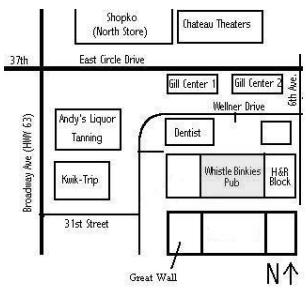
The Blue Gene/L Supercomputer: A Hardware and Software Story

Valentina Salapura, José Moreira
{salapura, jmoreira}@us.ibm.com
IBM T.J. Watson Research Center

The Blue Gene/L Team – Yorktown

The Blue Gene/L Team – Rochester, MN



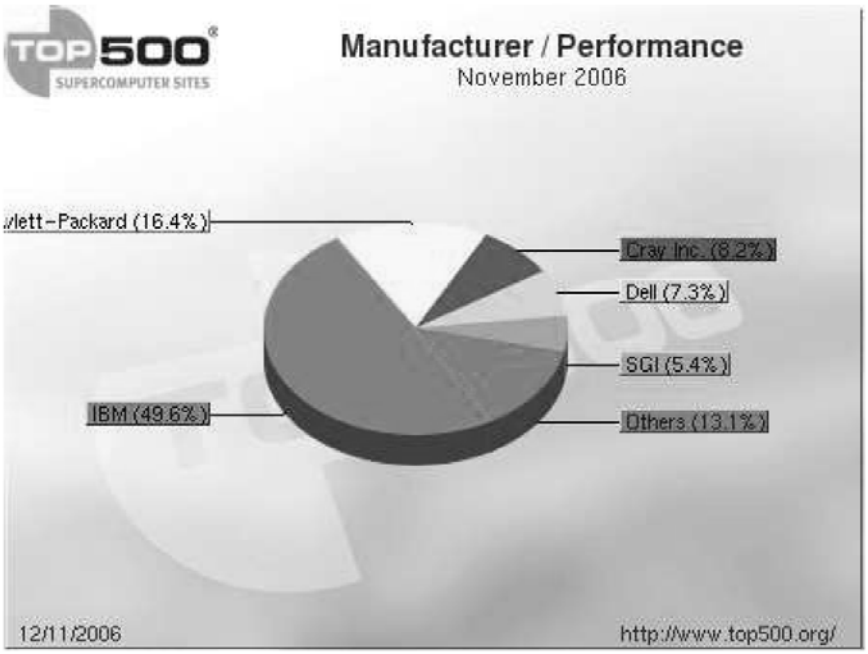
Phone: 507.289.9200
Address: 3210 Wellner Drive NE
Rochester, MN 55906

November 2006: 28th TOP10 of the TOP500 list

Year	Computer	Measured Tflop/s	Theoretical Peak Tflop/s	Number of Processors	% peak	Flops/ Hz	Processor MHz
2005	IBM BlueGene/L DD2, LLNL	280.6	367.00	131072	76	4	700
2006	Cray Redstorm, Sandia	101.4	127.411	26544	80	2	2400
2005	IBM BlueGene/L DD2, IBM Yorktown	91.29	114.688	40960	80	4	700
2006	IBM ASC Purple	75.76	92.78	12208	82	4	1900
2006	Mare Nostrum, IBM JS21 PowerPC970	62.63	94.21	10240	66	4	2300
2006	Dell PowerEdge 1850 Thunderbird, Sandia	53.00	64.97	9024	82	2	3600
2006	Bull Tera-10 - NovaScale 5160, Quadrics, CEA	52.84	63.80	9968	83	4	1600
2004	Columbia, SGI Altrix 3700, Itanium 2 for NASA	51.87	60.96	10160	84	4	1500
2006	TSUBAME Grid Cluster - Sun Fire x4600 Cluster	47.38	82.12	11088	58	2	2400
2006	Jaguar – Cray XT3, ORNL	43.48	54.20	10424	80	2	2600

IBM is the clear leader in the TOP500 list.

Manufacturer/Performance in November 2006



IBM is the clear leader in the TOP500 list.

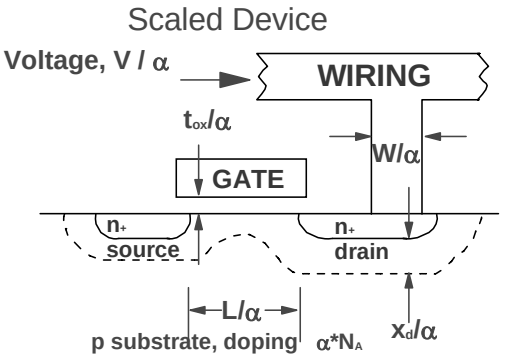
Outline

- **Blue Gene system motivation**
- **Architecture**
- **Packaging**
- **Reliability**
- **System software**
- **Applications – power and performance**

The Supercomputer Challenge

- **More Performance ➔ More Power**
  - systems limited by data center cooling capacity
    - new buildings for new supercomputers
  - FLOPS/W not improving from technology

CMOS Scaling

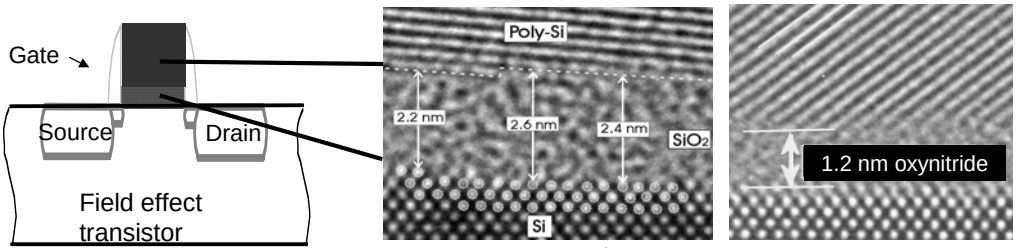


- **semiconductor scaling**
  - Dennard's scaling theory
  - Increases compute density
  - Enables higher compute speed

SCALING:		**RESULTS:**	
Voltage:	V/α	Higher Density:	$\sim\alpha^2$
Oxide:	t_{ox}/α	Higher Speed:	$\sim\alpha$
Wire width:	W/α	Power/ckt:	$\sim 1/\alpha^2$
Gate width:	L/α	**Power Density:**	
Diffusion:	x_d/α		**~Constant**
Substrate:	$\alpha * N_A$		

Source: Dennard et al., JSSC 1974.

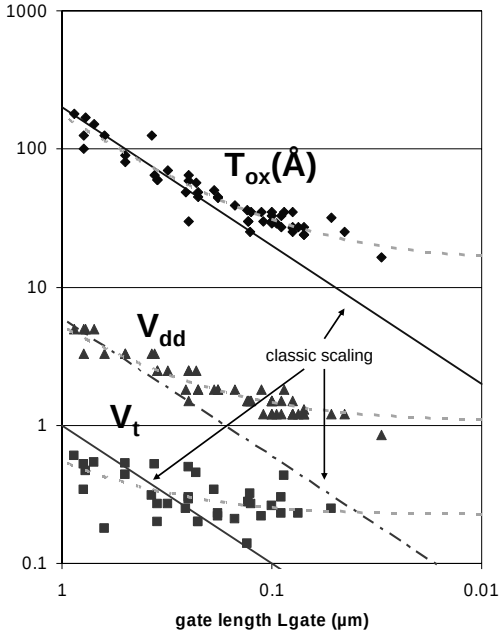
The End of Scaling – Atoms Don't Scale!



"Thick" gate oxide Scaled gate oxide

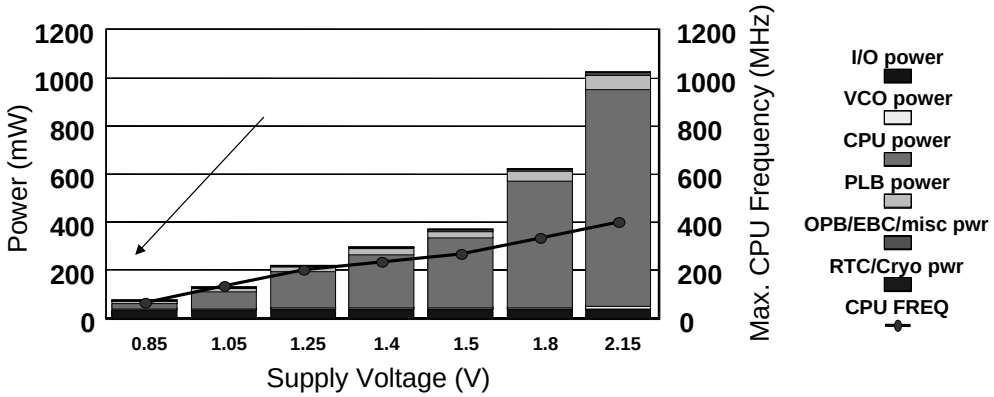
- **Consider the gate oxide in a CMOS transistor (the smallest dimensions today)**
  - Assume only 1 atom high "defects" on each surrounding silicon layer
  - for a "modern" 5-6 atoms thick oxide, 33% variability is induced
- **The bad news**
  - Single atom defects can cause local current leakage 10-100x higher than average
  - Oxides scaled below ~9 angstroms are too "leaky" and thus unreliable
- **The really bad news**
  - Such "non-statistical behaviors" are appearing elsewhere in technology

Voltage Scaling



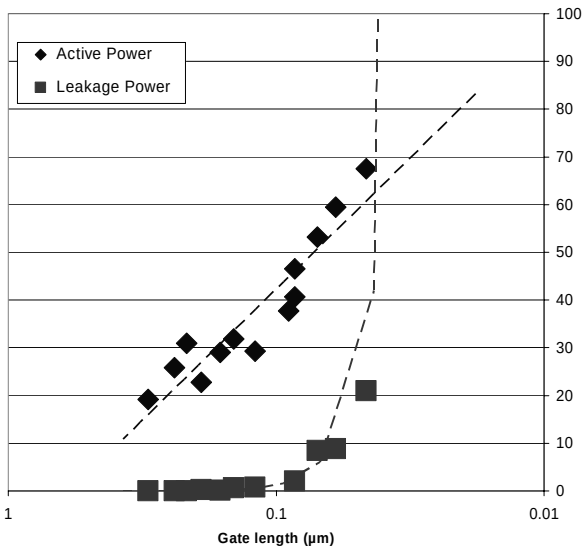
- **Yesterday**
  - deviate from "ideal" scaling: V not scaled according to scaling theory
  - additional performance & higher reliability at higher voltages
  - Old approach: Performance tailoring in manufacturing
- **Today**
  - supply voltage increase causes dramatic rise in power density (n^2)
  - can not be ignored any more
  - New approach: Performance tailoring in design

Voltage Scaling and Power Management



- **SoC 405LP power consumption breakdown**
  - used nominal part at 25 C°, Dhrystone 2.1 under Linux, dynamic power management applied
  - power dissipation dominated by CPU
- **Supply voltage increase causes dramatic rise in power density !!!**

Leakage Power



- **Yesterday**
  - Power to clock latches dominant power dissipation component
  - Old metric: Active power dominates
- **Today**
  - Power consumed even if not clocking latches
  - Leakage power has become a significant component
  - Must develop means to “disconnect” unused circuits
  - New metric: Leakage power significant contributor

The Supercomputer Challenge

- **More Performance ➔ More Power**
  - FLOPS/W not improving from technology

➔traditional supercomputer design hitting power & cost limits

Low Power Design

- **Compute density can be achieved only with low power design approach**
  - capacity of data center limited by cooling, not floor space
- **Focus on low power**
  - Air cooling – power budget per rack 25 KW

The BlueGene/L Concept

- **Parallelism can deliver higher aggregate performance**
  - Efficiency is key: (deliver performance / system power)
    - Power budget scales with peak performance
    - Application performance scales with sustained performance
- **Avoid scaling single core performance into regime with diminishing power/performance efficiency**
  - deliver performance by exploiting application parallelism
- **Focus design effort on improving efficient MP scaling**
  - e.g., special purpose networks for synchronization and communication
- **Compute density can be achieved only with low power design approach**
  - capacity of data center limited by cooling, not floor space

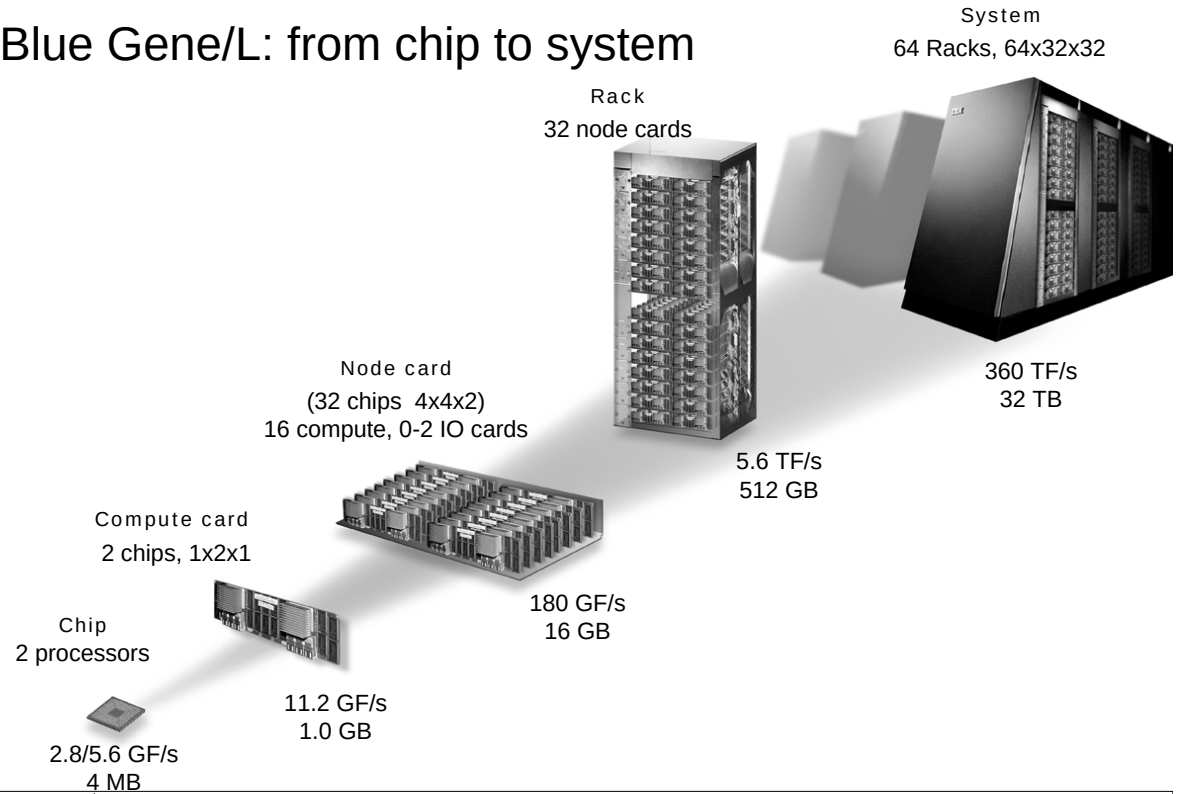
BlueGene/L Design Philosophy

- **Use standard embedded system-on-a-chip (SoC) design methodology**
- **Utilize PowerPC architecture and standard messaging interface (MPI)**
  - standard programming model
  - mature compiler support
- **Focus on low power**
  - Air cooling – power budget per rack 25 KW
- **Improve cost/performance (total cost/time to solution)**
  - Use & develop only two ASICs: node and link
  - Leverage industry-standard PowerPC design
- **Single-chip nodes, less complexity**
  - Enables high density

The BlueGene/L System

- A 64k-node highly integrated supercomputer
- 360 teraflops peak performance
- Strategic partnership with LLNL and high-performance computing centers
  - **Validate and optimize architecture using real applications**
  - **LLNL is accustomed to new architectures and experts in application tuning**
  - **Help us investigate the reach of this machine**
- Focuses on numerically intensive scientific problems
- “Grand challenge” science projects

Blue Gene/L: from chip to system



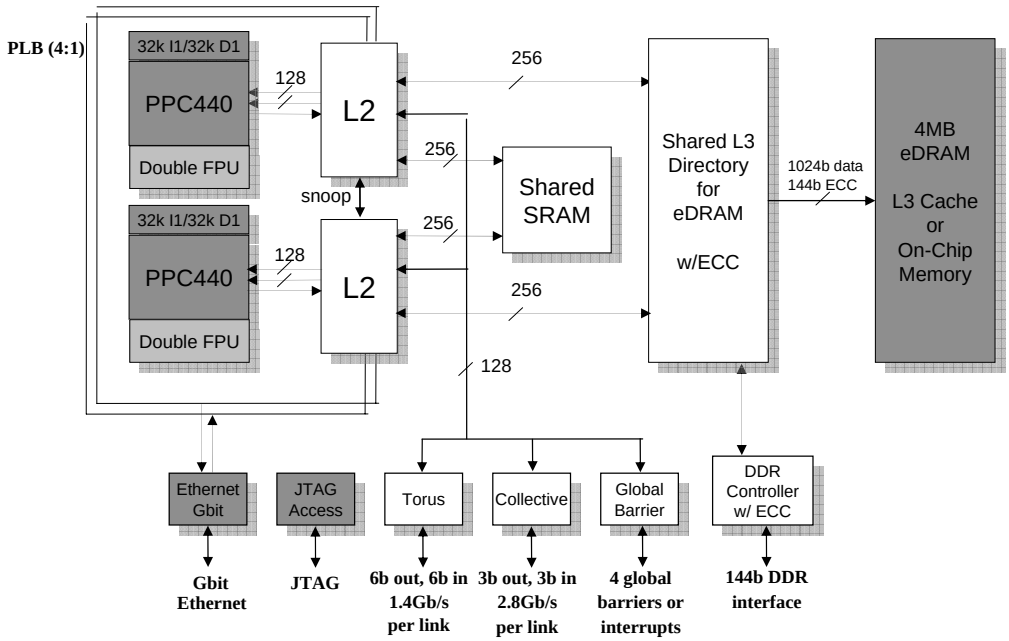
System Characteristics

- **Chip multiprocessor**
  - 2 PowerPC cores per chip
- **Data parallelism**
  - Double floating point unit for advanced SIMD operations
- **High integration**
  - 2 PowerPC cores + EDRAM cache + DDR memory interface + network interfaces on a single chip
- **High performance networks**
  - Directly on chip ➔ reduce latency
  - Multiple optimized, task-specific networks
    - Synchronization, data exchange, I/O

IBM

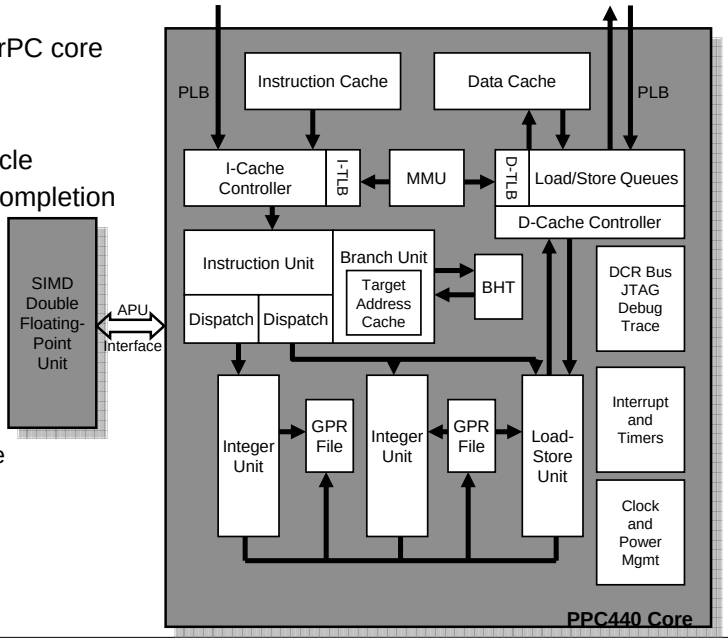
BlueGene/L Architecture

BlueGene/L Compute SoC ASIC



440 Processor Core Feature

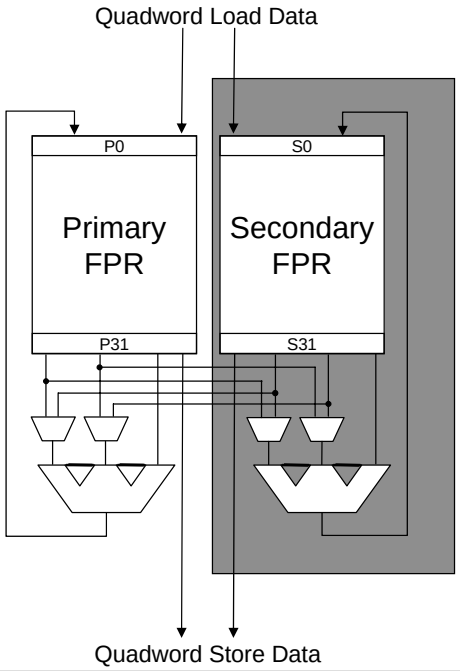
- High performance embedded PowerPC core
- 2.0 DMIPS/MHz
- Book E Architecture
- Superscalar: two instructions per cycle
- Out of order issue, execution, and completion
- 7 stage pipeline
- 3 Execution pipelines
- 32 32 bit GPR
- Dynamic branch prediction
  - BHT & BTAC
- Caches
  - ►32KB instruction & 32KB data cache
  - ►64-way set associative, 32 byte line
- 36-bit physical address
- 128-bit CoreConnect PLB Interface



Execution Pipelines

- **Three 440 execution pipelines**
  - I pipe - complex integer pipeline
    - All arithmetic, logical, branch operations including multiply, divide
    - System management instructions
      - interrupt, TLB management
    - 3 cycle latency, 1 cycle throughput
  - J pipe - simple integer pipeline
    - Most arithmetic and logical operations
  - L pipe - load/store pipeline
    - Load & store up to 16 byte
    - Supports little or big endian data
    - Cache management instructions
- **Fourth execution pipeline**
  - F pipe - FPU attached via APU (auxiliary processor unit) interface
    - Double precision 2-way SIMD FPU
    - 16 byte aligned load and store

Double Floating-Point Unit



- **Two replicas of a standard single-pipe PowerPC FPU**
  - 2 x 32 64-bit registers
- **Attached to the PPC440 core using the APU interface**
  - Issues instructions across APU interface
  - Instruction decode performed in Double FPU
  - Separate APU interface from LSU to provide up to 16B data for load and store
  - Datapath width is 16 bytes
    - Feeds two FPUs with 8 bytes each every cycle

Enhanced instruction set

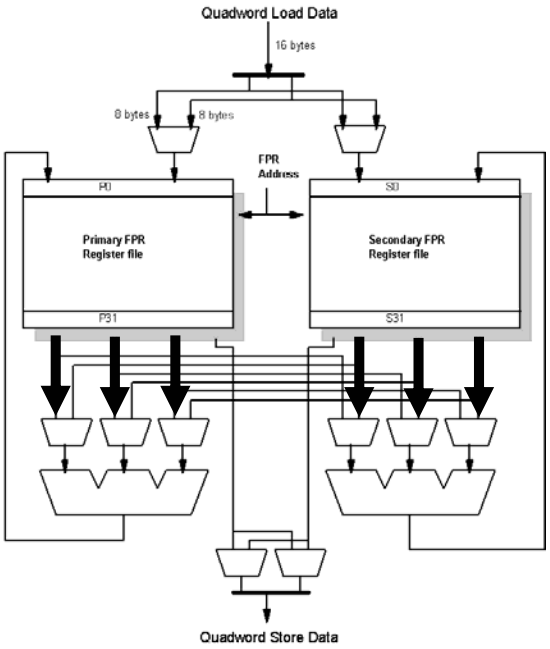
- **Enhanced ISA, includes instructions**
  - Executed on only one side
  - Simultaneously execute the same operation on both sides – SIMD instructions
  - Simultaneously execute two different operations of limited types on different data
- **Most instructions have a single cycle throughput**
  - All computation performed on double-precision format
    - Load/store operations convert single-precision operands to double precision
- **Two FP multiply-add operations per cycle**
  - 2.8 GF/s peak

FP instruction extension examples

Instruction	Mnemonic	Description	
		Primary	Secondary
Floating parallel multiply–ad	fpmadd	$P_T = P_A \cdot P_C + P_B$	$S_T = S_A \cdot S_C + S_B$
Floating parallel negative multiply subtrac	fpnmsub	$P_T = -P_A \cdot P_C + P_B$	$S_T = -S_A \cdot S_C + S_B$
Floating cross multiply–ad	fxmadd	$P_T = S_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Floating cross copy–primary multiply–ad	fxcpmadd	$P_T = P_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Floating cross copy–secondary multiply–ad	fxcsmadd	$P_T = S_A \cdot P_C + P_B$	$S_T = S_A \cdot S_C + S_B$
Asymmetrical cross copy–primary nsub–primary multiply–add	fxcpnpma	$P_T = -P_A \cdot P_C + P_B$	$S_T = P_A \cdot S_C + S_B$
Asymmetrical cross copy–secondary nsub–secondary multiply–add	fxcsnsma	$P_T = S_A \cdot P_C + P_B$	$S_T = -S_A \cdot S_C + S_B$
Floating cross cmplx nsub–primary multiply–add	fxcxnpma	$P_T = -S_A \cdot S_C + P_B$	$S_T = S_A \cdot P_C + S_B$
Floating cross cmplx nsub–secondary multiply–add	fxcxnsma	$P_T = S_A \cdot S_C + P_B$	$S_T = -S_A \cdot P_C + S_B$
Floating parallel reciprocal estimat	fpre	$P_T = \text{RecipEst}(P_B)$	$S_T = \text{RecipEst}(S_B)$
Floating parallel selec	fpsel	$P_T = P_A \,?\, P_C : P_B$	$S_T = S_A \,?\, S_C : S_B$
Floating secondary compare	fscmp		BF] = $S_A <> S_B$
Floating cross mov	fxmr	$P_T = S_B$	$S_T = P_B$

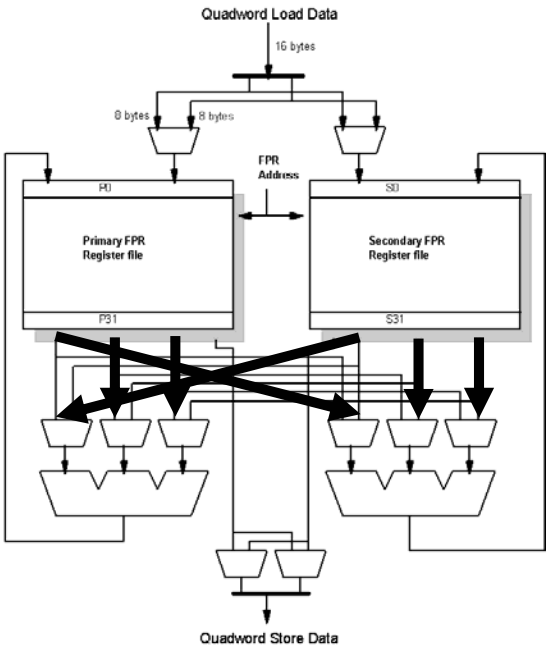
Double Floating Point Unit

- **Data from the same pipe**
- **Both pipes execute the same instruction**
- **Example instructions**
  - fpmadd
  - fpnmsub



