

BG/L:

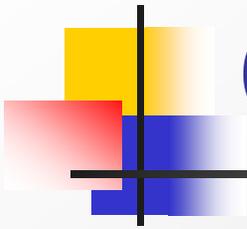
# Tuning for One Node

---

John A. Gunnels

Mathematical Sciences Dept.

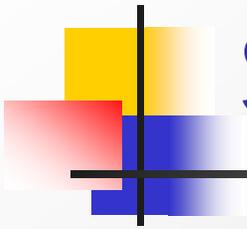
IBM T. J. Watson Research Center



# Outline

---

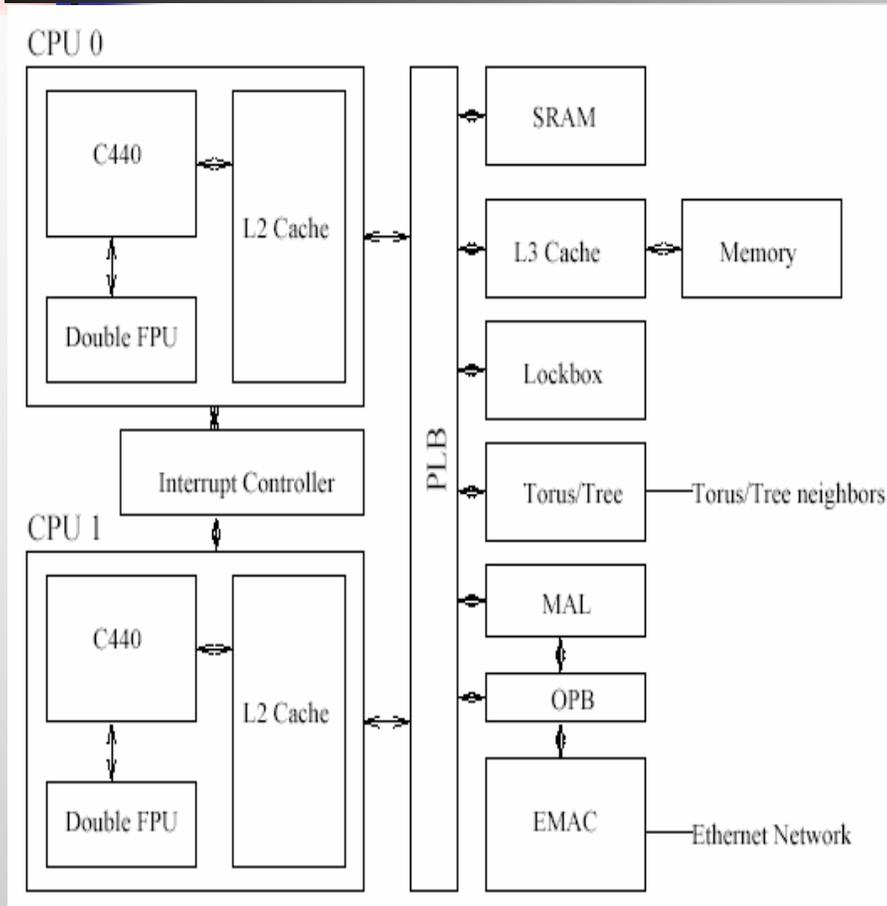
- Single Core
  - Architecture (brief)
    - Memory/FPU (BLAS-1 vs. BLAS-3)
  - Programming Options
  - Data Structures
- Dual Core
  - Why?
  - How?
  - When?



