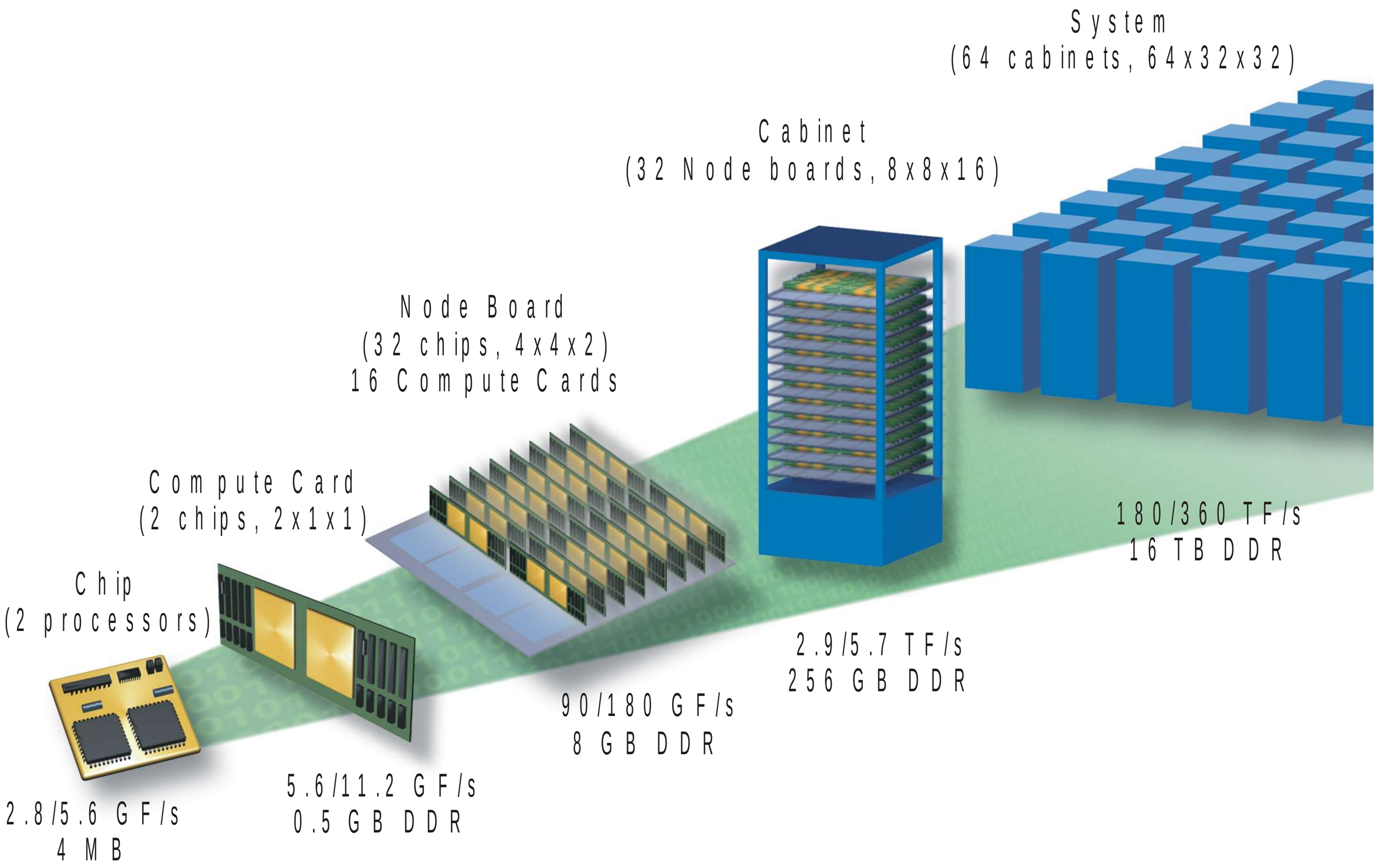
Tuning Numerics for the Memory Hierarchy: High-Performance Linear Algebra Routines for BlueGene/L

