sub-comms in collectives
  - this *may* exploit special BlueGene hardware features like **multicast along a torus axis**

BlueGene/P MPI runtime support of MPI topologies

- **Only cartesian topologies, no graphs**
  - most apps are cartesian, and BlueGene is a torus
- **input communicator to MPI_Cart_Create() must be a rectangular part of the torus**
- **one- to three-dimensional topologies supported in all three execution modes (VN, DUAL and SMP)**
- **four-dimensional topologies only for DUAL or VNM mode**
  - one dimension must have size 2 (DUAL) or 4 (VNM)
- **higher-dimensional cartesian topologies and graphs are accepted, but result is a NO-OP (same as MPI_COMM_WORLD)**
- **DCMF_TOPOLOGY environment variable controls optimization:**
  - Set it to 1 → on (default), or 0 → off

Blue Gene/P MPI communicators

- int **MPIX_Cart_comm_create** (MPI_Comm *cart_comm)
  - This function creates a four-dimensional (4D) Cartesian communicator that mimics the exact hardware on which it is run
- int **MPIX_Pset_same_comm_create** (MPI_Comm *pset_comm)
  - This function is a collective operation that creates a set of communicators, where all nodes in a given communicator are part of the same pset
- int **MPIX_Pset_diff_comm_create** (MPI_Comm *pset_comm)
  - This function is a collective operation that creates a set of communicators, where no two nodes in a given communicator are part of the same pset

Deep Computing Messaging Framework (DCMF)

- **Low level message layer API**
- **Understands and exploits Blue Gene network hardware**
- **Implements various protocols**
  - Point-point low level message passing
  - Multisend to broadcast to multiple destinations
  - Remote get/put
  - Component Collective Messaging Interface
    - allows implementation of optimized collectives
  - Pluggable user provided protocols
- **Manages Threads**
- **Manages Mappings**
- **DMCF allows direct use as well as multiple higher level layers such as MPI, GA/ARMCI, to coexist in a single application**
- **/bgsys/drivers/ppcfloor/comm/lib, /bgsys/drivers/ppcfloor/comm/include**
- **Doxygen documentation:** http://bgweb.rchland.ibm.com/~jratt/

System Programming Interfaces (SPI)

- **Lowest level access to “bare metal”**
- **Building block for higher level layers**
- **Generally inline interfaces with direct hardware access**
- **May be used with higher level layers if *carefully* coordinated**
- **Not thread safe!**
- **Discouraged except in extreme cases such as QCD**
- **Examples:**
  - setup and start DMA on Torus
  - inject or receive packets on Collective
  - access to low level hardware “lockbox”
  - access to SRAM

- **Only Doxygen documentation (/bgsys/drivers/ppcfloor/arch/include)**