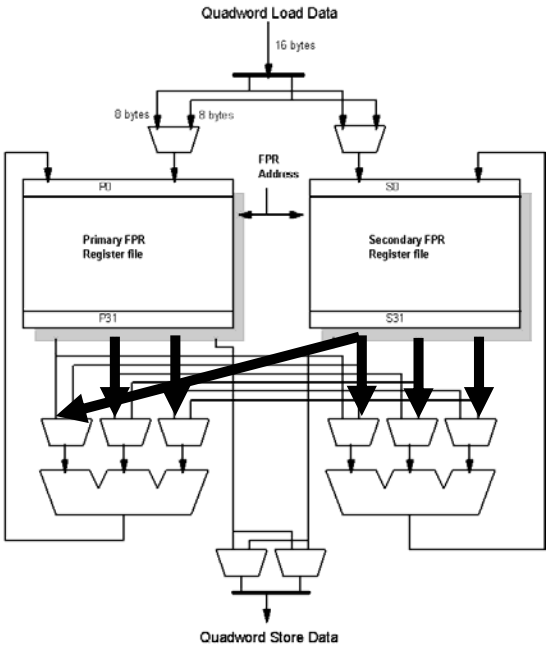
Double Floating Point Unit

- **Symmetrical data cross-copy**
  - One or two arguments from the other side
- **Both sides execute the same instruction**
- **Example instruction:**
  - fxmadd

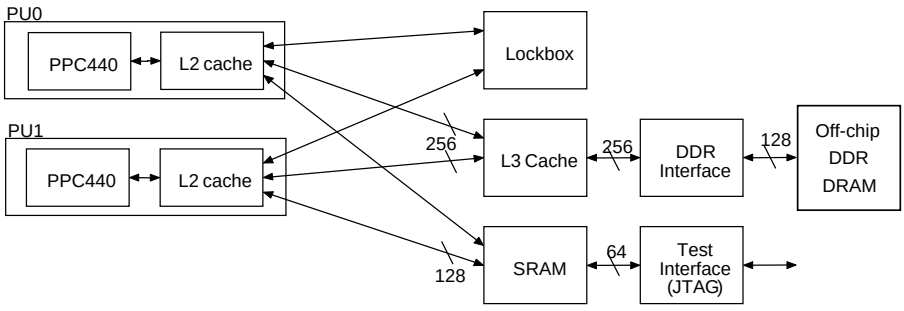


Double Floating Point Unit

- **Asymmetrical data cross-copy**
- **Different operations on each side**
- **Example instruction**
  - fxcsnpma
  - $P_T = S_A * P_C + P_B$
  - $S_T = -S_A * S_C + S_B$
- **Other combinations of data cross-copy also represented**



Memory Hierarchy



- **Main characteristics**
  - 32kB D&I private cache per processor
  - Small private L2 data prefetch caches
  - On-chip 4MB L3 cache
  - Access to main memory via L3 cache
  - SRAM for fast exchange of control information
  - Synchronization via lockbox semaphores

L1 Data and Instruction Caches

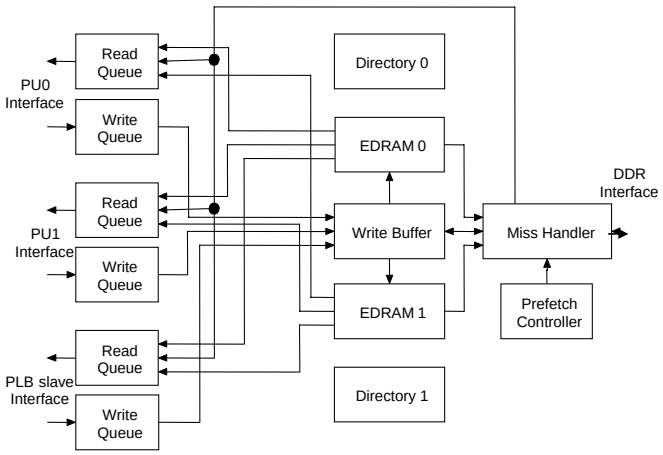
- **Private L1 data and instruction caches**
- **Instruction cache 32kB**
  - 32-byte cache line
- **Data cache 32 kB**
  - 32-byte cache line
  - Organized as 1024 lines, 64 way set associative
  - Round robin replacement policy
  - Write-back, write-through with, or without allocate modes
- **Handling cache miss**
  - non-blocking, with up to 3 outstanding loads
  - critical word first
- **Software managed coherency**
  - using lockbox and L1 cache invalidation before access
  - overall performance gain by eliminating snoop overhead

L2 Prefetch Cache

- **Private L2 cache**
  - L2 size < L1 size
  - non-inclusive
- **Separate L2R and L2W paths**
  - L2R prefetch buffer
  - L2W write buffer
- **Anchor point for memory hierarchy**
  - Coherency point
    - Memory hierarchy coherent from L2 downwards
  - Transition from private per-core cache to wide shared eDRAM
- **L2R organization:**
  - 128 byte line size, 16 lines
  - Fully associative

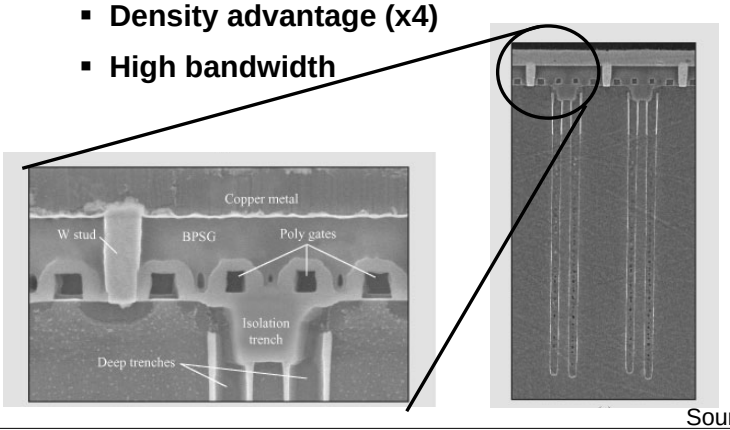
L3 Cache Implementation

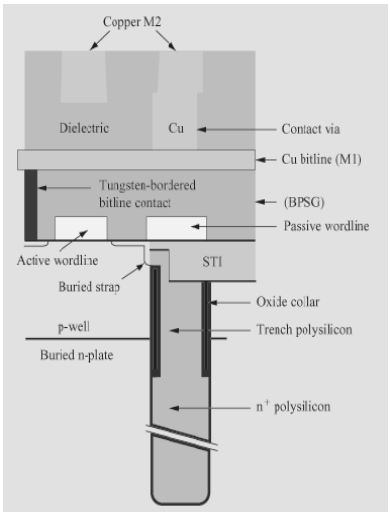
- **On-chip 4 MB shared L3 cache**
- **Use EDRAM**
- **Two-way interleaved**
  - 2MB EDRAM per bank
  - 8-way set-associative, LRU
  - 32k lines
  - 128-byte lines
- **ECC protected**



Embedded DRAM

- **DRAM integrated in logic process technology**
  - 1 transistor and 1 capacitor
- **Improved leakage power (x6)**
- **Reduced SER rates (x1000)**
- **Density advantage (x4)**
- **High bandwidth**





Source: Iyer et al., IBM Journal of R&D, 2005

L3 Cache Characteristics

- **Features:**
  - Can be partitioned into cache and directly addressed memory
    - in 512kB increments
    - for shared scratch area
  - Shared write combining buffer, 4+1 cache lines deep
- **Coherent access to L3 cache by both 440 cores**
- **Sufficient bandwidth for demand access and L2 cache prefetch**
- **32-byte read and write bus per core @ 350MHz**
- **2 x 64-byte EDRAM access @ 175MHz**
  - Can support 250MHz operation at 1.5V

Main Memory - Synchronous DRAM

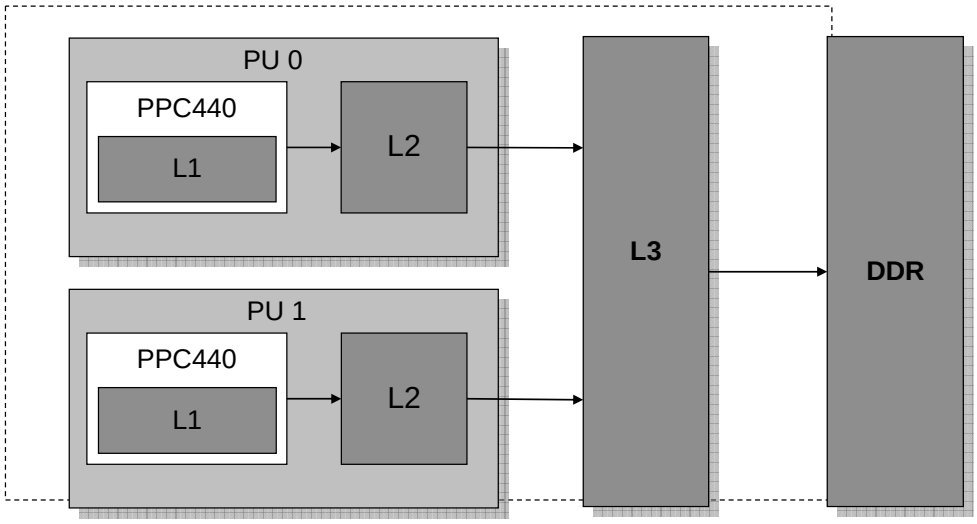
- **Double data rate (DDR) off-chip main memory**
  - 512MB or 1 GB per compute node
  - architecturally can support up to 2GB DDR
  - 9 x 512Mb DDR SDRAM chips per compute ASIC, 350 Mb/s/pin x16 pins/DRAM
  - Organized in 4 internal banks
- **16-byte wide I/O**
- **ECC, soft error scrub, nibble kill**

Memory System Access Latency

- **Pointer chasing (in cycles)**

Memory Type	Latency (cycles)
L1 cache	3
L2 cache	11
L3 cache	28/36/40
Main memory	86

Designing a High-Performance Memory System



L1
- 32 kB, 32B cache line
- Latency ~3cyc.

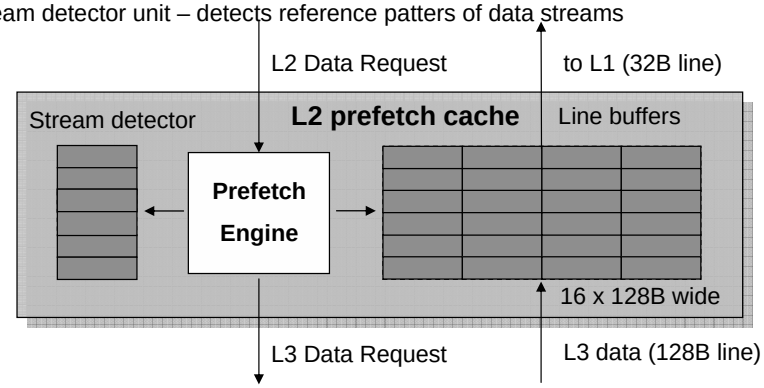
L2
- Hide L3 latency
- Capture locality

L3 eDRAM
- 4MB, 128B cache line
- Latency ~40 cyc.

Memory
- 512 MB off- chip
- Latency ~90 cyc.

L2 Prefetch Cache Architecture

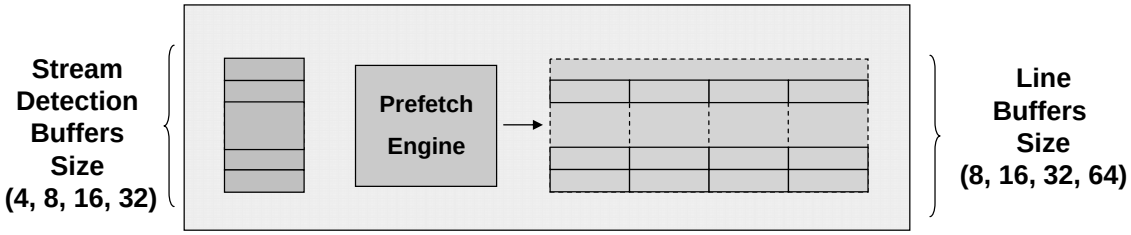
- **Traditional cache not viable**
  - Due to size and power consumption
  - Only fraction of size compared to a typical L2 cache
- **Prefetch cache components:**
  - Line buffers - hold lines from L3
  - Prefetch engine - initializes prefetch requests, predicts prefetch address, selects line buffer to replace
  - Stream detector unit – detects reference patters of data streams



L2 Prefetch Cache – Modes of Operation

- **Two modes of operation:**
  - Optimistic
  - Stream detector mode
- **Operation in optimistic mode**
  - L2 Miss: issue L3 data request
  - L2 Hit: issues prefetch to next L3 line
- **Operation in stream detector**
  - Stream detector buffer (SDB) holds addresses accessed
  - Address not in SDB : issue L3 data request + add address to SDB
  - Address in SDB: issues prefetch to next L3 line

Evaluated L2 Configurations



- **Different replacement algorithms for line buffers**
- **Deeper analysis on prefetching modes: optimistic vs. stream detection**
- **Evaluation metrics:**
  - –L2 hit, and L2 miss rates
  - –Execution time
    - •considers whole system interactions

Workloads Used for L2 Prefetch Cache Evaluation

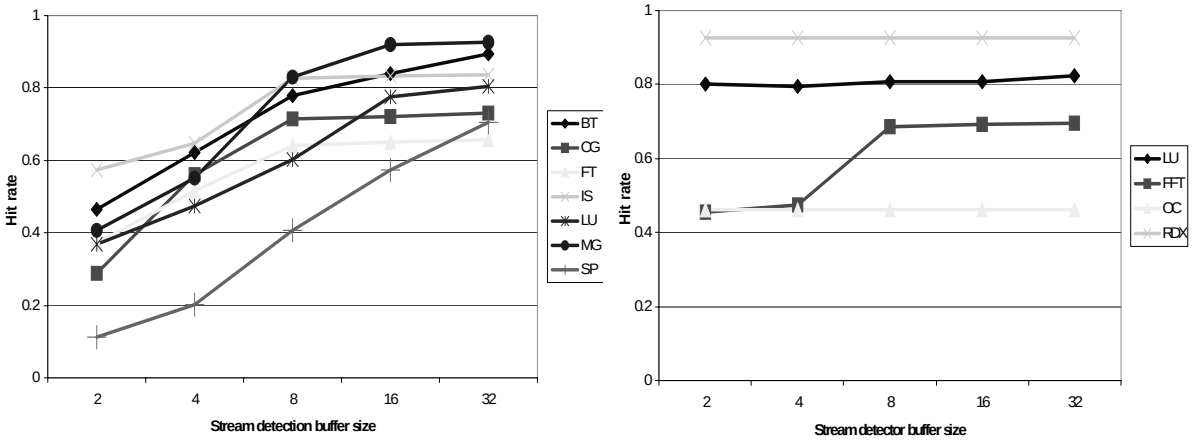
NAS	Instructions	L1 Misses	L1 Misses %
BT	547,414,050	30,788,712	5.62
CG	349,304,498	19,824,670	5.68
FT	645,116,212	37,248,944	5.77
IS	30,697,133	564,715	1.84
LU	238,891,062	10,934,076	4.58
MG	56,399,797	2,897,583	5.14
SP	273,988,939	20,660,969	7.54

Table 1: NAS Benchmarks Characteristics

Splash	Instructions	L1 Misses	L1 Misses %
LU	57,687,452	343,118	0.59
FFT	60,373,803	712,177	1.18
Radix	87,116,807	582,659	0.67
Ocean	30,005,066	1,843,293	6.14

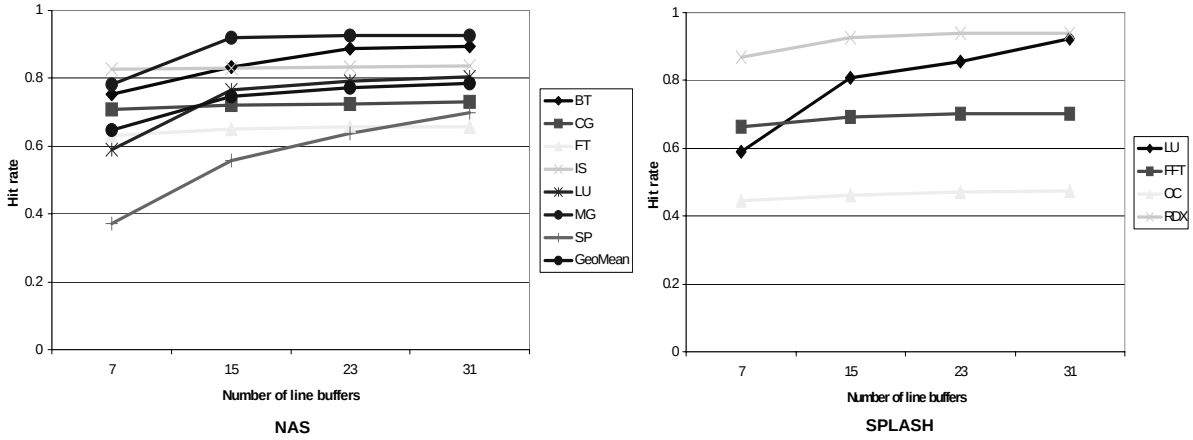
Table 2: Splash Benchmarks Characteristics

Dependence on Stream Detector Size



- Stream detector size greater than 16 does not significantly improve hit rate

Dependence on Number of Line Buffers

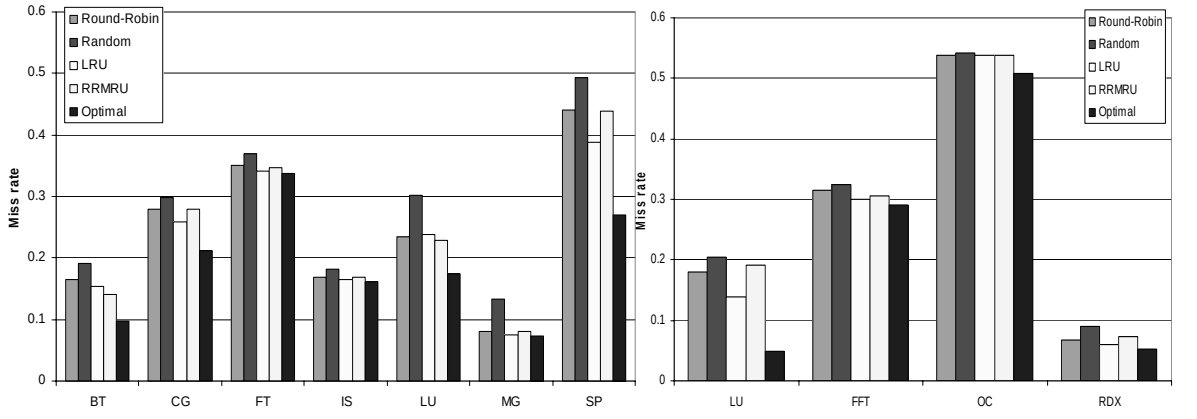


- Prefetch cache having 15 lines buffers seems the best compromise; few benchmarks benefit from a larger cache

Dependency on Prefetch Cache Replacement Policy

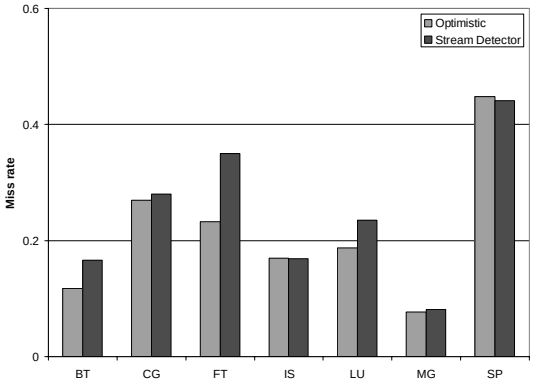
- **Explore various prefetch cache line replacement policies**
  - Round-robin
  - Random
  - Least Recently Used (LRU)
  - Round-robin skipping most recently used (RRMRU)
- **Policies compared against theoretical Optimal (one that relies on future knowledge) for upper boundary in replacement policy effect**

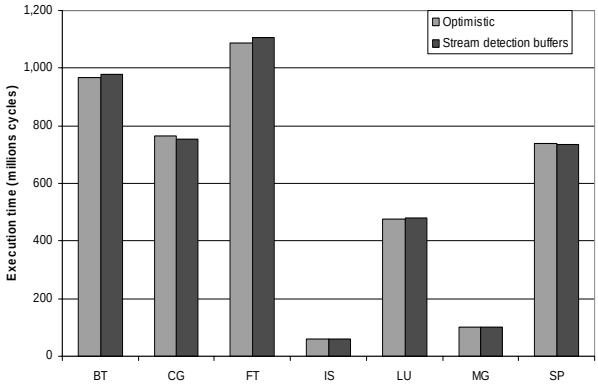
Dependency on Prefetch Cache Replacement Policy



- **LRU seems the best choice, closely followed by RRMRU**

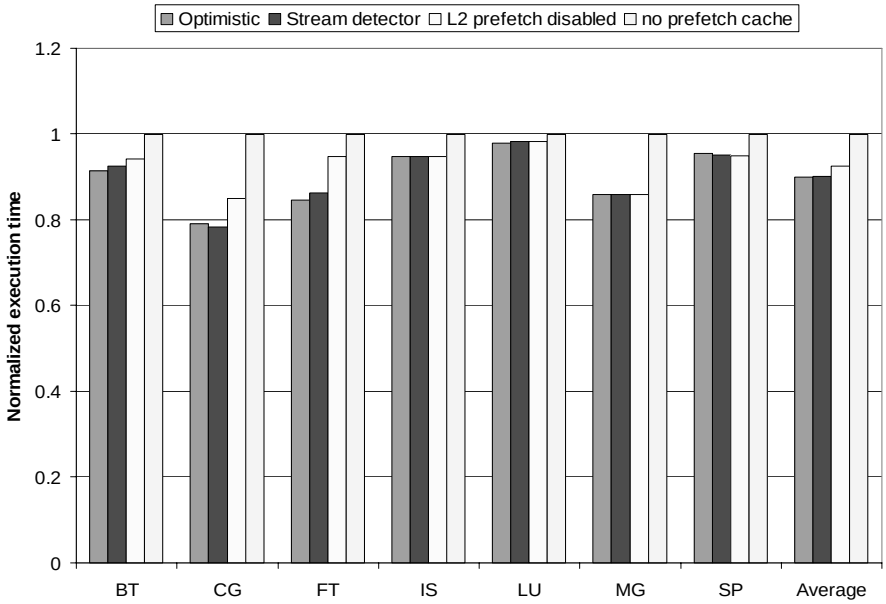
Stream Detector vs. Optimistic Modes





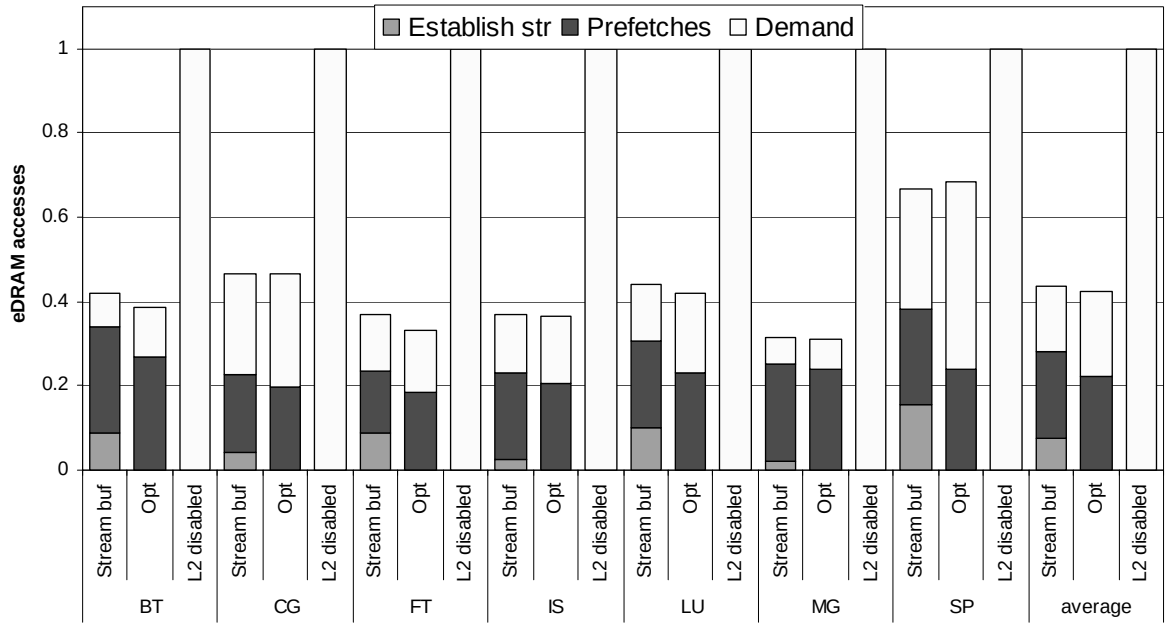
- **Optimistic mode yields lower miss rate for some applications**
- **Execution time for both approaches remarkably similar – 1.8%**

Prefetching Performance Advantage

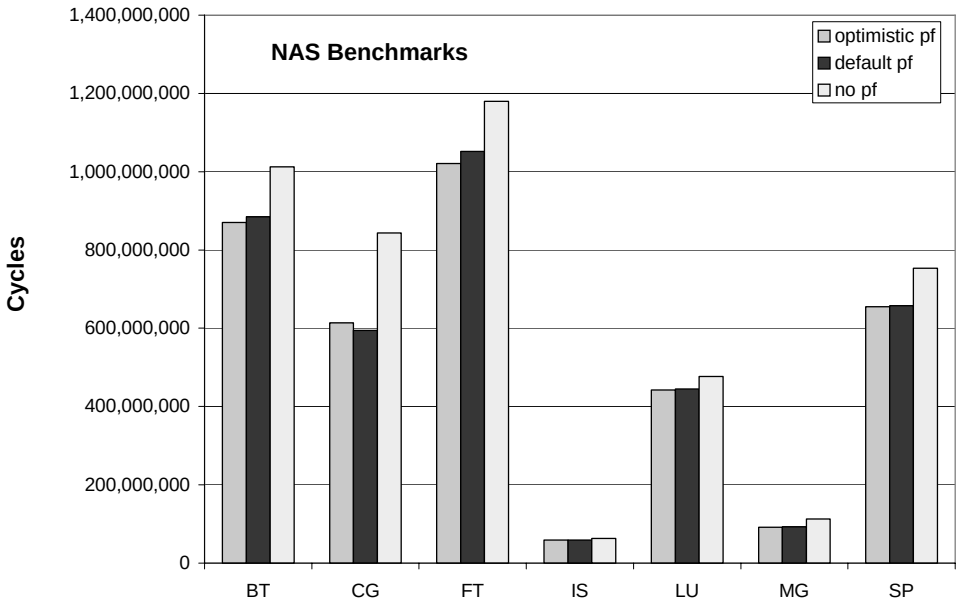


- **Prefetching brings 2% - 5% kicker**

L3 Bandwidth Requirements

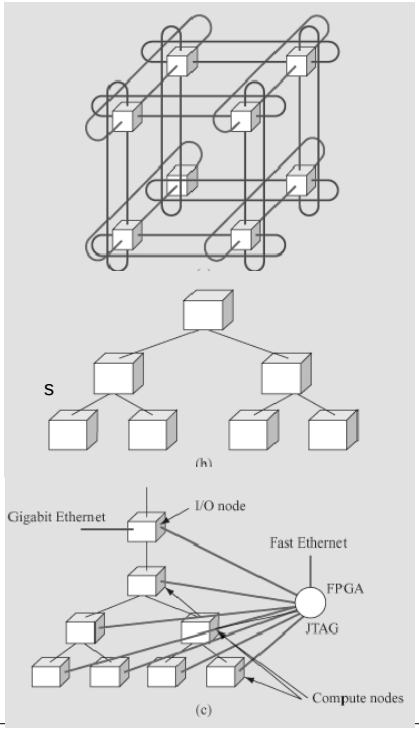


Hardware Measurements



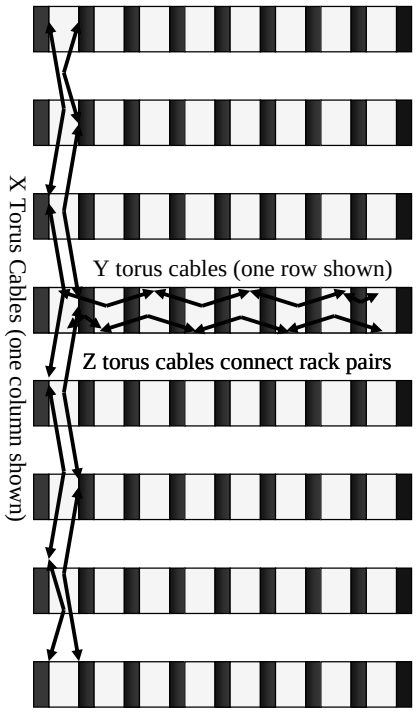
Five Independent Networks

- **3 Dimensional torus**
  - Point-to-point
- **Collective network**
  - Global operations
- **Global barriers and interrupts**
  - Low latency barriers and synchronization
- **Gbit Ethernet**
  - File I/O and host interface
- **Control network**
  - Boot, monitoring and diagnostics



