New and Innovative Features of IBM System BlueGene/Q

19 June 2012

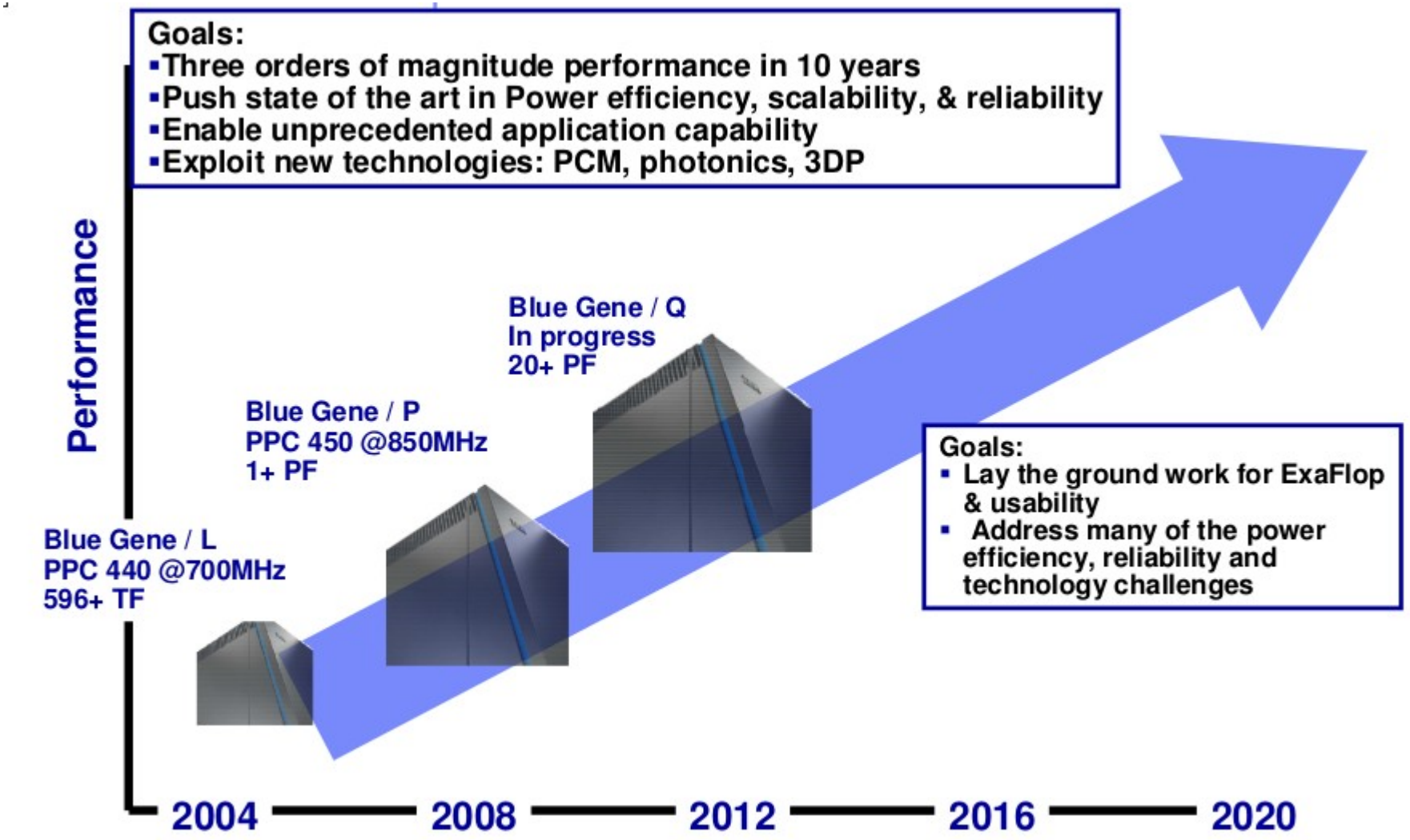
Indra Mani
indramani@in.ibm.com

Overview of presentation

- Part I
  - General Overview of IBM BlueGene Systems.
  - Current strengths of BlueGene/Q system.
  - New features added to BGQ
- Part II
  - More on selective innovations in BGQ node.

