Memory Performance and Optimization

- Locality
- Concurrency
- AMAT
- C-AMAT

Xian-He Sun
Illinois Institute of Technology
sun@iit.edu

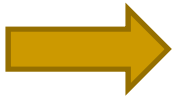
Data Access Optimizations

- Cache performance optimizations
  - Reducing cache miss rate
  - Reducing cache hit time
  - Increasing cache bandwidth
  - Reducing cache miss penalty
  - Prefetching

Reduce Miss Rate

- Code Transformations
- Utilize spatial and temporal locality
- Statement Reordering
  - Move one statement past another to increase temporal locality

```
a = x+y;          //S1
b = v+w;          //S2
c = x * 10;       //S3
```



```
a = x+y;          //S1
c = x * 10;       //S3
b = v+w;          //S2
```

Loop Unrolling

- Duplicating the body of a loop some number of times
- loop is executed less times, i.e. the test and branch code executed less
- Used in combination with other optimizations, such as software level data prefetching

```
for (i = 0; i < N; i++)
{
    A[i] = A[i] + B[i];
}
```

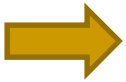
```
for (i = 0; i < N; i = i + 4)
{
    A[i] = A[i] + B[i];
    A[i + 1] = A[i + 1] + B[i + 1];
    A[i + 2] = A[i + 2] + B[i + 2];
    A[i + 3] = A[i + 3] + B[i + 3];
}

for (i = i - 4; i < N; i++)
    A[i] = A[i] + C[i];
```

Loop Fusion

- Two loops that have an identical index set may be merged together into one loop
- reduction in loop overhead and enhanced scheduling possibilities and the potential for reuse

```
for (i = 0; i < N; i++) {          //S1
    A[i] = B[i] + C[i];            //S2
}                                  //S3
for (i = 0; i < N; i++) {          //S4
    D[i] = B[i] + A[i];            //S5
}                                  //S6
```

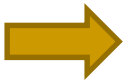


```
for (i = 0; i < N; i++) {
    A[i] = B[i] + C[i];            //S2
    D[i] = B[i] + A[i];            //S5
}
```

Loop Interchange

- Switching the nesting of loops
- moves parallelism into the inside of a set of nested loops
- If a loop-carried dependence is carried by an outer loop nesting, then the inner independent loops could be executed in parallel

```
for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
            A[i][j] = B[i][j];
      }
}
```



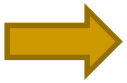
```
for (j = 0; j < M; j++){
      for (i = 0; i < N; i++) {
            A[i][j] = B[i][j];
      }
}
```

loop interchange would switch the loop nestings of the *i* loop and the *j* loop:

Strip Mining

- Strip mining converts a single-loop nest into a doubly nested loop
- Used for changing the locality characteristics of a loop, for vectorization, and for parallelization
- If N is large, after a certain point, each access may force a cache line fill

```
for (i = 0; i < N; i++) {
    a[i] = b[i] + c[i] + d[i] + e[i];
}
```

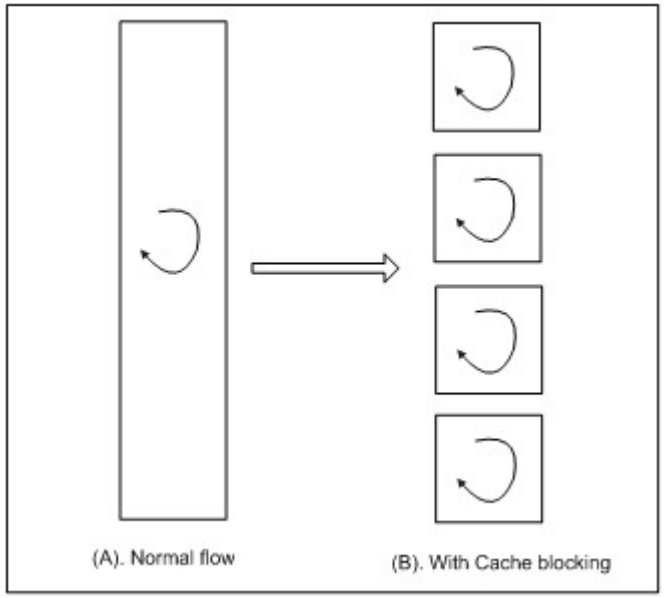


```
for (It = 0; It < N; It = It + S){
    for (i = It; i < min(N, It + S - 1); i++) {
        a[i] = b[i] + c[i] + d[i] + e[i];
    }
}
```

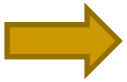
Loop Tiling

- Loop tiling partitions a loop's iteration space into smaller chunks or blocks, so as to help ensure data used in a loop stays in the cache until it is reused
- To "block" the execution of a loop for the purpose of better spatial locality
- By tiling the loops, we can take advantage of the locality of the fetched segments of the arrays as they fit into the cache

Loop Tiling



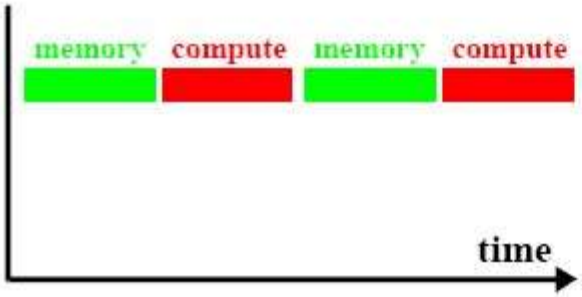

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        c[i] = c[i]+ a[i,j]*b[j];
```

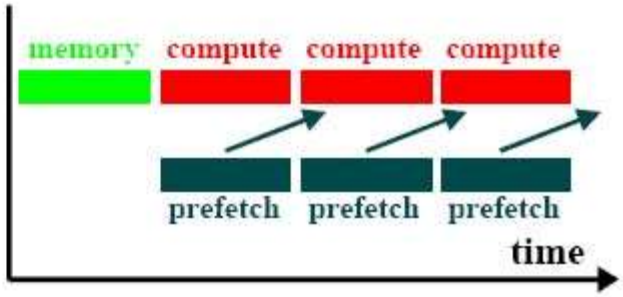