Torus Network

- **3 Dimensional Torus**
  - nearest neighbor interconnect
  - 6 neighbors to each node
- **Interconnects all compute nodes (65,536)**
  - point-to-point communications
  - all-to-all communications
- **Communications backbone for computations**

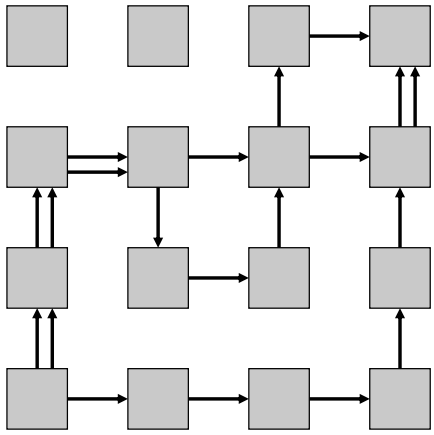


Torus Network (64x32x32)

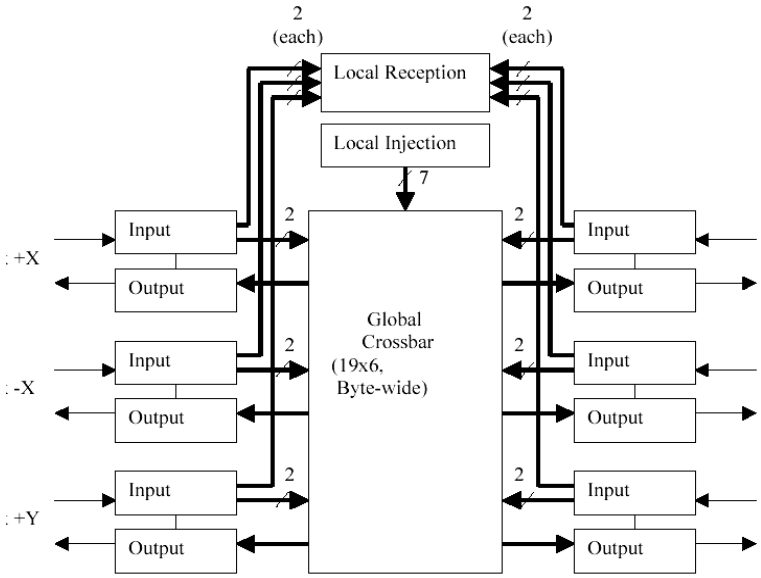
- **Features**
  - Can be partitioned down to the midplane level (8x8x8)
  - Synchronous network
  - Reliable, deadlock free
  - Packets from 32 to 256 bytes long
    - In 32-byte increments
    - Each packet includes a header with (X,Y,Z) of destination
    - 4-byte trailer
- **Links 1 bit wide, 6 bidirectional links/chip**
  - 1.4Gb/s per link
  - 12 links per node
  - 2.1 GB/s per node

Torus Network

- **Minimal, adaptive routing in HW**
- **Destination based, no routing table**
  - Multiple routes are possible
  - Always moves closer to destination
  - Packet arrival may be out of order
- **Virtual cut-through routing with multipacket buffering on collision**



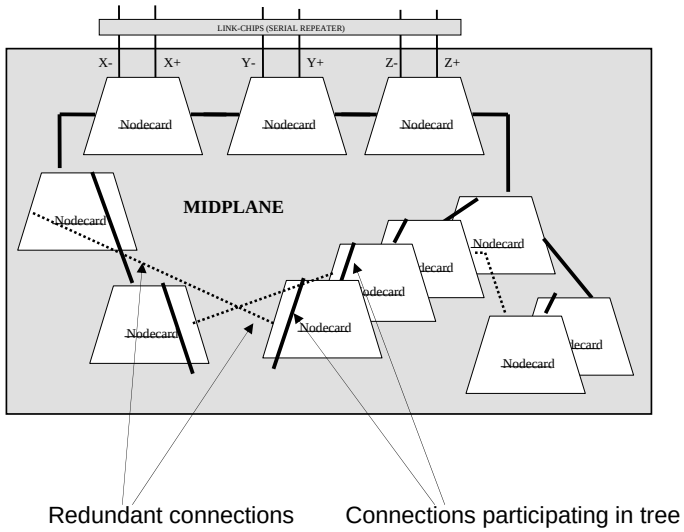
Three-Dimensional Torus Network



Torus router

- **Cat-through routing without software intervention**
- **Per-hop latency ~100 ns (avg.)**
- **Worst case latency for 64k machine (64 x 32 x 32)**
  - 6.4 µs (64 hops)
- **350/700 GB/s bisection bandwidth**

Collective Network



Redundant connections Connections participating in tree

- **Global reduction support**
- **Efficient for collective communication**
  - for broadcast messages
  - arithmetic reductions implemented in hardware
- **Bidirectional 3 links per node**
- **Fault-tolerant for tree topologies**
- **Connect every node to I/O node for file system**

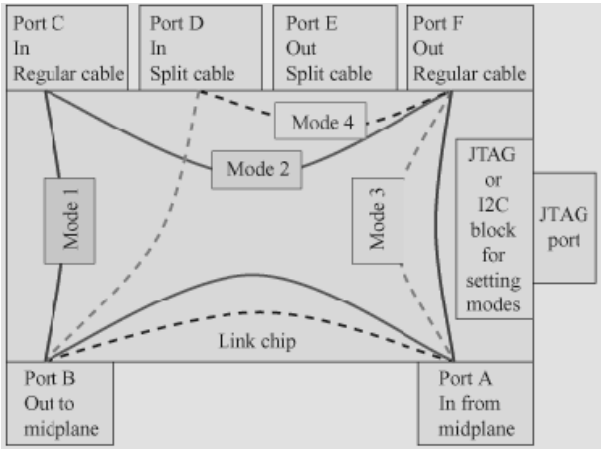
Collective Network

- **Implements efficient collectives**
  - reduction
    - MPI_REDUCE
  - broadcast
    - MPI_BCAST
- **Features**
  - Static (deterministic) routing with routing table
  - Low latency
- **Bandwidth**
  - 2.8 Gb/s per link
  - 2.1 GB/s per node
- **Worst case latency (round trip) 5.0µs**

BlueGene/L Link Chip

- **Placed at the midplane boundaries**
  - All networks pass through the BG/L link chip
- **Two functions**
  - Re-drives signals between the midplanes
  - Switches signals between its different ports
- **Allows partitioning into multiple logically separate systems**
- **Supports arbitrary static routing**
  - set by the host, static until another partition is created
- **Per midplane, 24 link chips**
  - 4 link cards
  - 6 BG/L chips per link card

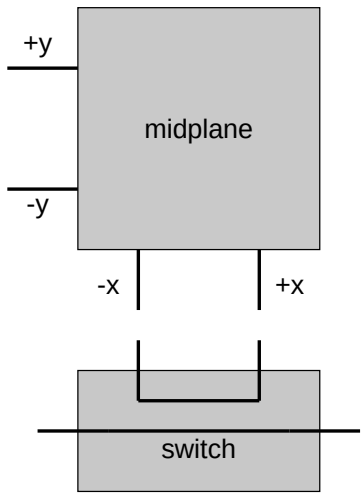
Link Chip Modes of Operation



- **Six ports**
  - 3 In and 3 Out
  - A and B are connected to nodes in a midplane
  - other four ports connected to cables
- **Per port**
  - 16 unidirectional torus links
  - Two global interrupt signals, 1 collective
  - A spare and a parity signal
- **Different modes of using the link chip:**
  - Mode 1: includes the ASICs of the local midplane in the larger torus
  - Mode 2: isolates the local midplane torus

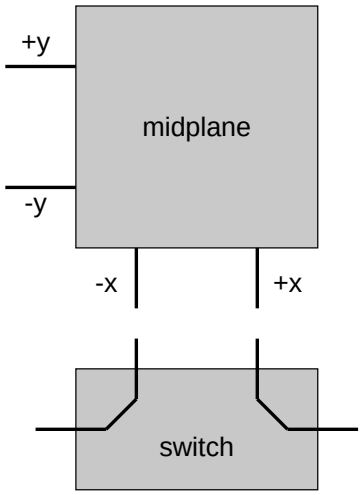
Midplanes and Switches

- **Each midplane contains 512 compute nodes and 8 I/O nodes**
- **Each midplane has six dimensions +x, -x, +y, -y, +z, -z**
- **For clarity, let us just consider two dimensions: x and y**
- **Each switch operates on one dimension**
- **Switch can be in one of two positions (simplification)**
  - –loopback

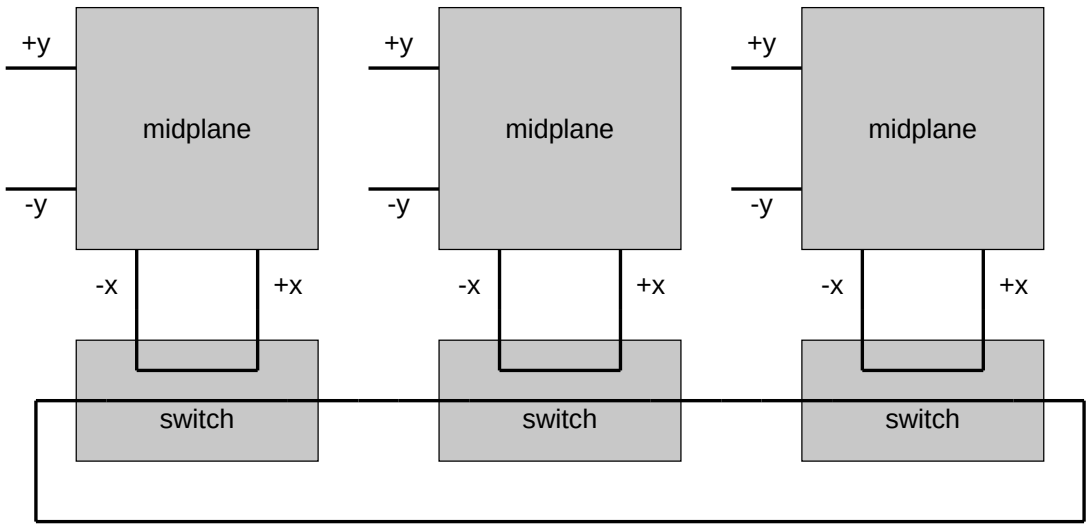


Midplanes and Switches

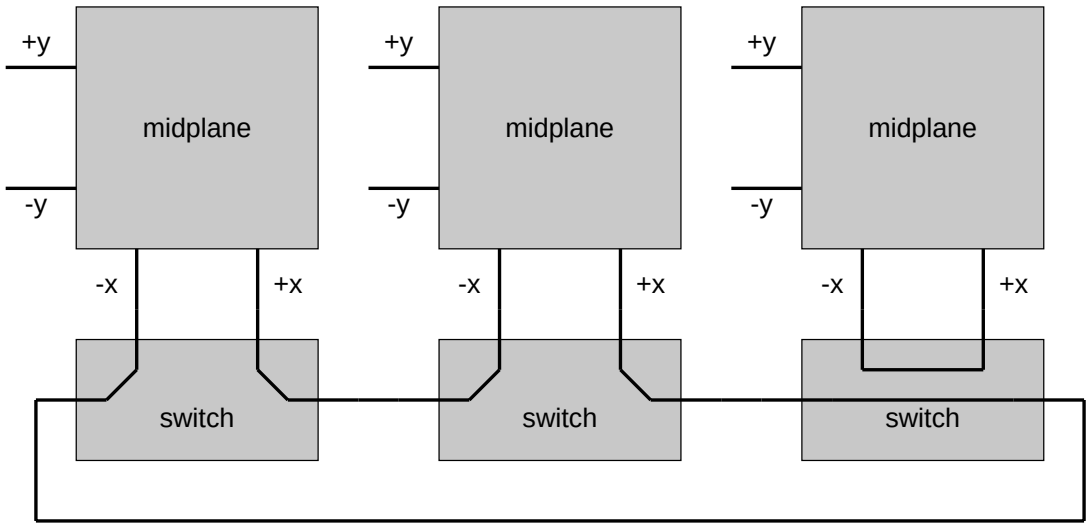
- **Each midplane contains 512 compute nodes and 8 I/O nodes**
- **Each midplane has six faces: +x, -x, +y, -y, +z, -z**
- **For clarity, let us just consider two dimensions: x and y**
- **Each switch operates on one dimension**
- **Switch can be in one of two positions (simplification)**
  - –pass-through



1D example: 3 x 1-midplane partitions



1D example: 1 x 2- + 1 x 1-midplane partitions

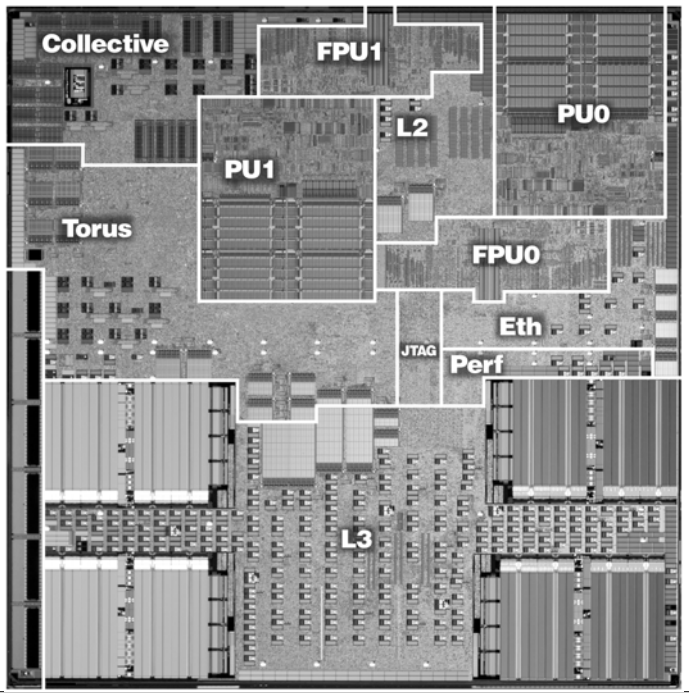


BlueGene/L ASIC

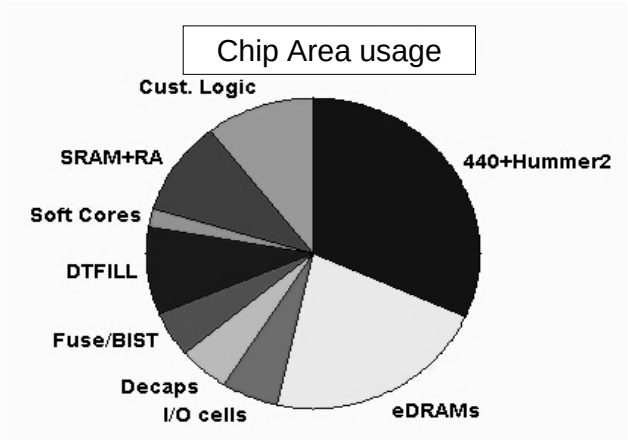
BlueGene/L ASIC

- **IBM Cu-11 0.13µ CMOS ASIC process technology**
- **11 x 11 mm die size**
- **1.5/2.5V**
- **CBGA package, 474 pins**
- **Transistor count 95M**
- **Clock frequency 700 MHz**
- **Power dissipation 12.9 W**

Careful floorplaning



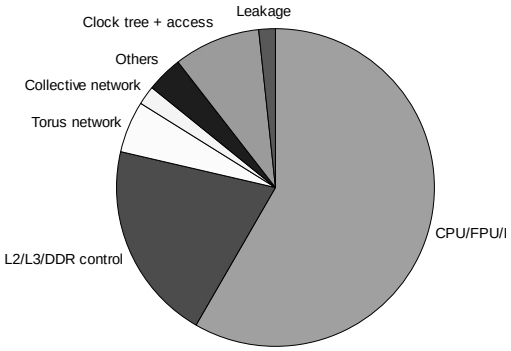
System-on-a-Chip Design Characteristics



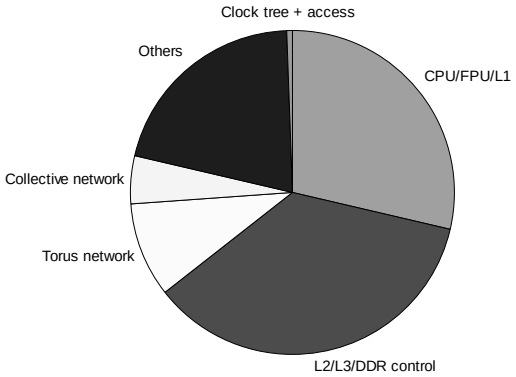
Cell Count	**57M**
Transistor Count	**95M**
Placeable Objects	**1.1M**
Clock Freq.	**700 MHz**
Power Dissipation	**12.9 W**
Bit Count eDRAM	**38M**
Bit Count eSRAM	**2.6M**

BlueGene/L Compute Chip Power and Area

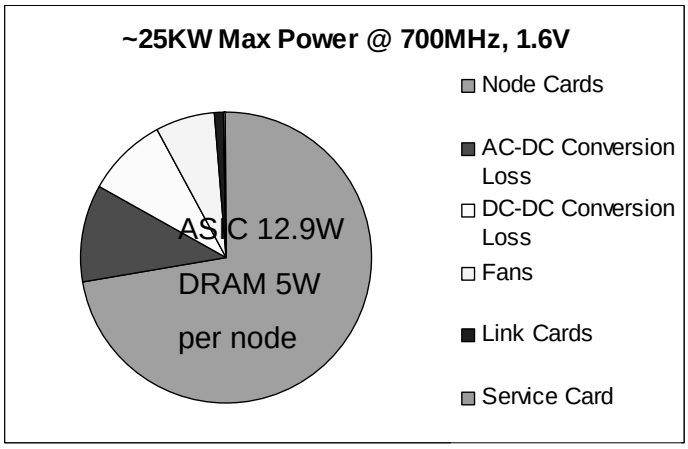
Power



Area

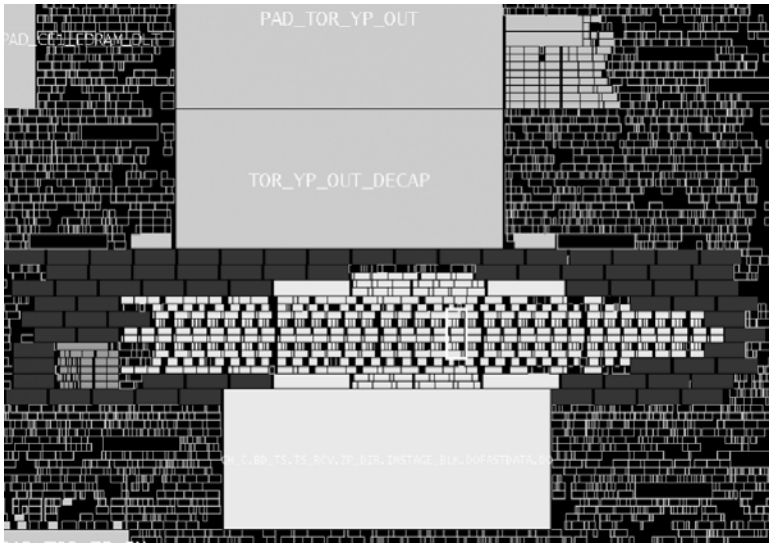


BlueGene/L Compute Rack Power

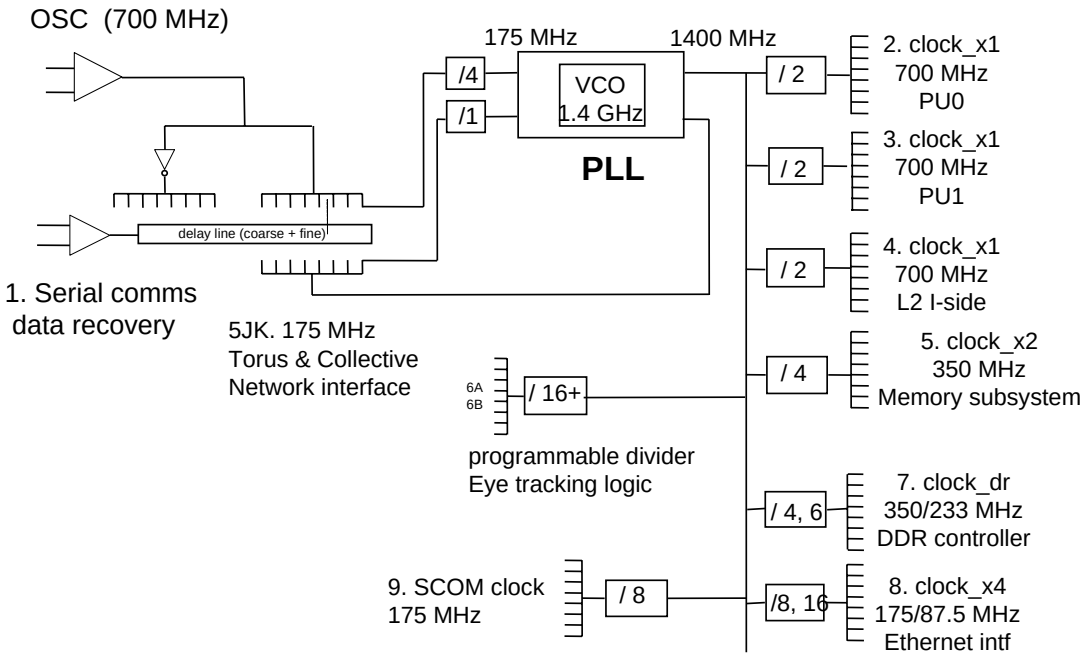


MF/W (Peak)	**230**
MF/W (Sustained-Linpack)	**172**

Bit Stacking



- **Technique used to achieve timing goals**
- **Example: high-speed data reception macro**

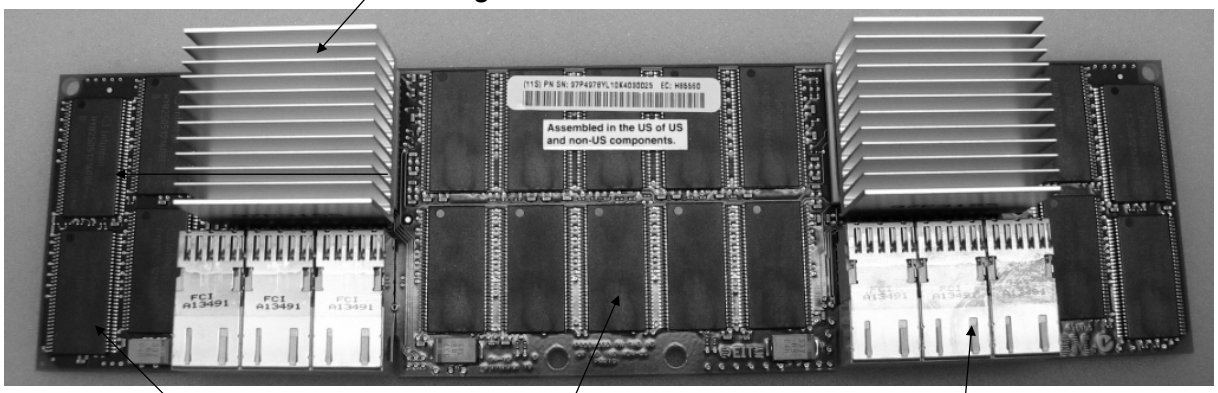


BlueGene/L System Package

Dual Node Compute Card

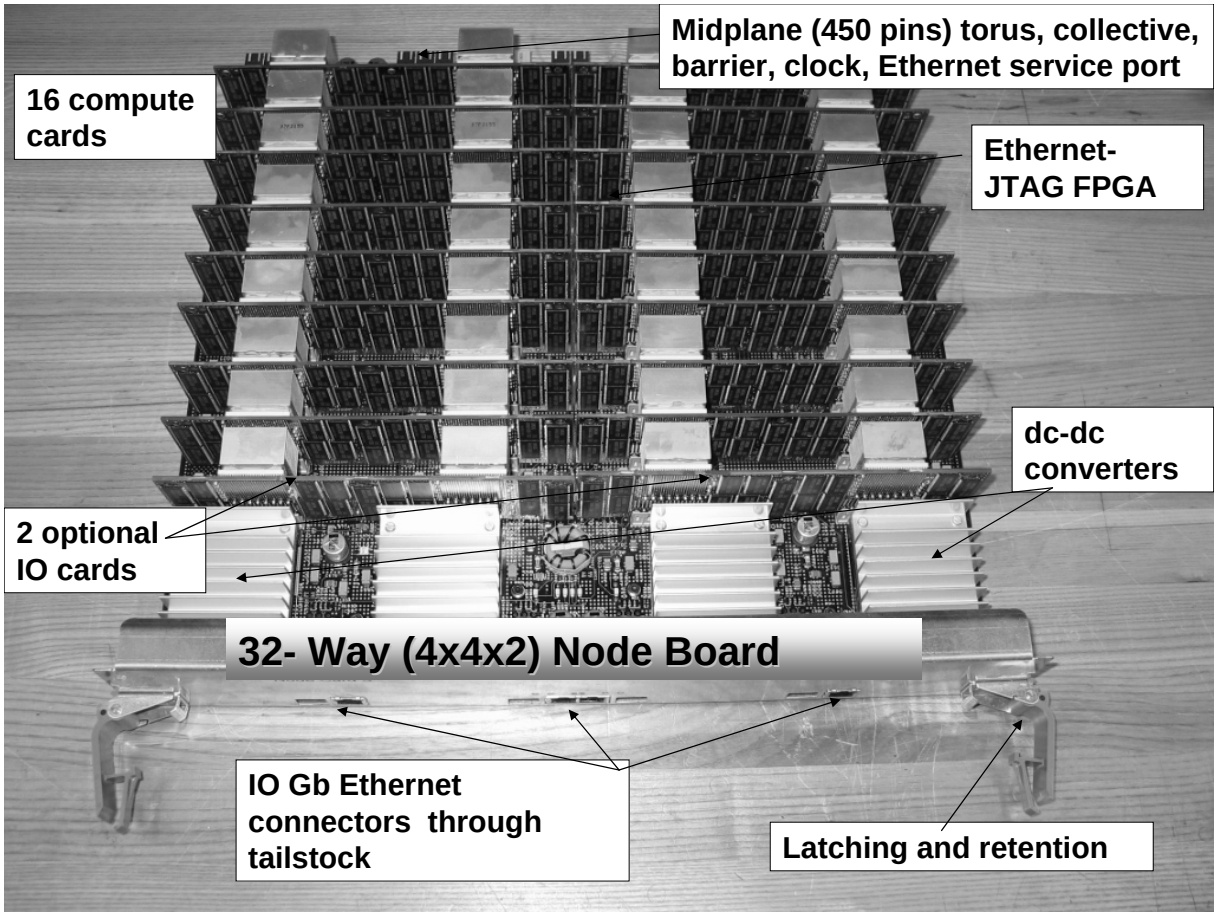
206 mm (8.125”) wide, 54mm high (2.125”), 14 layers, single sided, ground referenced