General Overview of IBM BlueGene Systems

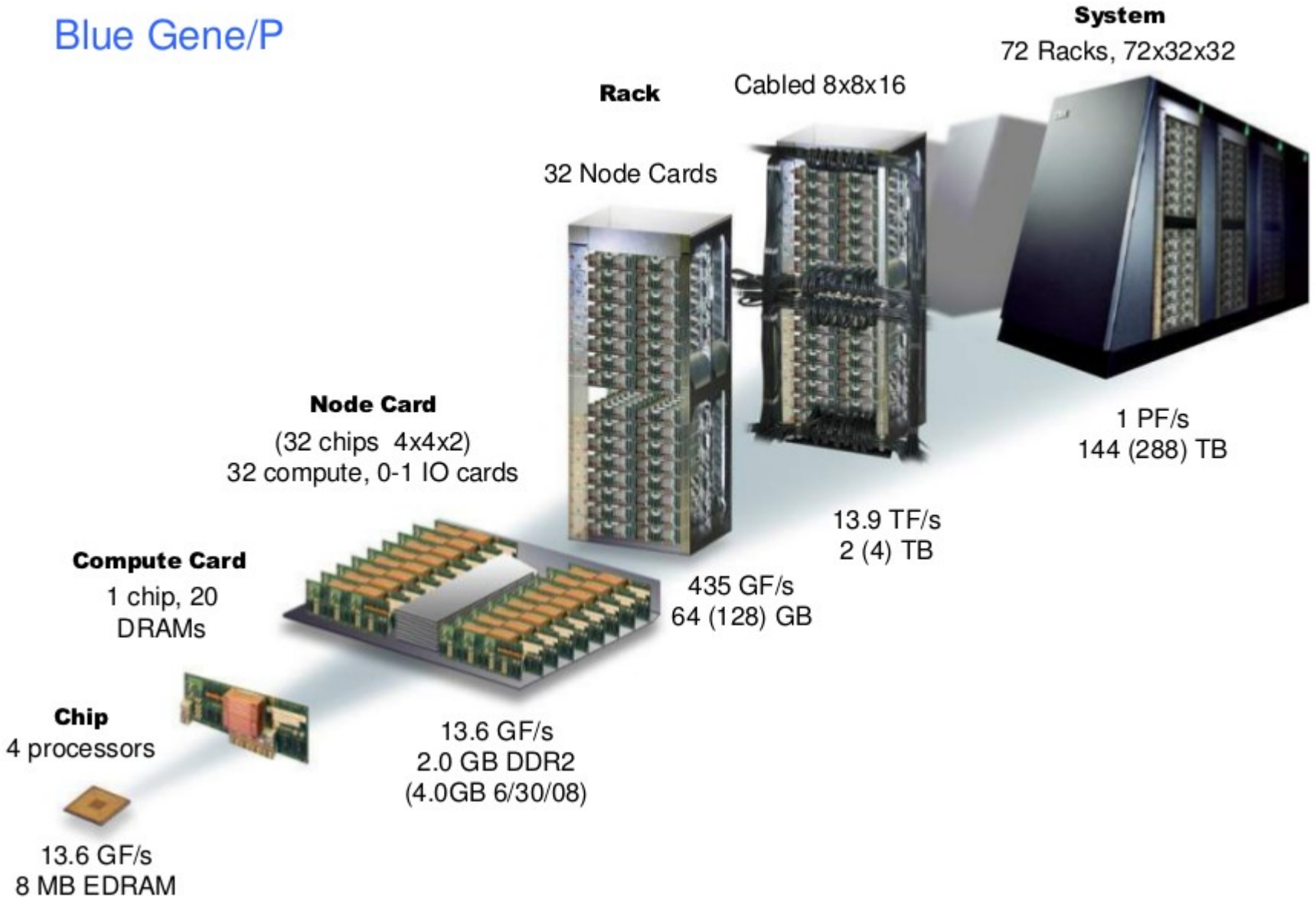
Evolutionary Road Map of Blue Gene Systems