```
for (i=0; i<N; i+=2)
    for (j=0; j<N; j+=2)
        for (ii=i; ii<min(i+2,N); ii++)
            for (jj=j; jj<min(j+2,N); jj++)
                c[ii] =c[ii]+ a[ii,jj]*b[jj];
```

Prefetching

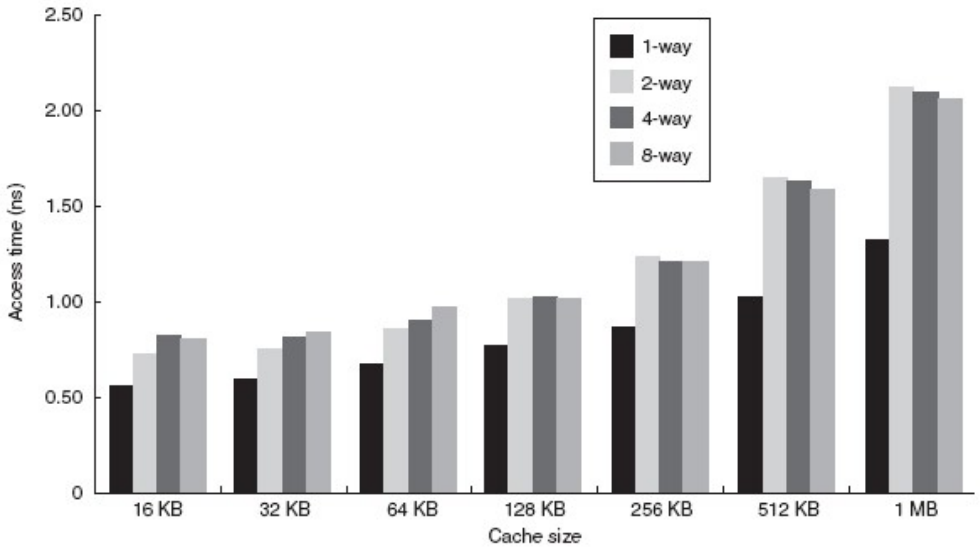
- Move data closer to the processor before it is demanded
- Prefetch data as close as possible to the processor in the memory hierarchy
- Key to prefetching
  - When should prefetching occur?
  - What data should be prefetched?





Reducing cache hit time

- Small and Simple caches
- L1 cache is usually kept small



These data are based on the CACTI model 4 4.0 by Tarjan, Thoziyoor, and Jouppi [2006]. They assumed 90 nm feature size, a single bank, and 64-byte blocks. The median ratios of access time relative to the direct-mapped caches are 1.32, 1.39, and 1.43 for 2-way, 4-way, and 8-way associative caches, respectively.

Increasing cache bandwidth

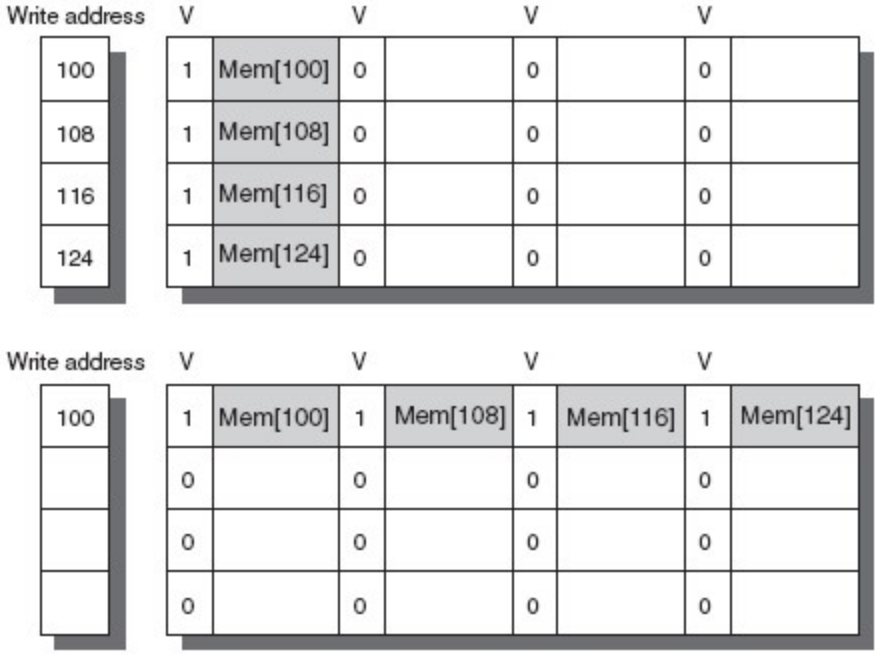
- Pipelined cache access
  - Effective latency of L1 cache hit is multiple cycles
  - Fast clock cycle time and high bandwidth, but slow hits
- Non-blocking caches
  - Fetching instructions from instruction cache while waiting on data cache misses
- Multi-banked caches
  - Dividing cache into independent blocks to support simultaneous access

Reducing Miss Penalty

- Critical word first and Early restart
  - Request the missed word first from memory and send it to the processor first as soon as it arrives
  - Fetch the block, but send the requested word of the block first to the processor
  - Only works for cache memories with large cache lines
  - What if there is spatial locality?

Reducing Miss Penalty

- Write merging for write-through caches
- Merge multiple stores into one cache block if they are contiguous

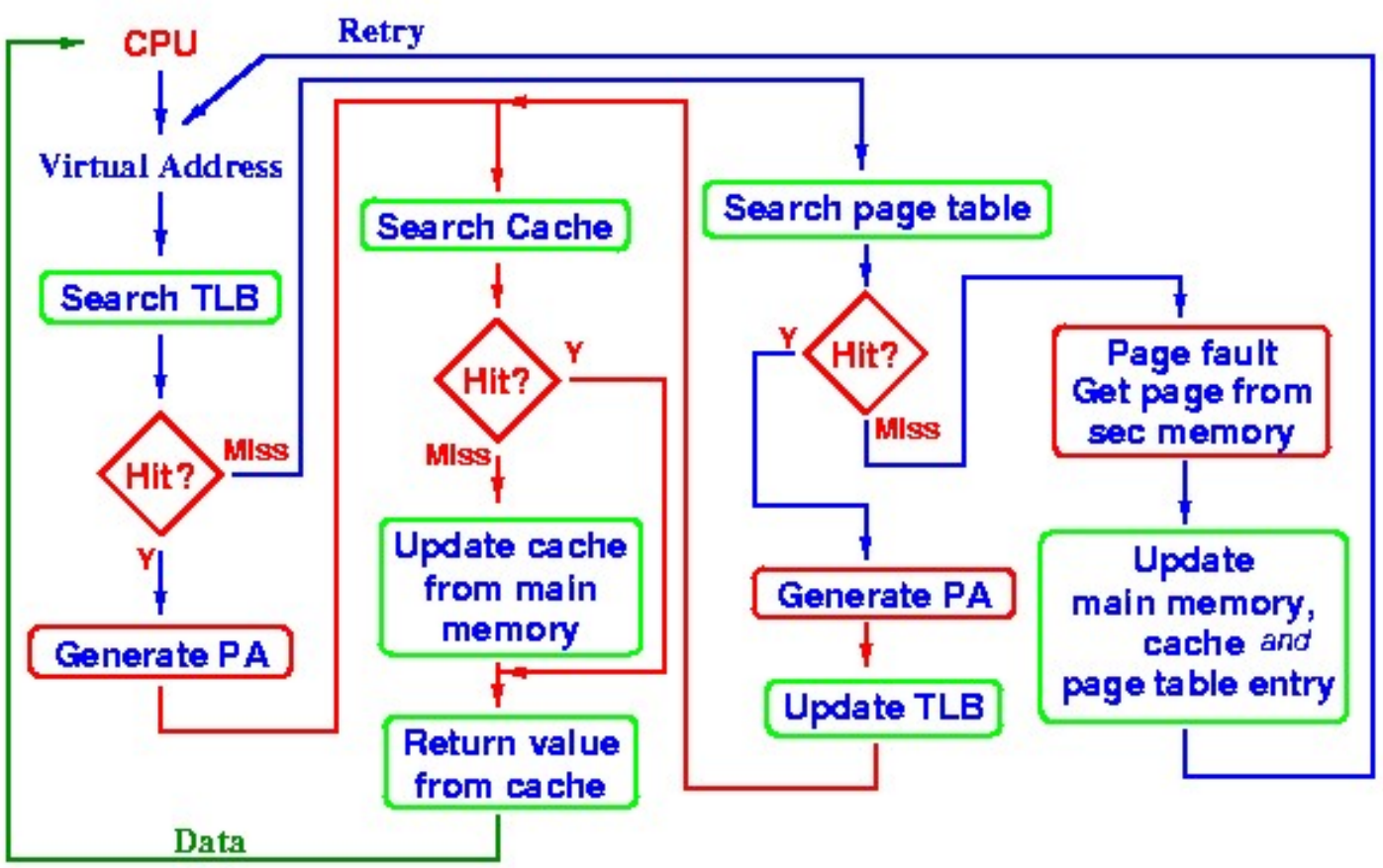


Summary of cache optimizations

Technique	Hit time	Band-width	Miss pen-alty	Miss rate	Hardware cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of Opteron and Niagara