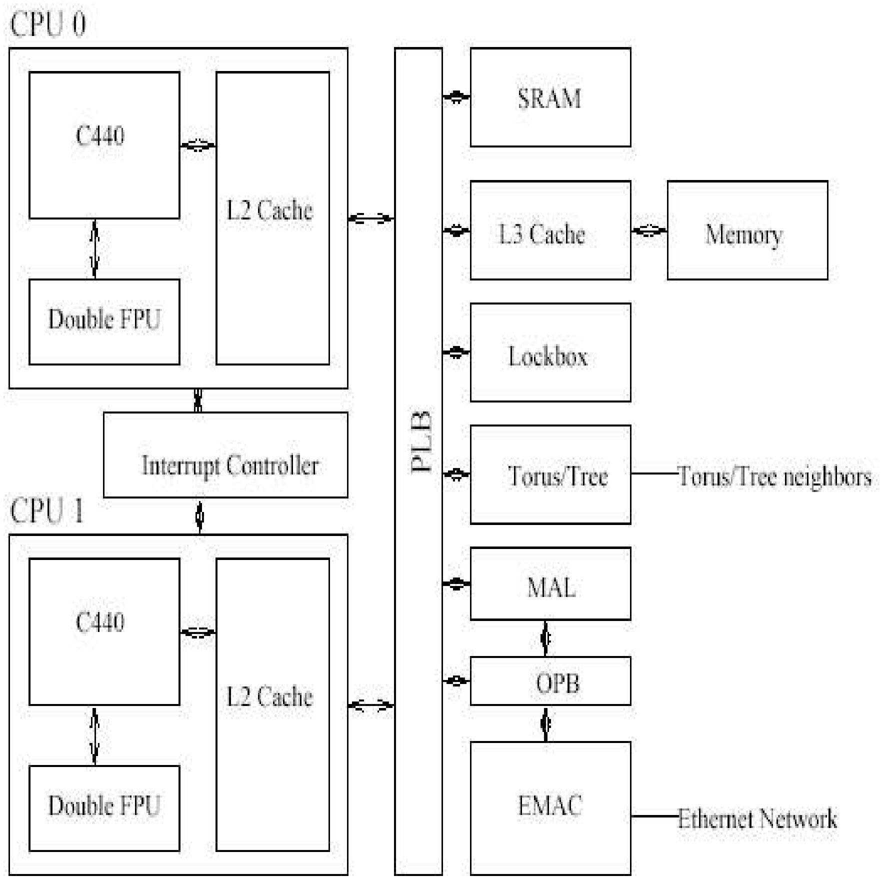
Siddhartha Chatterjee

IBM T. J. Watson Research Center
Yorktown Heights, NY

BlueGene/L


System
(64 cabinets, 64x32x32)
Cabinet
(32 Node boards, 8x8x16)
Node Board
(32 chips, 4x4x2)
16 Compute Cards
Compute Card
(2 chips, 2x1x1)
Chip
(2 processors)
180/360 TF/s
16 TB DDR
2.9/5.7 TF/s
256 GB DDR
90/180 GF/s
8 GB DDR
5.6/11.2 GF/s
0.5 GB DDR
2.8/5.6 GF/s
4 MB