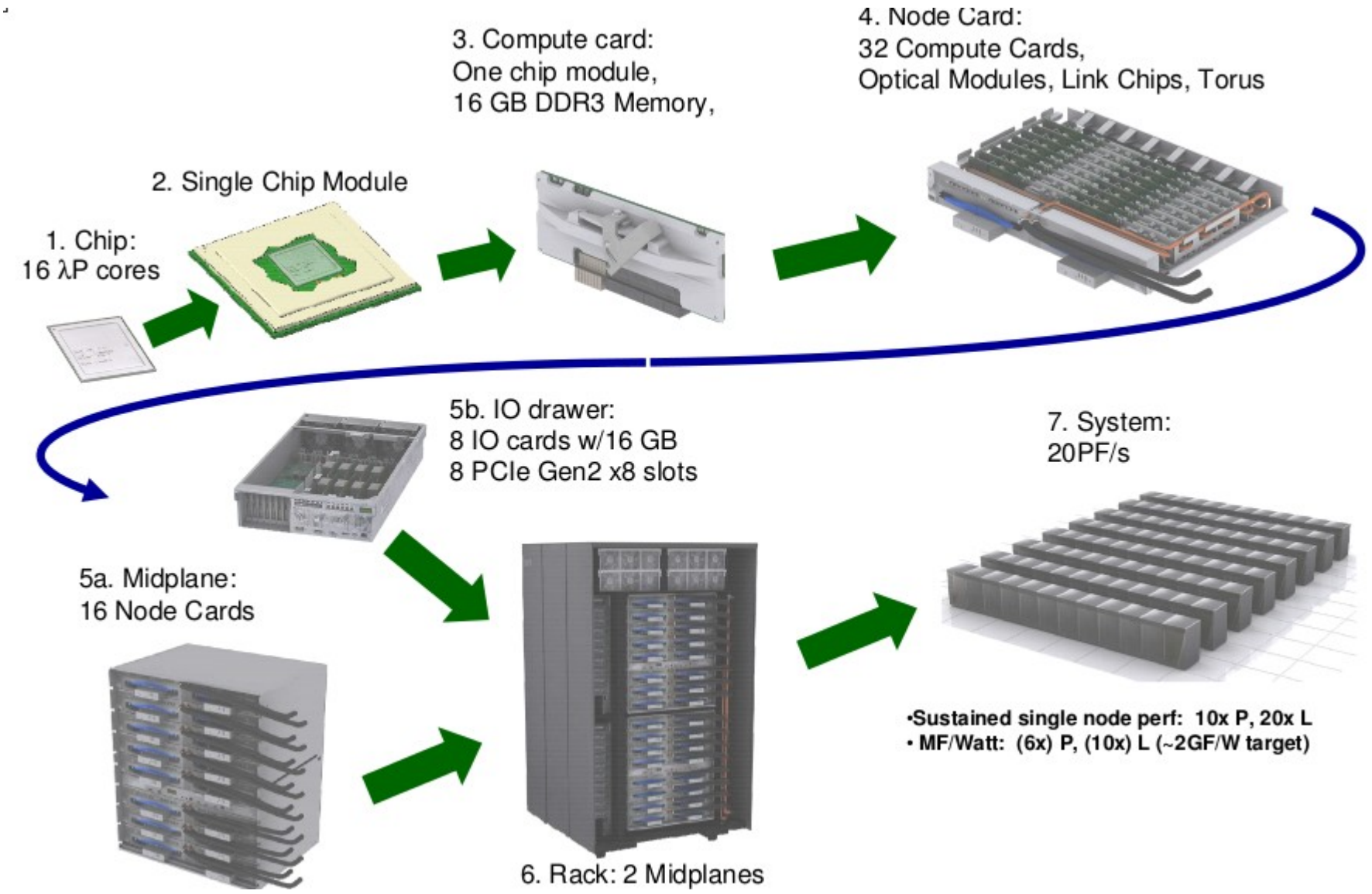


Blue Gene Evolution

- BG/L (5.7 TF/rack) – 130nm ASIC (1999-2004 GA)
  - Embedded 440 core, dual-core system-on-chip
  - Memory: 0.5/1 GB/node
  - Biggest installed system (LLNL): 104 racks, 212,992 cores, 596 TF/s, 210 MF/W
- BG/P (13.9 TF/rack) – 90nm ASIC (2004-2007 GA)
  - Embedded 450 core
  - Memory: 2/4 GB/node, quad core SOC, DMA
  - Biggest installed system (Jülich): 72 racks, 294,912 cores, 1 PF/s, 357 MF/W
  - SMP support, OpenMP, MPI
- BG/Q (209 TF/rack) – 45nm ASIC+ (2007-2012 GA)
  - A2 core, 16 core/64 thread SOC
  - 16 GB/node
  - Biggest installed system (LLNL): 96 racks, 1,572,864 cores, 20 PF/s, 2 GF/W,
  - Speculative execution, sophisticated L1 prefetch, transactional memory, fast thread handoff, compute + IO systems.

Blue Gene/P





Current Strengths of BlueGene/Q

Key strengths of Blue Gene/Q

- Performance/Power
- Scalability
- Fault Tolerance
- Ease of programmability

Performance/Power

- Top 5 entries in Green500 list (Nov 2011) are BlueGene/Q systems
- BlueGene/Q systems are 1.6x better than best non BlueGene/Q system in the list
- BlueGene/Q systems are generally more than 2x better than all but 2 non BlueGene/Q system in the list
- BlueGene/Q systems are around 9.6x better than BlueGene/L systems which had best performance/power numbers 6 years back
- BlueGene/Q systems are around 5.6x better than BlueGene/P systems which had best performance/power numbers 4 years back

Scalability and Fault Tolerance

- BlueGene systems are designed to scale well
- Largest BlueGene/L systems had 212992 processors.
- Largest BlueGene/P system had 294,912 processors