# Single Core

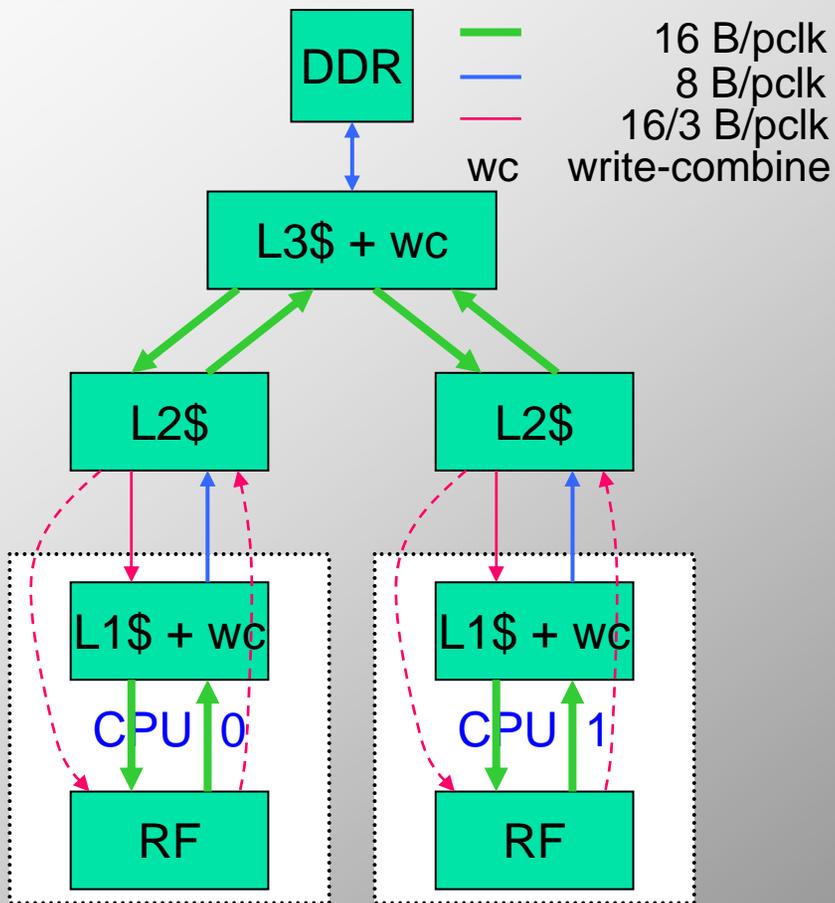
---

# Compute Node Structure of BlueGene/L



- Dual core
- Dual FPU
- Three-level cache memory hierarchy
- Non-coherent L1 cache
  - 32 KB, 64-way, RR
  - L2 & L3 \$'s coherent

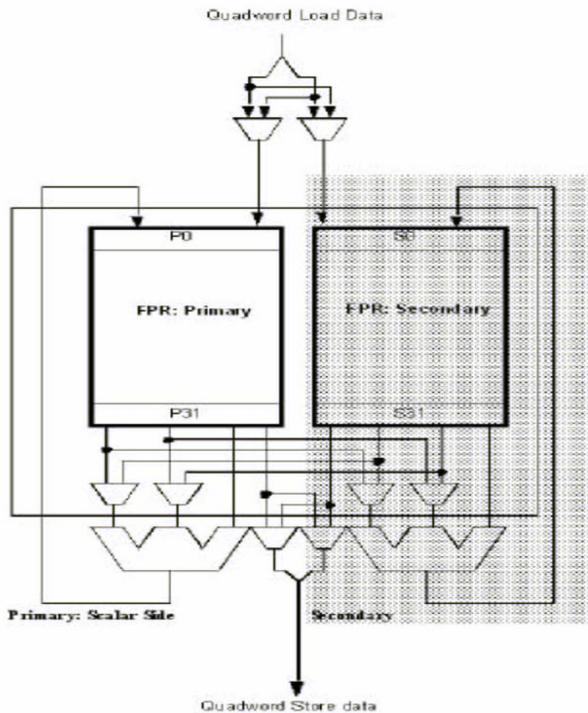
# Data Source Issue



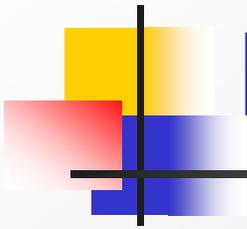
- Prefetching
  - L3-to-L2 and DDR-to-L3 “push”-based prefetching mask latency for stream accesses
  - L2-to-L1 prefetching is “pull”-based
- Typical bottlenecks are
  - The DDR-to-L3 path
  - The L2-to-L1 path
  - Outstanding L1 misses
- Bottleneck analysis

# The Dual FMA Unit

## Floating Point Unit



- Many special instructions
- Alignment is vital
- Instruction Level Parallelism
  - SIMD (Length 2 Vectors)
- Important for the compiler (or library developer) to exploit this (4 FLOPS/cycle)