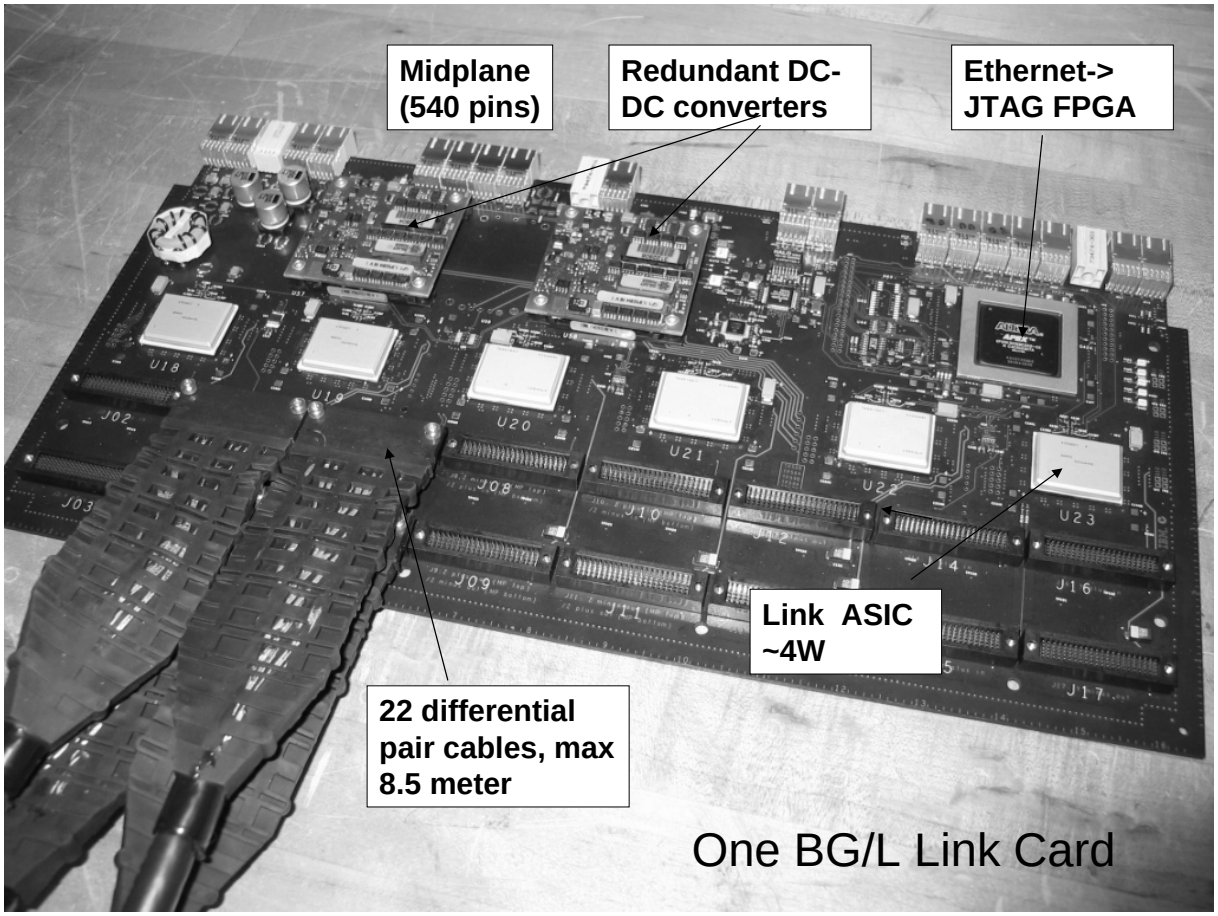
Heatsinks designed for 15W



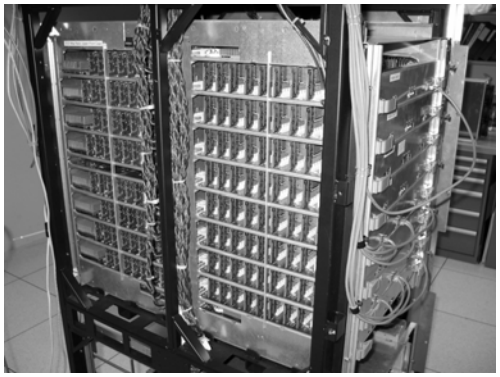
9 x 512 Mb DRAM

Metral 4000 high speed differential connector (180 pins)


Midplane (450 pins) torus, collective, barrier, clock, Ethernet service port
16 compute cards
Ethernet-JTAG FPGA
dc-dc converters
2 optional IO cards
32- Way (4x4x2) Node Board
IO Gb Ethernet connectors through tailstock
Latching and retention


Midplane (540 pins)
Redundant DC-DC converters
Ethernet-> JTAG FPGA
Link ASIC ~4W
22 differential pair cables, max 8.5 meter
One BG/L Link Card

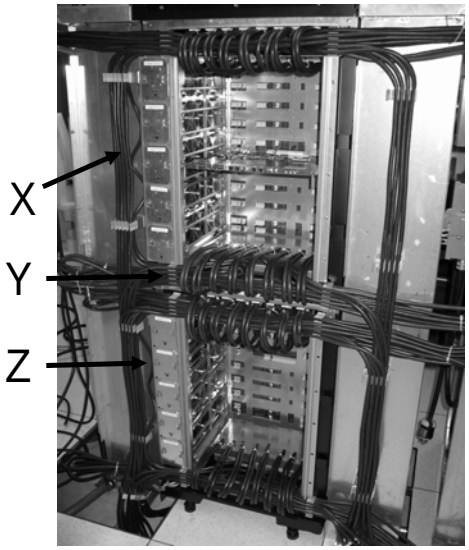
BlueGene/L Rack



512 – way (8 x 8 x 8) “midplane” (half-cabinet)

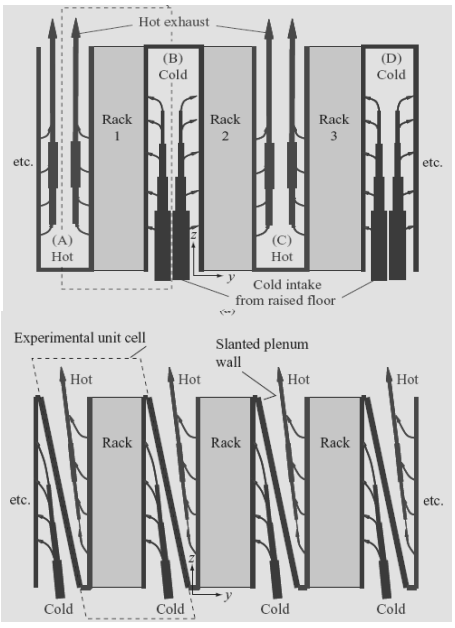
16 node boards

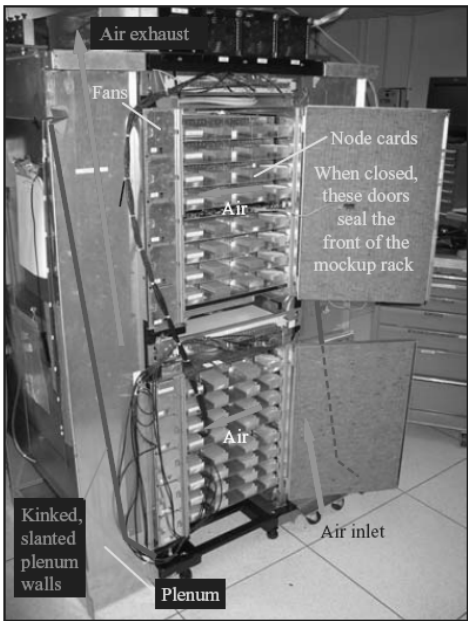
All wiring up to this level (>90%) card-level



Two midplanes interconnected with data cables

Cooling Strategy: Redefine the Box!





Source: Coteus et al., IBM Journal of R&D, 2005

The World's Fastest Supercomputer: Blue Gene/L at LLNL

BlueGene at LLNL

BlueGene/L reliability

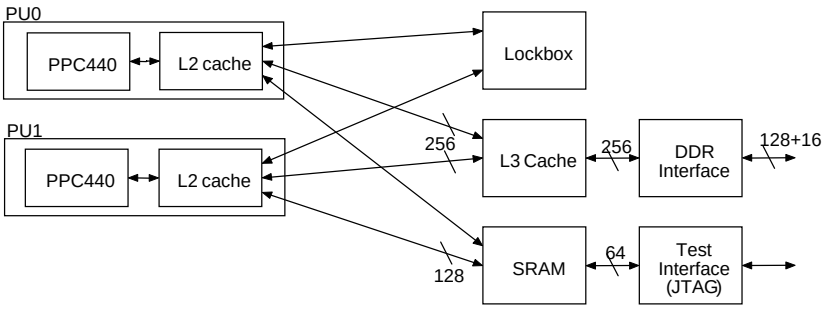
System Reliability

- **Massively parallel systems represent reliability challenge**
  - The problem amplifies with increase of system size
- **Focus on reliability from the beginning of design process**
  - High-end server reliability requirements
  - Reliability as important as high performance and power consumption
- **Advanced Reliability, Availability and Serviceability (RAS) architectural support**

Building a Reliable System

- **Protect signaling links**
  - Massive amounts of communication requires better bit error rate
- **Ensure data integrity**
  - Recovery from soft errors
- **Address component failure rate**
  - Allow isolation of errors
  - Use software checkpointing to resume operation
- **Track correctable errors to detect failing parts**
  - Collecting statistics on correctable errors
  - Used for predictive failure analysis

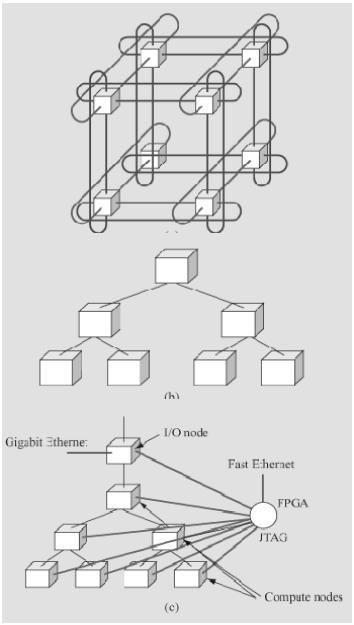
Memory Hierarchy Reliability



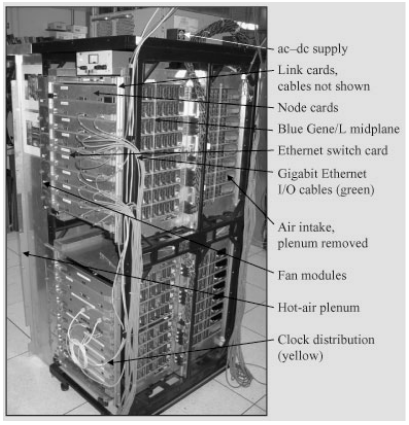
- **32kB D&I private cache per processor**
  - Write-through mode for lower soft error rate (SER)
- **All SRAMs are ECC protected**
  - L1 cache with parity protection supports write-through
- **Parity protected internal buses**
- **External memory uses ECC and spare data bits**
- **Controller for external DRAM supports scrub and ECC with nibble kill reliability**

BlueGene Network Reliability

- **Differential signaling for torus and collective links for low error rates**
- **Redundant connections in the collective network for tree topology**
- **Network packet transmissions protected with protocols for error detection and re-send**
- **Torus and collective networks CRC protected**
  - 24 bit per packet CRC
  - 32 bit link-level CRC over all data sent / received, checked at the end of run or at the checkpoint

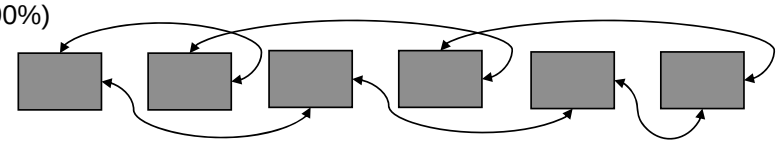


Packaging and Interconnection for Reliability



All wiring in the midplane (>90%) card-level

- **Midplanes interconnected with cables**
  - Include redundant cables
  - Cables designed to be never moved after installation
- **Link chip at edges of midplanes for signal re-buffering and routing**
  - Link chip failure rate: 2 in 3 years for 64k node system
  - Allow isolating failing midplane
- **Interchangeable pattern of midplanes and cables to avoid long wires**



System Level RAS Support in BlueGene/L

- **Reliability achieved thru**
  - Simplicity
    - Reduced number of components – Compute and link ASICs
  - Redundancy
    - Spare power supplies, memory chips, cooling fans, cables between racks
  - Check-pointing
    - Application writes results regularly to the disk
- **Availability**
  - Flexible partitioning
    - Partition sizes: 32, 64, 128, 256, 512, etc. with torus in each partition
- **Serviceability**
  - Separate power-supplies for each mid-plane
    - Only affected mid-plane powered down
  - Compute and I/O nodes can be removed without removal of cables
    - Requires removal of only node card from the mid-plane

BlueGene/L System Software

System Software Course Outline

- Introduction
  - Motivation of system software design
- Kernels
  - CNK in the compute nodes
  - Linux in the io nodes
  - CIOD daemons
  - Function shipping
- MPI
  - BG/L MPI Software Architecture
  - Optimizing performance
    - Point-to-point messaging
    - MPI collective operations
    - Mapping MPI applications to network geometry
- Compilers and libraries
  - Architecture of IBM XL compilers
  - Sample loop optimizations
  - SIMD optimizations
  - Math libraries
- Tools
  - Profiling tools

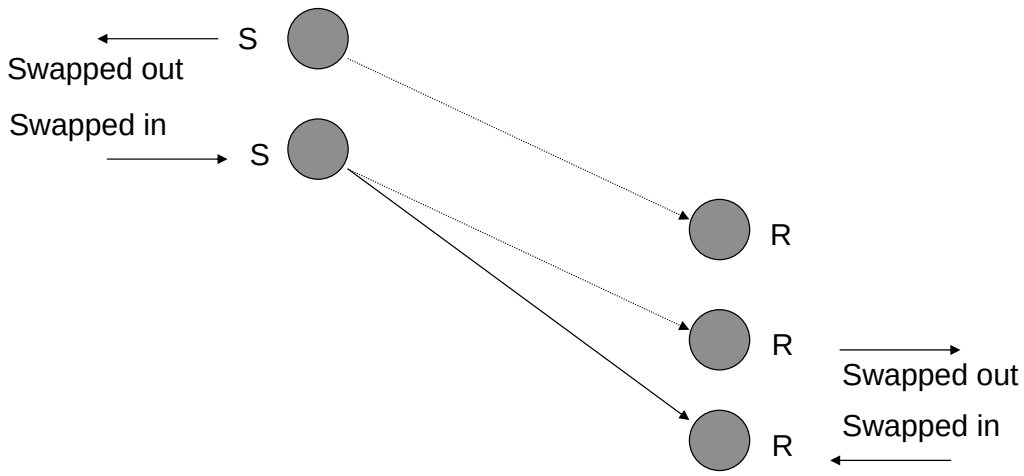
How do you write software for Blue Gene?

- Issues for application writer
  - Express parallelism – sufficient level of parallelism to exploit 128K processors – SPMD (Single Program Multiple Data)
  - Manage overheads of parallelism
    - Communication costs
    - Load balancing
  - Good single node performance
    - Multiplicative effect when scaling up

How do you write software for Blue Gene?

- Issues for system software developers
  - Scalability: overcome barriers like
    - Interference in parallel program execution
    - High communication costs
  - Good single node performance
  - Reliability

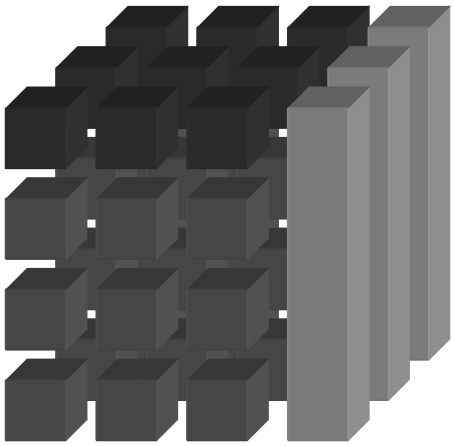
Interference problem



Principles of BlueGene/L system software design

- Simplicity
  - Need for an operating environment for 64k nodes, 128k processors
  - Limited purpose machine – enabled simplifications
  - Reliability through simplicity
- Efficiency
  - Simplicity enables efficiency by dedicating hardware to function
  - High performance without sacrificing security
- Familiarity
  - Standard programming languages and libraries
  - Enough functionality to deliver a familiar system without sacrificing simplicity or high performance

BlueGene/L System Software Architecture



- User applications execute exclusively in the **compute nodes**
  - Avoid asynchronous events (e.g., daemons, interrupts)
  - Avoid complex operations
  - Custom solution: Compute Node Kernel
- The outside world interacts only with the **I/O nodes**, an offload engine
  - Standard solution: Linux
  - Provide a more complete range of OS services – files, sockets, process launch, signaling, debugging, and termination
- Machine monitoring and control also offloaded to **service nodes**: Linux cluster
  - IBM middleware
  - Custom components for booting, configuration, partitioning, monitoring

Simplicity

- Strictly space sharing
  - One job (one user) per electrical partition of the machine
  - One thread of execution per processor in **application volume**
- Virtual memory constrained to physical memory size
  - Implies no demand paging, only static linking
- Hierarchical organization for management and operation
  - More comprehensive system services (I/O, process control, debugging) are offloaded to I/O nodes in **functional surface**
  - System control and monitoring offloaded to service node in **control surface**
  - Form processing sets (psets) consisting of a collection of compute nodes under control of an I/O node (for LLNL machine, pset = 1 I/O node + 64 compute nodes)
    - Each processing set is under control of Linux image in the I/O node
- Flat view for application programs – collection of compute processes

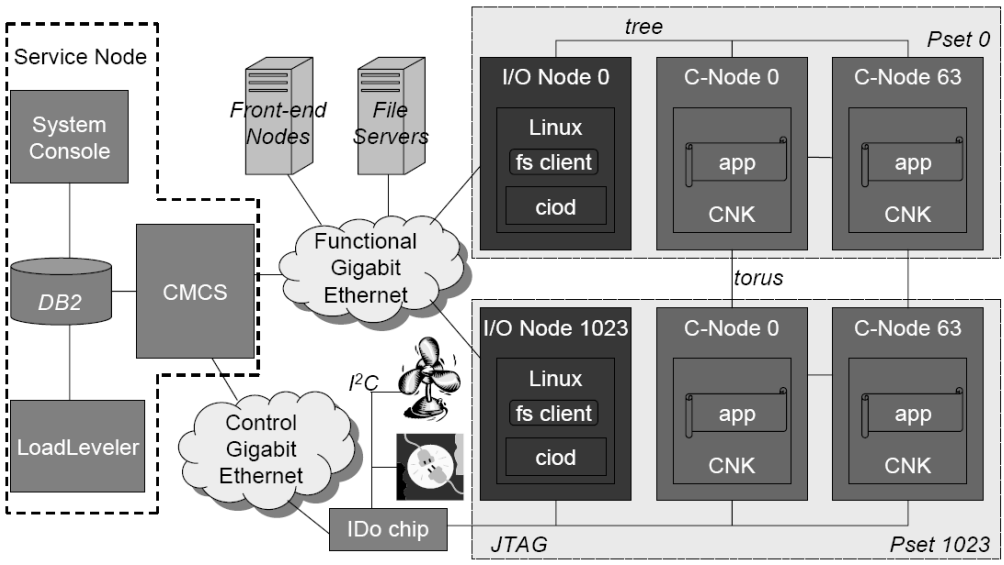
Efficiency

- Dedicated processor for each application-level thread
  - Deterministic, guaranteed execution
  - Maximum performance for each thread
  - Physical memory directly mapped to application address space – not TLB misses (also, no paging)
  - Statically-linked executables only
- System services executed on dedicated I/O-node
  - No daemons interfering with application execution
  - No asynchronous events on computational volume
- User-mode access to communication network
  - Electrically isolated partition dedicated to one job + compute node dedicated to one process = no protection necessary!
  - User-mode communication = no context switching to supervisor mode during application execution (except for I/O)

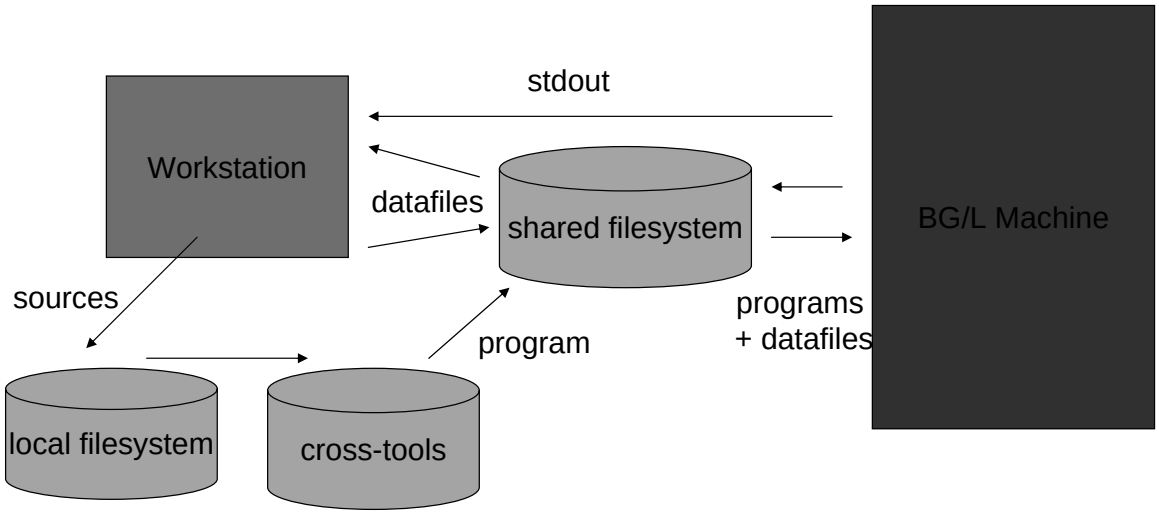
Familiarity

- 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), hardware performance monitors (HPMLib), trace based visualization (Paraver)

Blue Gene/L System Architecture



Compiling and Running on BG/L



Kernels:

- **CNK in Compute nodes**
- **Linux in IO nodes**
- **Function shipping**

The Kernels

- Linux kernel on IO nodes
- Custom IBM kernel on compute nodes
- Why not Linux everywhere?
  - non-cache coherent memory – “hard” to port smp Linux
  - general purpose multi-process kernel ill-suited for HPC
    - TLB management
    - time-slice scheduler

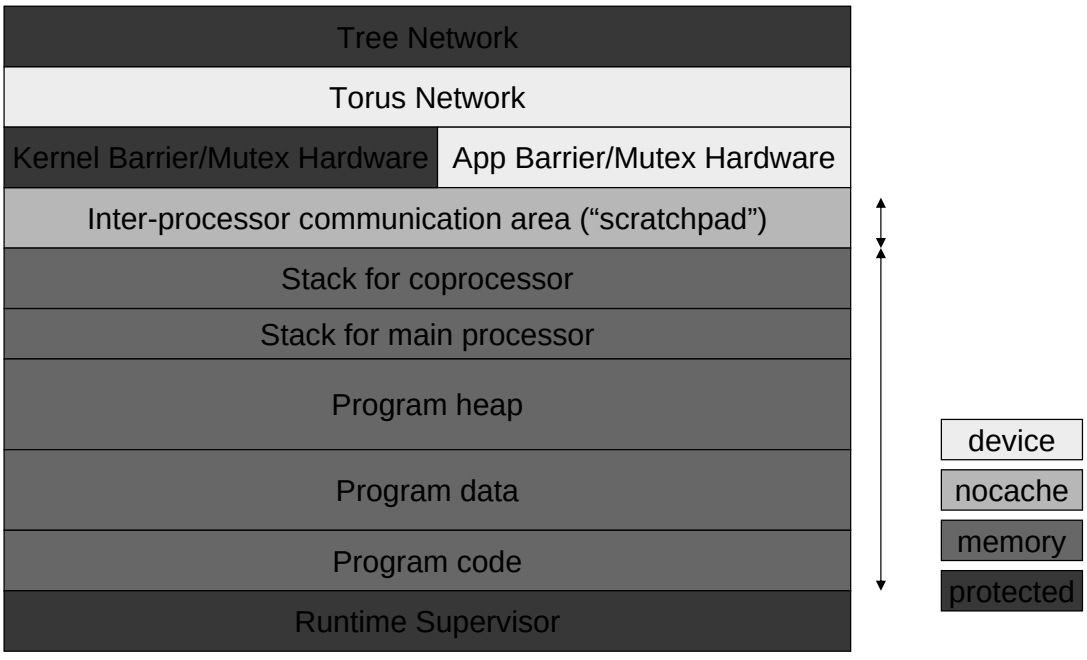
CNK Features

- A simple Linux-like kernel
  - Runs one process at a time
  - Uses small amount of memory – rest for the application
  - Supports attaching debuggers
- CNK provides a subset of the Linux calls
  - File I/O
  - Directory operations
  - Signals (ANSI C only)
  - Process information
  - Time
  - Sockets
- Non-preemptive
  - Application program has full control of all timing issues

CNK Features (II)

- Kernel provides
  - Program load / start / debug / termination
  - File access
  - All via message passing to IO nodes
- Direct access to most hardware
  - Tree and torus fifos
  - Global OR
  - Mutexes and barriers
  - Performance counters
- Mantra
  - Keep the compute kernel simple
  - Kernel stays out of the way and lets the application program run

CNK Memory Map



CNK System calls

- Traditional ANSI + “a little” POSIX
  - GLIBC runtime
- I/O
  - Open, close, read, write, …
  - Files, client side sockets
- Time
  - Gettimeofday, etc
- Signals
  - Synchronous (sigsegv, sigbus, etc)
  - Asynchronous (timers and hardware events)

CNK Limitations

- No “unix” (multi-processing) functions
  - fork, exec, pipe
  - setuid, setgid
- No support for
  - asynchronous signals using sigaction()
  - interprocess communication
  - server-side sockets APIs
  - poll() or select()
- Limited support for timers

IO Node Kernel

- IBM-endorsed “MCP” version of Linux kernel
- Vary small changes compared to a standard kernel
  - Boot sequence
  - Interrupt controller
  - Device drivers
    - Tree, console, emac (Ethernet)
  - Memory layout
  - Save and restore FPU registers
- Purpose is to run
  - network filesystem
  - job control and i/o daemon (“ciod”)

Processing sets (Pset)

- Logical grouping of
  - 8 to 64 compute nodes
  - one IO node, servicing the compute nodes
  - Compute nodes and IO nodes connected through the tree
- Permits communication to/from the compute nodes
  - Through the IO nodes
  - Over the tree