- BlueGene/Q will have 1,572,864 processors in largest installation with 20 PF capability

Ease of Programming

- System software built around open standards.
- Supports shared memory and hybrid programming models.
- Supports MPI, OpenMP, UPC, ARMCI, global arrays, Charm++.
- Supports dynamic linking.
- **Support for Transactional Memory.**
- **Supports Speculative Execution.**
- **Compiler support for auto-simdization.**

New features added to BGQ

BGQ changes from BGL/BGP (New Node)

- New voltage scaled processing core (A2) with 4-way SMT
- New SIMD floating point unit (8 flop/clock) with alignment support
- New “intelligent” prefetcher also called perfect prefetcher
- 17th Processor core for system functions.
- Speculative multi-threading and transactional memory support with multiversioning L2 Cache
- Hardware mechanisms to help with multi-threading (wakeup unit)
- Dual SDRAM-DDR3 memory controllers with up to 16 GB/node

BGQ changes from BGL/BGP (New Network)

- 5 D torus in compute nodes
  - 2 GB/s bidirectional bandwidth on all (10+1) links
  - Bisection bandwidth of 65TB/s (26PF/s) / 49 TB/s (20 PF/s) BGL at LLNL is 0.7 TB/s
- Collective and barrier networks embedded in 5-D torus network.
- Floating point addition support in collective network
- Performance
  - All-to-all: 97% of peak
  - Bisection: > 93% of peak
  - Nearest-neighbor: 98% of peak
  - Collective: FP reductions at 94.6% of peak
  - No performance problems identified in network logic

BGQ changes from BGL/BGP (New IO system)

- I/O nodes in separate drawers/rack with private 3D (or 4D) torus
- PCI-Express Gen 2 on every IO node with full sized PCI slot

