NATIONAL ENERGY RESEARCH
SCIENTIFIC COMPUTING CENTER

National Energy Research Scientific Computing Center (NERSC)

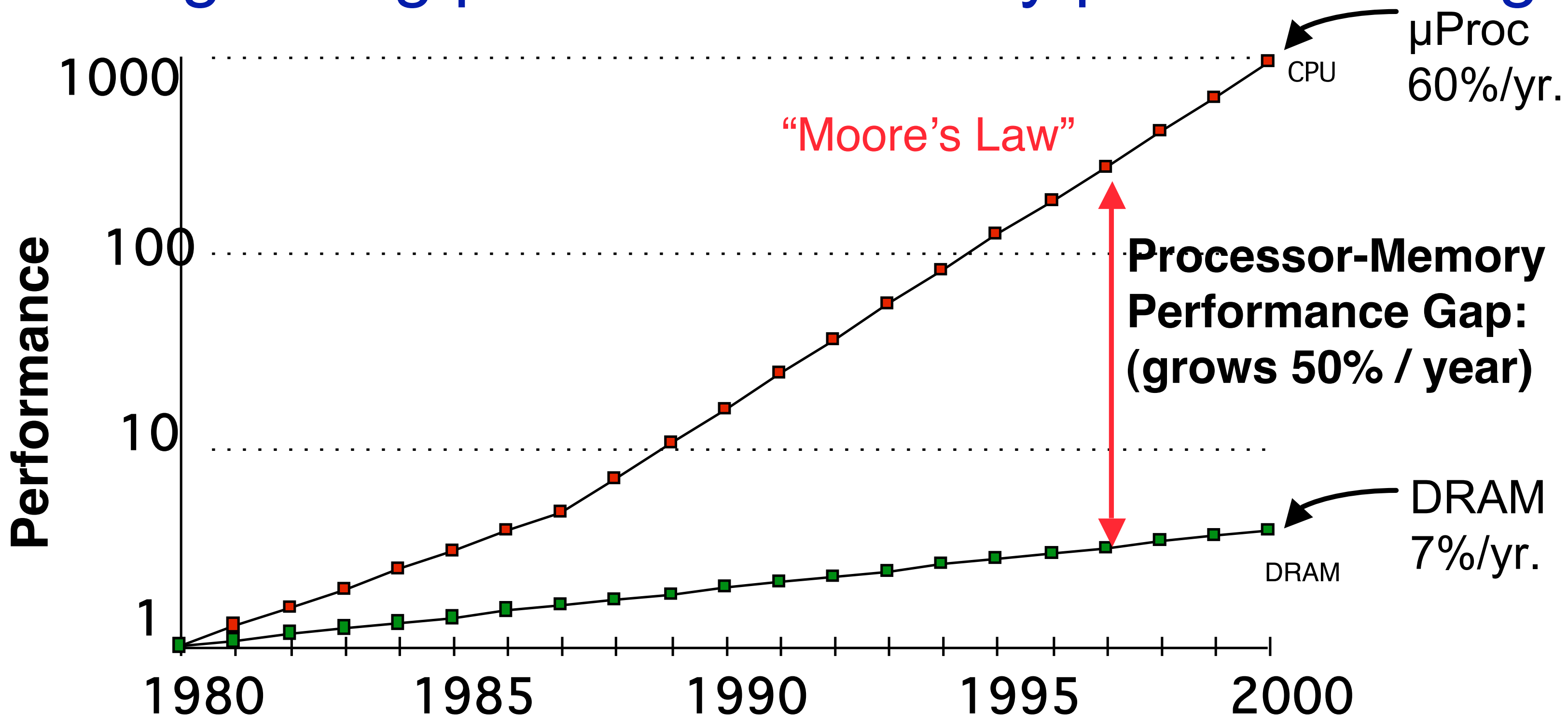
About Memory Bandwidth

John Shalf
NERSC Center Division, LBNL

June 13, 2007

Memory Performance is Key

Ever-growing processor-memory performance gap

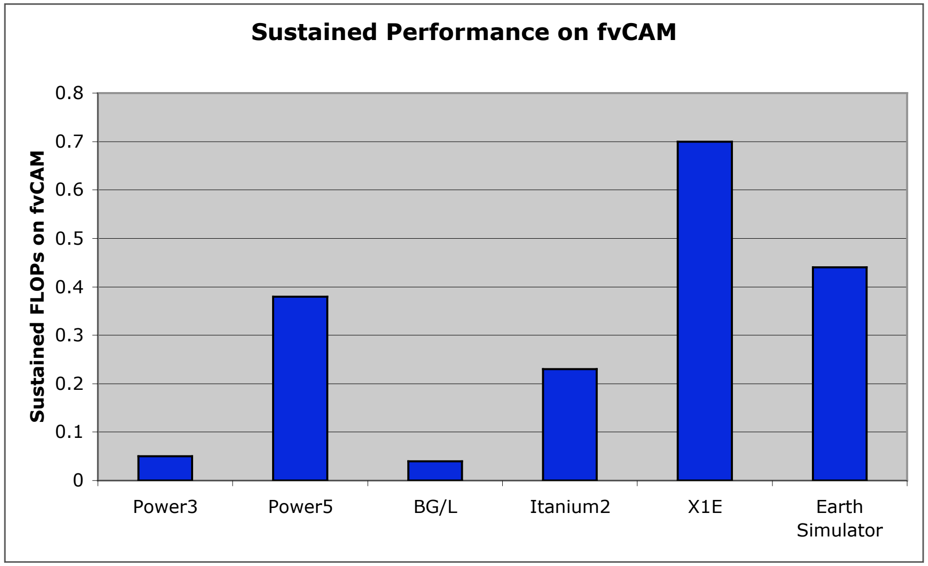


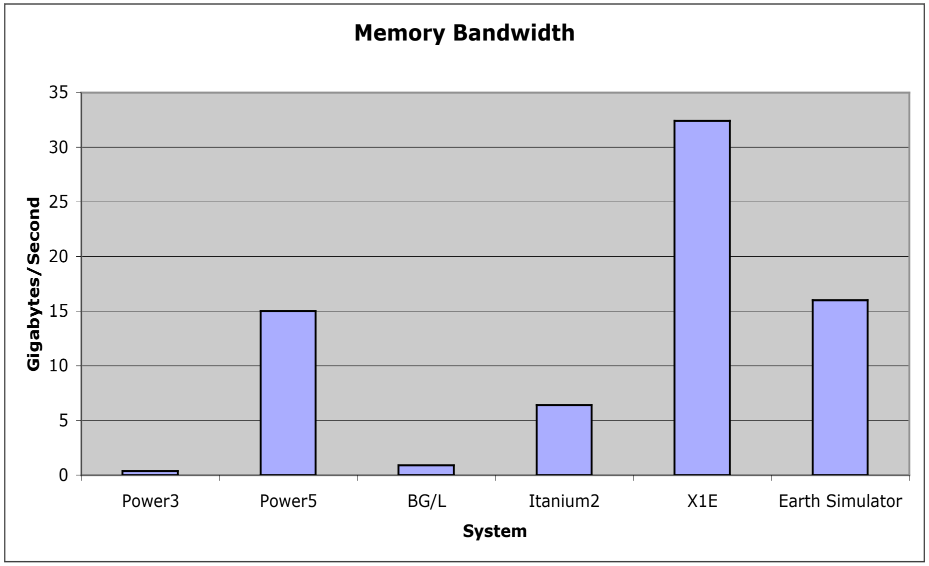
- Total chip performance following Moore's Law
- Increasing concern that memory bandwidth may cap overall performance

Concerns about Multicore

- Memory Bandwidth Starvation
  - *"Multicore puts us on the wrong side of the memory wall. Will CMP ultimately be asphyxiated by the memory wall?"* Thomas Sterling
  - While true, multicore has not introduced a *new* problem
    - "memory wall" first described in 1994 paper by Sally McKee et al. about uniprocessors
    - Bandwidth gap matches historical trends FLOPs on chip doubles every 18months (just by different means)
  - Regardless it is a worthy concern

CCSM3 FVCAM Performance

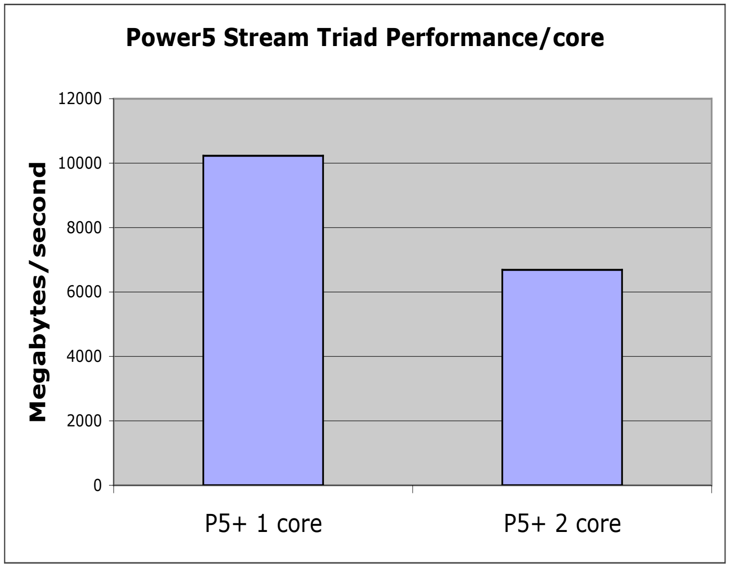


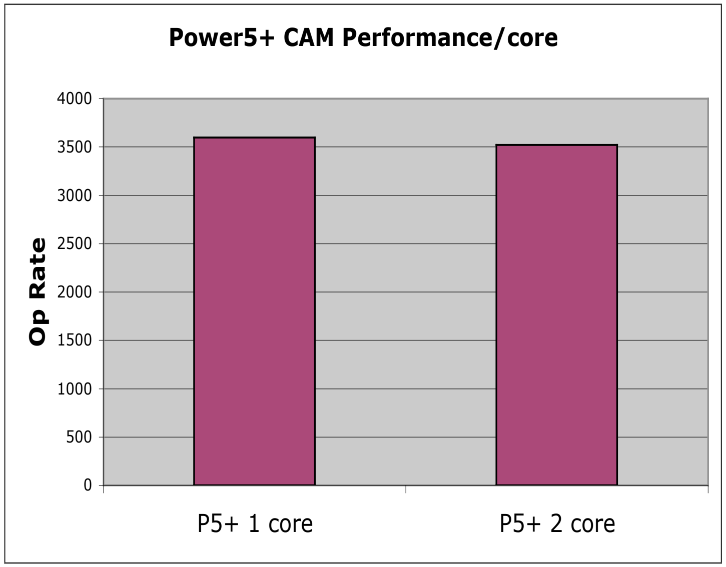


- FVCAM (atmospheric component of climate model) *OBVIOUSLY* correlated with memory bandwidth
- More memory bandwidth means more performance!
- So my theory is "If I move from single-core to dual-core, my performance should drop proportional to effective memory bandwidth delivered to each core!" *(right?)*