# Memory: DAXPY

---

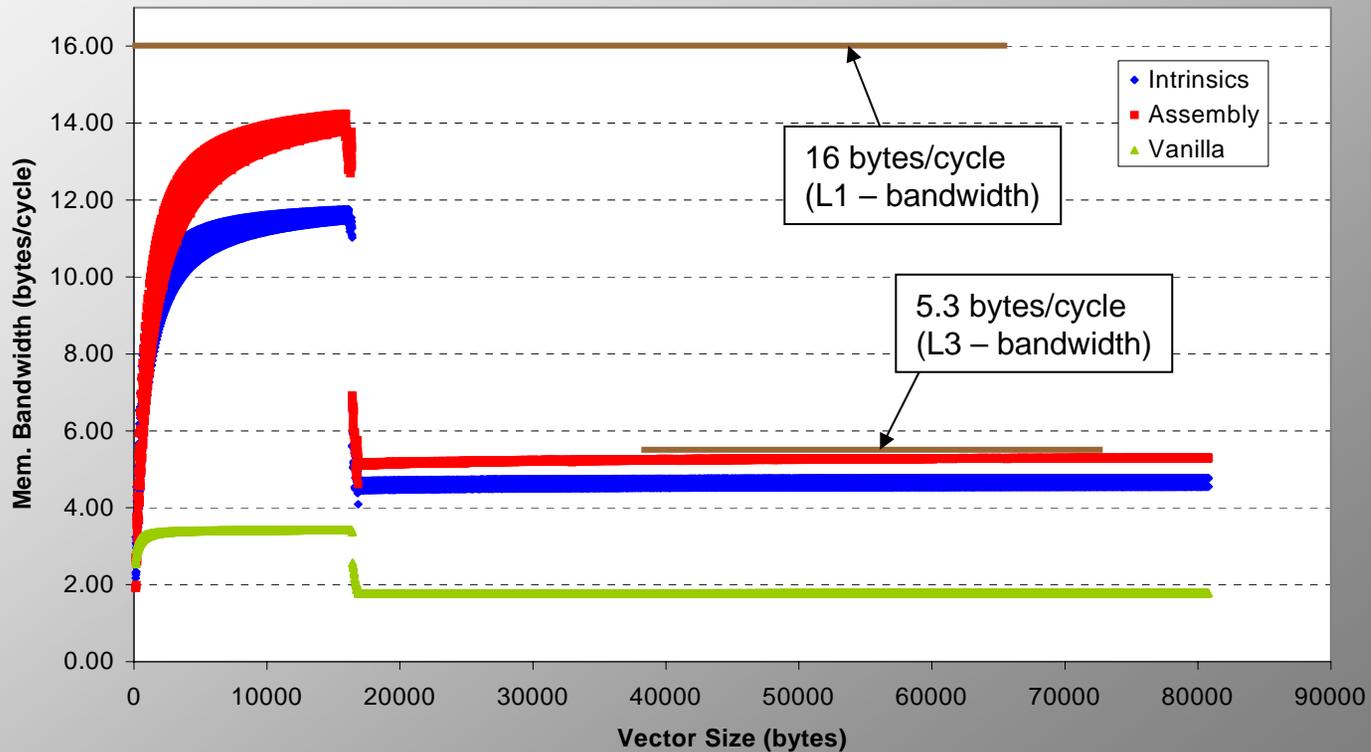
- Vector scaling in the form:

$$y = a * x + y$$

- BLAS Level 1 operation
- Memory-bound 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

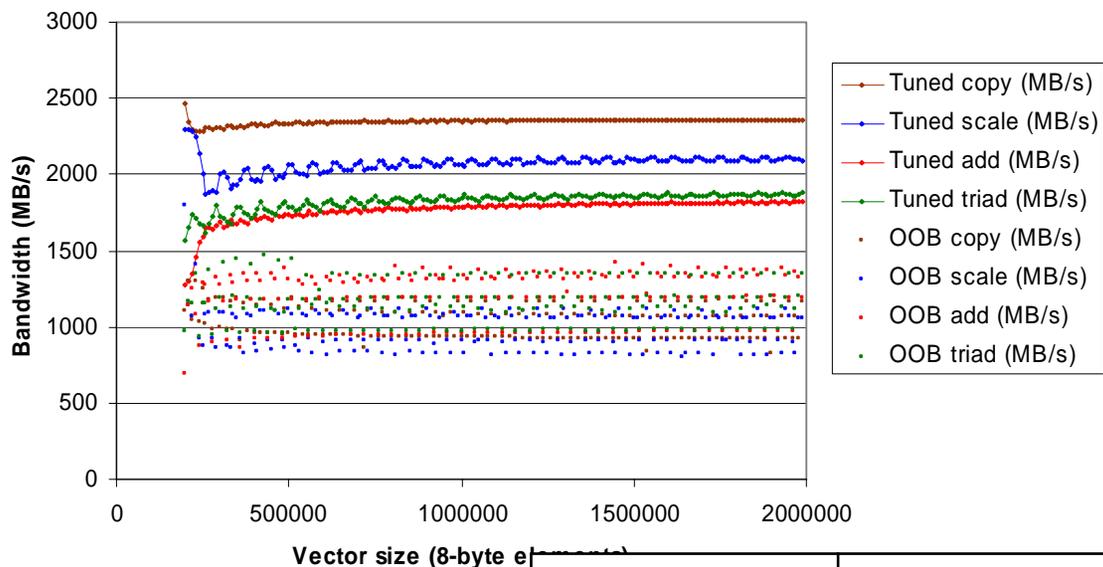
Memory Bandwidth Utilization vs Vector Size for Different Implementations of DAXPY



# STREAM Performance

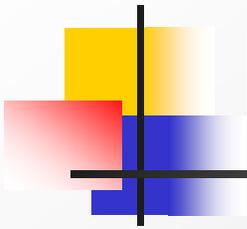
BG/L Single-Node STREAM Performance (444 MHz)

28 July 2003



- Out-of-box performance is 50-65% of tuned performance
  - Lessons learned in tuning will be transferred to compiler where possible
- Comparison with commodity microprocessors is competitive