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data			+	+	2 instr., 3 data	Many prefetch instructions; Opteron and Pentium 4 prefetch data
Compiler-controlled prefetching			+	+	3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Source: Computer Architecture – A Quantitative Approach, 4th edition, David Patterson

Operation of Memory Hierarchy

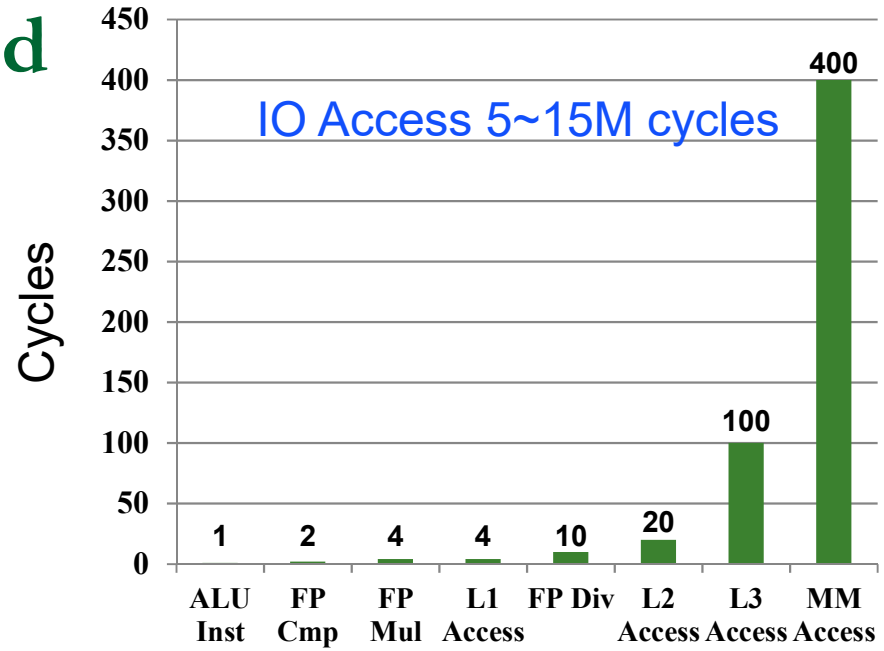


Assumption of Current Solutions

- Memory Hierarchy: Locality
- Concurrence: Data access pattern
  - Data stream

Extremely Unbalanced Operation Latency

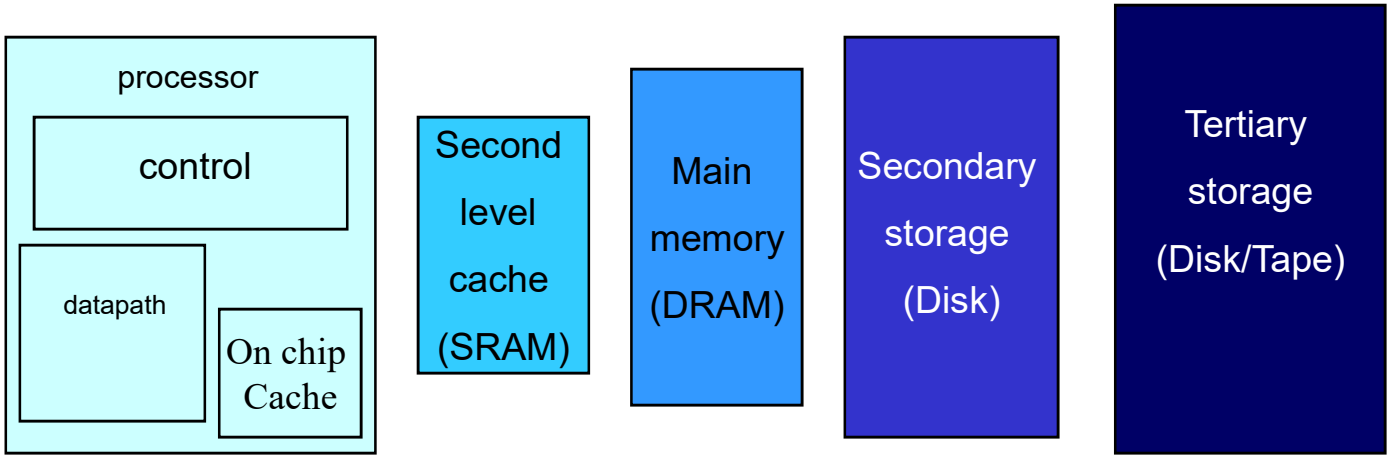
Performances vary largely



Levels of the Memory Hierarchy

Deeper levels of cache memory

Large memories are slow, fast memories are small and expensive.



See http://en.wikipedia.org/wiki/Orders_of_magnitude_(time)

The Role of Memory

- SPEC CPU 2000 Codes spend 65% TTS in memory stalls
- Memory Gap Grows 50% per year

- Example: 1 GHz CPU + 40ns DRAM access
  - With 2.5 IPC ideally could execute 100 instructions per miss
  - Change to 2.8 GHz + 30 ns DRAM
    - Ideally could execute 210 instructions per miss
    - Thousands of instructions soon (comparable to context switch)
  - I/O is much, much worse (millions of cycles lost)

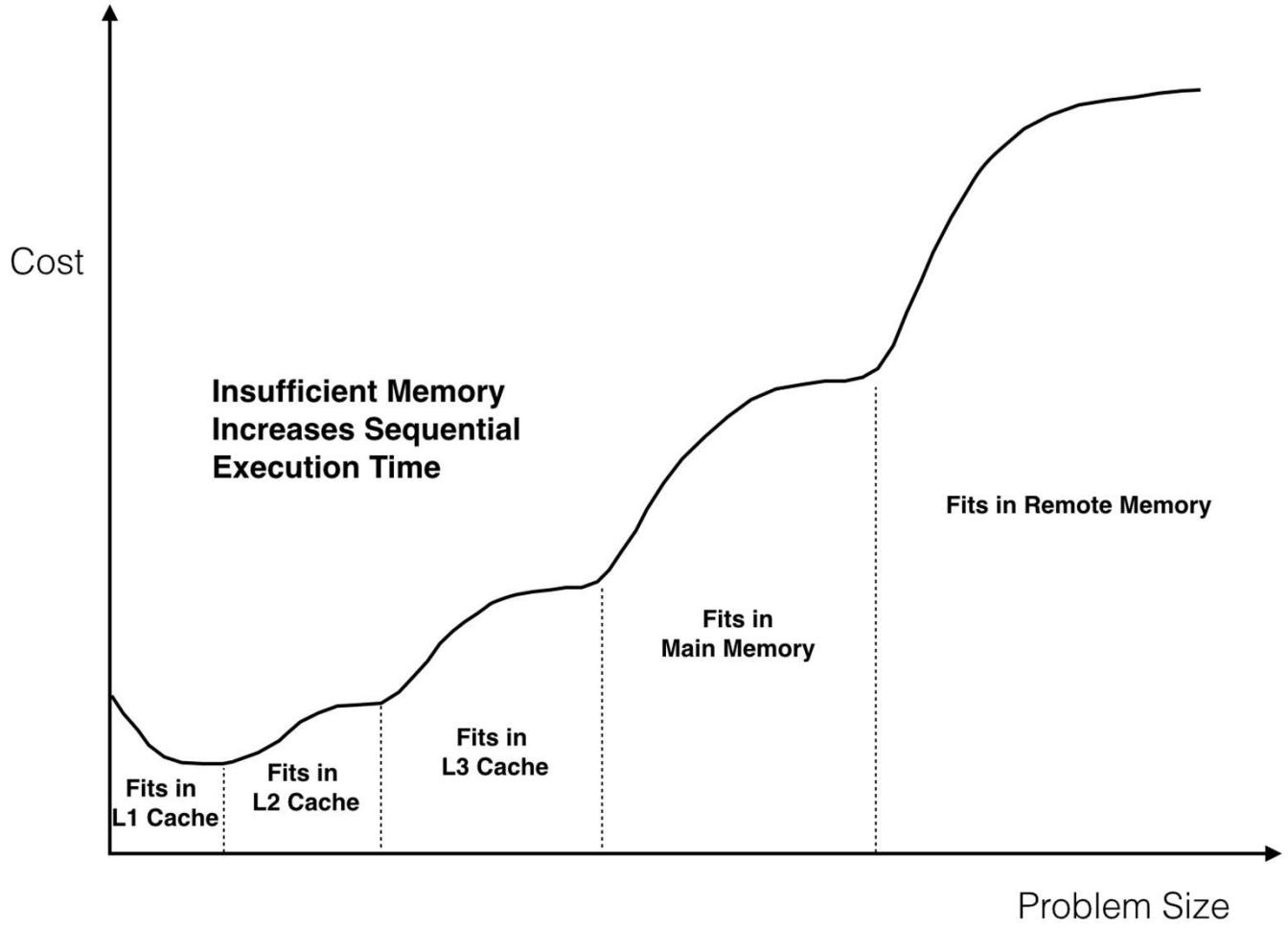


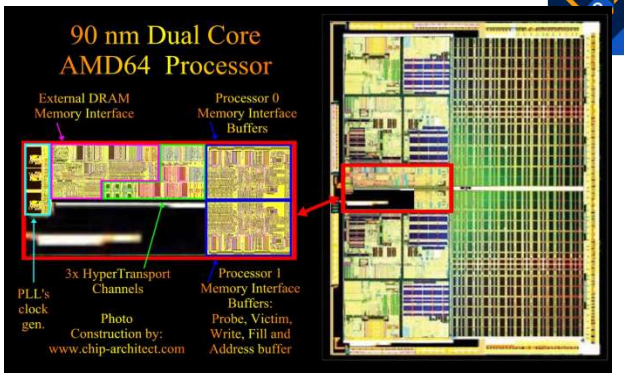
Fig 1. Cost variation pattern

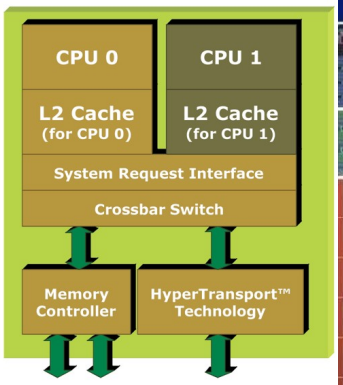
In the Era of Multicore

- Maximizing memory (or shared cache) hierarchy
- Minimizing memory traffic
  - Fewer cache misses are helpful
  - Loop optimizations are useful for multicore processors as well
- Multicore specific optimizations
  - Better task partitioning and mapping
  - Memory access scheduling to avoid data access contention
  - Minimize inter-socket communication
  - Aggressive prefetching engines

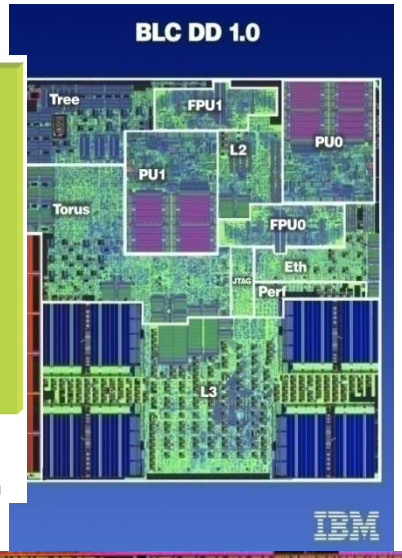
Multi-Core

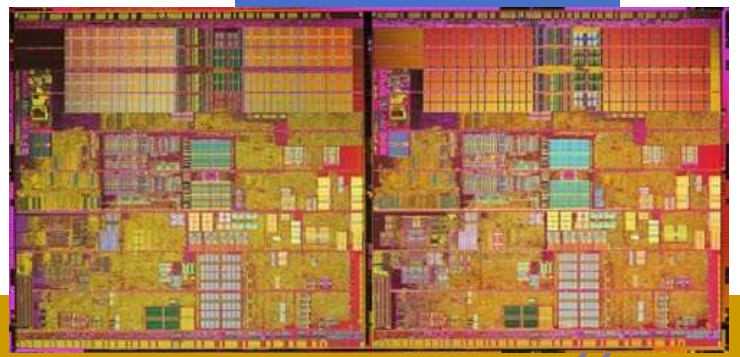
- Motivation for Multi-Core
  - Exploits improved feature-size and density
  - Increases functional units per chip (spatial efficiency)
  - Limits energy consumption per operation
  - Constrains growth in processor complexity