Compute Node Structure of BlueGene/L



Dual core

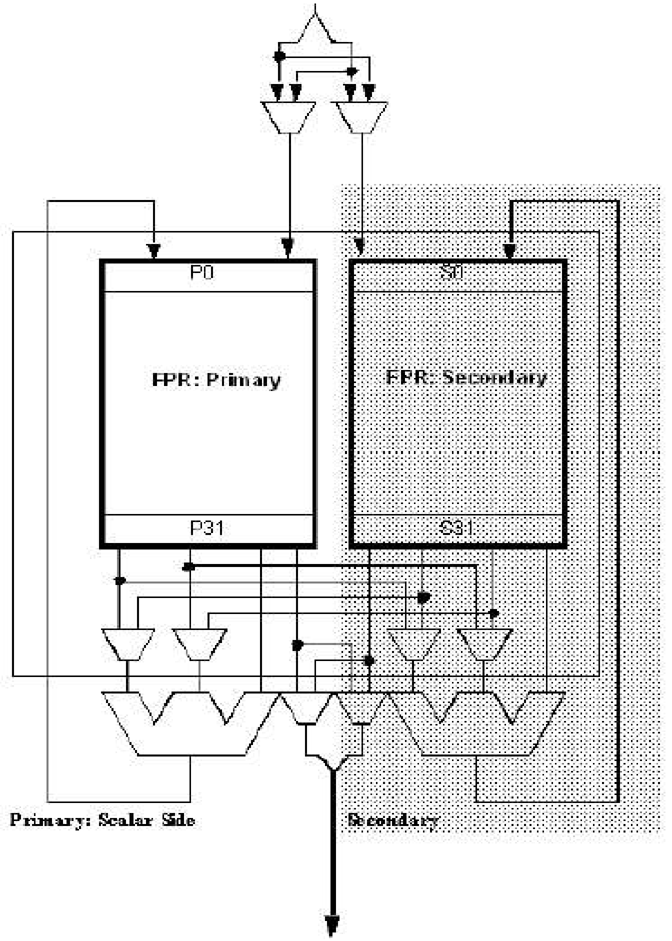
Dual FPU

Three-level cache memory hierarchy

Non-coherent L1 cache

L2 and L3 caches coherent

Dual FPU Architecture



Dual floating-point unit

SIMD instructions over both register files

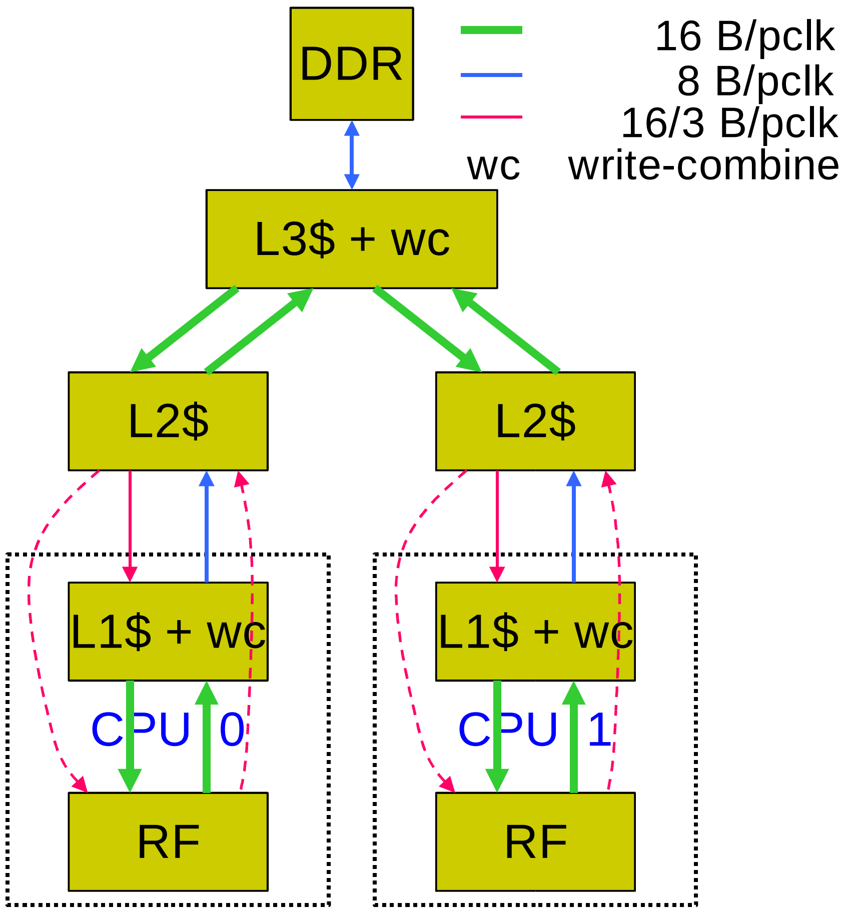
FMA available

Quadword loads/stores

Motivation

- Issues in designing high performance and robust math library routines that are memory-bound
  - Where does data come from (L1 vs L3)?
  - Is the data aligned?
- Robustness
  - Would like the performance to approach the machine limits and not vary significantly based on level of cache residency or data alignment
  - Would prefer to avoid writing assembly code
- We are writing libraries, not complete applications
  - Separate compilation
  - No global knowledge