CAM on Power5+

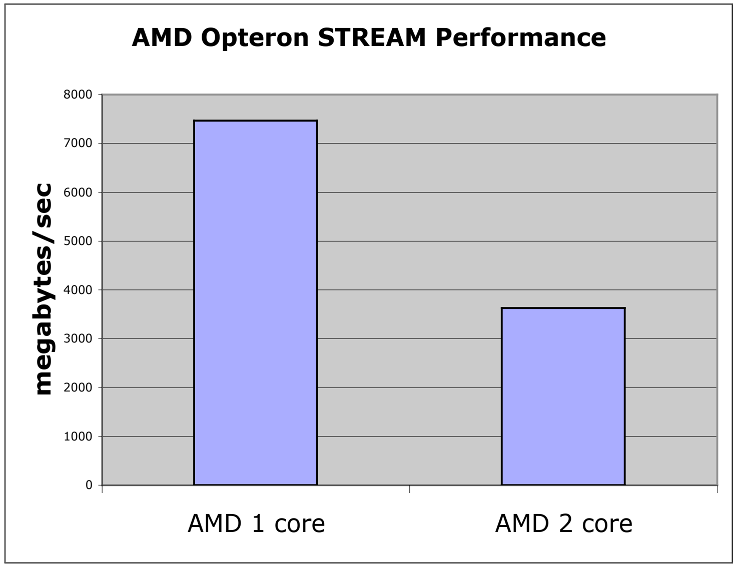
(test our memory bandwidth theory)

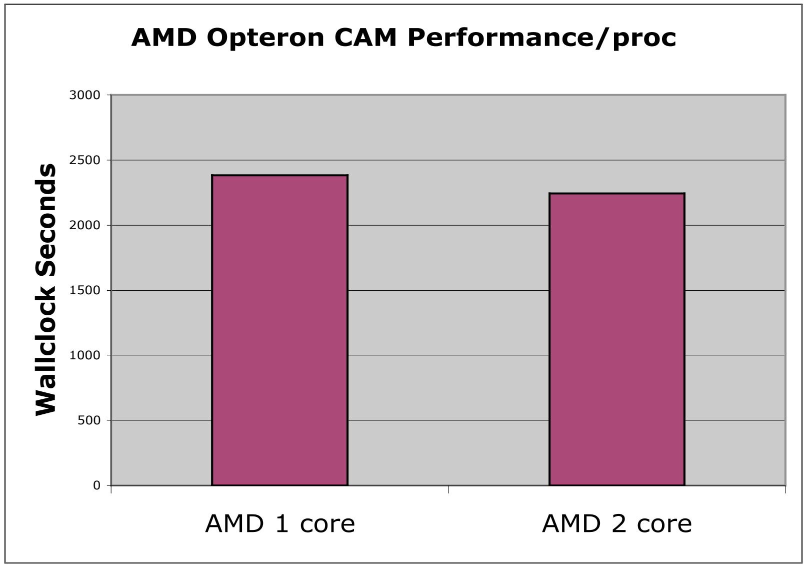




- T85 model (spectral CAM) run sparse and dense mode. (turn off timers for MPI operations)
- 2% performance drop (per core) when moving from 1-2 cores
- Does not meet expectations
  - Perhaps the Power5 is weird… Lets try another processor to support my theory