Machine	Frequency (MHz)	STREAM (MB/s)	FP peak (Mflop/s)	Balance (B/F)
Intel Xeon	3060	2900	6120	<b>0.474</b>
BG/L	444	2355	3552	<b>0.663</b>
BG/L	670	3579	5360	<b>0.668</b>
AMD Opteron	1800	3600	3600	<b>1.000</b>



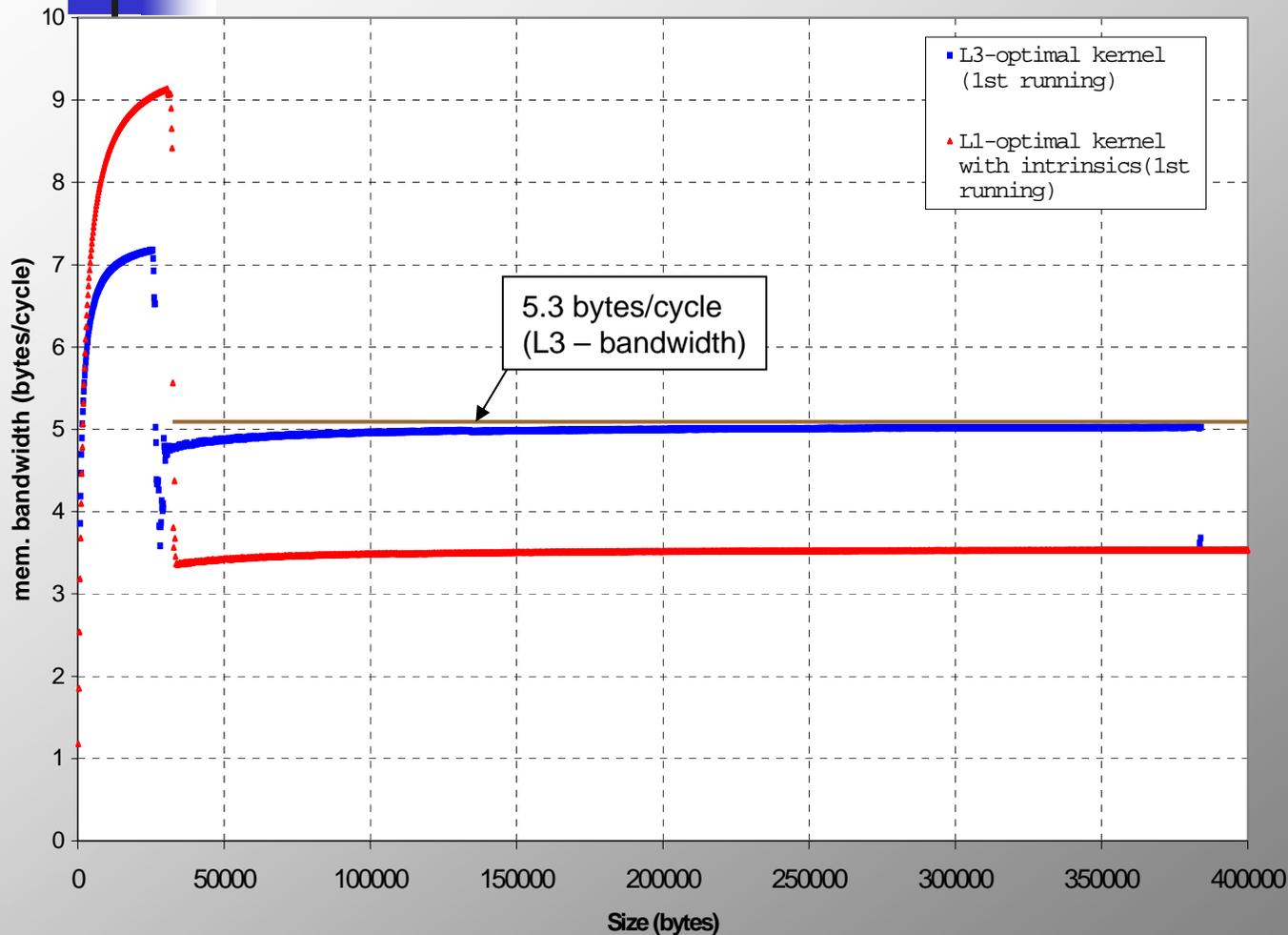
# DGEMV

---

- 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

Memory Bandwidth Utilization for L3-optimal DGEMV kernel

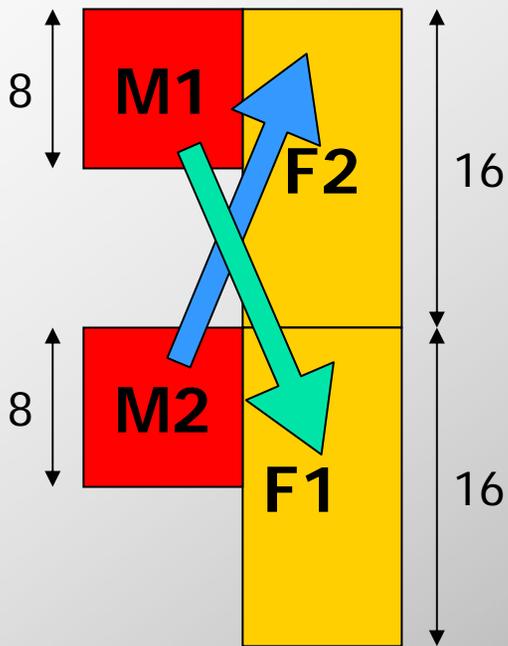


Two different kernels are needed to deal with data when:

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

# Matrix Multiplication

## Tiling for Registers (Analysis)



- Latency tolerance (not bandwidth)
  - Take advantage of register count
- Unroll by factor of two
  - 24 register pairs
  - 32 cycles per unrolled iteration
  - 15 cycle load-to-use latency (L2 hit)
- Could go to 3-way unroll if needed
  - 32 register pairs
  - 32 cycles per unrolled iteration
  - 31 cycle load-to-use latency

# Recursive Data Format

- Mapping 2-D (Matrix) to 1-D (RAM)
  - C/Fortran do not map well
- Space-Filling Curve Approximation
- Recursive **Tiling**
- Allows us to **efficiently stream** the data through the memory hierarchy