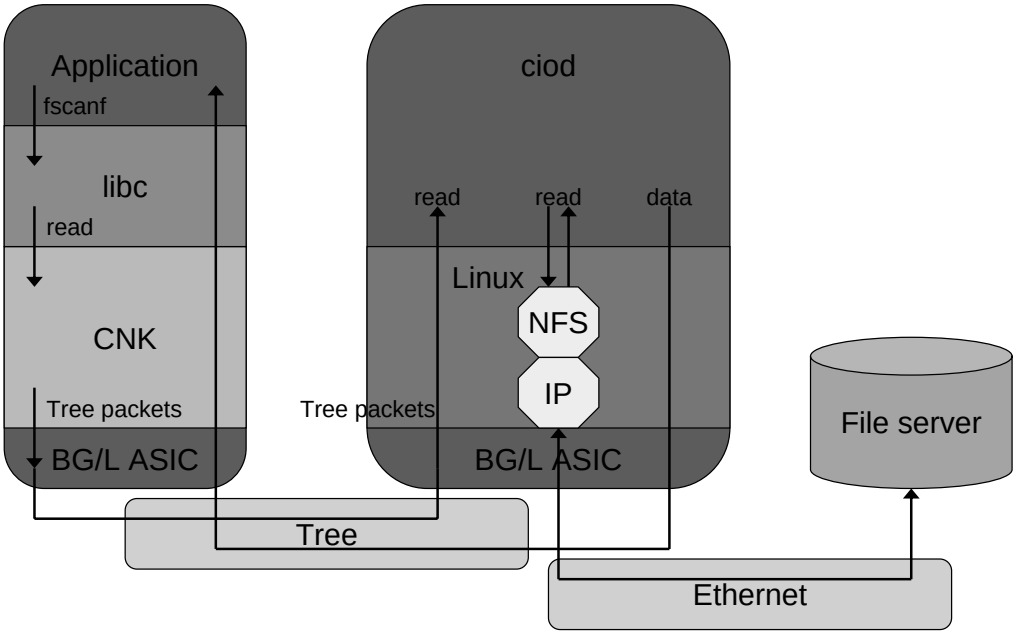
CIOD

- Stands for control and IO daemon
- Control
  - Connects to external daemons (ciodb) using the Gbit network in every IO node
    - Forwards job start, kill, signals to the compute nodes (over the tree)
- Receives IO from compute nodes
  - Ioremaps tree network fifos into user space
    - Continuously polls the tree
  - Buffers multiple tree packets (~4KB) to create a message
  - For stdout/stderr
    - Directly sends the IO to ciodb over a socket
    - Stdin redirected from /dev/null
  - For file IO
    - Remaps file descriptors
    - Calls Linux system calls
      - Ciod runs as a user process, with user privileges
      - Setuid bit converts to/from root at job start/terminate

CNK Function Shipping Example

- Application calls write() system call
- CNK breaks requests into multiple packets
  - ~4 KB
  - Sends each message and calls write
- CIOD receives the message, does a file descriptor translation and calls write()
- CIOD sends result back to the compute node
- CNK collects results from each message
- CNK returns result to application after either all data is sent or an error occurs
- CIOD never blocks on a system call
  - All sockets are implicitly non-blocking

File I/O in BlueGene/L

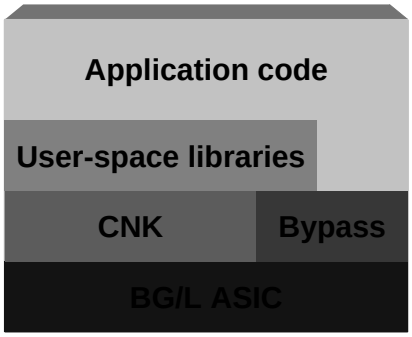


Execution Modes for Compute Node

- **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
- **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

Software Stack in BG/L Compute Node

- CNK controls all access to hardware, and enables bypass for application use
- User-space libraries and applications can directly access torus and tree through bypass
- As a policy, user-space code should not directly touch hardware, but there is no enforcement of that policy
- Application code can use both processors in a compute node



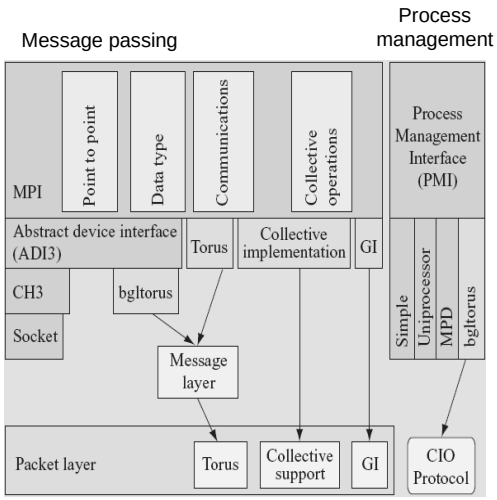
MPI

MPI Outline

- Preliminaries
- BG/L MPI Software Architecture
  - Description
  - Development methods, timeline
- Optimizing performance
  - Point-to-point messaging
  - MPI collective operations
  - Mapping MPI applications to network geometry

MPI

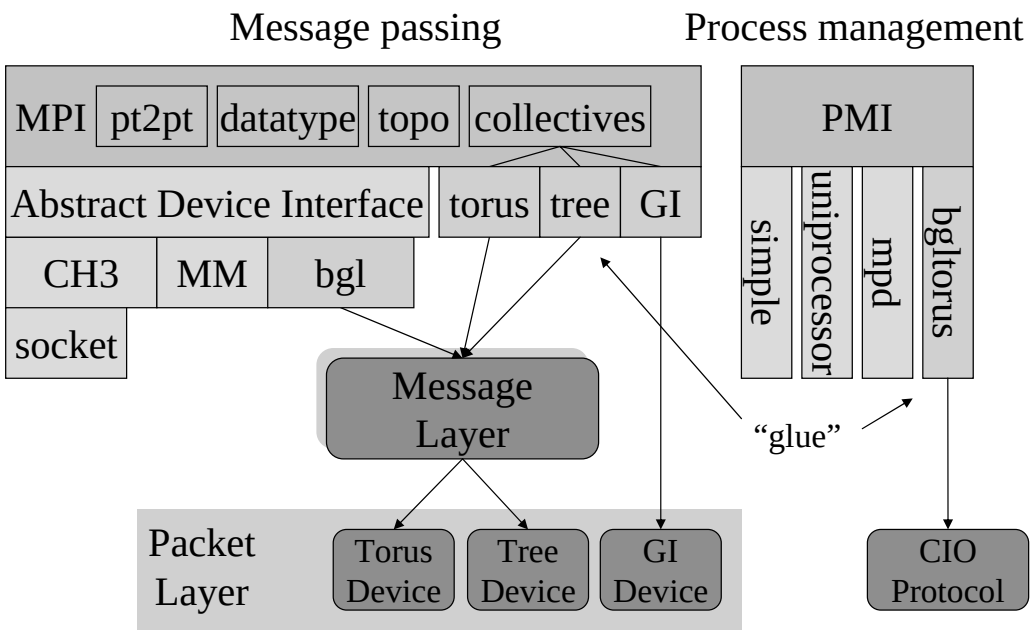
- MPI 1.1 compatible implementation for message passing between compute nodes
  - **only the most widely used features of MPI implemented**
- Based on MPICH2 from ANL
- Point-to-point
  - **utilizes Torus**
  - **implements a BlueGene/L version ADI3 on top of message layer**
- Global operations
  - **Utilizes both torus and collective network**
- Process management
  - **use BlueGene/L's control system rather than MPICH's process managers**



Layers of BlueGene/L Communication Software

- Packet layer
  - Initialize network HW, send and receive packets
  - As simple as we can afford to make it
- Message layer
  - Active message layer similar to LAPI and GAMMA
    - on top of packet layer
  - Handles hardware complexity
    - alignment, ordering, transmission protocols
    - Cache coherence, processor use policy
- MPI
  - A port of Argonne National Labs' MPICH2
  - BlueGene/L is primarily an MPI machine

MPICH2 based BG/L MPI Software Architecture



Process Management in BGL/MPI

- **Process startup and termination**
  - MPICH2 provides PM interface:
  - Implemented using the BG/L CIO protocol
    - `-ciorun` asks control system to start up job
    - –Control system contacts CIO daemons residing on 1024 I/O nodes
    - –CIO daemons issue commands to 64 compute nodes through tree network
  - Does not (and will not) support dynamic MPI process creation
  - Integrated with scheduler
- **Mapping of torus coordinates to MPI ranks**
  - Multiple fixed torus rank mappings can be selected through environment variables at startup
  - Arbitrary mapping function provided at job startup time
  - MPI programs are topology portable; MPI performance is not

Performance Limiting Factors in the MPI Design

Hardware

- Torus Network link bandwidth
  - 0.25 Bytes/cycle/link (theoretical)
  - 0.22 Bytes/cycle/link (effective)
  - 12*0.22 = **2.64 Bytes/cycle/node**
- Streaming memory bandwidth
  - **4.3 Bytes/cycle/CPU**
  - memory copies are expensive
- CPU/network interface
  - 204 cycles to read a packet;
  - 50 – 100 cycles to write a packet
  - Alignment restrictions
    - –Handling badly aligned data is expensive
  - Short FIFOs
    - –Network needs frequent attention
- Network order semantics and routing
  - Deterministic routing: in order, bad torus performance
  - Adaptive routing: excellent network performance, out-of-order packets
  - In-order semantics is expensive
- Dual core setup, memory coherency
  - Explicit coherency management via “blind device” and cache flush primitives
  - Requires communication between processors
  - Best done in large chunks
  - Coprocessor cannot manage MPI data structures

Software

- CNK is single-threaded; MPICH2 is not thread safe
- Context switches are expensive
  - Interrupt driven execution is slow

Using the Communication Co-processor

- Constraint 1: one CPU cannot keep up with network
- Constraint 2: BG/L chip has two non-coherent 440 cores
  - Original design point: second processor acts as an intelligent DMA engine (“co-processor mode”)
  - Initial software development done with 2nd processor in an idle loop (“heater mode”)
  - Considered: “virtual node mode” (2nd processor has its own O/S image and stack, shares all resources equally)
- Simple co-processor solution (1 extra memory copy):
  - CPU0 and CPU1 interact through common non-cached area (scratchpad)
  - Simple, but low performance
- Complex 0-copy solution:
  - Main CPU, coprocessor execute software cache coherency protocol
    - –Sequences of cache flush and invalidate instructions
    - –Need kernel support
    - –Danger of false sharing
    - –Complicated, fragile implementation (“heroic programming”)

Scaling MPI to 64k processors

- Constraint: 64k nodes all talking to each other
  - MPICH2 has lazy virtual connections by default:
    - Mitigates startup costs
  - BG/L: hardware allows virtual connections to be effectively pre-established - MPI “born” with connections up
    - Per-connection state required for flow control, protocol
    - 64k endpoints @ 50 bytes each = 3 Mbytes/node
    - Memory need varies with number of simultaneous messages in system
- Problem: Receiving 64k simultaneous eager unexpected messages?
  - Answer 1: MPI_Abort is a legal way to solve the memory problem
  - Answer 2: flow control for eager message protocol

The BlueGene/L Message Layer

- Looks very much like LAPI, GAMA
  - Just a lot simpler
- Simplest function: Deliver a buffer of bytes from one node to other
  - Can do this using one of many protocols
    - One-packet protocol
    - Eager protocol
    - Rendezvous protocol
    - Virtual node mode copy protocol!
    - Adaptive eager protocol!
    - Collective function protocols!
    - … and others

Point-to-point Communication

- Basic MPI functionality
  - MPI_Send(), MPI_Recv()
  - Enough to get MPI-1 compliance in MPICH2.
    - MPICH2 implements collectives using point-to-point communication
  - Do-or-die: no high performance MPI implementation without good point-to-point communication performance
  - Implemented using message layer
    - Provides an implementation of the MPICH2 Abstract Device Interface

Outline

- Preliminaries
- BG/L MPI Software Architecture
  - Description
  - Development methods, timeline
- Optimizing performance
  - Point-to-point messaging
  - MPI collective operations
  - Mapping MPI applications to network geometry
- Conclusions

Optimizing point-to-point communication (short messages: 0-10 KBytes)

- The thing to watch is overhead

- Bandwidth
- CPU load
- Co-processor
- Network load

Not a factor: not enough network traffic

protocol	cycles	μs
short	2350	**3.35**
eager	4000	5.71
rendezvous	11000	15.71

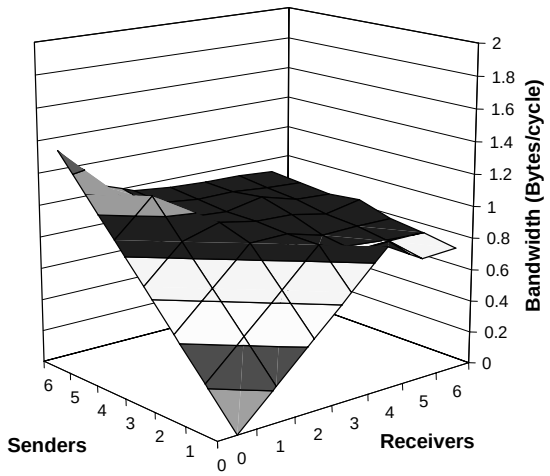
- BlueGene/L network requires 16 byte aligned loads and stores
  - Memory copies to resolve alignment issues
- Compromise solution:
  - Deterministic routing insures good latency but creates network hotspots
  - Adaptive routing avoids hotspots but doubles latency
  - Currently: deterministic routing more advantageous at up to 4k nodes
  - Balance may change as we scale to 64k nodes: shorter messages, more traffic

Optimizing point-to-point communication (long messages)

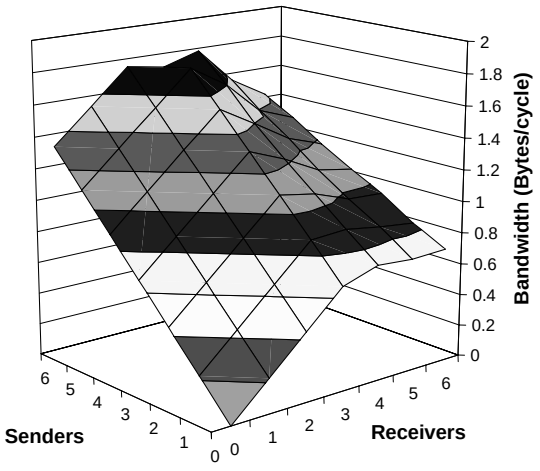
- Most important thing to optimize for: CPU per packet overhead
  - At maximum torus utilization, only 90 CPU cycles available to prepare/handle a packet!
- Sad (measured) reality:
  - READ: 204, WRITE: 50-100 cycles
  - Plus MPI overhead
- Packet overhead reduction
  - Cooked packets:
    - –Contain destination address
    - –Assume initial dialog (rendezvous)
  - Rendezvous costs ≈ 3000 cycles
  - Saves ≈ 100 cycles/packet
  - Allows adaptively routed packets
  - Permits coprocessor mode
- Comm. copro. mode essential
  - (Allows 180 cycles/CPU/packet)
  - Explicit cache management
    - – ≈ 5000 cycles/message
  - System support necessary
    - –Coprocessor library
    - –Scratchpad library
- Adaptive routing essential
  - MPI semantics achieved by initial deterministically routed scout packet
- Packet alignment issues handled with 0 memory copies
  - Overlapping realignment with torus reading
- Drawback: only works well for long messages (10KBytes+)

Optimizing point-to-point communication (long messages)

Per-node bandwidth in heater mode



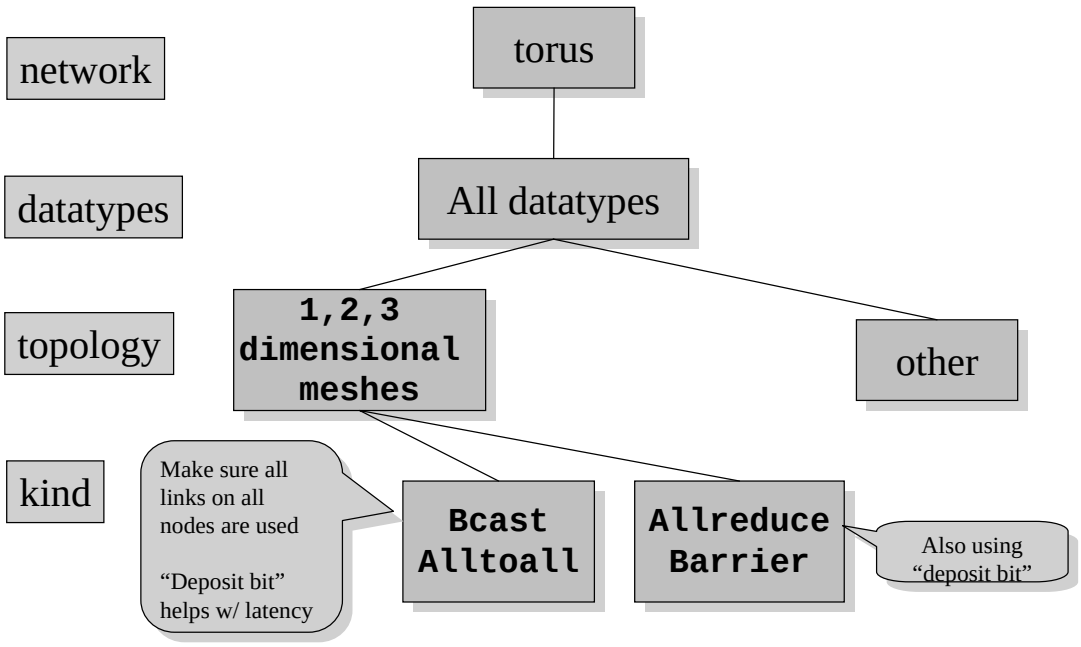
Per-node bandwidth in coprocessor mode



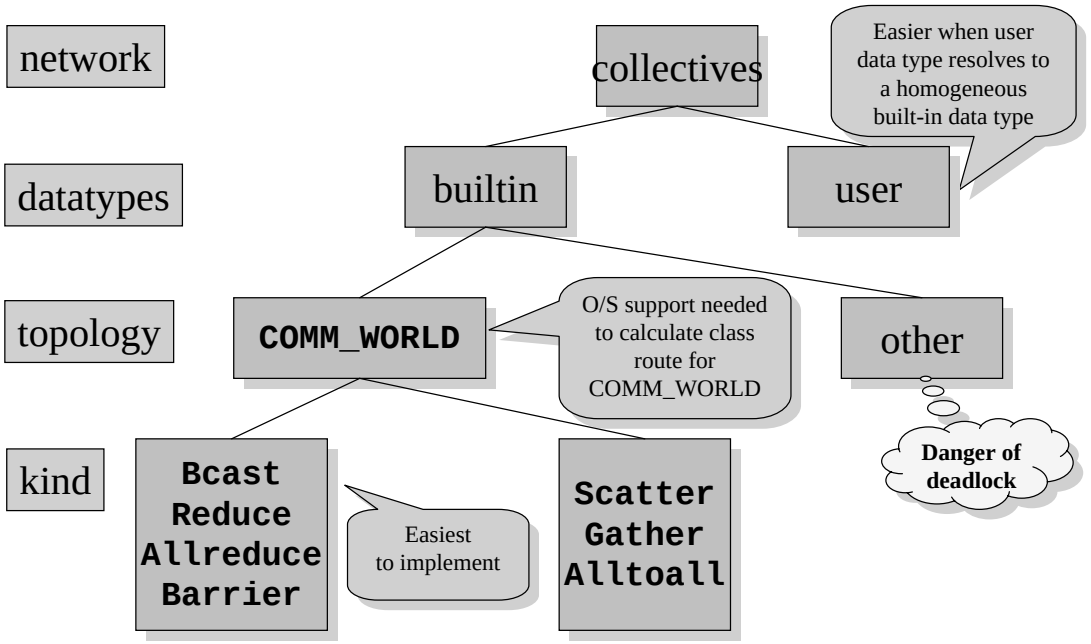
Optimizing Collective Operations

- MPICH2 comes with default collective algorithms
  - Bcast: MST or scatter/allgather
  - Alltoall: recursive dbl., pairwise exchanges
  - Alltoallv: post & waitall
  - Scatter: MST
- Default algorithms not necessarily suitable for torus topology
  - Designed for Ethernet, switched (crossbar) environments
  - E.g. a good plane broadcast algorithm uses the four available links of a node to the maximum
- Taxonomy of possible optimizations

Implementing collectives on the torus network



Implementing collectives on the collectives network



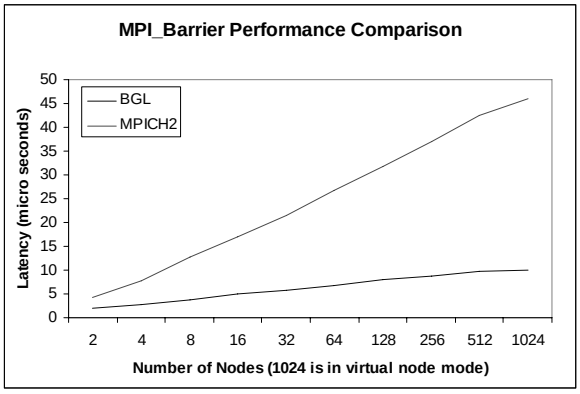
Optimizing MPI Collective Operations

- MPI Collective operations:
  - Executed by all nodes in a group at the same time
  - **barrier**
  - **broadcast**, **scatter**
  - **allreduce**
  - **allgather[v]**
  - **alltoall[v]**
- MPICH2 implementations stable, slow
- Torus, tree based implementations faster

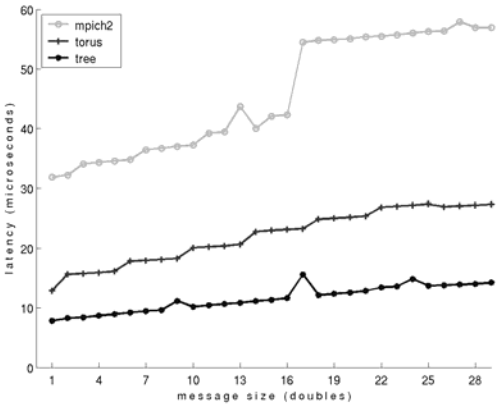
Collective:	MPICH2	torus	tree	GI	Today's best
barrier	40 μs	10 μs	5 μs	**1.5 μs**	**1.5 μs**
broadcast	30MB/s	**> 700MB/s**	350MB/s	-	350MB/s
allreduce	30MB/s	**350MB/s**	350MB/s	-	120MB/s
alltoall	30-40%	**90%**	-	-	**90%**

Barrier and short message Allreduce: Latency and Scaling

Barrier latency vs. machine size

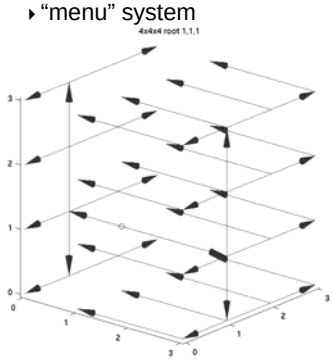


Short-message Allreduce latency vs. message size

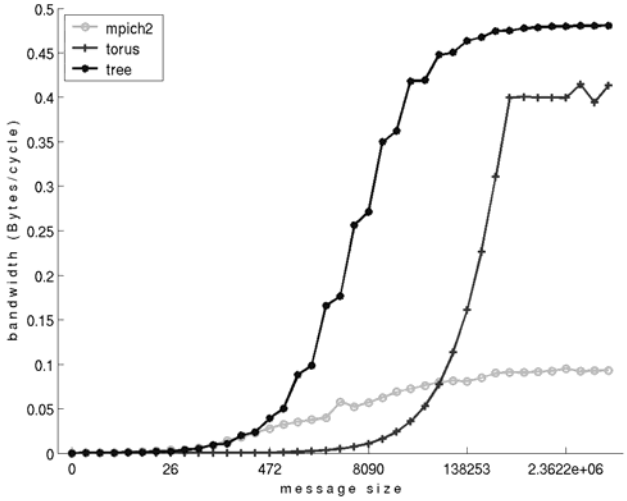


Optimized collectives: Broadcast

- MPICH2: stable but slow
- Tree broadcast:
  - only for MPI_COMM_WORLD
- Torus broadcast:
  - any rectangular communicator
  - Uses deposit bit
  - "menu" system

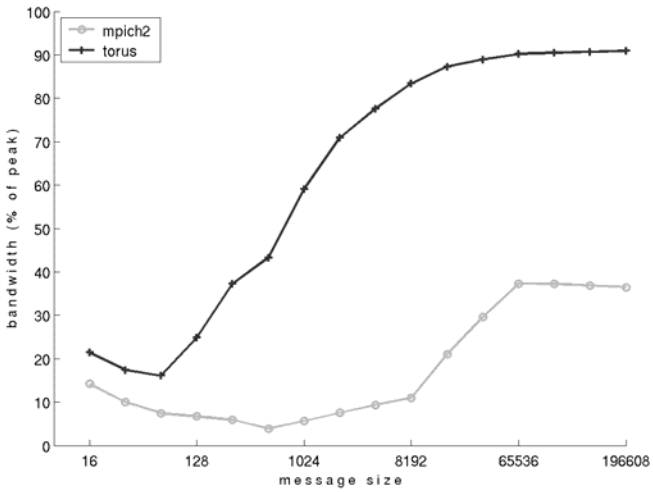


Broadcast bandwidth



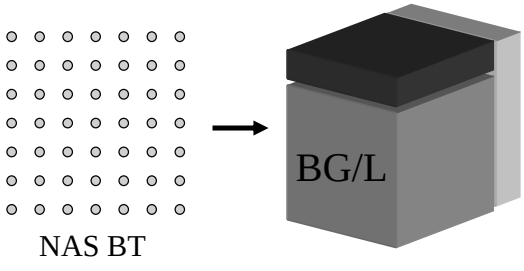
Optimized collectives: Alltoall[v]

- Performance measured as percentage of peak, which is function of partition “shape”
- MPICH2 implementation not suitable for torus network
- Optimized implementation: 90% of peak
- Implemented by Charles Archer (Rochester); measured on an 8x8x8 partition

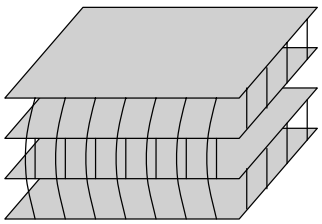


Mapping MPI tasks: NAS BT

- CFD application
  - Navier-Stokes
  - Finite-difference method, 3D
  - Parallelized in 2D
  - Application laid out in a 2D square grid
- BG/L torus network: 3D mesh/torus



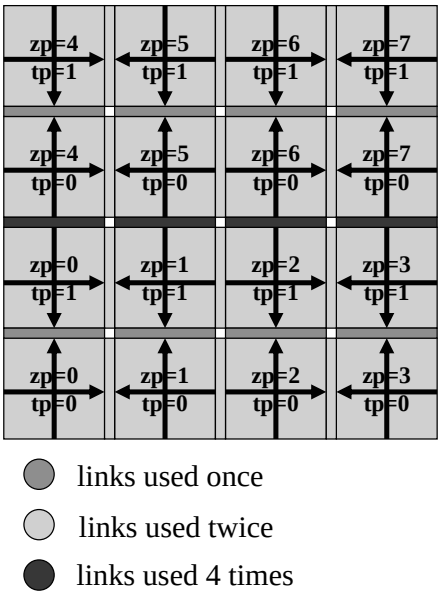
- Naïve mapping: lexicographic ordering
  - Many of the logical links map to the same physical link
  - Link hotspots, bad performance
- Better mapping
  - Minimize number of hotspots
  - Intuition: fold up 2D grid like a napkin
  - Not a perfect solution