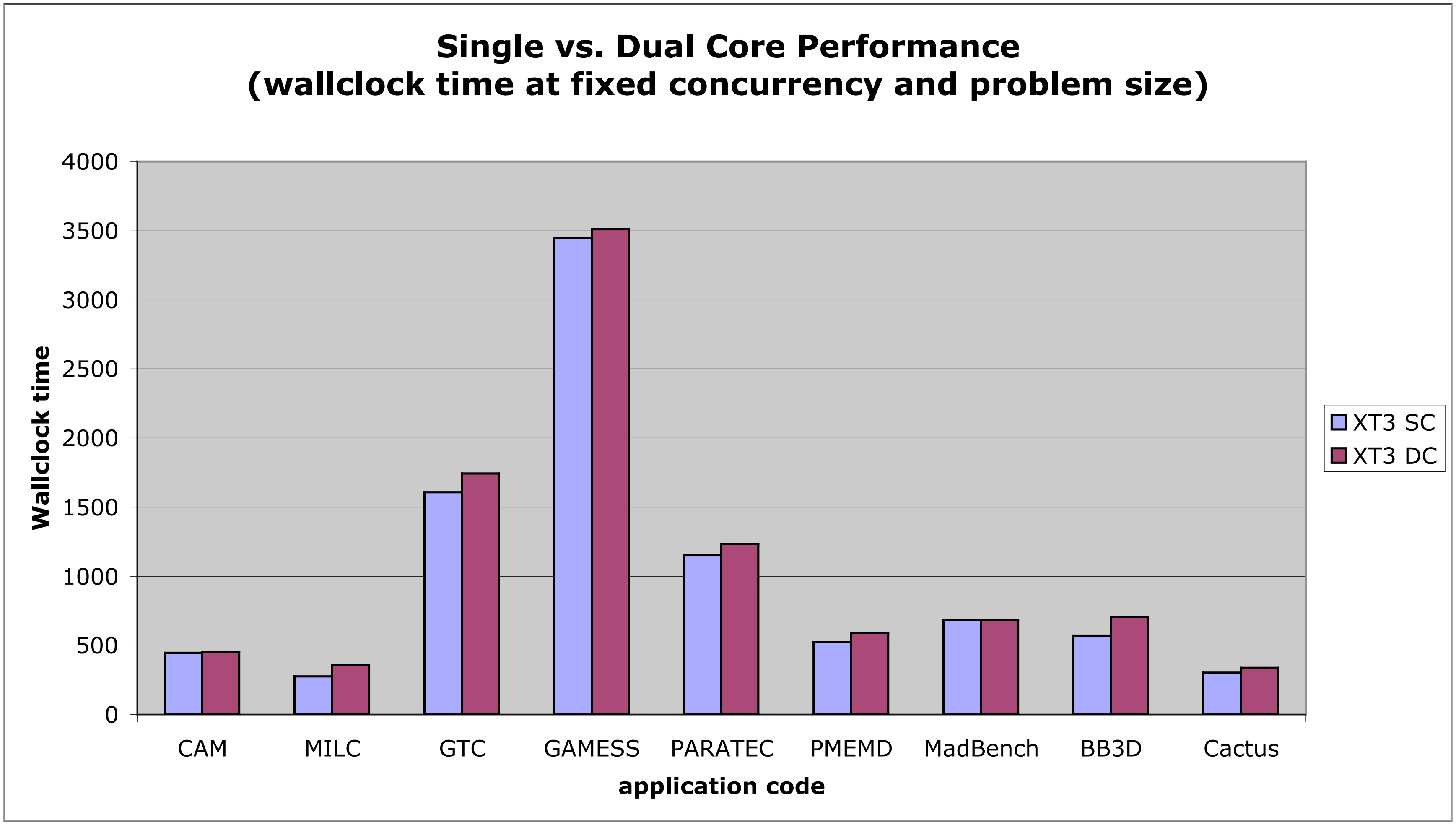
CAM on AMD Opteron



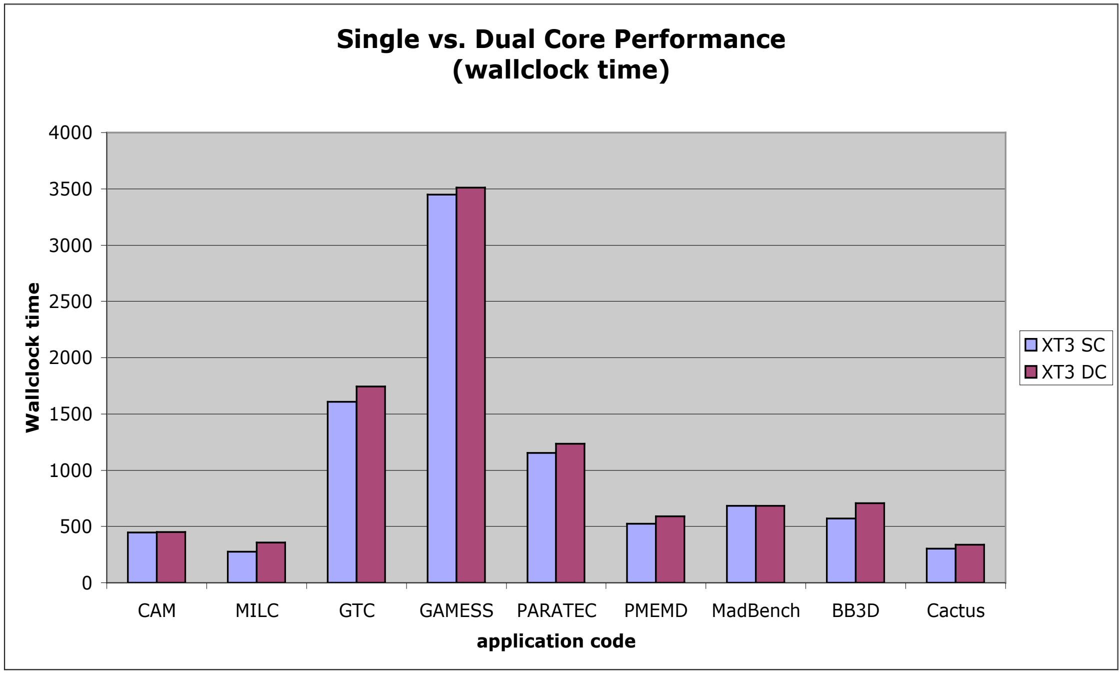


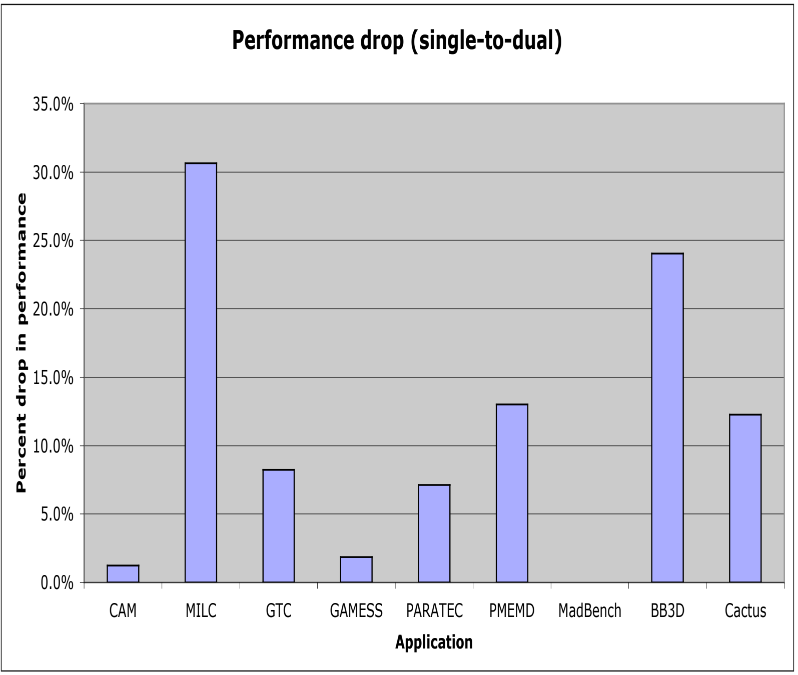
- 3% drop in performance going from single to dual core
  - Still not what I wanted
  - Need to find application to support my theory
  - Lets look at a broad spectrum of applications!

NERSC SSP Applications



NERSC SSP Applications



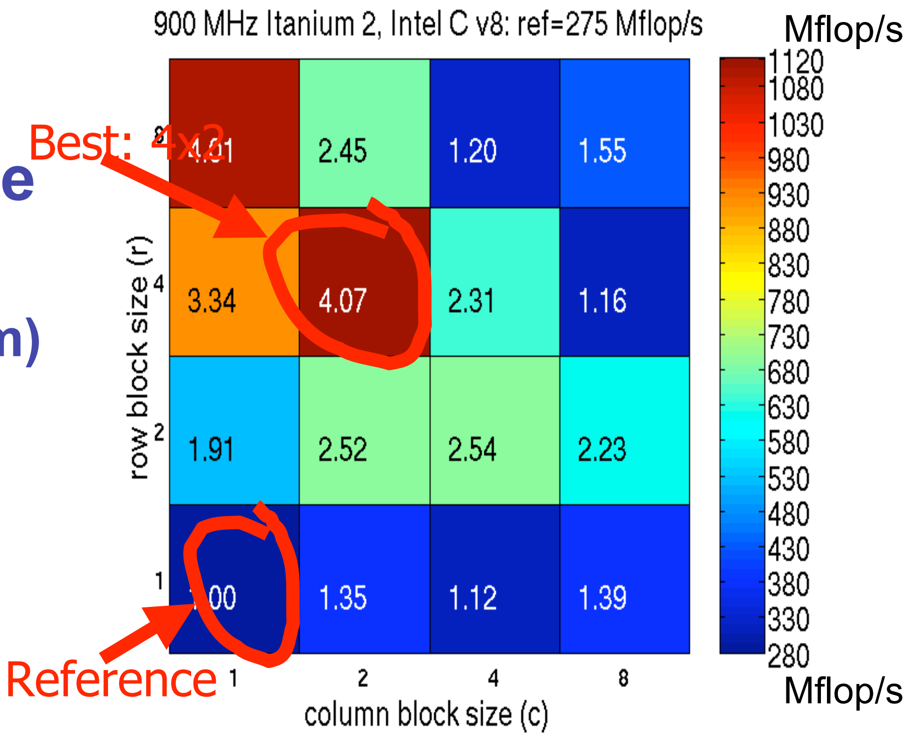


- Still 10% drop on average when halving memory bandwidth!
  - #$%^&* application developers write crummy code!
  - Lets pick an application that *I KNOW* is memory bandwidth bound!

Lets Try SpMV

- Perhaps full application codes are a bad example
- Lets try a kernel like SpMV
  - Should be memory bound!
  - Small kernel
- Highly optimized to maximize memory performance
  - Hand coded (sometimes in asm) by highly motivated GSRA
  - Carefully crafted prefetch
  - Exhaustive search for optimal block size
  - Auto-search for optimal blocking strategy!

For finite element problem (BCSR) [Im, Yelick, Vuduc, 2005]


Best: 4x2

Reference

Example: Sparse Matrix * Vector

Name	Clovertown	Opteron	Cell
Chips*Cores	2*4 = 8	2*2 = 4	1*8 = 8
Architecture	4-/3-issue, SSE, OOO, caches, prefetch		2-VLIW, SIMD, local store, DMA
Clock Rate	2.3 GHz	2.2 GHz	3.2 GHz
Peak MemBW	21.3 GB/s	21.3	25.6 GB/s
SPMv MemBW	[redacted]	[redacted]	[redacted]
Efficiency %	[redacted]	[redacted]	[redacted]
Peak GFLOPS	75	18	15 (DP Fl. Pt.)
SPMv GFLOPS	[redacted]	[redacted]	[redacted]
Efficiency %	[redacted]	[redacted]	[redacted]

What the #$%^& is going on Here!!!

- Cannot find data to support my conclusion!
  - And it was a good conclusion!
  - Theory was proved conclusively by correlation to memory bandwidth shown on slide #1!

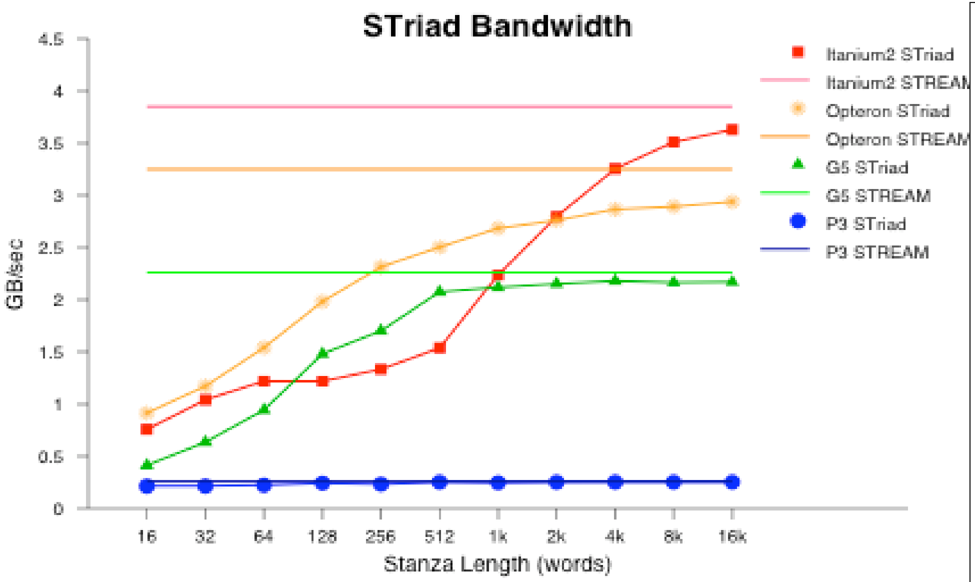
- Correlations do not guarantee causality
  - Consumption of memory bandwidth limited by ability to tolerate latency!
  - Vendors sized memory bandwidth to match what processor core could consume (2nd order effect manufactured a correlation)

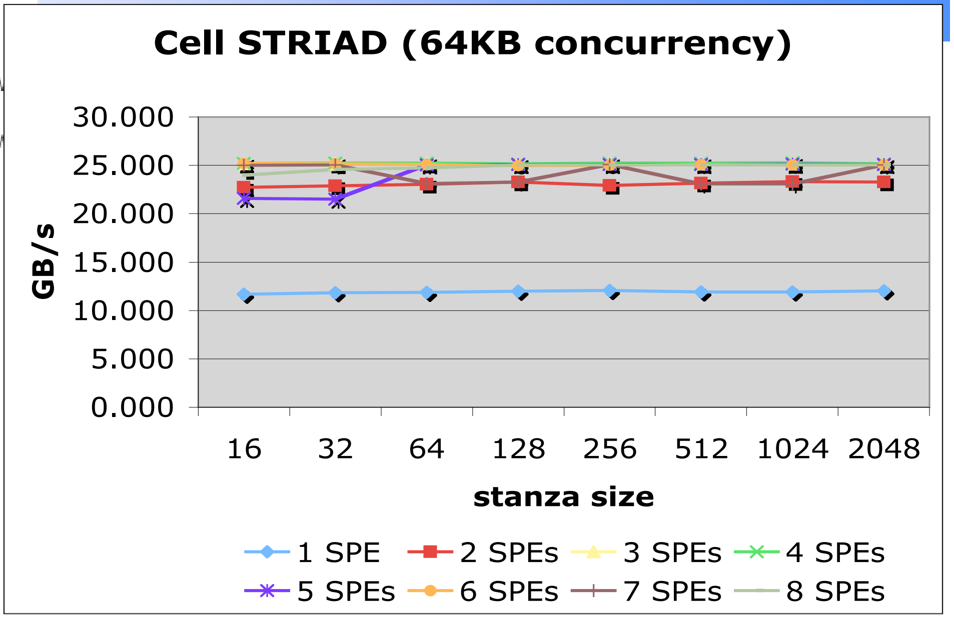
Short Diversion about Latency Hiding

- Little’s Law: bandwidth * latency = concurrency
  - bandwidth * latency = #outstanding_memory_fetches
- For Power5+ single-core (theoretical):
  - 120ns * 25 Gigabytes/sec
  - 3000 bytes of data in flight (375 DP operands)
  - 23.4 cache lines (very close to 24 RCQ depth on Power5)
  - 375 operands must be in flight to balance Little’s Law!
    - But I only have only 32 FP registers
    - Even with OOO, only ~100 FP shadow registers, and instruction reordering window is only ~100
    - Means, must depend on prefetch (375 operand prefetch depth)
- Various ways to manipulate memory fetch concurrency
  - 2x memory bandwidth: Need 6000 bytes/flight
  - 2x cores: Each only needs 1500 bytes/flight
  - 2 threads/core: Each needs 750 bytes/flight
  - 128 slower cores/threads?: 24 bytes in flight (3 DP words)
  - Vectors *(not SIMD!):* 64-128 words per vec load (1024 bytes)
  - Software Controlled Memory (eg. Cell, ViVA)
- Need mem queue depth performance ctr!