# Recursive Data Format

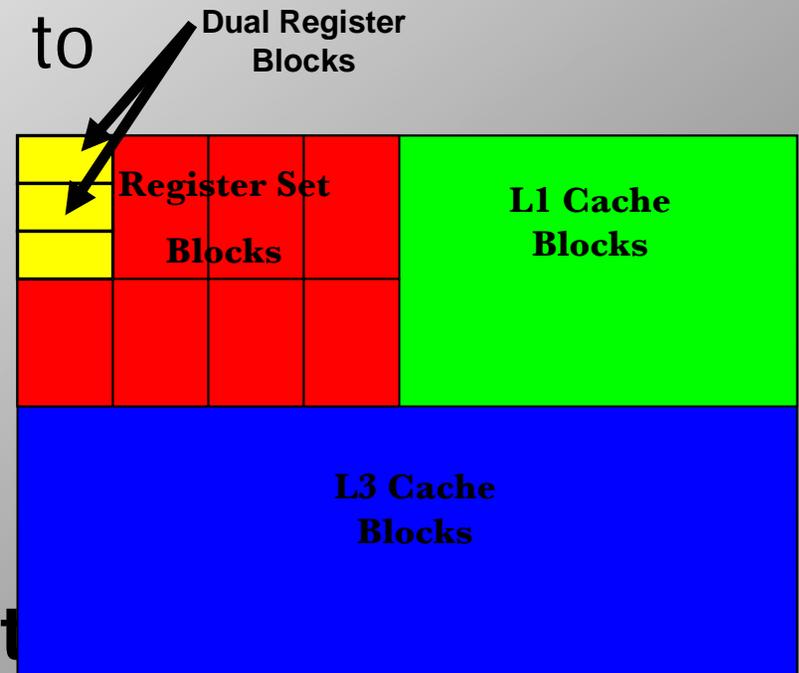
- Mapping 2-D (Matrix) to 1-D (RAM)

- C/Fortran do not map

- Space-Filling Curve Approximation

- Recursive **Tiling**

- Allows us to **efficiently stream** the data through the memory hierarchy



# Programming Options

## High Level

---

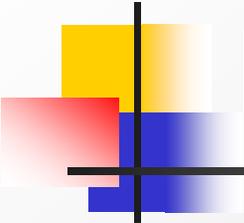
- Compiler optimization to find SIMD parallelism (XL/IBM Only)
  - Currently uses Larsen-Amarasinghe “Superword Level Parallelism” algorithm (PLDI’00) to detect and generate SIMD operations
  - Needs user input for specifying memory alignment and lack of aliasing
    - `__alignx` assertion
    - `disjoint` pragma
  - Currently limited to parallel SIMD and memory operations

# Programming Options

## Low Level

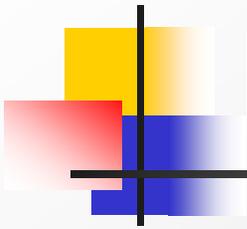
---

- In-line assembly
  - User responsible for instruction selection, register allocation, and scheduling
- Dual FPU intrinsics
  - Complex data type used to model pair of double-precision numbers that occupy a (P, S) register pair
  - User responsible for instruction selection
  - Compiler responsible for register allocation and scheduling
  - Supported in C99 and Fortran (not in C++)