- Challenges resulting from multi-core
  - **Aggravates memory wall**
    - Memory bandwidth
      - Way to get data out of memory banks
      - Way to get data into multi-core processor array
    - Memory latency
    - **Fragments L3 cache**
  - **Relies on effective exploitation of multiple-thread parallelism**
    - **Need for parallel computing model and parallel programming model**
  - Pins become strangle point
    - Rate of pin growth projected to slow and flatten
    - Rate of bandwidth per pin (pair) projected to grow slowly
  - Requires mechanisms for efficient inter-processor coordination
    - Synchronization
    - Mutual exclusion
    - Context switching





AMD Athlon™ 64 X2
Dual-Core Processor Design





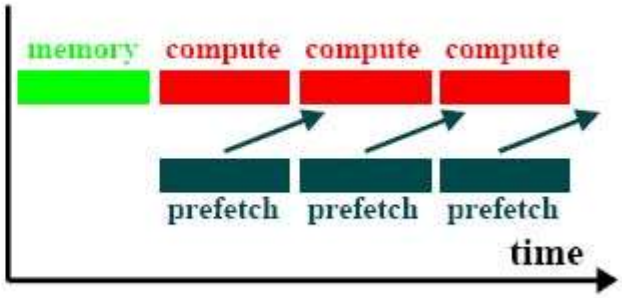
Increasing cache bandwidth

- Pipelined cache access
  - Effective latency of L1 cache hit is multiple cycles
  - Fast clock cycle time and high bandwidth, but slow hits
- Non-blocking caches
  - Fetching instructions from instruction cache while waiting on data cache misses
- Multi-banked caches
  - Dividing cache into independent blocks to support simultaneous access

These are concurrent data accesses

Prefetching

- Move data closer to the processor before it is demanded
- Prefetch data as close as possible to the processor in the memory hierarchy
- Key to prefetching
  - When should prefetching occur?
  - What data should be prefetched?

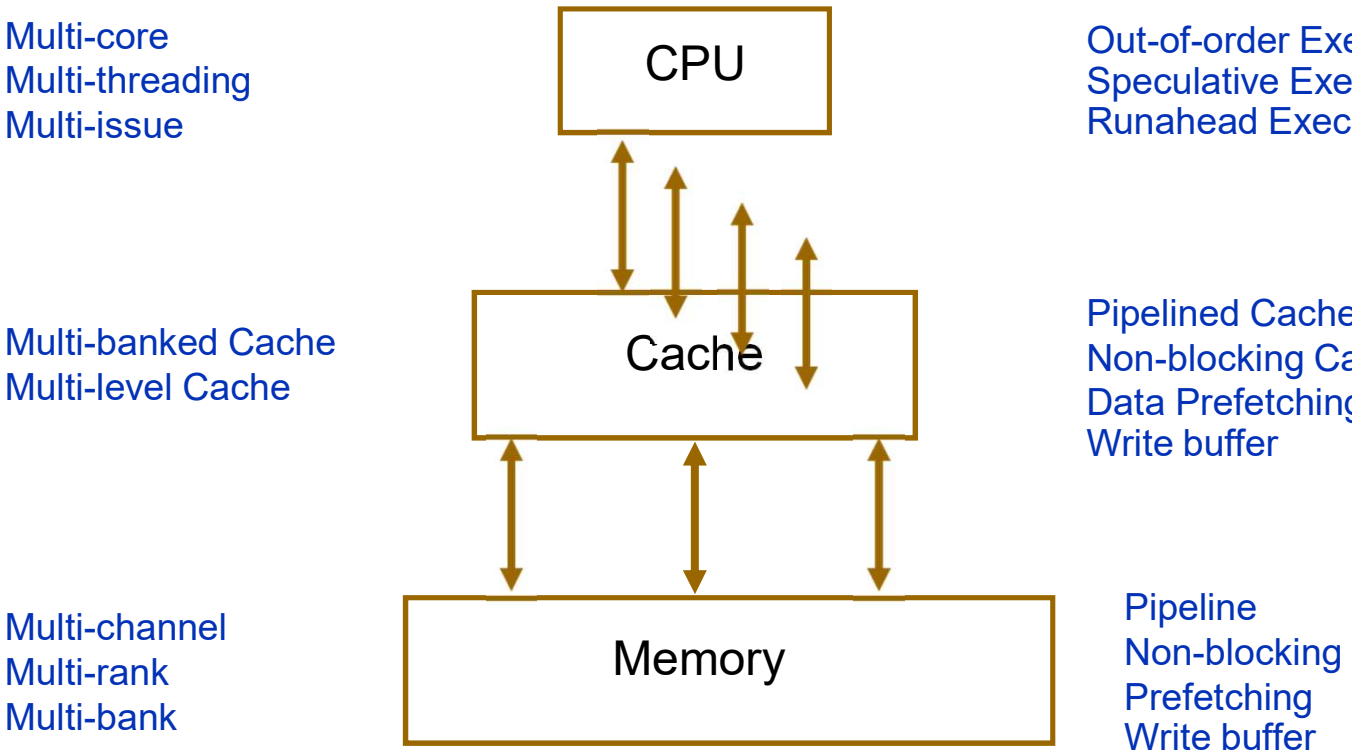


This is concurrent data access

Processor/Core Micro Architecture

- Execution Pipeline
  - Stages of functionality to process issued instructions
  - Hazards are conflicts with continued execution
  - Forwarding supports closely associated operations exhibiting precedence constarints
- Out of Order Execution
  - Uses reservation stations
  - Hides some core latencies and provide fine grain asynchronous operation supporting concurrency
- Branch Prediction
  - Permits computation to proceed at a conditional branch point prior to resolving predicate value