Why is the STI Cell So Efficient?

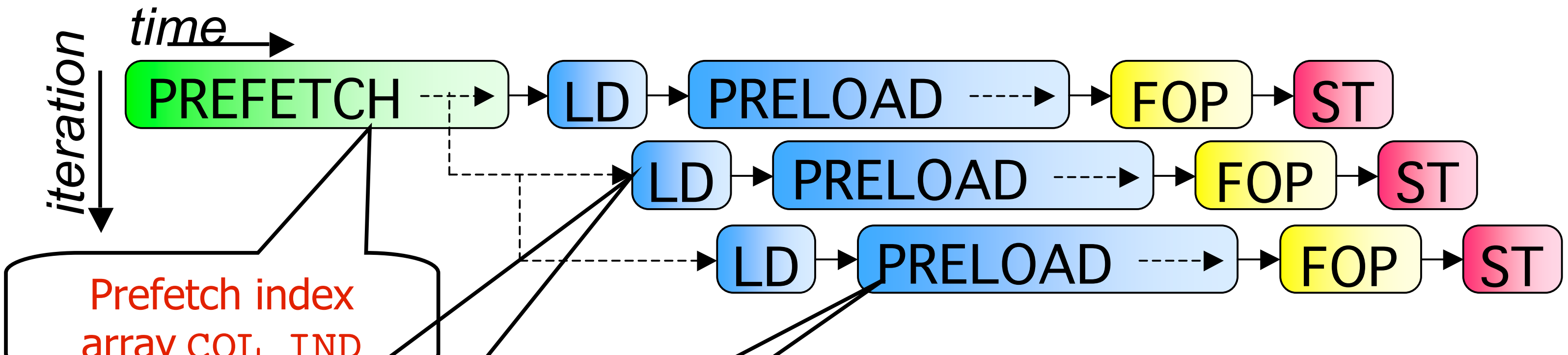
(Latency Hiding with Software Controlled Memory)





- **Performance of Standard Cache Hierarchy**
  - **Cache hierarchies underutilize memory bandwidth due to inability to tolerate latency**
  - **Hardware prefetch prefers long unit-stride access patterns (optimized for STREAM)**
  - **But in practice, access patterns are for shorter stanzas: so never reaches peak bandwidth (still latency limited)**
- **Cell "explicit DMA"**
  - **Cell software controlled DMA engines can provide nearly flat response for a variety of access patterns**
  - **Response is nearly full memory bandwidth can be utilized for all access patterns**
  - **Cell memory requests can be nearly completely hidden behind the computation due to asynchronous DMA engines**
  - **Performance model is simple and deterministic (much simpler than modeling a complex cache hierarchy),** min{time_for_memory_ops, time_for_core_exec}
  - **Problem: lack of tractable/broadly applicable programming model**

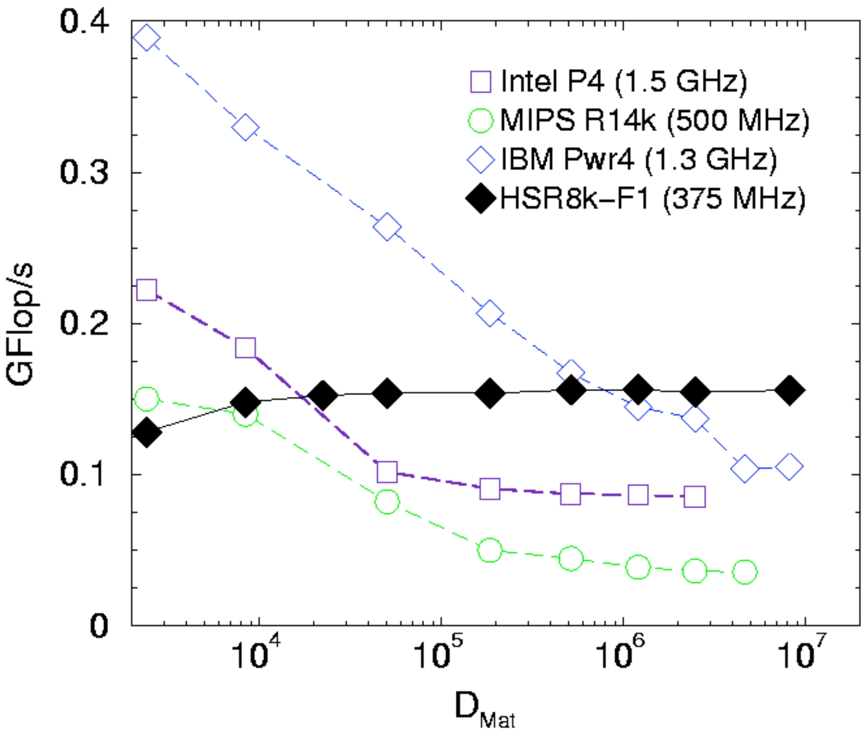
Deep Pipelining for Sparse MVM
(Gerhard Wellein: SR8k review)



Prefetch index array `COL_IND`

Load `index` from cache to reg

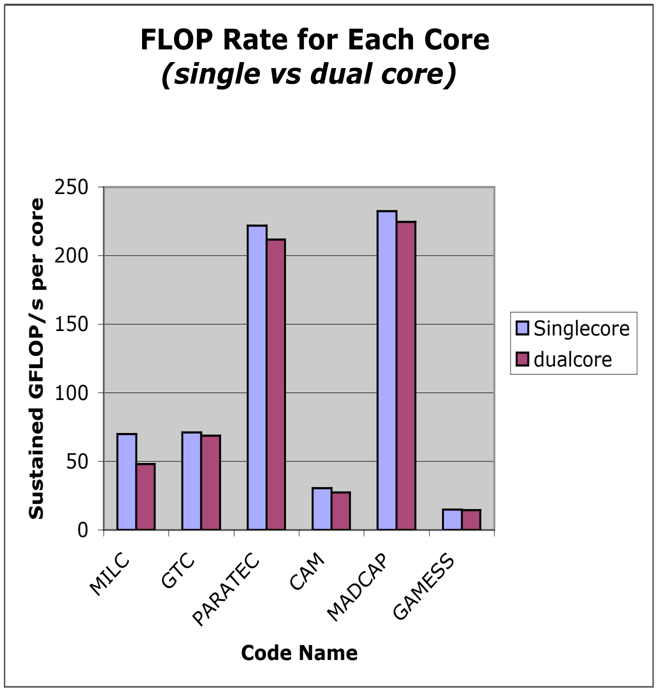
Preload single data item `x(index)`



Additional FPRs support loop unrolling of 24 iterations!

Will Multicore Slam Against the Memory Wall?

- **Memory Bandwidth Starvation**
  - *"Multicore puts us on the wrong side of the memory wall. Will CMP ultimately be asphyxiated by the memory wall?"* Thomas Sterling
  - Memory wall is NOT a problem that is caused by multicore (*term coined in 1994*).



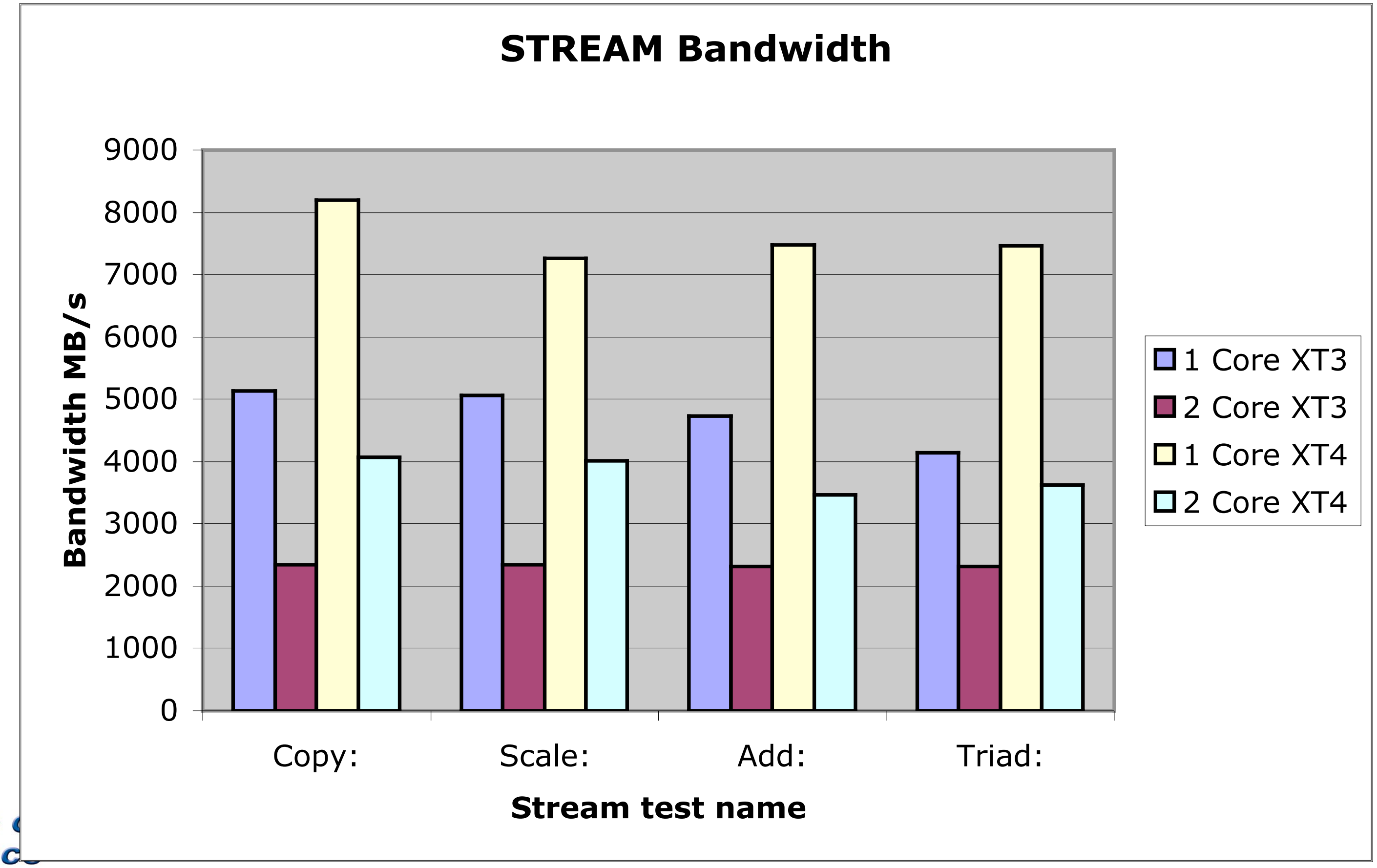
- **What about latency (other part of memory wall)**
  - Effective use of bandwidth is progressively inhibited by poor latency tolerance of modern microprocessor cores *(memory molassas rather than memory wall)*
  - Stalled clock rates actually halt growing gap of memory latency / operation