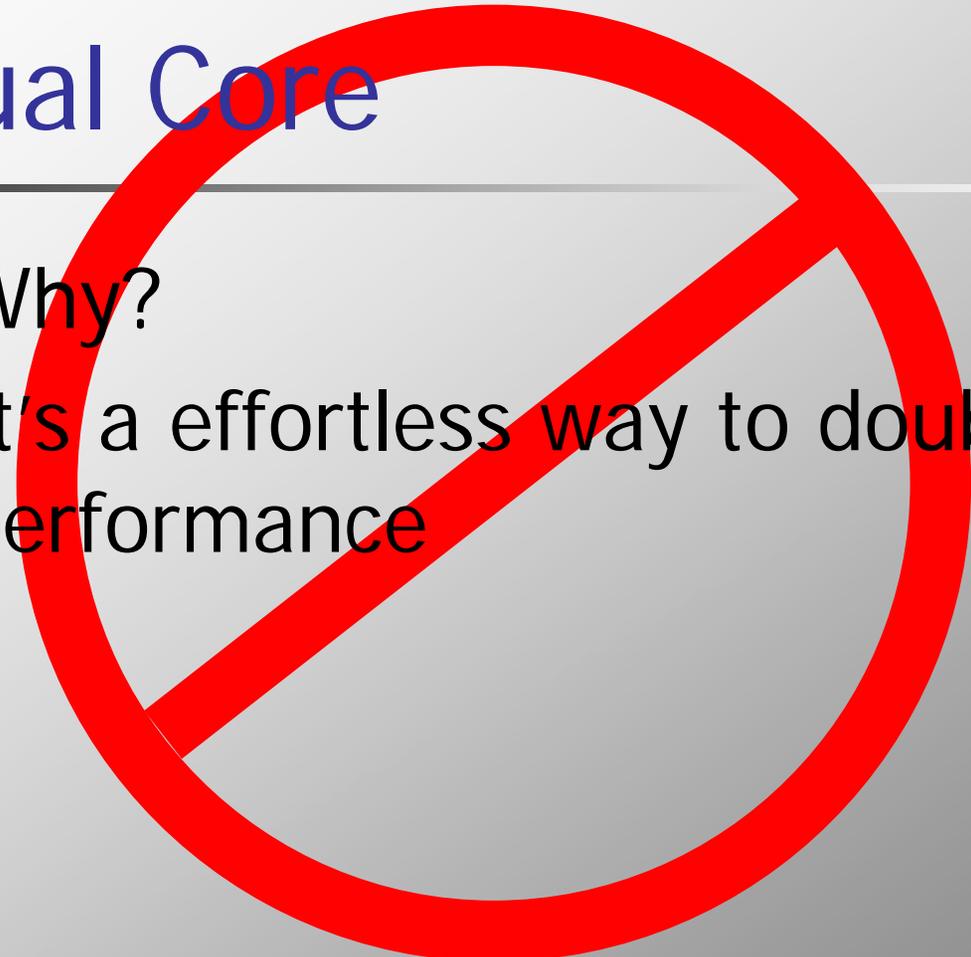
# Dual Core

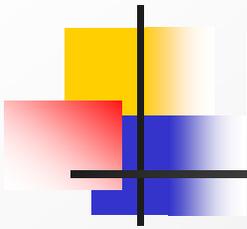
---



# Dual Core

---

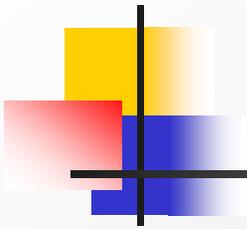
- Why?
  - It's a effortless way to double your performance
- 



# Dual Core

---

- Why?
- It exploits the architecture and may allow one to double the performance of their code in some cases/regions

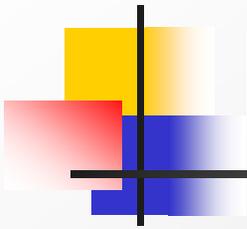


# Dual Core

---

- How?

- `grip=co_start(copro, func, args);`
  - copro: opaque object
  - func: function pointer
  - args: Void ptr to arguments
    - Must be flushed on originator (core 0)
    - Must be invalidated on receiver (core 1)
- `co_join(copro, grip);`