  - Overlaps follow-on computation with predicate resolution
  - Requires roll-back or equivalent to correct false guesses
  - Sometimes follows both paths, and several deep

Solution: Memory Hierarchy and Concurrency



Input-Output (I/O)

Parallel File System

Disks

Improve via Data Access Concurrence

- The complexity of CPU Design
  - Out-of-order Execution
  - Multithreading technology
  - Speculation mechanisms

- The complexity of Memory Design
  - Advanced Cache Technologies
  - Allow tens or hundreds of cache accesses to overlap with each other
  - Processor continue execution instructions under multiple cache misses

Concurrence is more than bandwidth

- Pipelined cache access
  - Effective latency of L1 cache hit is multiple cycles
  - Fast clock cycle time and high bandwidth, but slow hits
- Non-blocking caches
  - Fetching instructions from instruction cache while waiting on data cache misses
- Multi-banked caches
  - Dividing cache into independent blocks to support simultaneous access

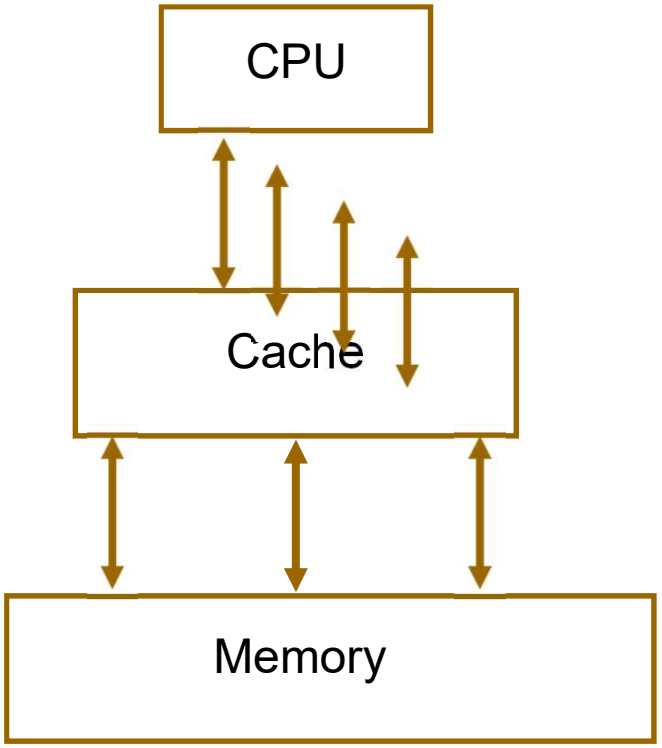
Concurrence can overlap computing and data movement
Hardware concurrence is not the achieved concurrency

How to measure and represent the concurrence contribution?

Also: (mostly hiding) Memory Concurrency

Multi-core
Multi-threading
Multi-issue

Out-of-order Execution
Speculative Execution
Runahead Execution



Multi-banked Cache
Multi-level Cache

Pipelined Cache
Non-blocking Cache
Data Prefetching
Write buffer

Multi-channel
Multi-rank
Multi-bank

Pipeline
Non-blocking
Prefetching
Write buffer

Input-Output (I/O)

Parallel File System

Disks

Existing Memory Metrics

- Miss Rate(MR)
  - {the number of miss memory accesses} over {the number of total memory accesses}
- Misses Per Kilo-Instructions(MPKI)
  - {the number of miss memory accesses} over {the number of total committed Instructions × 1000}
- Average Miss Penalty(AMP)
  - {the summary of single miss latency} over {the number of miss memory accesses}
- Average Memory Access Time (AMAT)
  - **AMAT = Hit time + MR×AMP**
- Flaw of Existing Metrics