- **We can fix bandwidth (but not latency)**
  - With current technology, we could put 8x more bandwidth onto chips then we currently do! *. . . GPUs and Cicso Metro already do this!*
  - So why don't we do it? . . . *because it is ineffective for current processor cores*
  - *Manycore can use memory bandwidth more effectively*
  - *Software controlled memory can use bandwidth even more effectively*
  - *Can use manycore to test system balance using controlled environment*

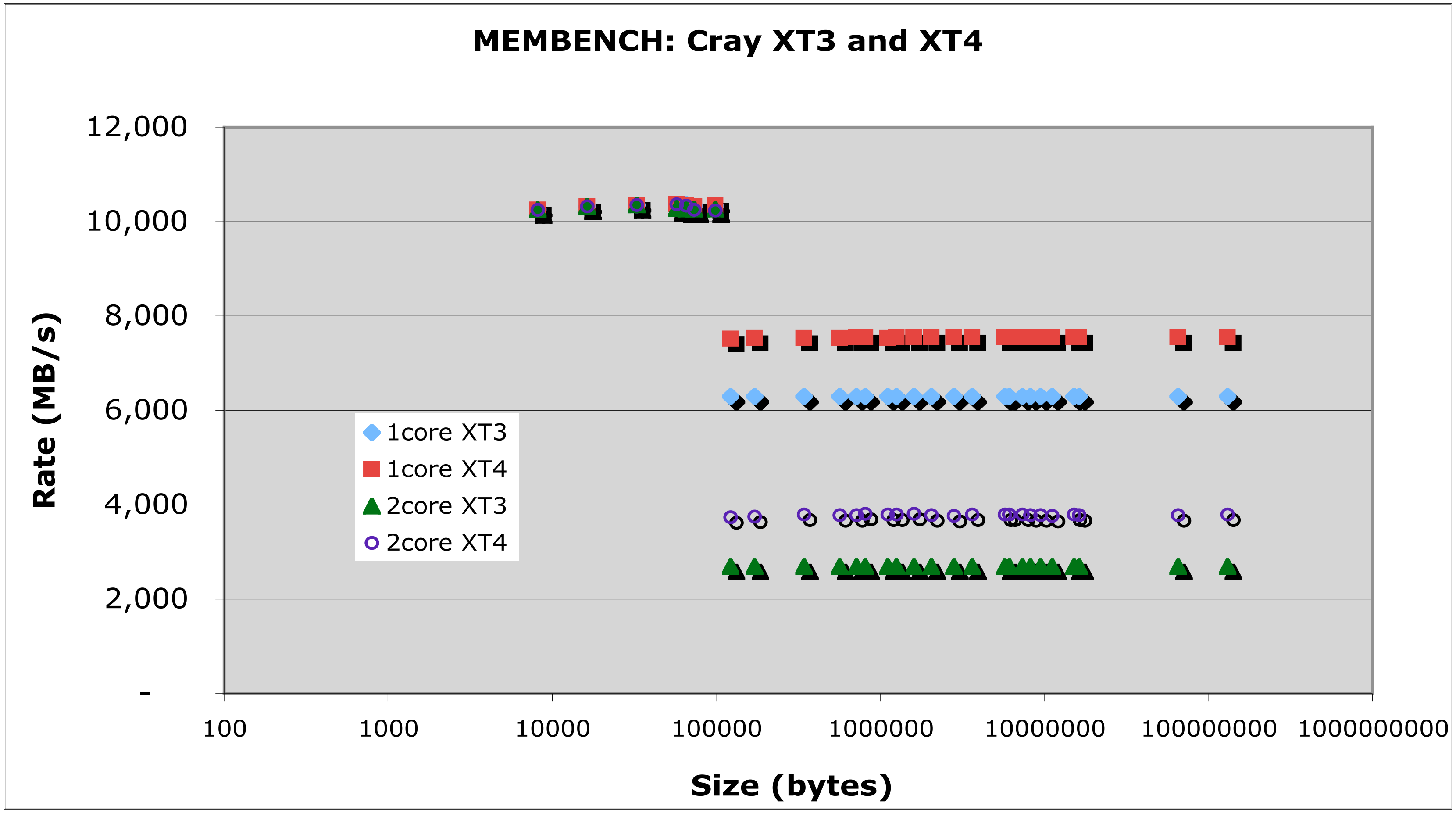
Predicting XT4 Quad Core Performance

STREAM

	1 Core XT3	1 Core XT4	2 Core XT3	2 Core XT4
Copy:	5137	8196	2345	4074
Scale:	5067	7257	2348	4012
Add:	4734	7482	2309	3469
Triad:	4135	7464	2310	3626

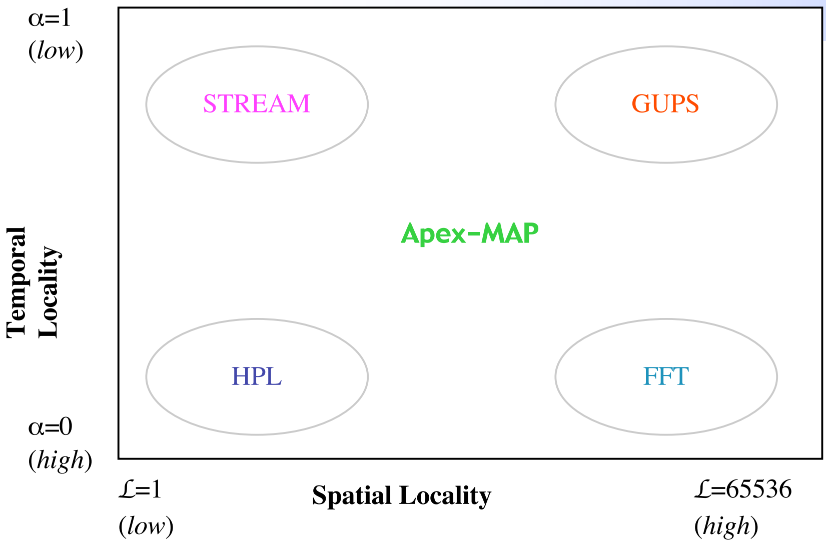


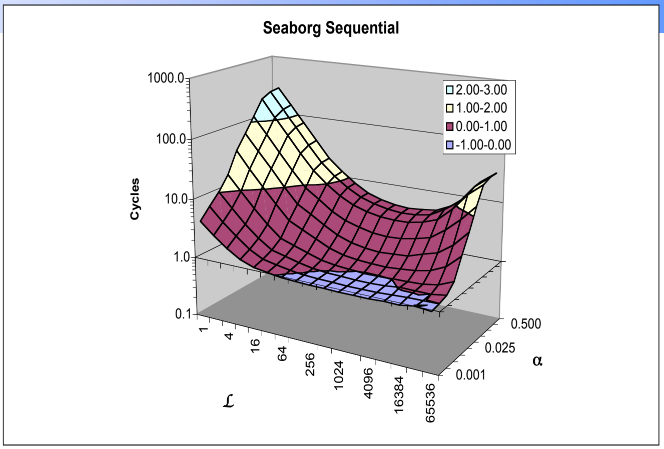
Membench



Membench results for XT3 and XT4 indicate primary source of contention is memory bandwidth (no signs of resource contention when data fits on-chip).

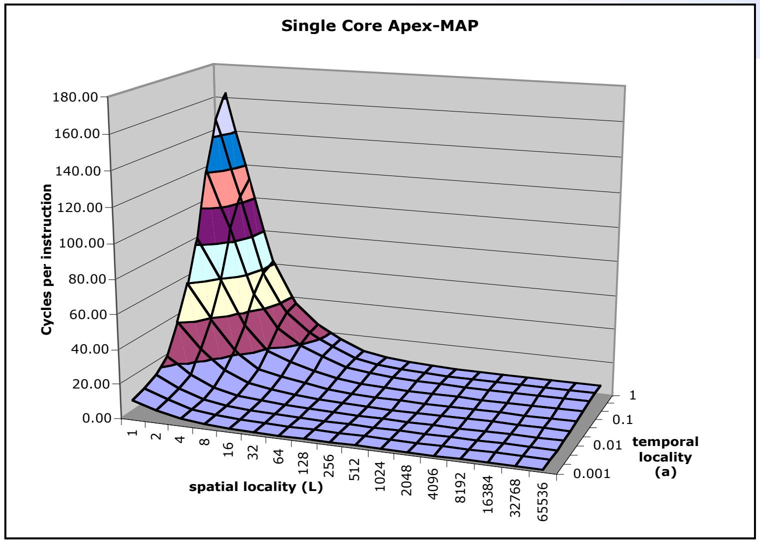
Apex-MAP



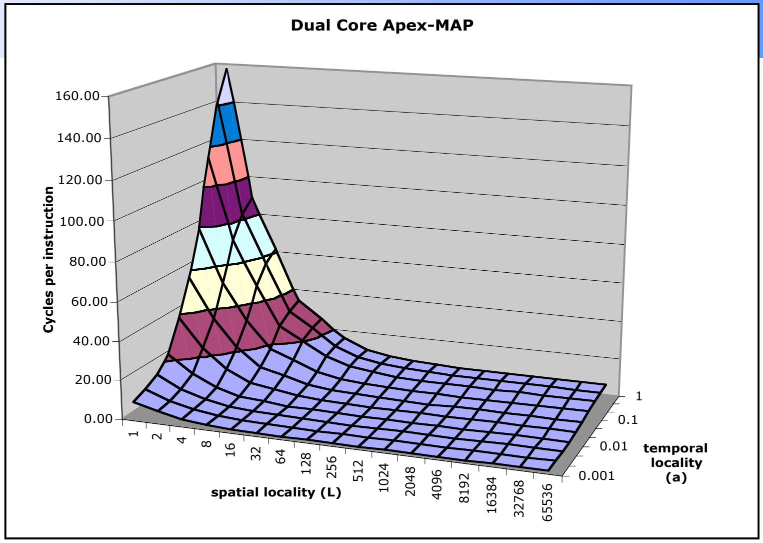


- Traces out 2D space of memory access patterns characterized by their spatial and temporal locality
- Parameter ℒ
  - Represents spatial locality
  - Describes size of contiguous accesses to a given memory location
- Parameter α
  - Represents temporal locality
  - Exponent of a power law distribution of memory addresses
- Performance (height of the graph) is given in cycles per memory access