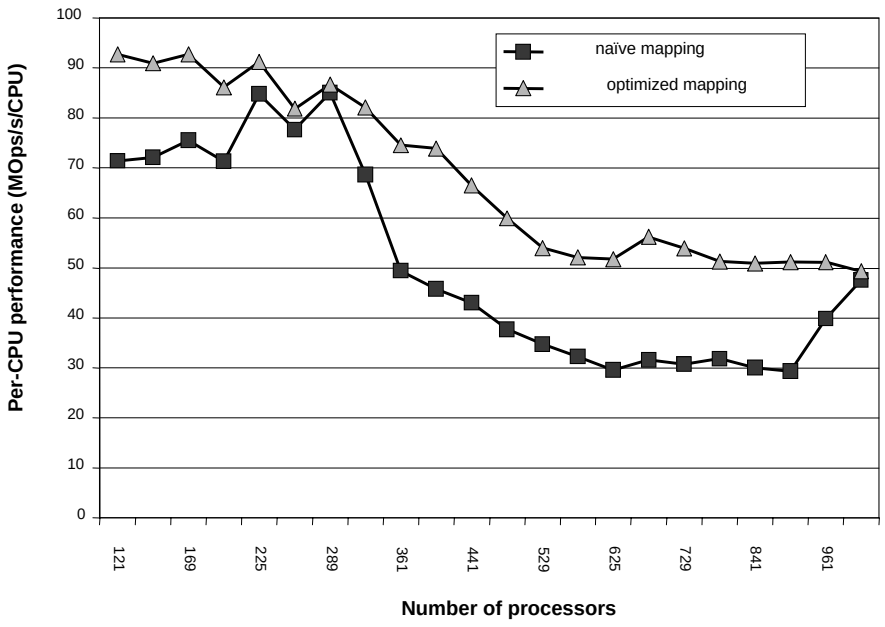
Mapping NAS BT onto 1024 nodes of BG/L

- BG/L: 1024 nodes (virtual node mode):
  - (3+1)D grid, 8 x 8 x 8 x 2 nodes
- Map NAS BT 11x11, 12x12, 13x13, … 32x32 onto the machine
- A function f: $\mathbf{N}^2$->$\mathbf{N}^4$ to map logical (BT) coordinates to physical (BG/L torus) coordinates
- f(x, y) = [xp,yp,zp,tp] where

```
xp = x mod 8      if x/8 is even
xp = 7 – x mod 8  otherwise

yp = y mod 8      if y/8 is even
yp = 7 – y mod 8  otherwise

zp = x/8 + 4*(y/16)

tp = y/8 mod 2
```



NAS BT Scaling (virtual node mode)



Compilers and Math Libraries

Compilers

- Support Fortran95, C99, C++
  - IBM xlf/xlc compilers have been ported to BG/L, with code generation and optimization features for SIMD FPU
- Backend enhanced to support PPC440 and to target SIMD FPU on nodes
  - Finds parallel operations that match SIMD instructions
  - Register allocator enhanced to handle register pairs
  - Instruction scheduling tuned to unit latencies
- Initial design of (2-way) SIMD FPU architecture was driven by key workload kernels such as matrix-matrix product and FFT
  - Identified several mux combinations for SIMD operations not usually seen on other SIMD ISA extensions (Intel SSE, PPC AltiVec), e.g.,

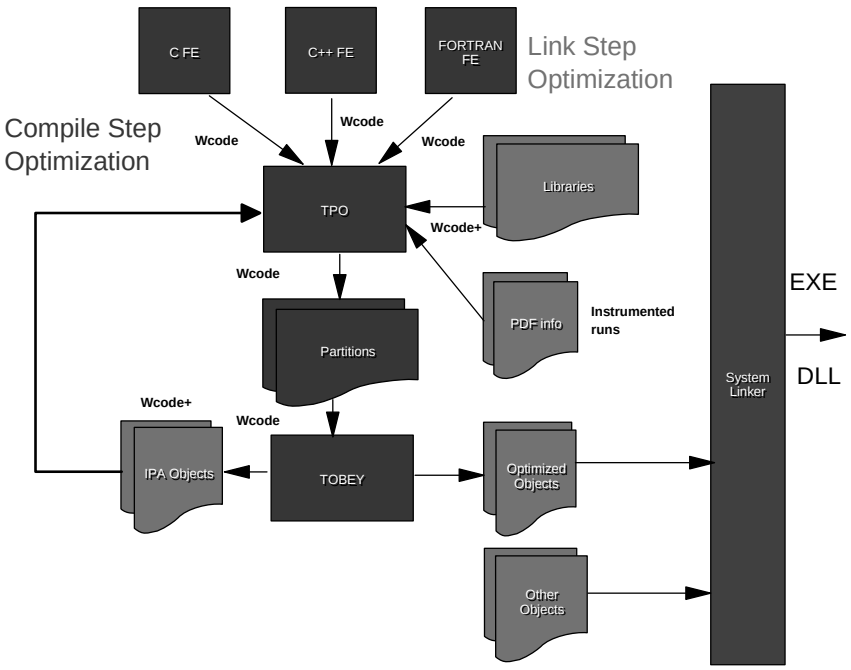
    $d_P = a_P + b_P * c_P \,||\, d_S = a_S - b_S * c_S$

    $d_P = a_P + b_P * c_P \,||\, d_S = a_S + b_P * c_S$

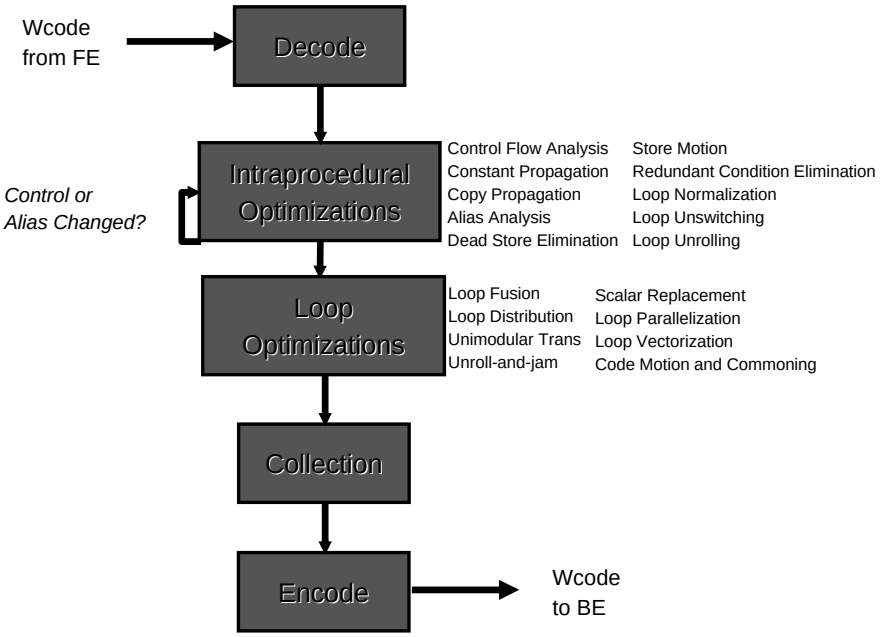
Outline

- Architecture of IBM Compilers
  - TOBEY backend
  - Loop Transformations – background
  - Toronto Portable Optimizer
- Exploiting BG/L SIMD Floating Point Unit
- Dual processor exploitation on compute node
  - Computation offload mode
  - Virtual node mode
- Math libraries for BlueGene/L

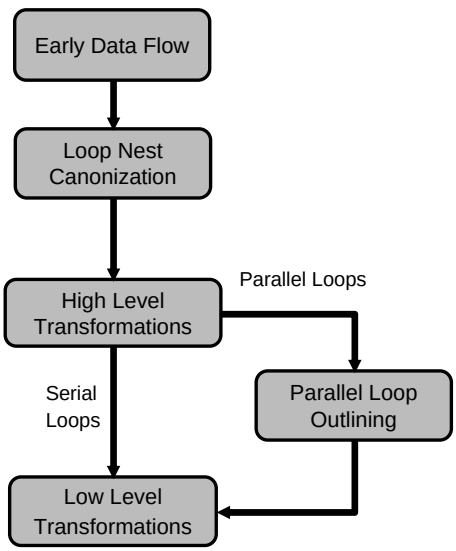
IBM Compiler Architecture



Inside TPO Compile Time Optimization



Loop Optimization Overview In TPO



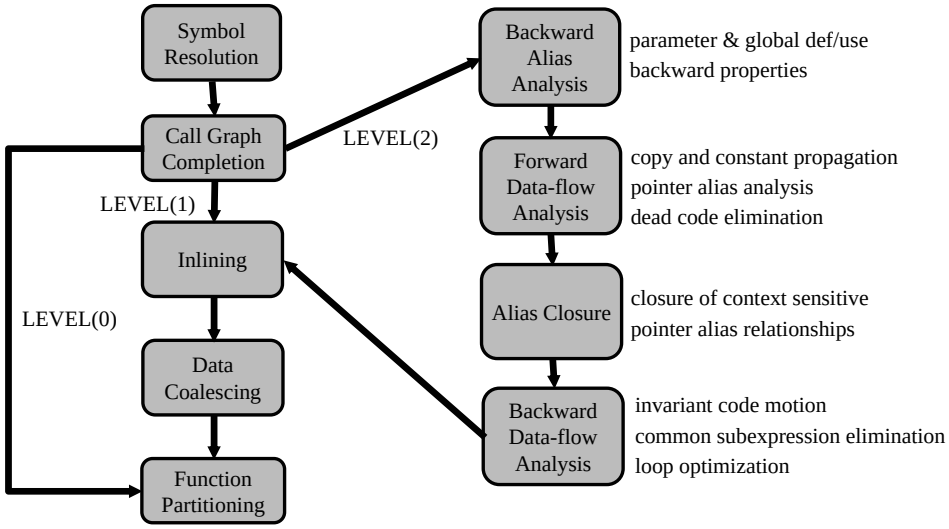
Control Flow Optimization
Data Flow Optimization
Loop Normalization

Aggressive Copy Propagation
Maximal Loop Fusion

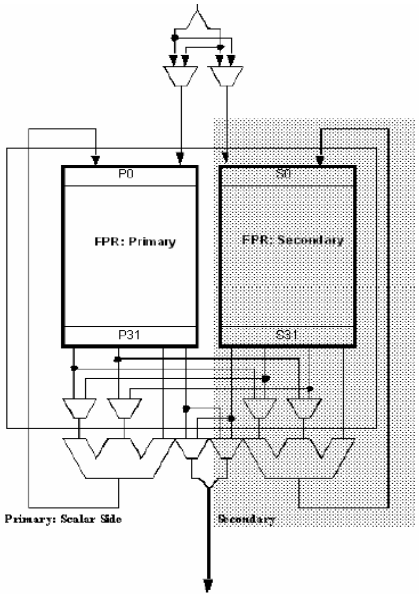
Code Sinking
Loop Nest Partitioning
Loop Interchange
Loop Unroll and Jam
Scalar Replacement
Loop Parallelization
Gather/Scatter

Index Set Splitting
Node Splitting
Inner Loop Distribution
Loop Vectorization
Prefetch Optimizations
Inner Loop Unrolling
Redundancy Elimination
Code Motion

Inside TPO Link Time Optimization



Dual FPU Architecture



- Dual floating-point unit
- 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
  - Load/store with swap order available
  - Data needs to be 16-byte aligned
    - Useful for matrix transpose

Strategy to Exploit SIMD FPU

- Automatic code generation by compiler
- User can help the compiler via pragmas and intrinsics
  - Pragma for data alignment: *__alignx(16, var)*
  - Pragma for parallelism
    - Disjoint: *#pragma disjoint (*a, *b)*
    - Independent: *#pragma ibm independent* loop
  - Intrinsics
    - Intrinsic function defined for each parallel floating point operation
      - E.g.: *D = __fpmadd(B, C, A) => fpmadd rD, rA, rC, rB*
    - Control over instruction selection, compiler retains responsibility for register allocation and scheduling
- Using library routines where available
  - Dense matrix BLAS – e.g., DGEMM, DGEMV, DAXPY
  - FFT

Example: Vector Add

```
void vadd(double* a, double* b, double* c,
  int n)
{
  int i;
  for (i=0; i<n; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```


Compiler transformations for Dual FPU

```
void vadd(double* a, double* b, double* c,
 int n)
{
  int i;
  for (i=0; i<n-1; i+=2)
    {
      c[i]   = a[i]   + b[i];
      c[i+1] = a[i+1] + b[i+1];
    }
  for (; i<n; i++) c[i] = a[i] + b[i];
}
```


Compiler transformations for Dual FPU

```
void vadd(double* a, double* b, double* c,
 int n)
{
  int i;
  for (i=0; i<n-1; i+=2)
    {
      c[i]   = a[i]   + b[i];
      c[i+1] = a[i+1] + b[i+1];
    }
  for (; i<n; i++) c[i] = a[i] + b[i];
}
```

```
LFPL  (pa, sa) = (a[i], a[i+1])
LFPL  (pb, sb) = (b[i], b[i+1])
FPADD (pc, sc) = (pa+pb, sa+sb)
SFPL  (c[i], c[i+1]) = (pc, sc)
```


Pragmas and Advanced Compilation Techniques