  - Focus on a single component or
  - A single memory access

Missing memory parallelism/concurrency

The Introduction of APC

- Access Per Cycle (APC)
- APC is measured as the number of memory accesses per cycle
  - Measures the overall memory system performance
  - Each memory level has its own APC value
  - Dominating overall CPU performance
- Benefits of APC
  - Separate memory evaluation from CPU evaluation
  - A better understanding of memory system as a whole
  - A better understanding of the match between computing capacity and memory system performance

APC: measure from the memory side

- Access Per memory active Cycle (APC)
  - APC = A/T
- APC is measured as the number of memory accesses per memory active cycle or Access Per Memory Active Cycle (APMAC)
- Benefits of APC (APMAC)
  - Separate memory evaluation from CPU evaluation
  - Each memory level has its own APC value

X.-H. Sun and D. Wang, "APC: A Performance Metric of Memory Systems", ACM SIGMETRICS Performance Evaluation Review, Volume 40 , Issue 2, 2012.

APC Measurement

- Measure **T** based on ***memory (active) cycle***
  - Can be measured for ***each layer of a memory hierarchy***
- Measure **A** based on the ***overlapping mode***
  - When there are several memory accesses co-existing during the same clock cycle, *T* only increases by one

- Difficulty in measure memory cycle A & T
  - Hundreds of memory accesses co-exist the memory system
- Hardware cost: one bit
- **Concurrence** and **Data-Centric** *(memory active cycles)* view

D. Wang, X.-H. Sun "Memory Access Cycle and the Measurement of Memory Systems", IEEE Transactions on Computers, vol. 63, no. 7, pp. 1626-1639, July.2014

APC in Detail

- APC is the overall memory accesses requested at a certain memory level (i.e. L1, L2, L3, Main Memory) divided by the total number of memory access cycles at that level
  - APC = A/T
  - Different level has different APC
    - APC_D L1 Data Cache
    - APC_I L1 Instruction Cache
    - APC_M Main Memory
- APC performance is hierarchical