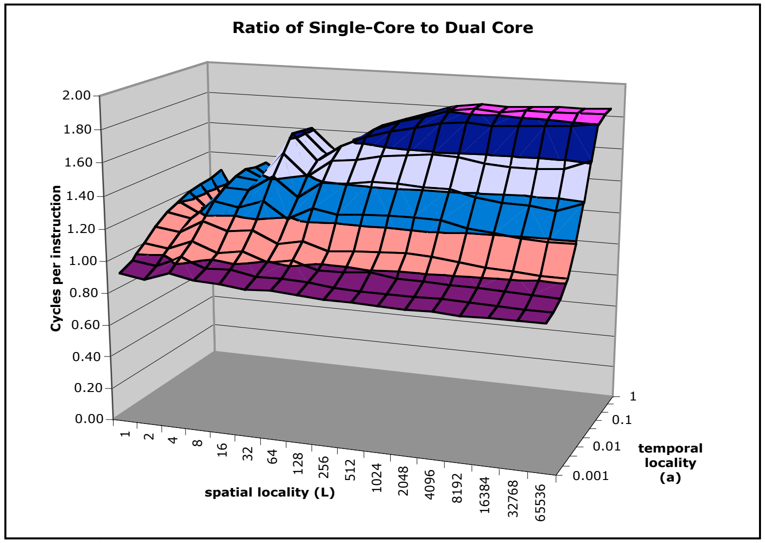
Apex-MAP



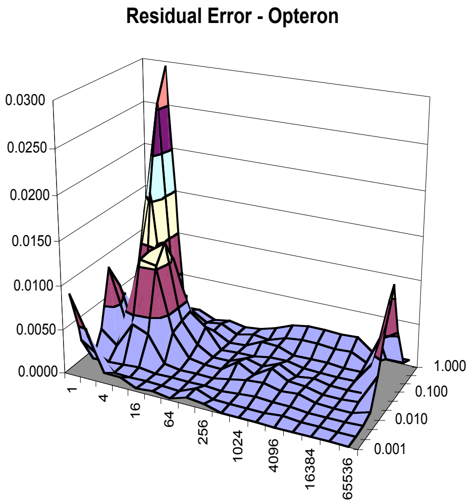
Single Core



Dual Core



Difference



Residual Error for model

Apex-MAP Conclusion

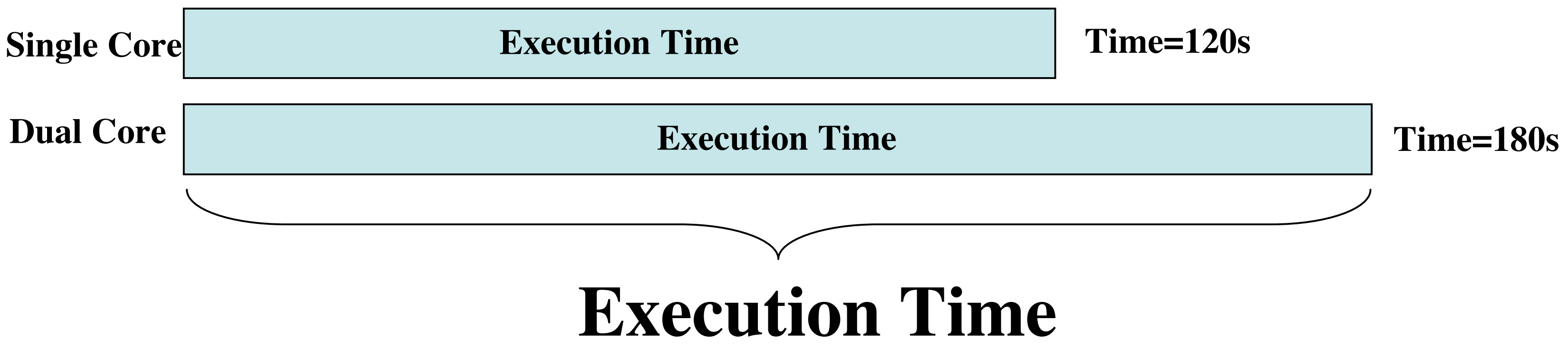
- A simple latency/bandwidth model provides a reasonably close estimation of actual Apex-MAP performance
  - Residual error of the model is low
  - That is to say, a simple explanation will suffice for dual-core performance penalty (*no need to chase down more complex models*)
- The simple model says that memory bandwidth contention explains most of the performance difference between single and dual core

Estimating Quad-Core Performance

- Assumptions
  - Memory bandwidth is the only contended resource
  - Can break down execution time into portion that is stalled on shared resources (*memory bandwidth*) and portion that is stalled on non-shared resources (*everything else*)
  - Estimate time spent on memory contention from XT3 single/dual core studies
  - Estimate # bytes moved in memory-contended zone
  - Extrapolate to XT4 based on increased memory bandwidth
    - Use to validate model
  - Extrapolate to quad-core