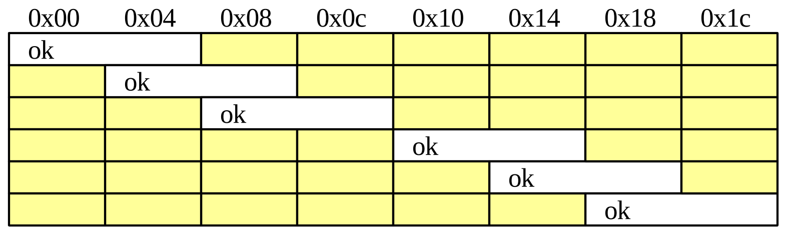
Data Source Issue



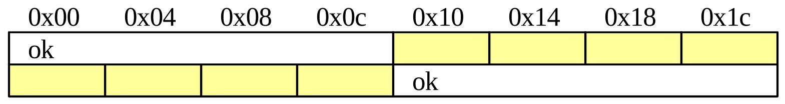
- Prefetching
  - L3-to-L2 and DDR-to-L3 "push"-based prefetching work well to mask latency for stream accesses
  - L2-to-L1 prefetching is "pull"-based
- Typical bottlenecks are
  - The DDR-to-L3 path
  - The L2-to-L1 path
  - The number of outstanding misses that L1 can support
- Bottleneck analysis useful to predict code performance

Data Alignment Issues

Possible accesses to memory data:

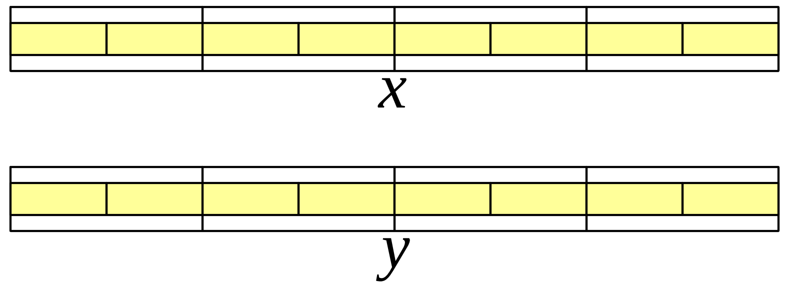


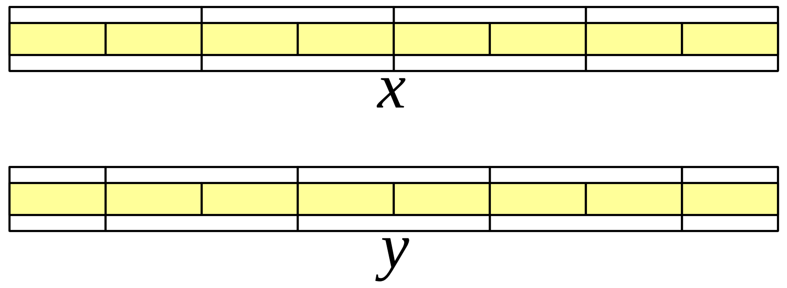
(doublewords)



(quadwords)

Relative alignment between data structures:

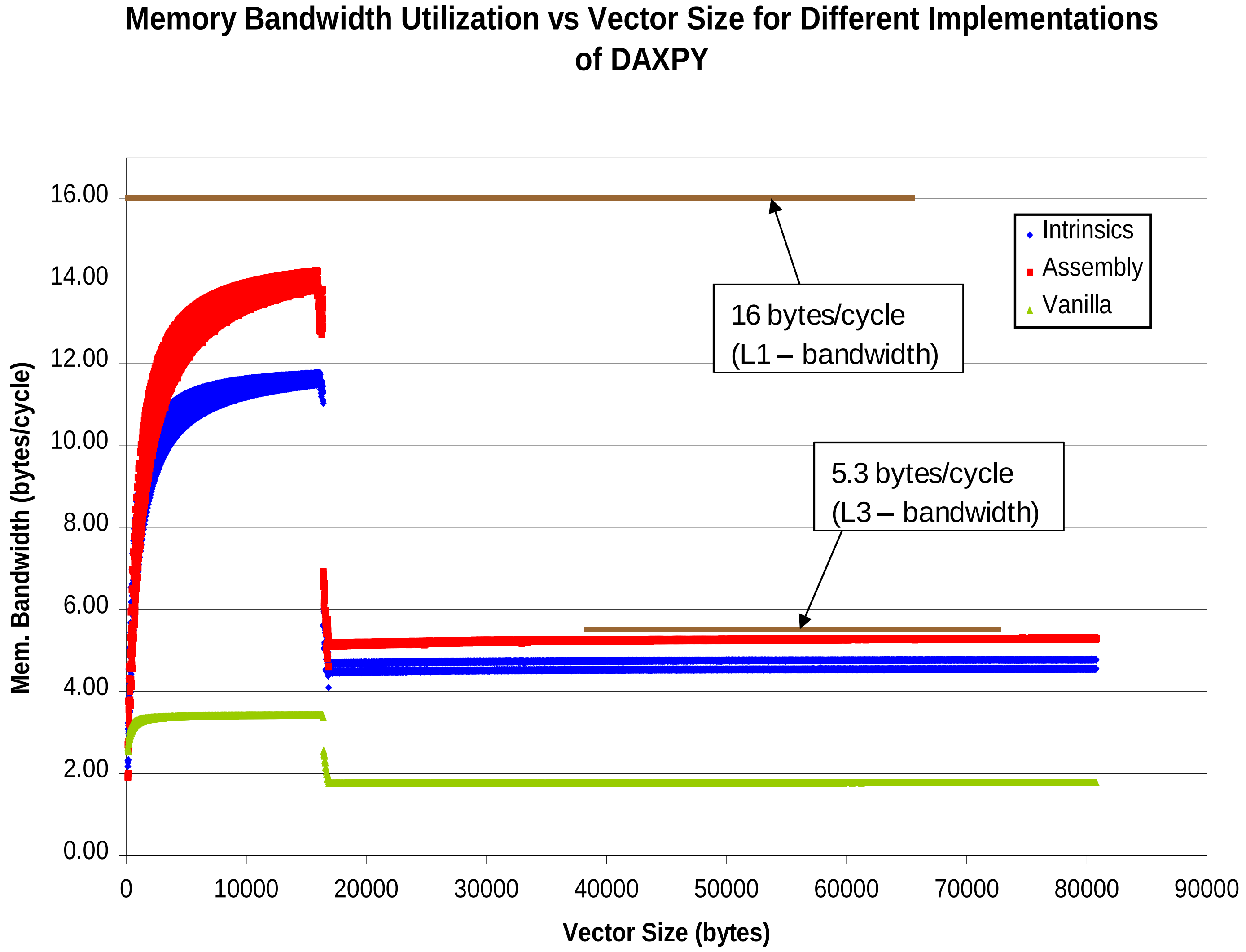




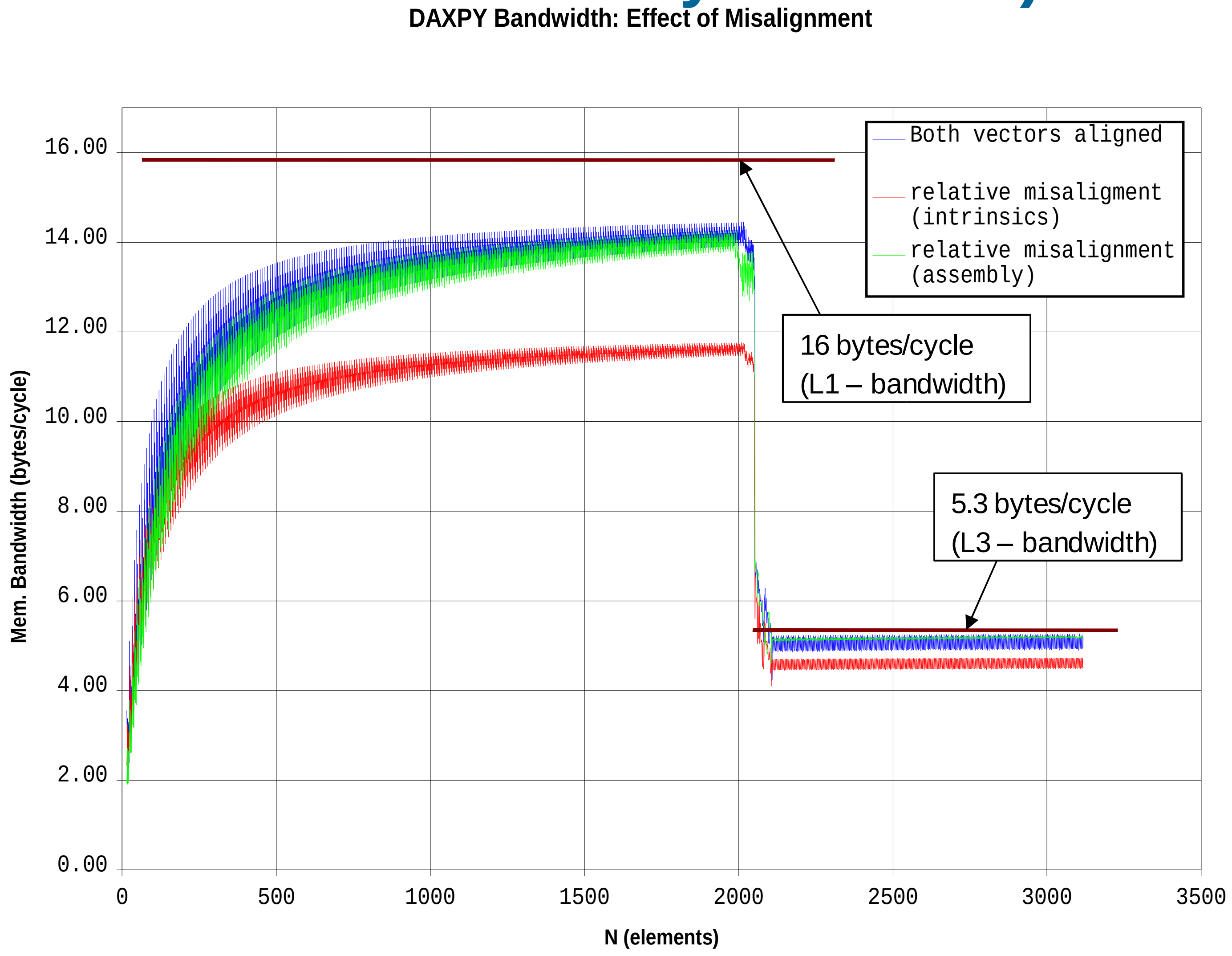
DAXPY

- Vector scaling in the form:
  - $y = a*x + y$
- BLAS Level 1 operation
- Memory bounded kernel: three accesses to memory for every computation:
  - Load of x[i] into p
  - Load of y[i] into q
  - Computation of $r = a*p + q$
  - Store of r into y[i]
- There is no reuse of the elements loaded

DAXPY Bandwidth Utilization



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



DGEMV

Two basic operations:

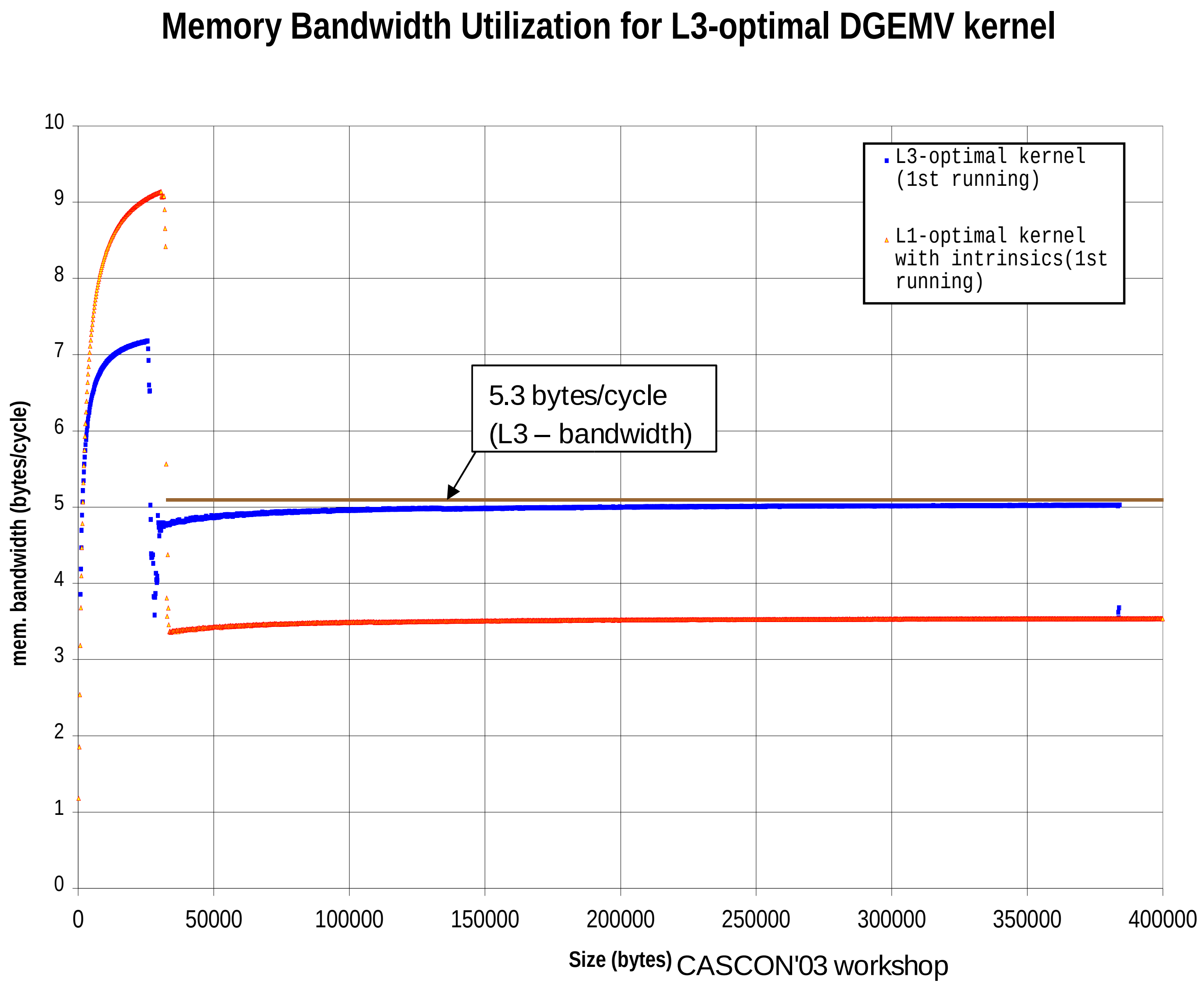
- $y += Ax$
- $y += A^T x$

Different optimizations for the situations:

- Data resides in L3 cache
- Data resides in L1 cache

There is some reuse of vector elements loaded

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

What More Could We Want?

- Open up the cache architecture more
  - It would be good if the library writer could specify that a particular access would be a miss in L1, or a hit in L3, or whatever
- Expose more microarchitectural constraints to the compiler
  - Example: maximum number of L1 cache misses before stall
- Better register scheduling algorithms
  - Currently, we have observed excessive spills when using close to all 32 registers