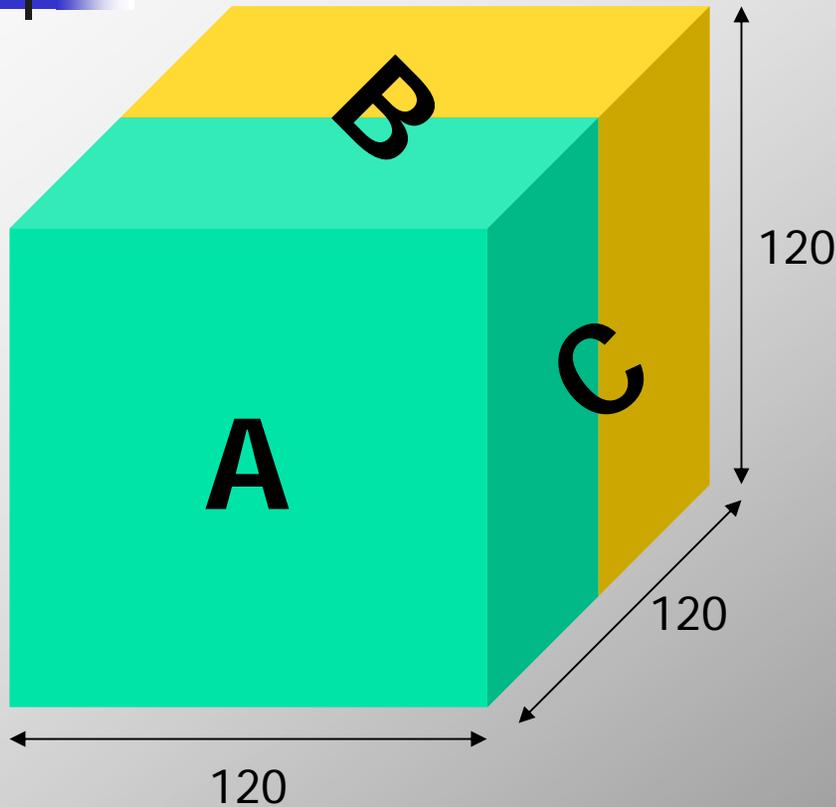
# Dual Core

---

- When ...
  - Your computation is not L2 or above memory-bound
    - Those elements are shared between cores
  - There is a good deal of (independent) computation to perform
    - Process fork is relatively cheap, but not free
  - The amount of altered memory is small
    - Washing out the caches takes cycles
      - Blind device (4200 obs. cycles vs. 4096 opt. cycles)
  - The computation is “almost” L1 resident
    - Two cores: twice as much L1

# Matrix Multiplication

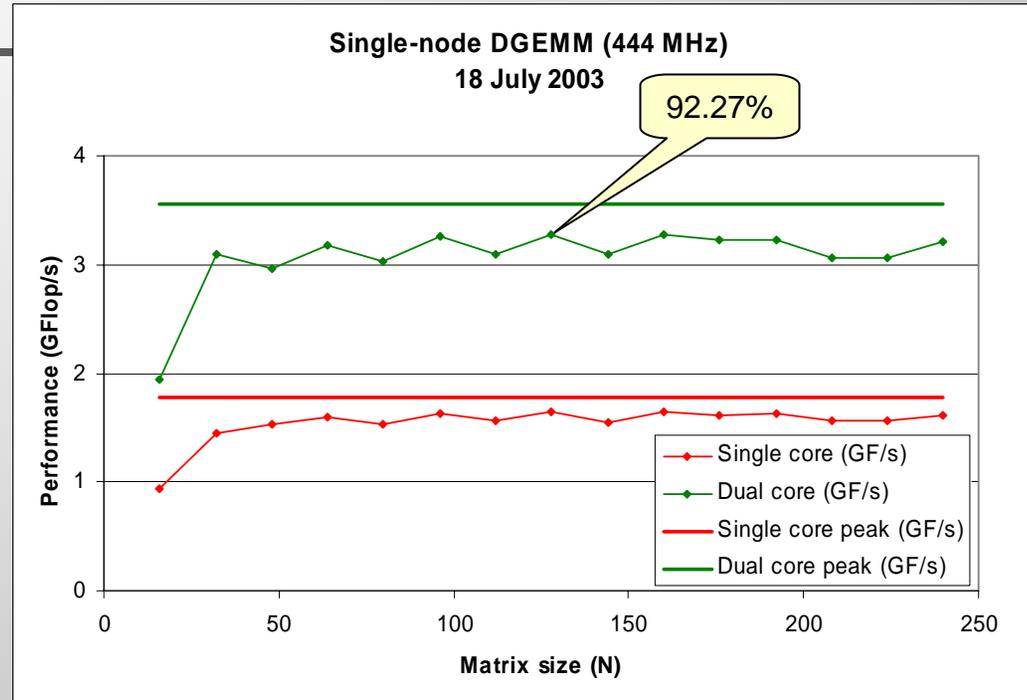
## Tiling for Dual Cores



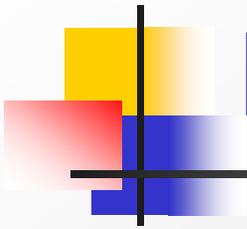
- Lack of coherence in L1 dictates split of C
- B “streams” through L1: split it to control stream traffic
- Tile for
  - Dual core
  - L3
  - L1
  - Registers