Part II

More on selective innovations introduced in BGQ node

Innovations that matter

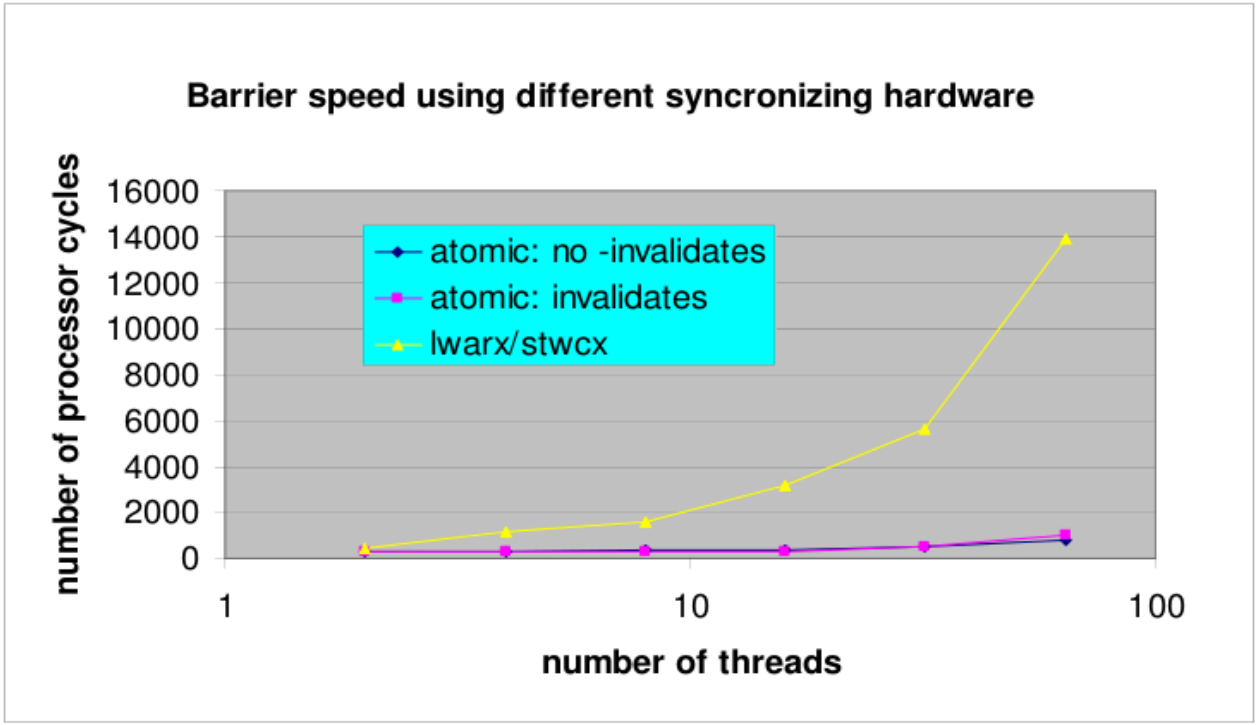
- To reduce the overhead to hand off work to high numbers of threads used in OpenMP and messaging through hardware support for atomic operations and fast wake up of cores.
  - L2 Atomics
  - Wakeup Unit
- Multiversioning cache to help in a number of dimensions such as performance, ease of use and RAS.
  - Transactional Memory
  - Speculative Execution
- Aggressive FPU to allow for higher single thread performance for some applications. Most will get modest bump (10-25%), some big bump (approaching 300%)
  - QPX and auto-simdization
- “perfect” prefetching for repeated memory reference patterns in arbitrarily long code segments. Also helps achieve higher single thread for some applications.
  - L1 perfect prefetcher

Atomic Operations : L2 Atomics

- L2 implements atomic operations on every 64-bit word in memory
  - Allows specific memory operations to be performed atomically:
    - Load: Increment, Decrement, Clear (plus variations)
    - Store: Twin, Add, OR, XOR, Max (plus variations)
  - Atomic operation is triggered by L2 when **specific shadow physical addresses** are read/written
- **CNK exposes these L2 atomic physical addresses**
  - Kernel allocates an atomic page using a different base address beyond the 16G of Installed memory
  - Applications must pre-register the location of the L2 atomic before access
    - Otherwise segfault
  - Fast barrier implementation using L2 atomics available
- CNK also uses L2 atomics internally
  - Fast performance counters (store w/ add 1)
  - Locking
- OpenMP exploits L2 atomics for lock/barrier

Atomic Operations : L2 Atomics (Continued)

- Pipelined at L2
- Low latency even under high contention
- Faster OpenMP work hand off



Wakeup Unit

- Used in conjunction with the PowerPC wait instruction
- When a hardware thread is in a wait state, the hardware thread stops executing and the other hardware threads will benefit from the additional available cycles.
- Sends a wakeup signal to a hardware thread
- Configurable wakeup conditions:
  - WakeUp Address Compare
  - Messaging Unit activity
  - Interrupt sources
- Kernel provides application interfaces to utilize the wakeup unit

Transactional Memory

- User labels atomic sections
- atomic
  {
  ...
  }
- Underlying system ensures atomicity
  – Executes In parallel when possible (speculation)
- Goals:
  – Simplicity: as easy as coarse-grain locks
  – Performance: as fast as fine-grain locks
- BG/Q: first commercially available HTM system from IBM!