Validation Testing Methodology

- System performance is the ultimate interest
- A good memory metric should influence system performance directly
- Use IPC (Instruction Per Cycle) as the system performance
- Use Correlation Coefficient to measure the correlation
  - Better correlation, better metric

Experiment Methodology

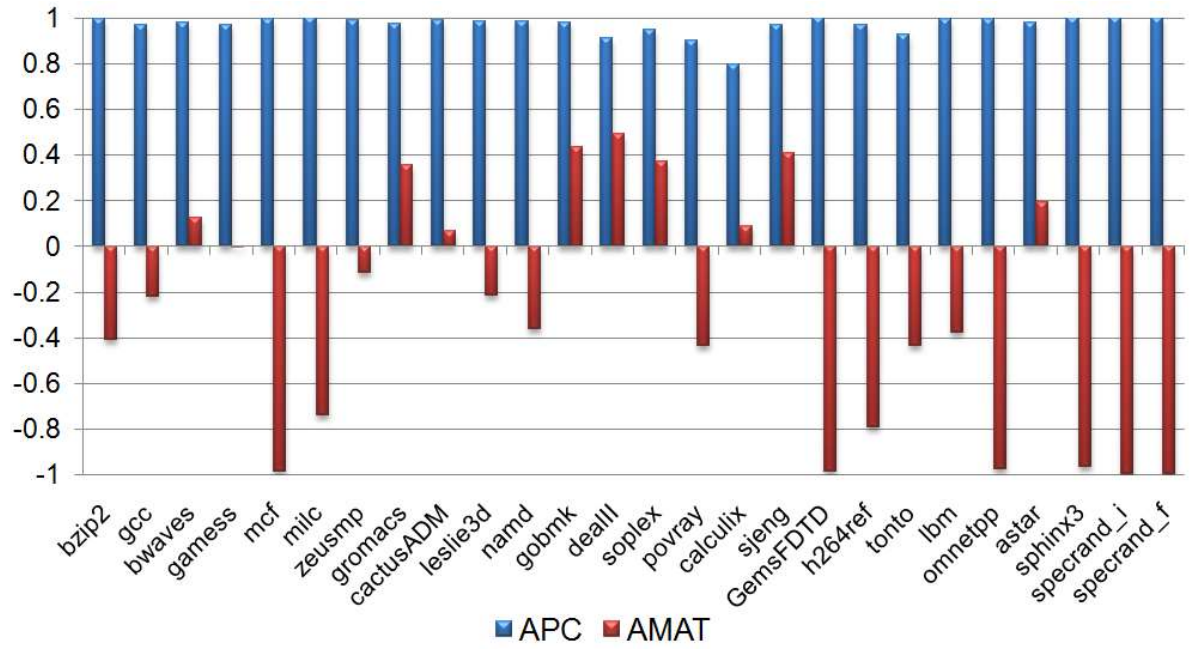
- Simulator
  - GEM5
- Benchmarks
  - 29 benchmarks from SPEC CPU2006 suite
- For each benchmark, 10 million instructions were simulated to collect statistics
- Average values of the correspondent memory metrics are shown
- A good memory metric should matches the actual design choices for modern processors

Default configuration

Device/parameter	Value
Processor Function units ROB (reorder buffer) and LSQ (load/ store queue) size	1 core, 4 GHz, 4-issue width 6 IntALU: 1 cycle; 1 IntMul: 3 cycles; 2 FPAdd: 2 cycles; 1 FPCmp: 2 cycles; 1 FPCvt: 2 cycles; 1 FPMul: 4 cycles; 1 FPDiv: 12 cycles ROB 64, LQ 48, SQ 24
L1 caches	32 KB inst/32KB data, 2-way, 64B line, 4-cycle hit latency, inst/4-cycle data, ICache 8 MSHR entry, DCache 8 MSHR entry
L2 cache	512 KB, 16-way, 64B line, 24-cycle hit latency, 16 MSHR entry
DRAM latency/width	240-cycle access latency/64 bits

Default processor and cache configuration parameters for simulated testing of C-AMAT

APC & IPC: Changing Cache Parallelism



- Changing the number of MSHR entries (1→2→10→16)
- APC still has the dominant correlation, with average value of 0.9656
- AMAT does not correlate with IPC for most applications
  - APC record the CPU blocked cycles by MSHR cycles
  - AMAT cannot records block cycles, it only measure the issued memory requests

D. Wang, X.-H. Sun "Memory Access Cycle and the Measurement of Memory Systems", IEEE Transactions on Computers, vol. 63, no. 7, pp. 1626-1639, July.2014

Exhausted Testing

- With different benchmarks, and with different configurations
- With advanced cache technologies
  - Non-block cache
  - Pipelined cache
  - Multi-port cache
  - Hardware prefetcher
- With single core or multicore