```
void vadd(double* a, double* b, double* c, int n)
{
#pragma disjoint(*a, *b, *c)
__alignx(16,a+0);
__alignx(16,b+0);
__alignx(16,c+0);
  int i;
  for (i=0; i<n; i++)
    {
      c[i] = a[i] + b[i];
    }
}
```

Coming soon

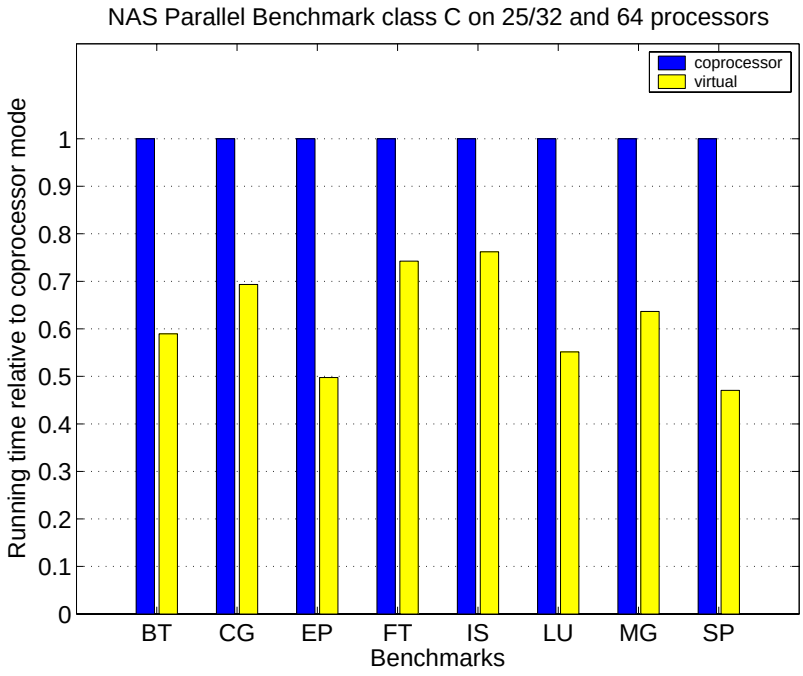
Coming Soon (Using TPO)

- **Interprocedural pointer alignment analysis**
- **Loop transformations to enable SIMD code generation in absence of compile-time alignment information**
  - **loop versioning**
  - **loop peeling**

Virtual Node Mode

- Can lead to better performance density
  - One MPI process per processor – two MPI processes per compute node
  - Communication between processes on same compute node using scratch space in L3 cache
- Issues
  - Compute node resources split into half – only half of memory (256 MB) available to each MPI process
  - Each CPU has to deal with both computation and communication
    - No overlap between computation and communication
    - Responsiveness to incoming messages

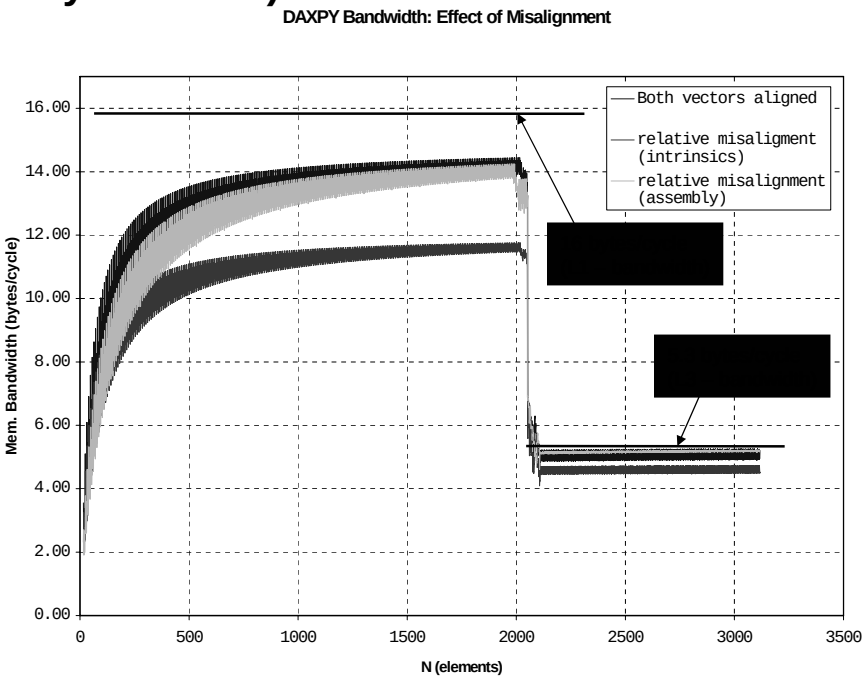
NAS



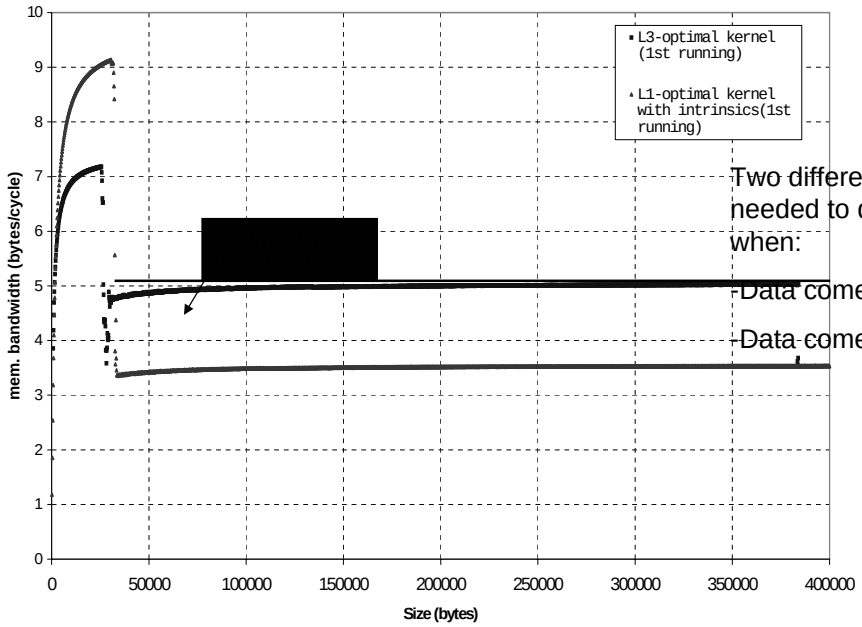
Math Library

- ESSL: Small subset (of ~500 routines)
  - Mainly dense matrix kernels – DGEMM, DGEMV, DDOT, DAXPY, DAAT, Cholesky and LU factorization
  - Exploiting second CPU for computation-intensive kernels
  - Using ESSL source code to drive compiler testing and exploration of complete ESSL support
- MASSV: Small subset (of ~50 routines)
  - Reciprocal, square root, reciprocal square root, exponential, logarithm, cube root – prioritized based on early applications
  - Exploring complete support using equivalent routines in C
- FFT
  - Technical University of Vienna developed FFT library optimized for BlueGene/L – effective use of the SIMD FPU

DAXPY: Effect of Data Alignment (Intrinsics and Assembly Versions)



L1 and L3-optimal DGEMV Bandwidth Utilization



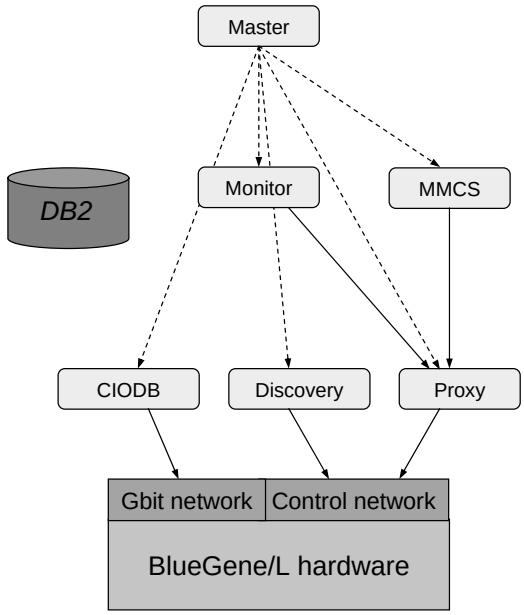
Two different kernels are needed to deal with data when:

- Data come out of L1
- Data come out of L3

Control System: Vision and Principles

- Separation between execution mechanisms in the core and policy decisions in the service node
- Global O/S (MMCS in the service node)
  - Makes all global and collective decisions
  - Interfaces with external policy modules (e.g., scheduler)
- Use commercial database technology (DB2) to store all static and dynamic state of the global O/S
  - Scalability, robustness, security, recovery, logging
  - Database is also used as a communication mechanism
- Local O/S (Linux for I/O node and CNK for compute node)
  - Implementation of services: I/O, creation of processes, signals, debug
  - Responsible for local decisions that do not affect overall operation of the machine (e.g., process scheduling in the I/O node)

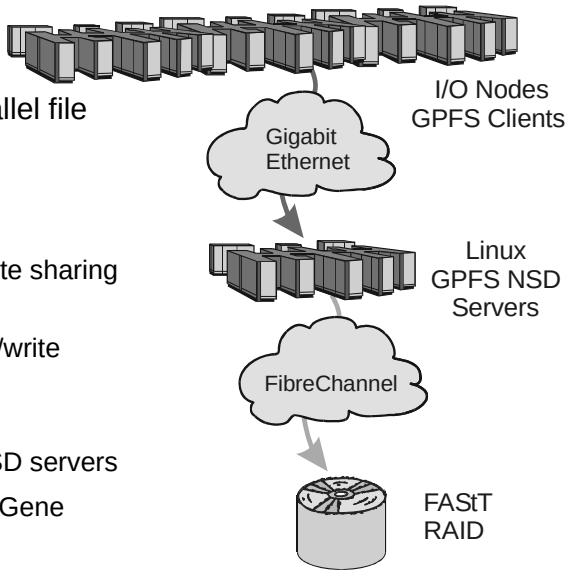
BG/L control system processes



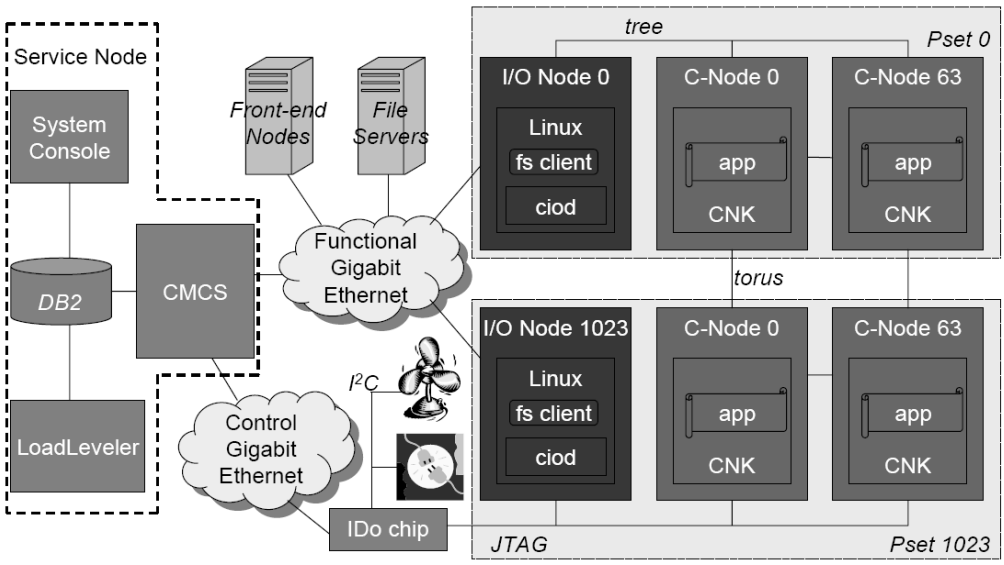
- Master
  - creates, monitors, and restarts the other processes
- Discovery
  - find and initializes new hardware
- Proxy
  - virtualizes the IDo hardware, providing reliable and atomic connection
- Monitor
  - monitors environmentals, such as temperature and voltages
- MMCS
  - configures and IPLs partitions of the machine, bringing those partitions to a user-architected state
- CIODB
  - connects to the ciods in the I/O nodes
- All processes interact with the DB

General Parallel File System (GPFS) for Blue Gene/L

- BlueGene/L can generate enormous I/O demand
  - 10 GB/sec of writes per I/O-rich rack
  - 6 GB/sec of reads per I/O-rich rack
- Serving this kind of demand requires a parallel file system
- NFS for file I/O
  - Limited scalability
  - NFS has no cache consistency, making write sharing difficult
  - Poor performance, not enough read ahead/write behind
- GPFS ported to Blue Gene
  - GPFS clients in Blue Gene call external NSD servers
  - Brings traditional benefits of GPFS to Blue Gene
    - I/O parallelism
    - Cache consistent shared access
    - Aggressive read-ahead, write-behind



Blue Gene/L System Architecture



The IBM High Performance Computing Toolkit on BlueGene/L

IBM High Performance Computing Toolkit on BG/L

- **MPI performance: MP_Profiler**
- **CPU performance: Xprofiler, HPM**
- **Visualization and analysis: PeekPerf**

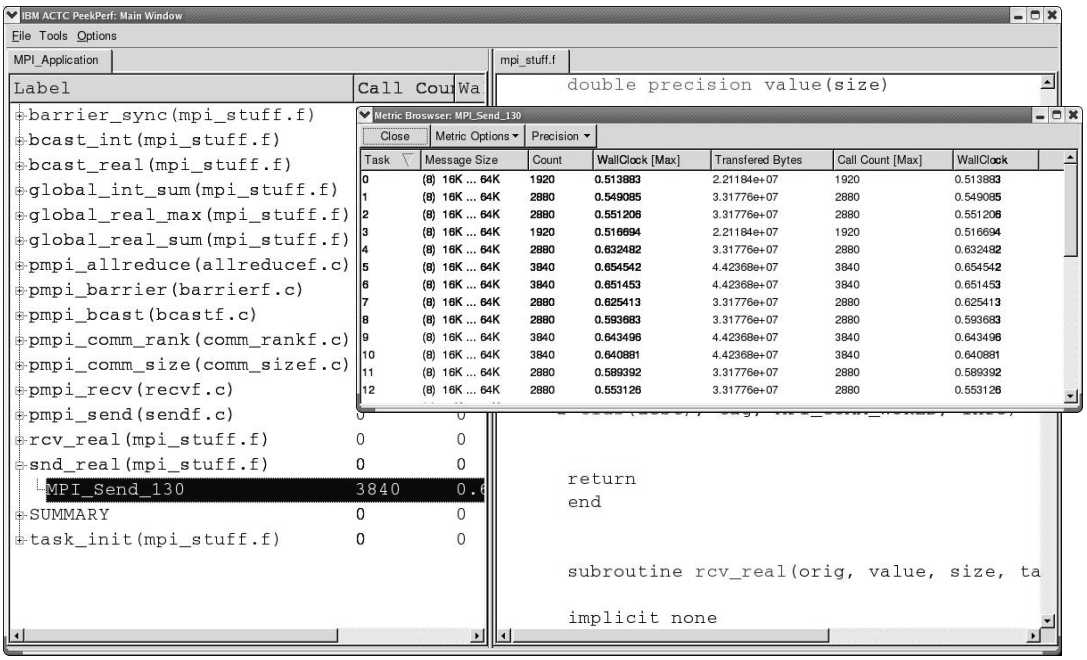
http://www.research.ibm.com/actc/

http://www.absoft.com/Products/Tools/hpc-toolkit/

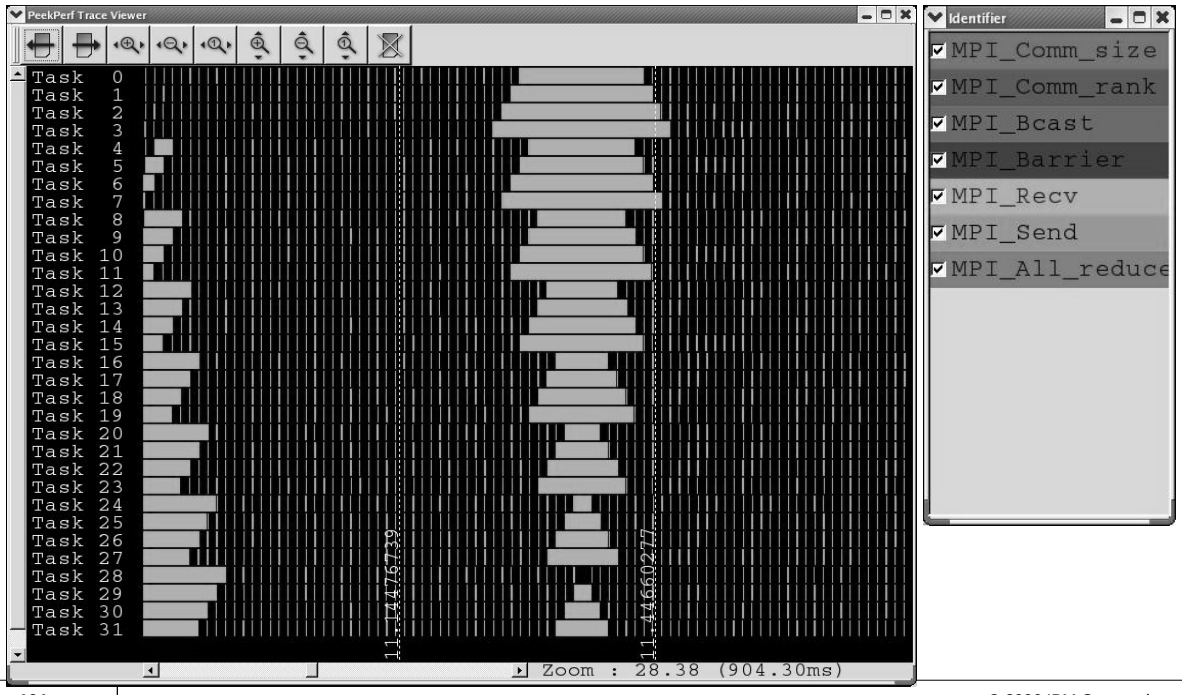
Message-Passing Performance:

- **MP_Profiler Library**
  - Captures “summary” data for MPI calls
  - Source code traceback
  - User **MUST** call MPI_Finalize() in order to get output files.
  - No changes to source code
    - MUST compile with –g to obtain source line number information
- **MP_Tracer Library**
  - Captures “timestamped” data for MPI calls
  - Source traceback

MP_Profiler Output with Peekperf



MP_Profiler - Traces



MP_Profiler Message Size Distribution

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Comm_size	1 (1)	0 ... 4	0	1E-07
MPI_Comm_rank	1 (1)	0 ... 4	0	1E-07
MPI_Isend	2 (1)	0 ... 4	3	0.000006
MPI_Isend	2 (2)	5 ... 16	12	1.4E-06
MPI_Isend	2 (3)	17 ... 64	48	1.3E-06
MPI_Isend	2 (4)	65 ... 256	192	1.3E-06
MPI_Isend	2 (5)	257 ... 1K	768	1.3E-06
MPI_Isend	2 (6)	1K ... 4K	3072	1.3E-06
MPI_Isend	2 (7)	4K ... 16K	12288	1.3E-06
MPI_Isend	2 (8)	16K ... 64K	49152	1.3E-06
MPI_Isend	2 (9)	64K ... 256K	196608	1.7E-06
MPI_Isend	2 (A)	256K ... 1M	786432	1.7E-06
MPI_Isend	1 (B)	1M ... 4M	1048576	9E-07

MPI Function	#Calls	Message Size	#Bytes	Walltime
MPI_Irecv	2 (1)	0 ... 4	3	4.7E-06
MPI_Irecv	2 (2)	5 ... 16	12	1.4E-06
MPI_Irecv	2 (3)	17 ... 64	48	1.5E-06
MPI_Irecv	2 (4)	65 ... 256	192	2.4E-06
MPI_Irecv	2 (5)	257 ... 1K	768	2.6E-06
MPI_Irecv	2 (6)	1K ... 4K	3072	3.4E-06
MPI_Irecv	2 (7)	4K ... 16K	12288	7.1E-06
MPI_Irecv	2 (8)	16K ... 64K	49152	2.23E-05
MPI_Irecv	2 (9)	64K ... 256K	196608	9.98E-05
MPI_Irecv	2 (A)	256K ... 1M	786432	0.00039
MPI_Irecv	1 (B)	1M ... 4M	1048576	0.000517
MPI_Waitall	21 (1)	0 ... 4	0	1.98E-05
MPI_Barrier	5 (1)	0 ... 4	0	7.8E-06

Xprofiler

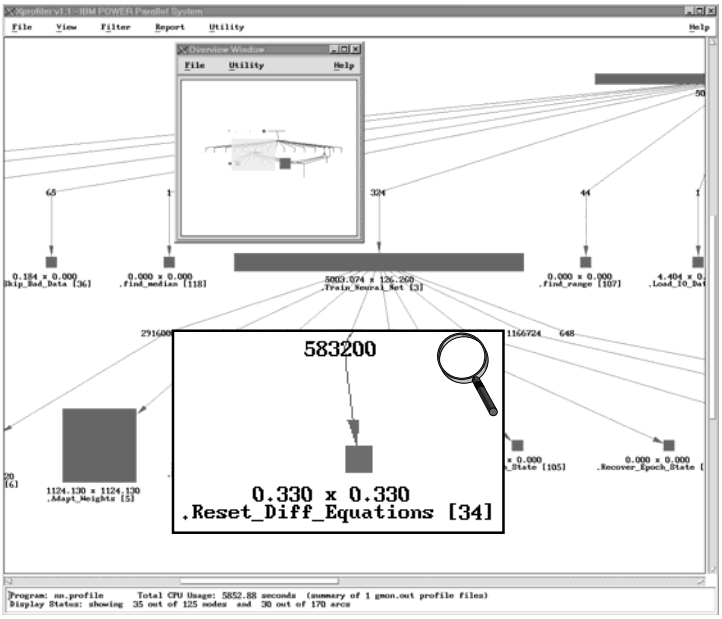
- **CPU profiling tool similar to gprof**
- **Can be used to profile both serial and parallel applications**
- **Use procedure-profiling information to construct a graphical display of the functions within an application**
- **Provide quick access to the profiled data and helps users identify functions that are the most CPU-intensive**
- **Based on sampling (support from both compiler and kernel)**
- **Charge execution time to source lines and show disassembly code**

Running Xprofiler

- **Compile the program with -pg**
- **Run the program**
- **gmon.out file is generated (MPI applications generate gmon.out.1, …, gmon.out.n)**
- **Run Xprofiler**

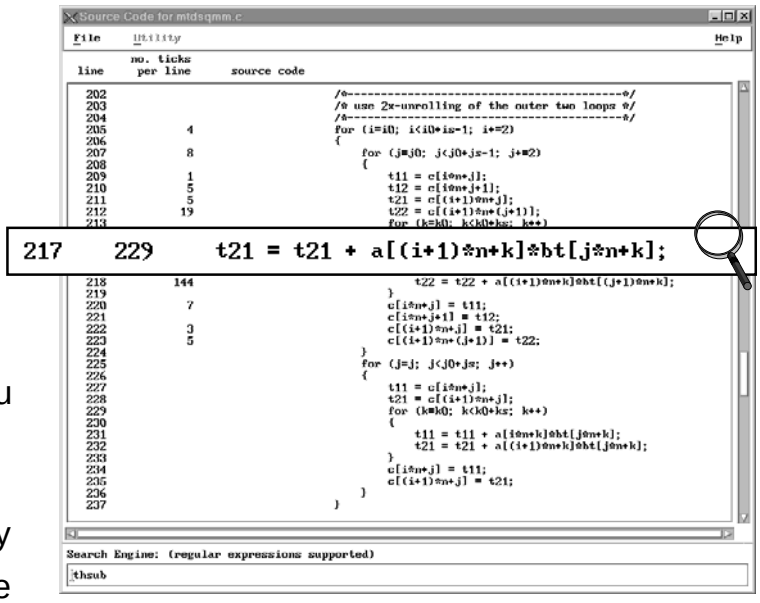
Xprofiler: Main Display

- **Width of a bar: time including called routines**
- **Height of a bar: time excluding called routines**
- **Call arrows labeled with number of calls**
- **Overview window for easy navigation (View → Overview)**

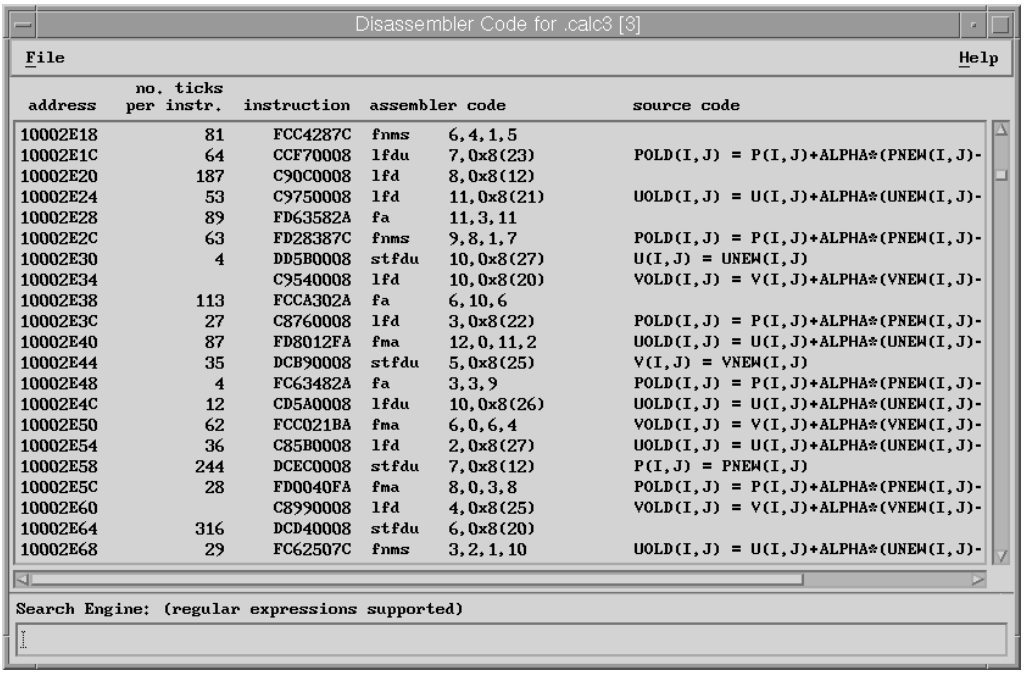


Xprofiler: Source Code Window

- **Source code window displays source code with time profile (in ticks=.01 sec)**
- **Access**
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



Xprofiler - Disassembler Code



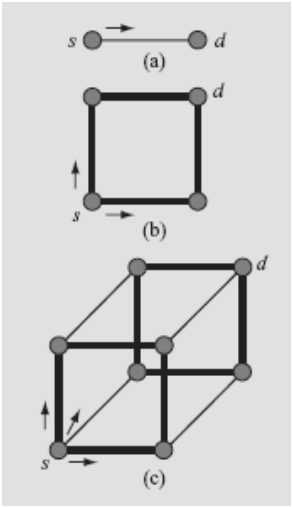
LIBHPM

- **Instrumentation library**
- **Provides performance information for instrumented program sections**
- **Supports multiple (nested) instrumentation sections**
- **Multiple sections may have the same ID**
- **Run-time performance information collection**
- **Based on bgl_perfctr layer**

BlueGene/L Application Performance and Power Analysis

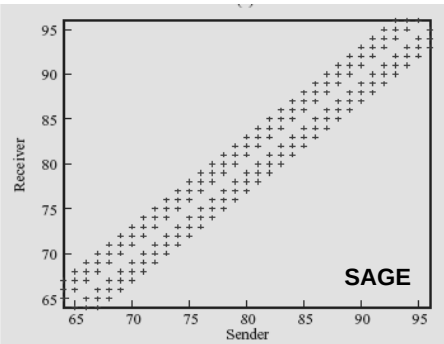
Application Mapping to BlueGene/L

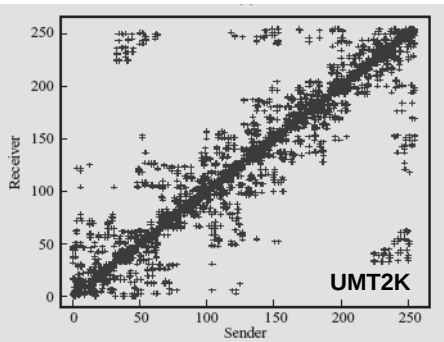
- **Continuous problem mapped to finite number of grid or mash nodes**
  - Spacing of grid determines the accuracy and cost of the computation
- **Mapping of grid onto compute nodes determines the cost of communication**
  - on BlueGene/L, nodes arranged in a 3D lattice and connect only to their six nearest neighbors through the torus
  - Increase in length of communication path increases latency and probability of link contention
  - If message can access multiple paths, available bandwidth increases



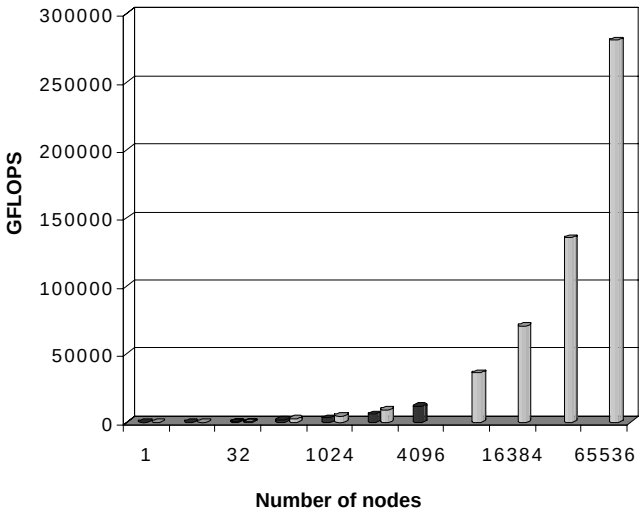
Communication Pattern

- **Communication pattern depends on application**
  - Typically having most of the communication with nearest neighbors and limited communication with remote nodes
- **Adaptive Mesh Refinement**
  - "difficult" selected regions (due to discontinuities, shocks, steep gradients, etc.) superimpose finer sub-grid spacing
  - Introduces irregularity in the torus nearest neighbor communication pattern
  - due to multiple paths, this has little or no effect on overall torus bandwidth





LINPACK Performance



DD1 hardware @ 500 MHz

#4 on June 2004 TOP500 list

11.68 TFLOP/s on 4K nodes

DD2 hardware @ 700 MHz

#1 on Nov 2004 TOP500 list

70.72 TFLOP/s on 16K nodes

#1 on November 2005 TOP500 list

280.6 TFLOP/s on 64K nodes

5 of top 10 IBM, 3 based on Blue Gene/L

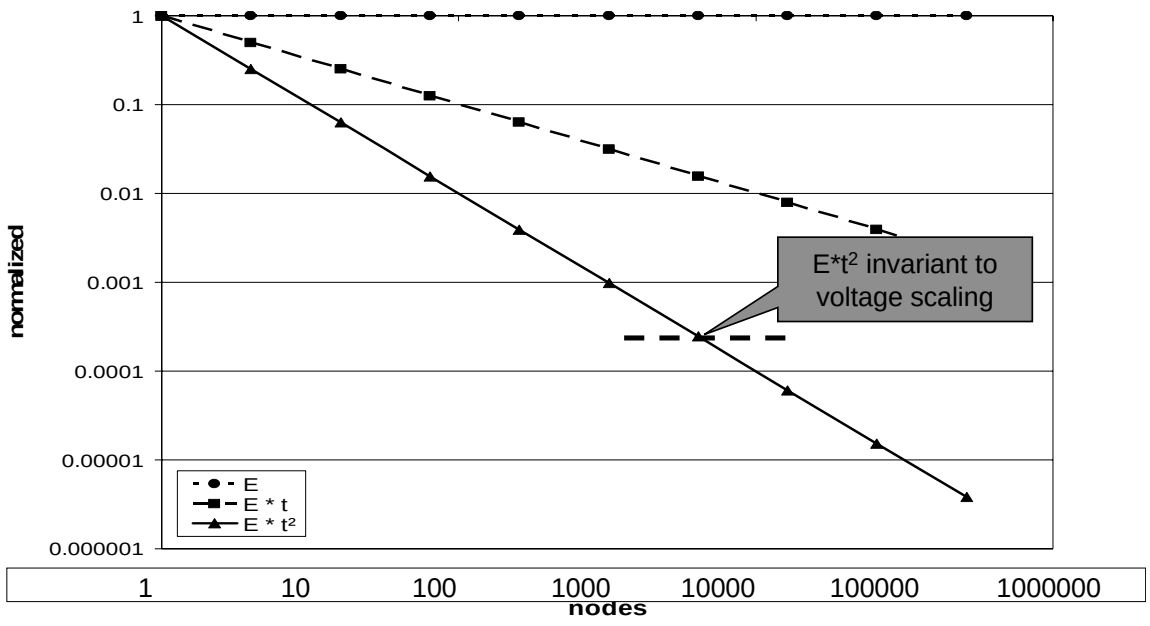
78% of peak

Still only system exceeding 100 TFLOP/s

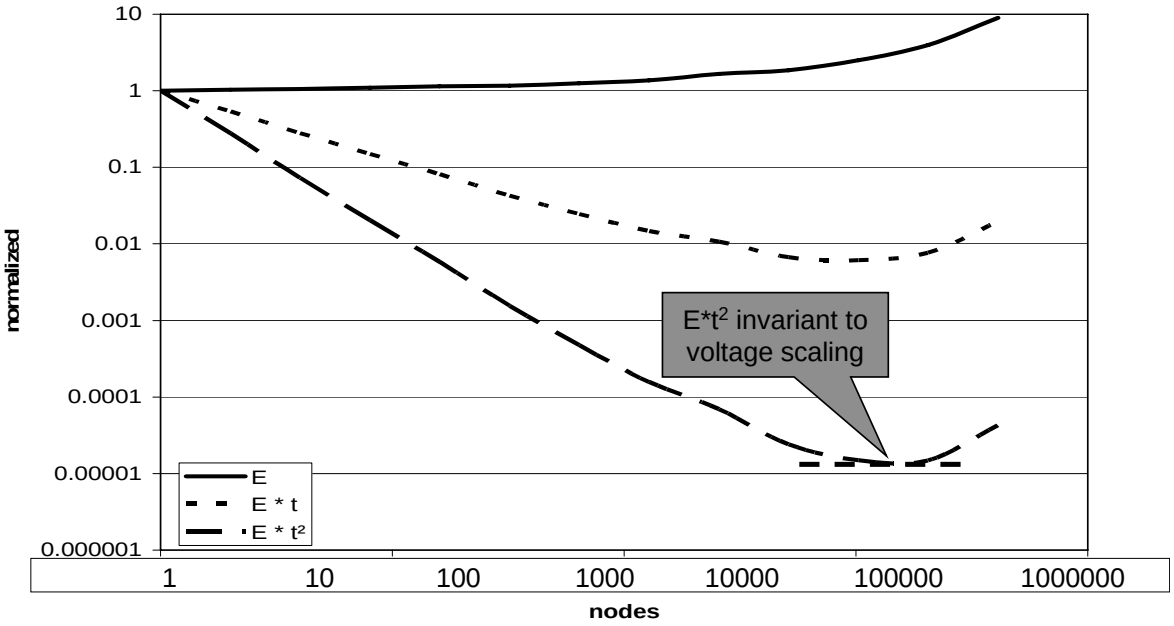
Application Performance and Power Efficiency

- **Figures of merit:**
  - t -- time (delay)
    - application execution time
  - E -- energy (W/MIPS)
    - energy dissipated to execute application
  - E * t -- energy-delay [Gonzalez Horowitz 1996]
    - energy and delay are equally weighted
  - $E * t^2$ -- energy-delay squared [Martin *et al.* 2001]
    - metric invariant on the assumption of voltage scaling

Low Power - High Performance System Concept



Power/Performance Efficiency for a Strong Scaling Problem

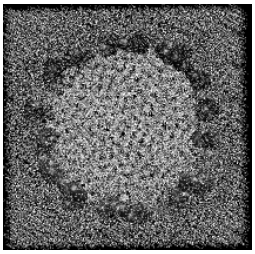


Applying Metrics to Actual Applications

- **LINPACK highly parallel – follows 78% of peak performance**
  - problem size matches the size of the system
  - week scaling
- **Many applications require constant amount of computation regardless of the size of the system**
  - fixed sized problems
  - strong scaling
  - more conservative performance evaluation
- **Apply metrics for several applications and problems**
  - e.g., NAMD, UMT2K

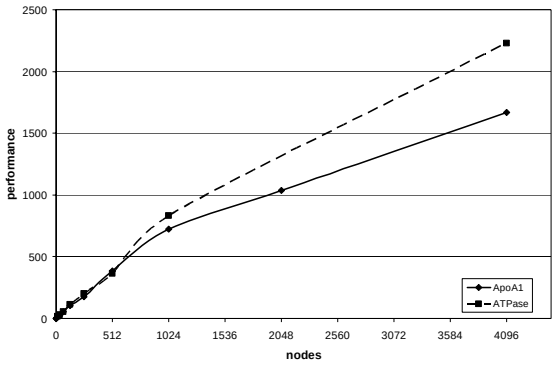
NAMD

- **Parallel, object-oriented molecular dynamics code designed for high-performance simulation of large biomolecular systems**
  - developed by the Theoretical Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign
  - Open source code
- **NAMD benchmarks**
  - ApoA1
    - one molecule of apoprotein A1 solvated in water
    - fixed size problem on 92,224 atoms
  - ATPase
    - F1 subunit of ATP synthase
    - protein and water
    - 327k atoms

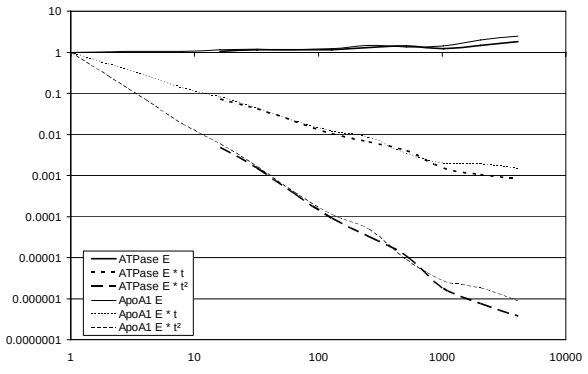


NAMD Performance and Power/Performance Efficiency

Performance scaling - normalized



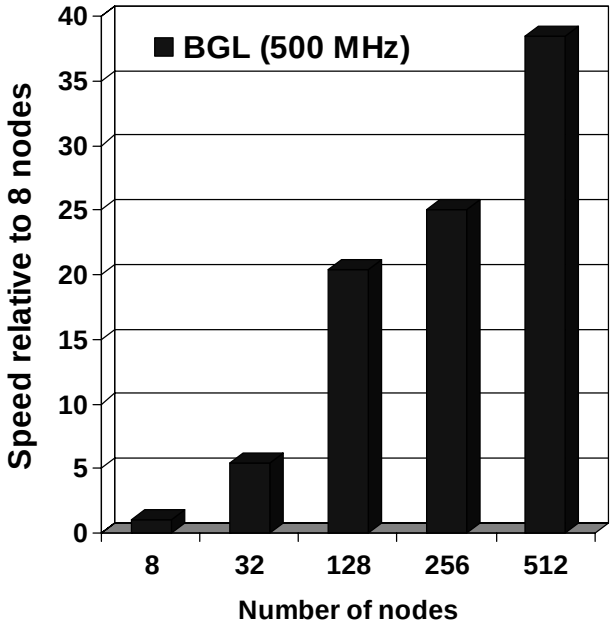
Power/Performance efficiency on log-log



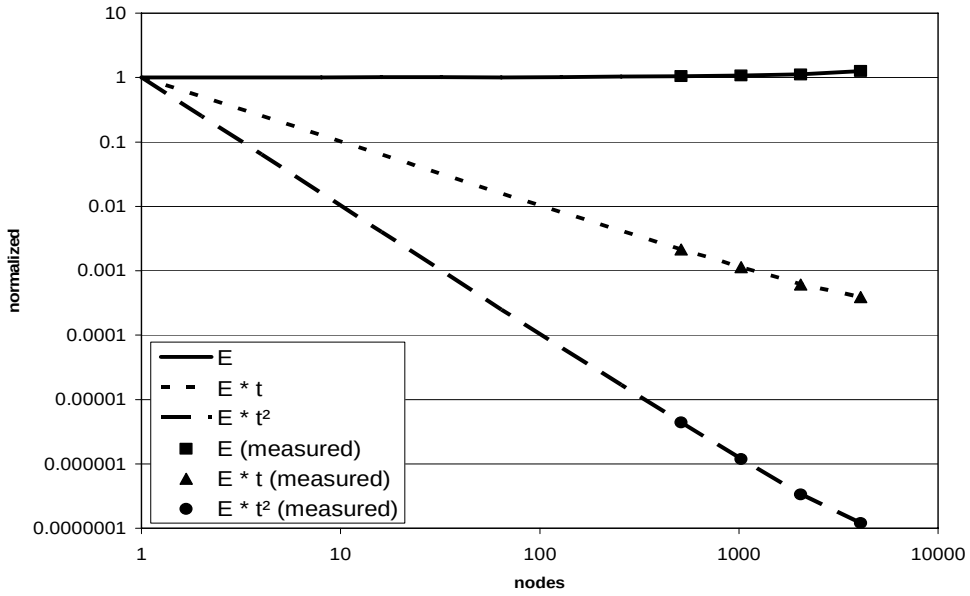
Source: Salapura et al., IEEE Micro, 2006

ASCI Purple Benchmarks – UMT2K

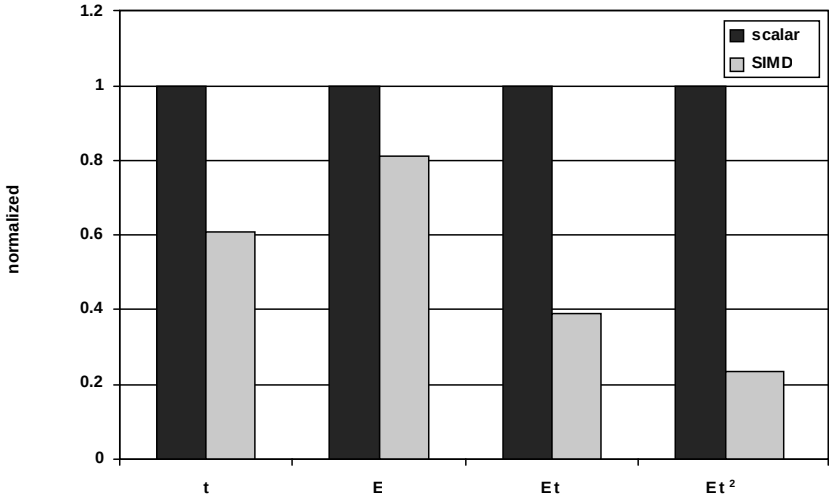
- **Unstructured mesh radiation transport**
  - Typically used as week scaling application
  - Can be used for fixed size problems
- **Excellent scalability up to mid-sized configurations**
  - load balancing problems when scaling to 2000 or more nodes
  - needed algorithmic changes in original program
  - tuned UMT2K version scales well beyond 8000 BlueGene/L nodes



UMT2K Power and Power Performance



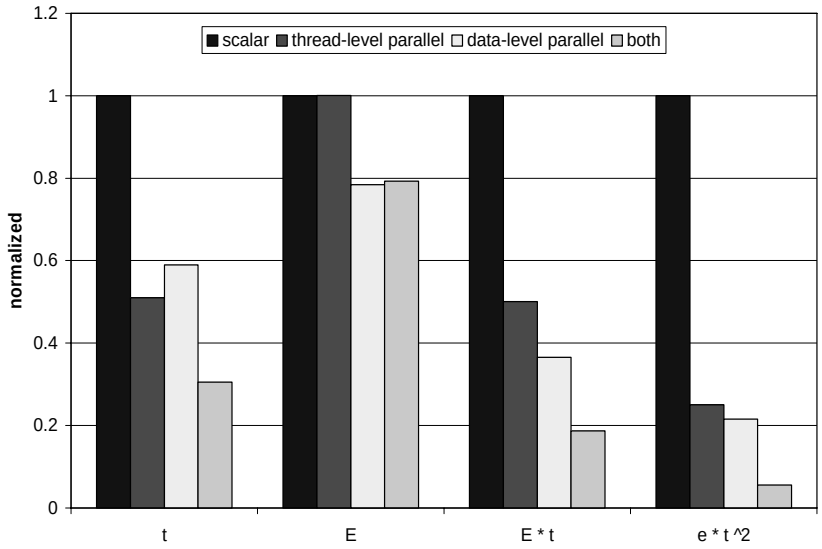
Exploring the Benefits of SIMD



- **Power efficient**
- **Low overhead (doubles data computation without paying cost of instruction decode, issue etc.)**

- **UMT2K runs on 1024 nodes**
- **Code optimized to exploit SIMD floating point**

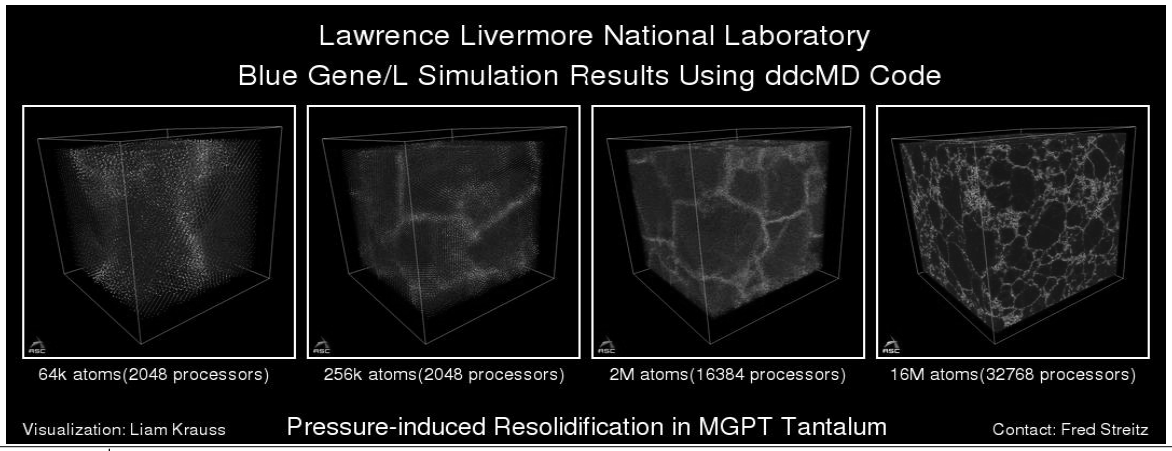