Estimating Quad-Core Performance

Cray XT3 Opteron@2.6Ghz DDR400



Single Core

Execution Time

Time=120s

Dual Core

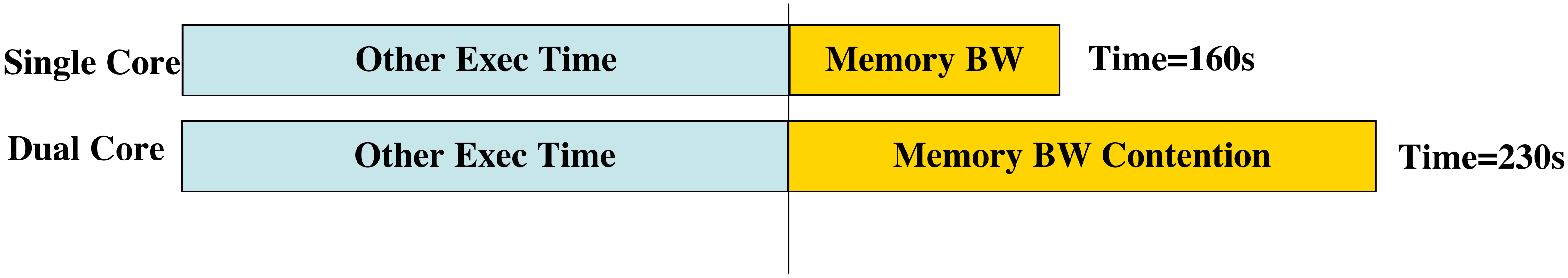
Execution Time

Time=180s

Execution Time

Estimating Quad-Core Performance

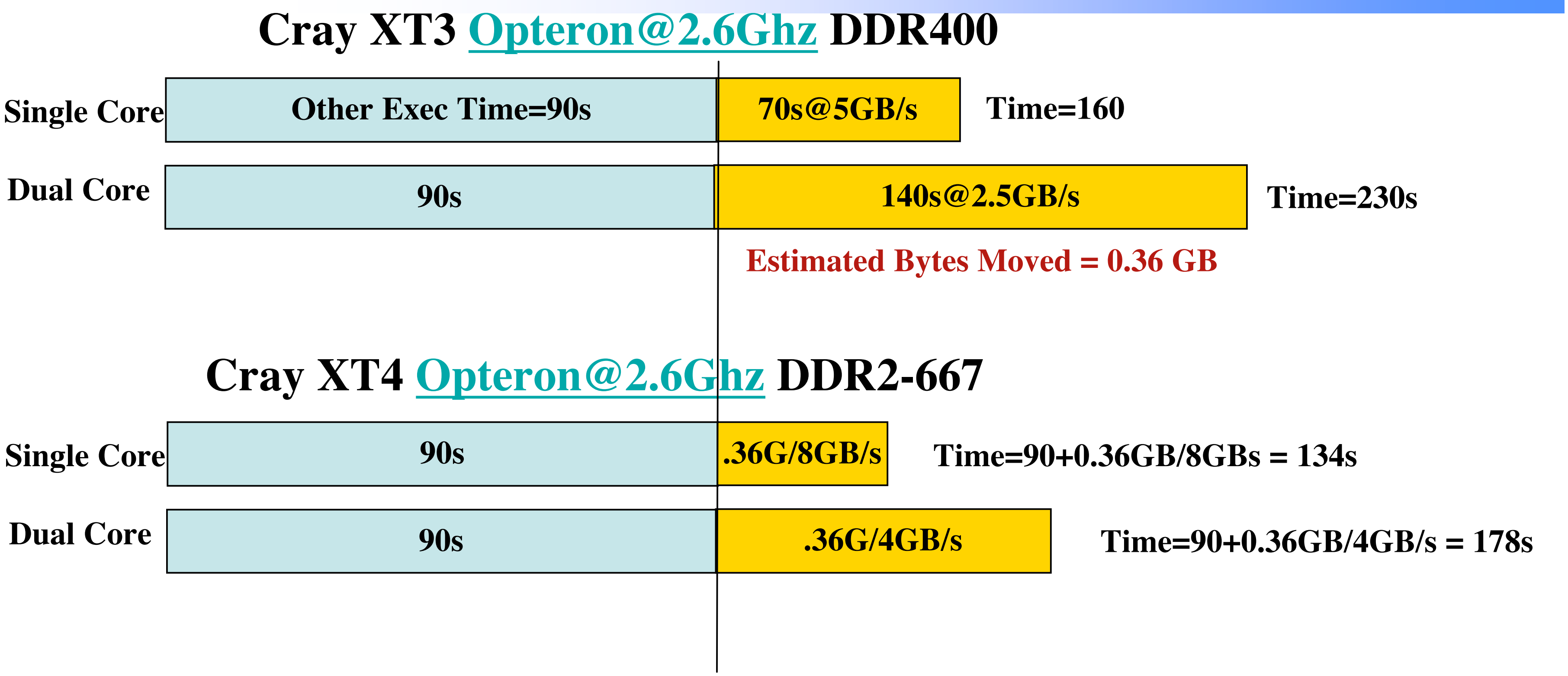
Cray XT3 Opteron@2.6Ghz DDR400



Single Core: Other Exec Time | Memory BW — Time=160s

Dual Core: Other Exec Time | Memory BW Contention — Time=230s

Estimating Quad-Core Performance



Cray XT3 Opteron@2.6Ghz DDR400

Single Core | Other Exec Time=90s | 70s@5GB/s | Time=160

Dual Core | 90s | 140s@2.5GB/s | Time=230s

Estimated Bytes Moved = 0.36 GB

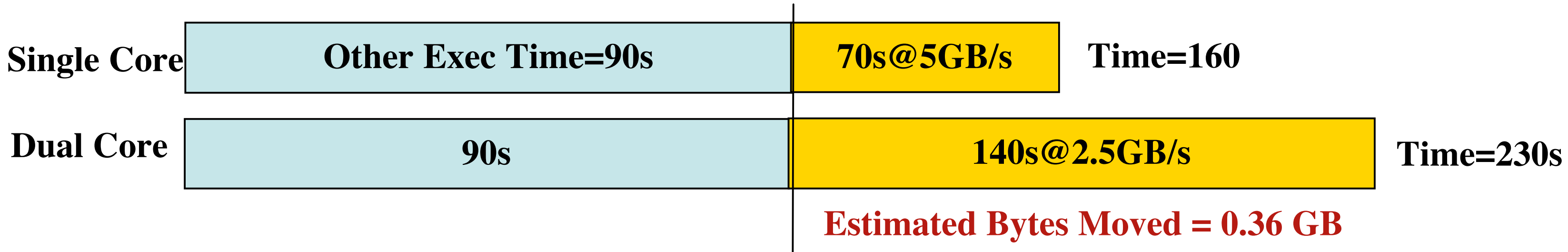
Cray XT4 Opteron@2.6Ghz DDR2-667

Single Core | 90s | .36G/8GB/s | Time=90+0.36GB/8GBs = 134s

Dual Core | 90s | .36G/4GB/s | Time=90+0.36GB/4GB/s = 178s

Estimating Quad-Core Performance

Cray XT3 Opteron@2.6Ghz DDR400

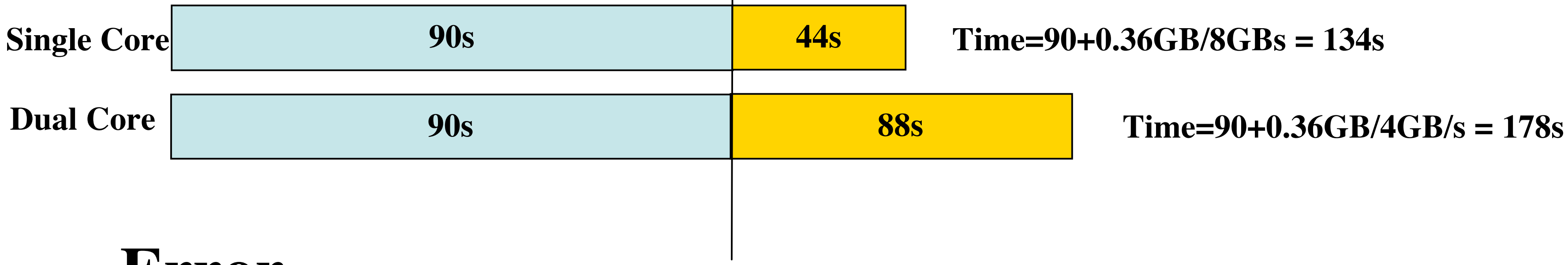


Single Core: Other Exec Time=90s | 70s@5GB/s — Time=160

Dual Core: 90s | 140s@2.5GB/s — Time=230s

Estimated Bytes Moved = 0.36 GB

Cray XT4 Opteron@2.6Ghz DDR2-667



Single Core: 90s | 44s — Time=90+0.36GB/8GBs = 134s

Dual Core: 90s | 88s — Time=90+0.36GB/4GB/s = 178s

Error

MILC Prediction for XT4 SC=134s

actual = 127s

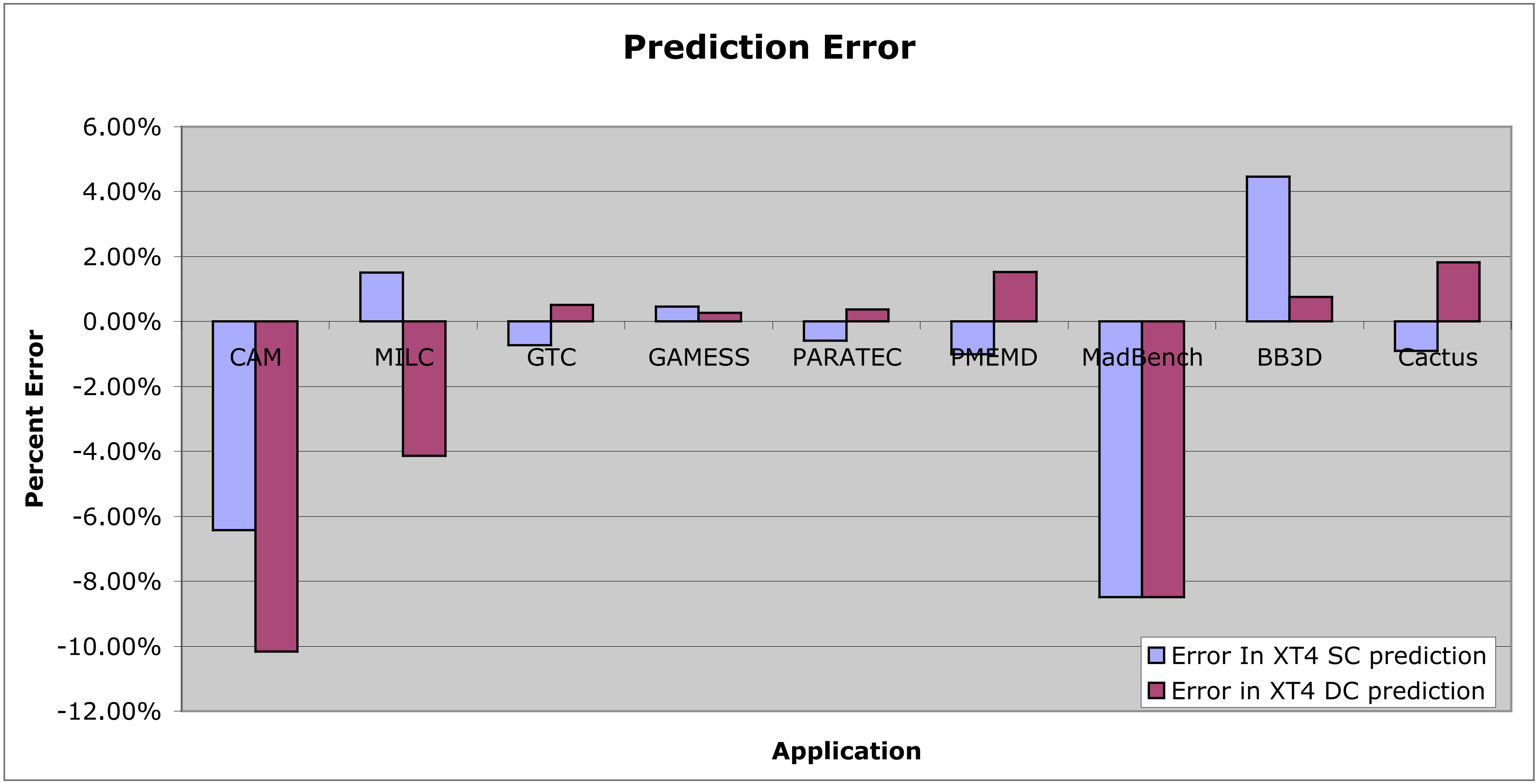
error = 5%

MILC Prediction for XT4 DC = 178s

actual = 181s

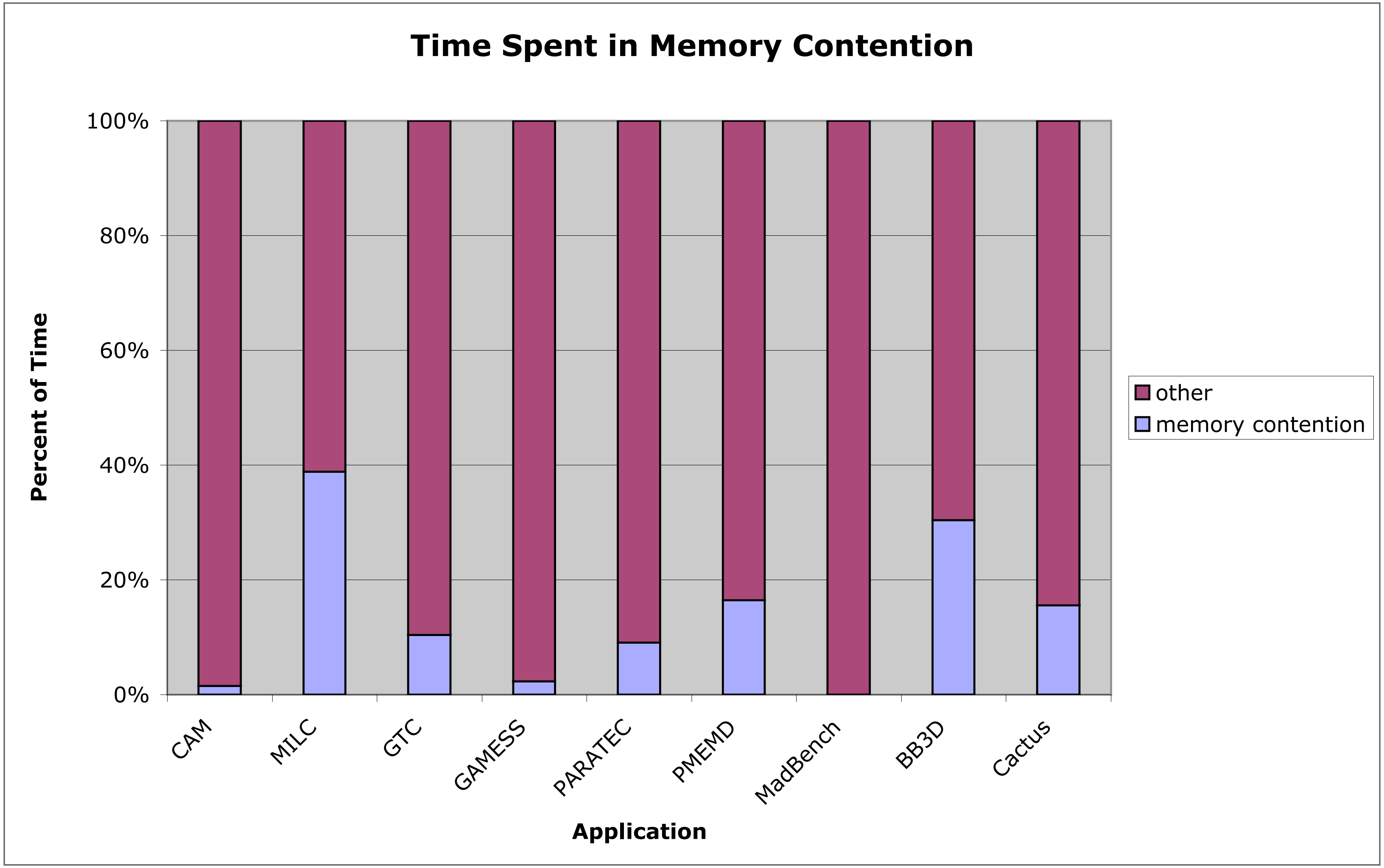
error = 1.5%

Testing the Performance Model



- **Reasonably accurate prediction of XT4 performance by plugging XT3 data into the analytic model**