# Single-Node DGEMM

## Performance at 92% of Peak



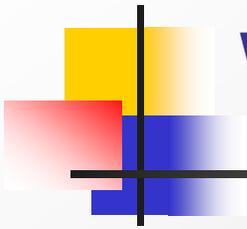
- Near-perfect scalability (1.99 $\times$ ) going from single-core to dual-core
- Dual-core code delivers 92.27% of peak flops (8 flop/pclk)
- Performance (as fraction of peak) competitive with that of Power3 and Power4



# Points to consider

---

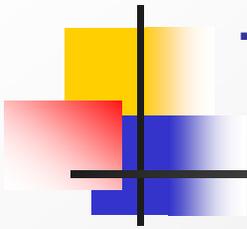
- Code fusion can enable one to
  - Perform a data re-format and/or make effective use of both cores for an operation
- The architecture is very rich
  - Corner cases have to be handled
  - Can be very powerful
  - Helpful in understanding performance
  - Semi-esoteric improvements exist
    - Fine-grained L1 data cache control



# 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, for example
  - 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



# Thanks to ...

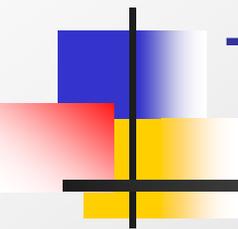
---

Leonardo Bachega: BLAS-1, performance results

Sid Chatterjee: Coprocessor, BLAS-1

Fred Gustavson, James Sexton: Data structure investigations, design, sanity tests

BG/L:



# Tuning for One Node

---

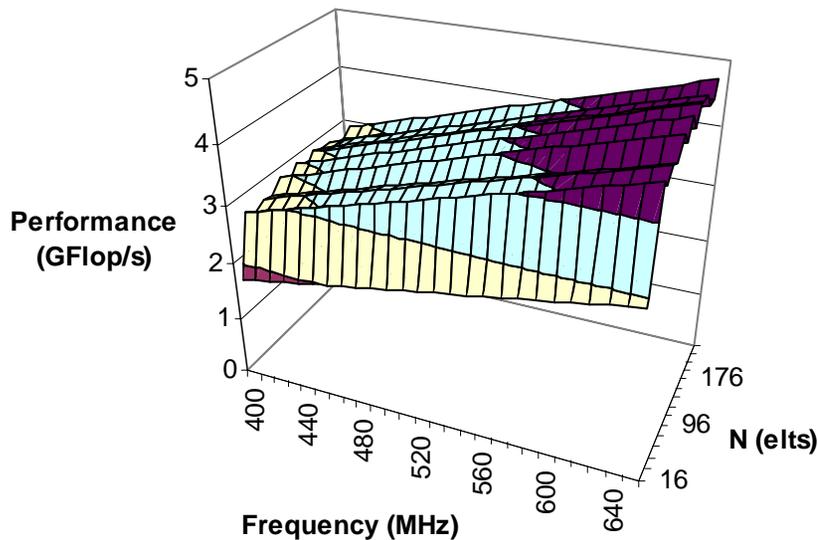
John A. Gunnels

Mathematical Sciences Dept.

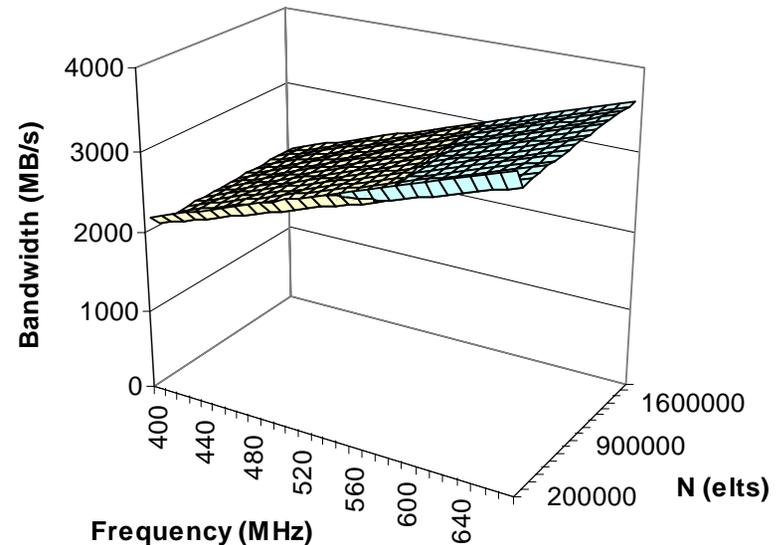
IBM T. J. Watson Research Center

# Performance Scales Linearly with Clock Frequency

Speed test of DGEMM, 25 July 2003



Speed test of STREAM COPY, 25 July 2003



- Measured performance of DGEMM and STREAM scale linearly with frequency
  - DGEMM at 650 MHz delivers 4.79 Gflop/s
  - STREAM COPY at 670 MHz delivers 3579 MB/s