SIMD and Thread Level Parallelism



- **Exploiting parallelism at all levels offers the biggest power/performance advantage**

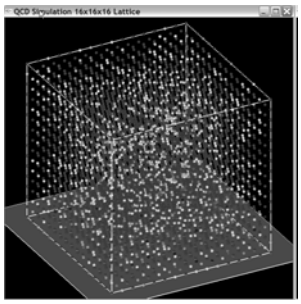
ddcMD: *2005 Gordon Bell Prize Winner*

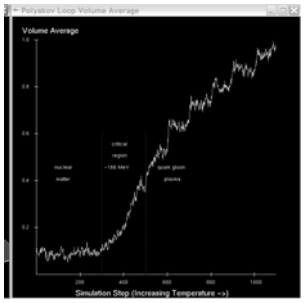
- **Molecular dynamics**
  - Models solidification process - determines structure and stability of metals under dynamic loading conditions
- **524 million atom simulations on 64K nodes are orders of magnitude larger than any previously attempted runs**
  - Example: Solidification of molten Ta at 5000K during isothermal compression to 250GPa
- **Blue Gene enables important scientific findings**



QCD: *2006 Gordon Bell Prize Winner*

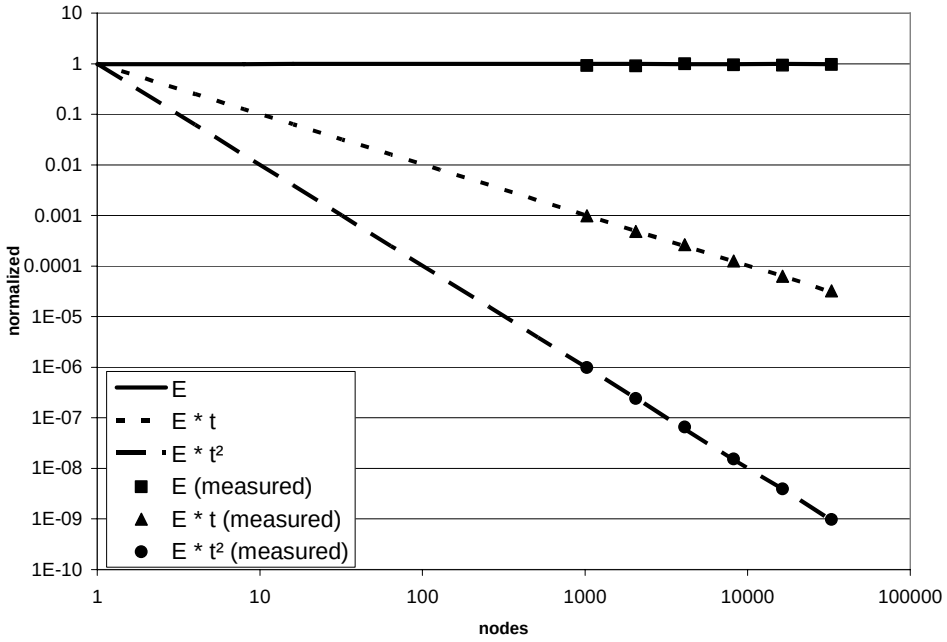
- **Quantum chromo dynamics**
  - the theory of the strong nuclear force that binds the constituents of sub-nuclear matter, the quarks and gluons, to form stable nuclei and therefore more than 90% of the visible matter of the Universe
- ***70.9 Tflops!***
- **Significance of QCD**
  - Cosmological models depend on the mechanism that transformed the primordial quark gluon plasma into stable nuclear matter
  - The Big Bang simulation
- **Employs Domain Wall Fermion (DWF) method**
  - uses a fifth space time dimension to eliminate systematic errors
- **LQCD breaks new ground in the understanding of sub-nuclear matter**





Source: Vranas et al., Supercomputing 2006

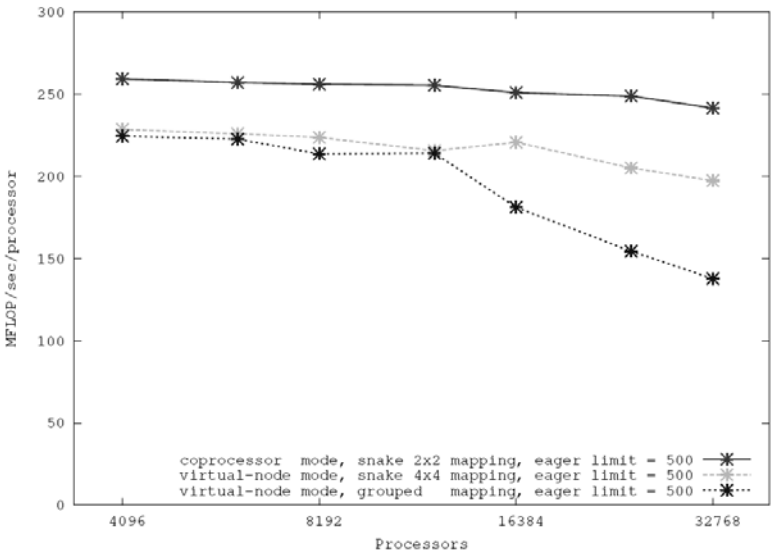
QCD Power/Performance Efficiency



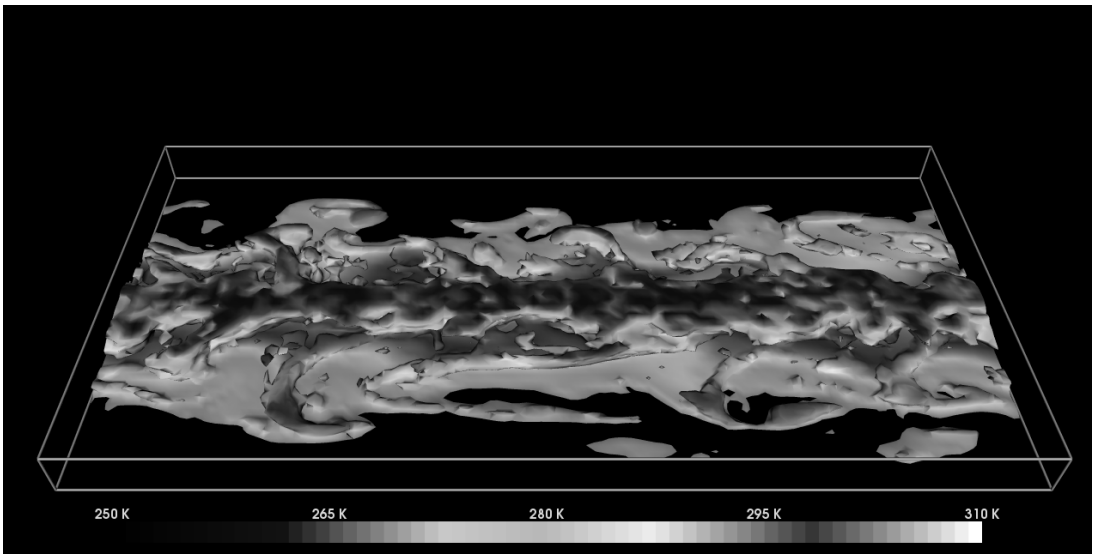
HOMME

- **National Center for Atmospheric Research**
  - cooperation NCAR, Boulder and IBM
- **Moist Held-Suarez test**
  - atmospheric moist processes fundamental component of atmospheric dynamics
    - most uncertain aspect of climate change research
  - moisture injected into the system at a constant rate from the surface
- **Importance of problem**
  - moist processes must be included for relevant weather model
    - formation of clouds and the development and fallout of precipitation
  - requires high horizontal and vertical resolution
    - order of 1 kilometer
  - key to a better scientific understanding of global climate change

HOMME: Strong Scaling

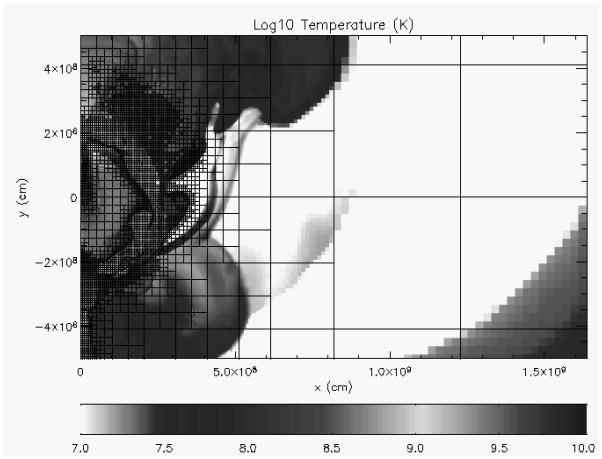


Homme: visualisation

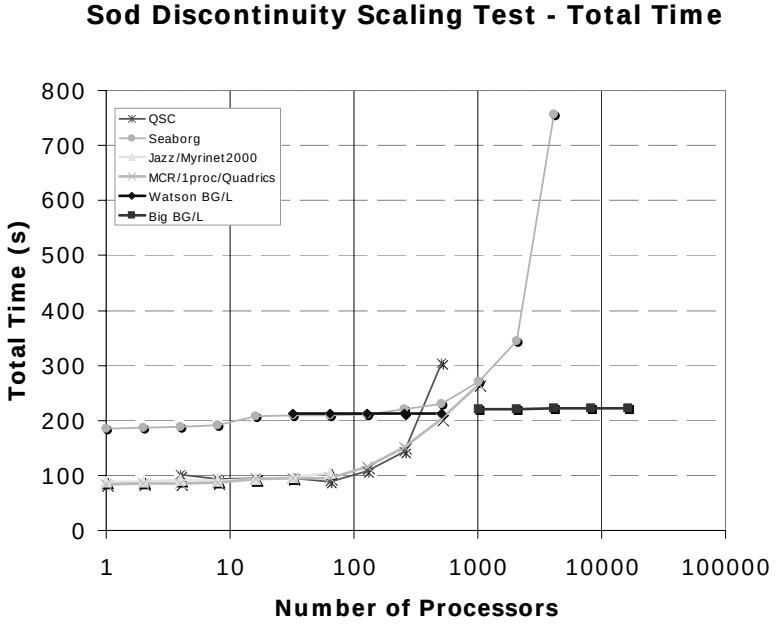


FLASH

- **University of Chicago and Argonne National Laboratory**
- **the Gordon-Bell prize winner for performance**
  - on the ASCI Red system at Sandia National Laboratory (6k processors)
- **for nuclear astrophysical problems related to exploding stars**
  - parallel adaptive-mesh
  - solves the Euler equations for compressible flow and the Poisson equation for self-gravity
- **Exploring the regime directly before and after the detonation prohibitively expensive for current systems**
  - BlueGene/L allows new study of 3D flame front instead of 2D

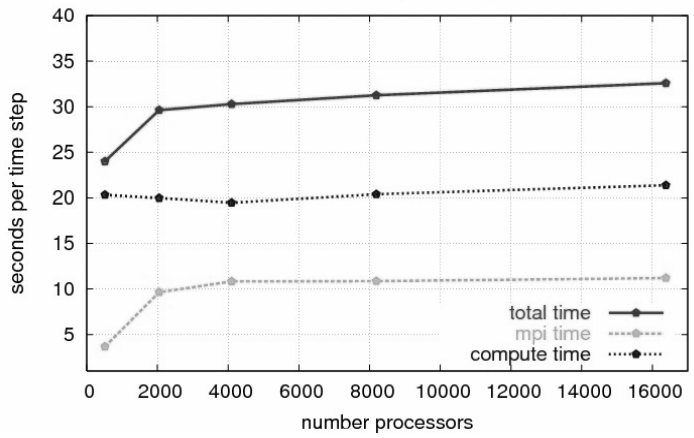


FLASH BlueGene/L Performance



Miranda

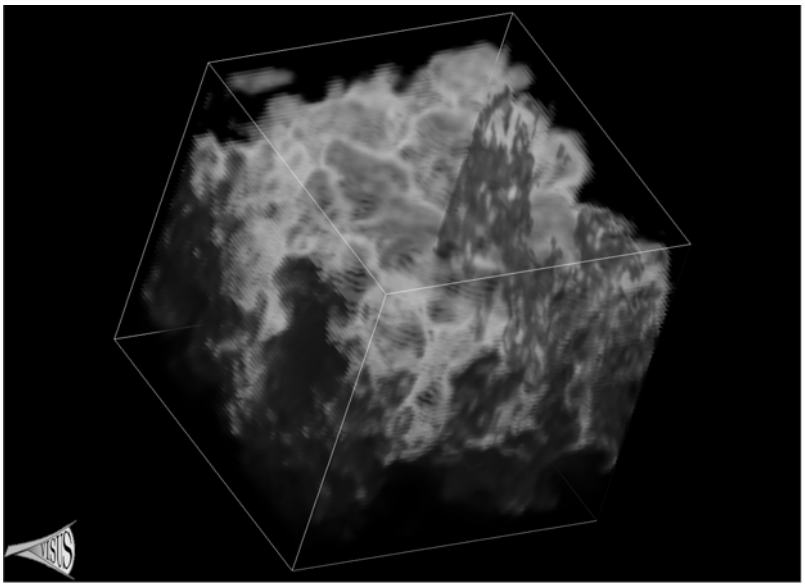
- **High order hydrodynamics code for computing fluid instabilities and turbulent mix**
  - LLNL application
  - solvers for both compressible and incompressible flows
- **three dimensional turbulent growth Rayleigh-Taylor (R-T)& Richtmyer-Meshkov (R-M) instability**
  - occurs in supernovae and Inertial Confinement Fusion



Coprocessor mode

Weak scaling

Miranda: Visualization



BlueGene/L-Tuned Applications

- **Amber7**: Classical molecular dynamics used by AIST and IBM Blue Matter.
- **Blue Matter**: (IBM: Robert Germain et al) Classical molecular dynamics for protein folding and lipids.
- **CPMD**: (Car-Parrinello (ab initio) quantum molecular dynamics by IBM) Strong scaling of SiC 216 atoms & 1000 atoms.
- **ddcMD**: (LLNL: Classical molecular dynamics; Fred Streitz, Jim Glosli, Mehul Patel)
- **Enzo:** (UC San Diego) simulation of galaxies, has performance problem on every platform.
- **Flash**: (University of Chicago & Argonne) Collapse of stellar core and envelope explosion. Supernova simulation.
- **GAMESS**: Quantum Chemistry
- **HOMME**: (NCAR, Richard Loft) Climate code, 2d model of cloud physics.
- **HPCMW** (RIST): Solver for finite elements
- **LJ** (Caltech): Lennard Jones molecular dynamics
- **LSMS**: (Oak Ridge National Lab: Thomas Schulthess and Mark Fahey) First principles Material Science.
- **MDCASK**: (LLNL: Classical molecular dynamics; Alison Kutoba, Tom Spelce)
- **Miranda** (LLNL: instability/turbulence; Andy Cook, Bill Cabot, Peter Williams, Jeff Hagelberg)
- **MM5** from NCAR: meso-scale weather prediction
- **NAMD**: Molecular Dynamics
- **NIWS** (Nissei): Financial/Insurance Portfolio Simulation
- **PAM-CRASH**: (ESI) Automobile crash simulation.
- **ParaDis**: (LLNL: dislocation dynamics;Vasily Bulatov, Gregg Hommes)
- **Polycrystal**: (Caltech) material science
- **Qbox**: Quantum Chemistry, ab initio quantum molecular dynamical calculation.
- **Quarks** (Boston University, Joe Howard)
- **Raptor** (LLNL: instability/turbulence; Jeff Greenough, Charles Rendleman)
- **QCD**: (IBM Pavlos Vranas) sustained 1 TF/s on one rack. 19% uni efficiency.
- **QMC**: (Caltech) Quantum Chemistry
- **SAGE**: (LANL: SAIC's Adaptive Grid Eulerian Code) AMR hydrodynamics. Heat and radiation transport with AMR.
- **SPHOT**: (LLNL) 2D photon transport
- **SPPM**: **S**implified **P**iecewise **P**arabolic Method. 3-D gas dynamics on a uniform Cartesian grid.
- **Sweep3d**: (LANL) 3-d neutron transport
- **TBLE**: magnetohydrodynamics
- **UMT2K**: (LLNL) photon transport 3d Boltzmann on unstructured grid

BlueGene/L: Performance and Density Breakthrough

Yesterday: Focus on technology performance

Today: Focus on system performance

Metric	ASCI White	ASCI Q	Earth Simulator	ASC Purple	BG/L
Memory/Space (GB/sq.m)	8.6	17	3.1	**80**	**140**
Speed/Space (GF/sq.m)	13	16	13	***150***	***1600***
Speed/Power (GF/kW)	12	7.9	4	***19.4***	***300***

Innovation that matters

- **Innovation and integration delivers what scaling can't**
  - Holistic design approach
  - Innovation across all aspects of system design
- **Aggregate performance is important**
  - not performance of individual chip
- **Simple building block**
  - high integration on a single chip
  - low power → allows high density packaging

➔ **As a result, breakthrough in compute power**

  - per Watt
  - per square meter of floor space
  - per dollar

➔ **BlueGene/L enables**

  - new unparalleled application performance
  - breakthroughs in science by providing unprecedented compute power

Additional Information

- **Exploiting eDRAM bandwidth with data prefetching: simulation and measurements**
  - V. Salapura, J. R. Brunheroto, and F. Redìgolo
  - HPCA-13 – IEEE International Symposium on High-Performance Computer Architecture, February 2007.
- **Exploiting Workload Parallelism for Performance and Power Optimization in Blue Gene**
  - Valentina Salapura, Robert Walkup and Alan Gara
  - IEEE Micro, September/October 2006
- **The BlueGene/L Supercomputer and Quantum ChromoDynamics**
  - ***2006 Gordon Bell Prize winner***
  - P. Vranas, G. Bhanot, M. Blumrich, D. Chen, A. Gara, P. Heidelberger, V. Salapura, J. Sexton and R. Soltz
  - Supercomputing 2006, November 2006
- **Designing a Highly-Scalable Operating System: The Blue Gene/L Story**
  - Jose Moreira, Michael Brutman, Jose Castanos, Tom Gooding, Todd Inglett, Derek Lieber, Pat McCarthy, Mike Mundy, Jeff Parker, Brian Wallenfelt, Mark Giampapa, Thomas Engelsiepen, Roger Haskin
  - Supercomputing 2006, November 2006
- **Power and Performance Optimization at the System Level**
  - Valentina Salapura and the IBM BlueGene team
  - ACM Computing Frontiers 2005, Ischia, Italy, May 2005.
- **Creating the BlueGene/L supercomputer from low power SoC ASICs**
  - A. Bright, M. Ellavsky, A. Gara, R. Haring, G. Kopcsay, R. Lembach, J. Marcella, M. Ohmacht, and V. Salapura
  - ISSCC 2005 – IEEE International Solid-State Circuits Conference, February 2005.
- **BlueGene: A Vision for Protein Science using a Petaflop Supercomputer**
  - The IBM BlueGene team
  - IBM System Journal, Vol. 40, No. 2, 2001.

Additional Information

- **Early Experience with Scientific Applications on the Blue Gene/L Supercomputer**
  - George Almasi, Gyan Bhanot, Dong Chen, Maria Eleftheriou, Blake Fitch, Alan Gara, Manish Gupta, Jose Moreira
  - Proceedings of Euro-Par 2005. August 30-September 2. Lisbon, Portugal.
- **Scaling Physics and Material Science Aplications on a Massively Parallel Blue Gene/L System**
  - G. Almasi, G. Bhanot, A. Gara, M. Gupta, J. Sexton, B. Walkup, V. Bulatov, A. W. Cook, B. R. de Supinski, J. A. Greenough, F. Gygi, A. Kubota, S. Louis, F. H. Streitz, R. Yates, C. Archer, J. Moreira
  - Proceedings of the 19th ACM International Conference on Supercomputing. June 20-22, 2005. Cambridge, MA
- **Optimization of MPI colective communication on BlueGene/L systems**
  - G. Almási, P. Heidelberger, C. J. Archer, X. Martorel, C. Erway, J. E. Moreira, B. Steinmacher-Burow, Y. Zheng,
  - Proceedings of the 19th ACM International Conference on Supercomputing. June 20-22, 2005. Cambridge, MA
- **Implementing MPI on the BlueGene/L Supercomputer (Distinguished paper).**
  - G. Almasi, C. Archer, J. G. Castanos, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, N. Smeds, B. Steinmacher-burrow, W. Gropp, B. Toonen
  - Proceedings of Euro-Par 2004, August, 2004. Pisa, Italy
- **Open Job Management Architecture for the BlueGene/L Supercomputer.**
  - Yariv Aridor, Tamar Domany, Oleg Goldshmidt, Yevgeny Kliteynik, Jose Moreira, and Edi Shmueli.
  - Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing. June 19, 2005. Cambridge, MA

Blue Gene/L on the Web

www.research.ibm.com/bluegene

System Features		BG/L
Node Properties	Node Processors	2* PowerPC440
	Processor Frequency	700MHz
	L1 Cache (private) I+D	32+32KB/processor
	L2 Cache (private)	14 (7) stream prefetching
	L3 Cache size (shared)	4MB
	Main Store	256MB/512MB/1GB
	Main Store Bandwidth	5.6GB/s
	Peak Performance	5.6GF/node
Torus Network	Bandwidth (per node)	6*2*175MB/s=2.1GB/s
	Hardware Latency (Nearest Neighbor)	200ns (32B packet) 1.6µs (256B packet)
	Hardware Latency (Worst Case)	6.4µs (64 hops)
Collective Network	Bandwidth (per node)	3*2*350MB/s=2.1GB/s
	Hardware Latency (round trip worst case)	5.0µs

Blue Gene/L at a Glance

Attribute	Details	Benefits
Processor	PowerPC 440 700MHz; two per node	Low power allows dense packaging; better processor-memory balance
Memory	512 MB SDRAM-DDR per node	
Networks	3D Torus - 175MB/sec in each direction Collective Network – 350MB/sec; 1.5 usec latency Global Barrier/Interrupt Gigabit Ethernet (machine control and outside connectivity)	Special networks speed up internode communications; designed for MPI programming constructs; improve systems management
Compute Nodes	Dual processor; 1024 per rack	Double FPU improves performance
I/O Nodes	Dual processor; 16 per rack (additional nodes optional)	Strengthens systems management
Operating Systems	Compute Node – Lightweight proprietary kernel I/O Node – Embedded Linux Front End and Service Nodes – SuSE SLES 9 Linux	Kernel tailored to processor design; industry-standard distribution preserves familiarity to end user
Performance	Peak per rack (virtual node mode) – 5.73 teraflops Peak per rack (coprocessor mode) – 2.86 teraflops Linpack per rack – 4.53 teraflops	Highest available performance benefits capability customers
Power	28.14 kW power consumption per rack (maximum) 208 VAC 3-phase; 100 amp service per rack	Low power draw enables dense packaging
Cooling	Air conditioning 8 tons/rack (minimum) 2800 CFM (compute rack); 350 CFM (power supplies)	Low cooling requirements enable extreme scale-up
Acoustics	9.0 LwAD and 8.7 LwAm	
Dimensions (includes air duct)	Height – 1958 mm Width – 915 mm Depth – 915 mm Weight – 750 Kg	Design allows "brickwall" layout for better floor space utilization