Memory Contention



“Other” may include *anything* that isn’t memory bandwidth) (eg. latency stalls, Integer or FP arithmetic, I/O.)

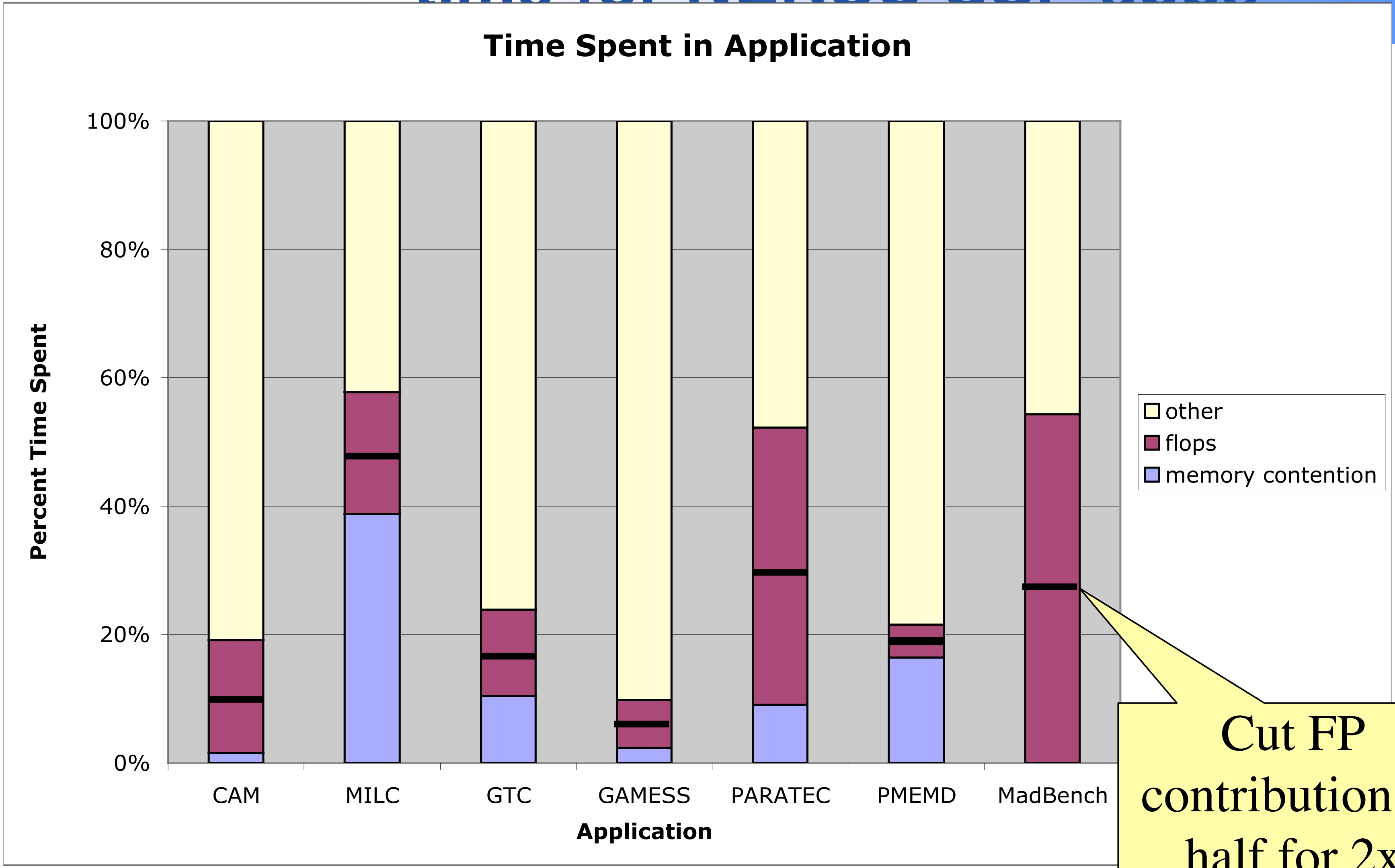
Conclusions

- Application codes see modest impact from move to dual-core (10.3% avg)
  - Exception is MILC, which is more dependent on memory bandwidth due to aggressive use of prefetch
  - Indicates most application performance bounded by other bottlenecks (memory latency stalls for instance)
- Most of the time is spent in “other” category
  - Could be integer address arithmetic
  - Could also be stalled on memory latency (could not launch enough concurrent memory requests to balance Little’s Law.
  - Could be Floating point performance
- Next generation x86 processors will double the FP execution rate
  - So, how much of “other” is FLOPs?

Refining Model for FLOPs

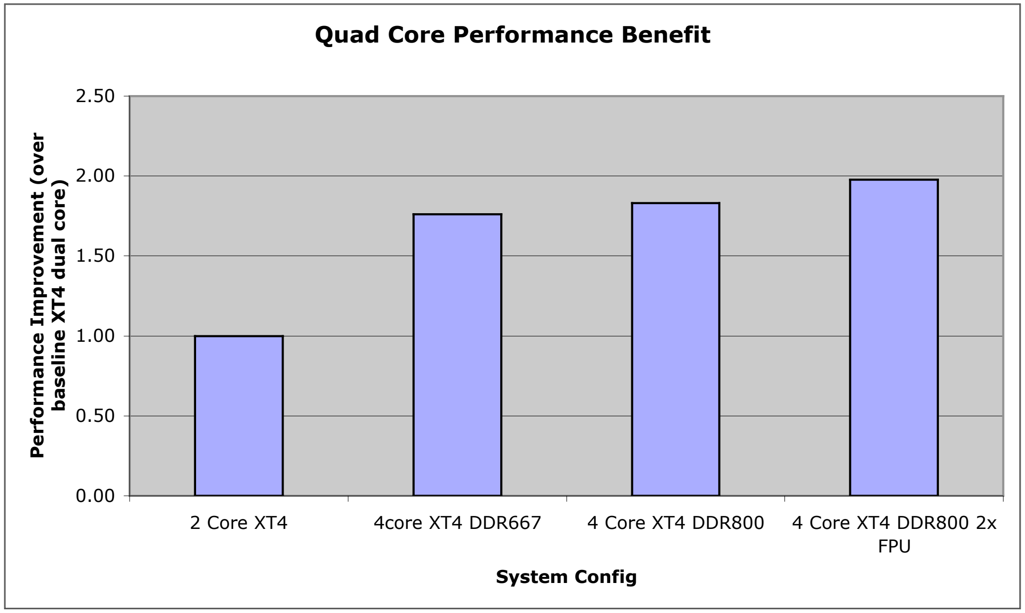
- Opteron Quad-core enhanced FPU
  - Each core has 2x the FLOP rate/cycle of the dual-core Rev. F implementation
  - Need to take into account how much performance may improve with 2x improvement in FLOP rate
- Approach
  - Count # flops performed per core
  - Estimate max total execution time spent in FLOPs assuming no overlap with other operations by dividing by peak flop rate on current FPU
  - Project for 2x faster FPU by halving that contribution to the overall exec time
- Result is the *maximum* possible improvement that could be derived from 2x FPU rate improvement

Contribution of FLOPs to exec time for NERSC SSP apps



Cut FP contribution in half for 2x faster FPU.

Quad Core Prediction



- **Conclusion: between 1.7x and 2.0x sustained performance improvement on NERSC SSP applications if we move from dual-core to quad-core**
  - **This is less than half the 4x peak performance improvement (but who cares about peak?)**
  - **But nearly 2x improvement is pretty good nonetheless (it matches the Moore’s law lithography improvement)**

Conclusions
(what I AM/AM_NOT saying!)

- **I am NOT saying that bandwidth is not important**
  - We *should* be bandwidth bound for many of these application
  - We *want* bandwidth to be the primary problem -- it is a failure that we are not bandwidth bound for most cases
- **I AM saying that we are *not* currently bandwidth bound because of inability to tolerate latency**
- **I am NOT saying that all applications are latency bound**
  - For example, elliptic solvers and BLAS-1 will suffer greatly from bandwidth starvation
  - I am saying that looking at the broader field of scientific applications will not benefit from a huge bandwidth boost on conventional microprocessor designs
  - I do not consider the status-quo acceptable -- our inability to benefit from improved memory bandwidth is an indication of serious deficiencies with existing CPU core design!

Conclusions
(what I AM/AM_NOT saying!)

- **I am NOT saying that the memory wall does not exist**
  - On the contrary, I am saying that it exists and we hit it a long time ago
  - We are still in denial about having hit this wall
  - The wall was softer than expected (memory molasses)
  - We are up to our necks in the molasses (and STILL in denial)
- **What I AM saying bandwidth *alone* will not rescue us from the molasses**
  - Throwing bandwidth at the problem sinks us deeper into the molasses
  - . . . . further unbalances Little's Law
  - . . . . Is simply throwing bandwidth away
  - We either need a new features core design that is better at hiding latency by increasing memory fetch concurrency (SW controlled memory), or a new approach to balancing Little's Law (manycore w/slower clocks or many HW threads for example)