Example using Locks and TM

```
// WITH LOCKS
void move(T s, T d, Obj key){
LOCK(s);
LOCK(d);
tmp = s.remove(key);
d.insert(key, tmp);
UNLOCK(d);
UNLOCK(s);
}

Thread 0 move(a, b, key1);

Thread 1 move(b, a, key2);

DEADLOCK!
```

- Coarse-grain locking limits concurrency
- Fine-grain locking difficult

```
// With TM
void move(T s, T d, Obj key){
#pragma tm_atomic
{
tmp = s.remove(key);
d.insert(key, tmp);
}
}
```

- Avoids Deadlock
- Fine Grain Locking

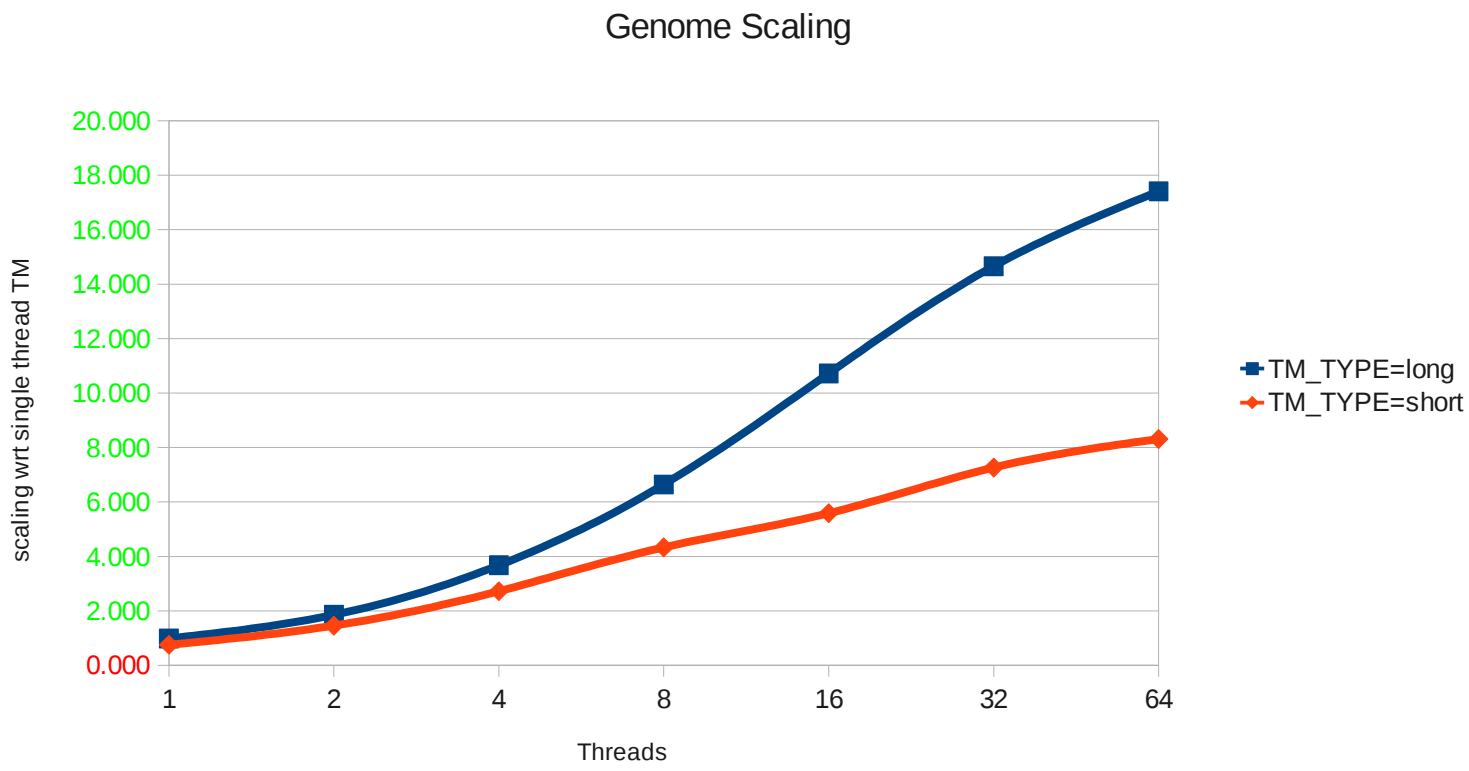
BGQ TM Programming Model

- Single entry, single exit code block
  - Boundary of a transaction statically determined
- Can be used with Pthreads, or openMP
- Flat nesting semantic
  - Support in software
- What can go inside a TM region?
  - Any computation is allowed inside the TM region
  - Entire ISA is allowed
- Compiler invocation
  - -qtm for the compilers to recognize the pragma or directive
  - Runtime report details specific success/failure metrics for each TM region such as number of transactions and rollbacks and reason for serialization.

Management of L1 Cache

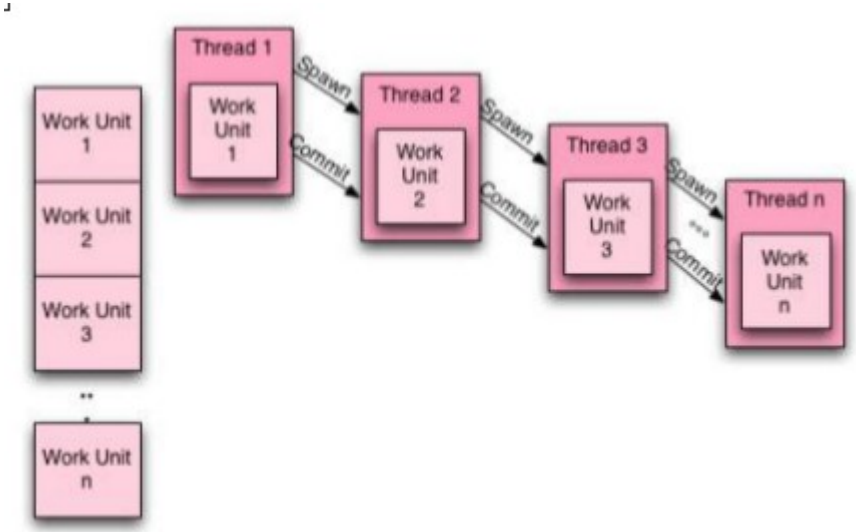
- How does the L1 cache keeps a speculative thread's writes invisible to the other three SMT threads?
- Support for two modes for transactional memory:
  - Short running (SR) mode (via L1 bypass)
    - Core evicts speculative written cache line from L1
    - Subsequent loads served from L2
    - Loaded data placed directly into the register of the thread
  - Long running (LR) mode (via TLB aliasing)
    - Software creates illusion of versioned address space
    - Bits in the physical address used by MMU to create the 'aliasing effect' at the L1 level
- Transaction memory runtime is implemented via both long and short running mode
  - Default: long running mode
  - Toggle to short running mode via environment variable: TM_SHORT_RUNNING_TRANSACTION=YES

STAMP/genome TM scaling performance



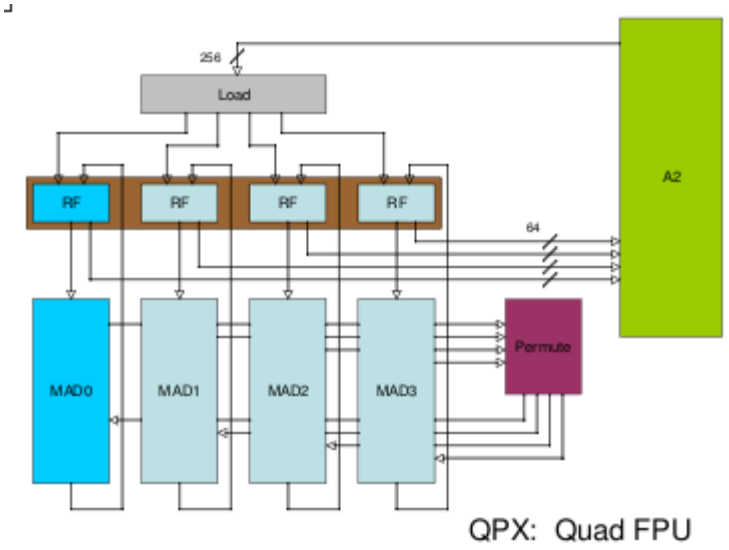
Speculative Execution

- Similar to Transactional Memory
  - Except Ordered thread commit and different usage model
- Leverages existing OpenMP parallelization
  - However compiler does not need to guarantee that there is no array overlap
  - Should allow the compiler to do a much better job of auto-parallelizing
- Total work is subdivided into workunits without locking
- If work units collide in memory:
  - SE hardware detects
  - Kernel rolls back transaction
  - Runtime decides whether to retry or serialize
- #pragma speculative
  - Inspired by OpenMP
  - parallel for and section
  - -qsmp=speculative
- XL Compiler only



Quad FPU (QPX)

- 4-wide double precision FPU SIMD
- 2-wide complex SIMD
- Supports a multitude of alignments
- Allows for higher single thread performance for some applications

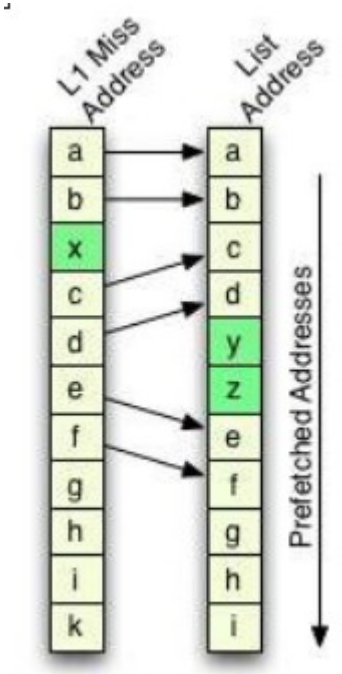


Auto-SIMDization Support

- Improving SIMD from BG/P/L to BG/Q
  - Moving from low level optimizer to high-level optimizer due to better loop optimization structure, and alignment analysis
- To enable SIMDization
  - Start to compile (assume appropriate –qarch=qp and –qtune=qp option):
    - -O3 (compile time, limited hot and SIMD optimizations)
  - Increase optimization Level
    - -O4 (compile time, limited scope analysis, SIMDization)
    - -O5 (link time, pointer analysis, whole-program analysis, and SIMD instruction)
  - -qhot=level=0 (enables SIMD by default)
- To disable SIMDization
  - Turn off SIMDization for the program
    - add -qhot=nosimd to the previous command line options
  - Turn off simdization for a particular loop
    - #pragma nosimd | !IBM* NOSIMD
- Tune your programs
  - Help the compiler with extra information (directives/pragmas)
  - Check the SIMD instruction generation in the object code listing (-qsource -qlist).
  - Use compiler feedback (-qreport -qhot) to guide you.

L1 Perfect Prefetcher

- Dedicated L1p (4K) resides between dedicated L1 (32K) and shared L2
- The L1p has 2 different prefetch algorithms:
  - Stream prefetcher
    - Similar to the prefetcher algorithms used on BGL/BGP
  - "Perfect" prefetcher
- Perfect Prefetcher:
  - First iteration through code during training run:
    - BQC records sequence of memory load accesses
    - Sequence is stored in DDR memory
  - Subsequent iteration through code:
    - BQC loads the sequence and tracks where the code is in the sequence
    - Prefetcher attempts to prefetch memory before it is needed in the sequence
- Kernel provides access routines to setup and configure the stream and perfect prefetchers



Thank you

Further Resources

- Scicomp 2012 Tutorial by Amy Wang

  http://spscicomp.org/wordpress/wp-content/uploads/2012/04/ScicomP-2012-Tutorial-BGQ-Amy-Wang.pdf
- PRACE winter school 2012 by Pascal Vezolle

  http://www.training.prace-ri.eu/uploads/tx_pracetmo/BG-Q-_Vezolle.pdf