- **APC always has the highest CC values among all the memory metrics**

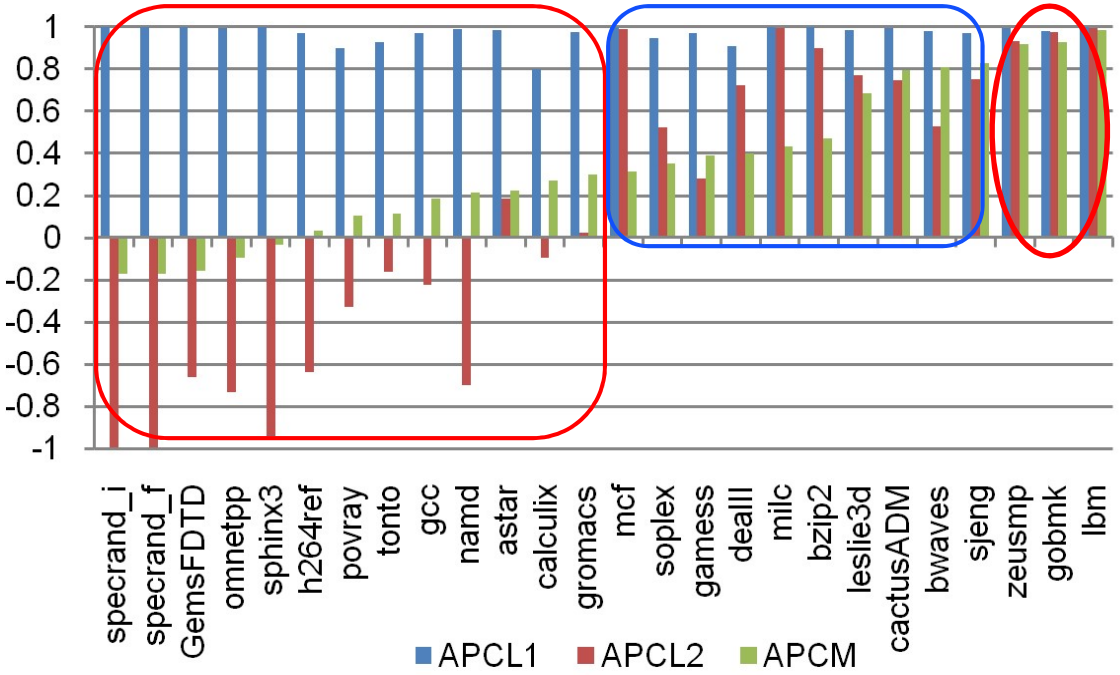
A Definition of Data Intensiveness

- The IPC and APC correlation value provides a quantitative definition of data intensive
- Use the correlation value of **APC_M** to quantify the degree of data intensive
  - Do not count data re-use as part of data-intensiveness unless it has to be read from main memory again
  - Assuming the "memory-wall" problem is actually due to the slow speed of main memory
  - Could define differently for small kernel application or off-core application

Definition

$$coe(APC_M, IPC) \geq 0.9$$

Data-intensive Definition



- The correlation value of APC_M are divided into three intervals, that is (-1, 0.3), [0.3, 0.9), [0.9, 1)
- Reason for picking 0.9 as the threshold

  According to mathematical definition of correlation coefficient

When CC >= 0.9, then the two variables have a dominant relation

APC's Correlation on POWER7

- IBM POWER7 provides performance counters to measure L1 Data Cache APC
  - PM_LSU_LMQ_SRQ_EMPTY_CYC
    - Number of cycles that Load/Store unit is empty
    - L1 DCache Cycles = CPU Cycles – LSU Empty Cycles
  - PM_LD_REF_L1
    - Accesses to L1 Data Cache memory
- APC_D = DCache Accesses / DCache Cycles
- Three applications to measure correlations
  - Vector multiplication CC value **0.9598**
  - Matrix multiplication CC value **0.9998**
  - Linpack CC value **0.9993**

Results

- High Performance Linpack compiled with different gcc optimization options

Size	O1		O2		O3	
Matrix / Block	IPC	APC	IPC	APC	IPC	APC
1000 / 16	1.20895	**0.34536**	1.57888	**0.40663**	1.65207	**0.42341**
1000 / 24	1.19629	**0.34313**	1.54968	**0.40455**	1.6231	**0.42156**
1000 / 48	1.20624	**0.3424**	1.58018	**0.4053**	1.66118	**0.42551**

Concurrent-AMAT: step to optimization

- The traditional AMAT(Average Memory Access Time) :

$$AMAT = HitCycle + MR \times AMP$$

- MR is the miss rate of cache accesses; and AMP is the average miss penalty

- **Concurrent-AMAT (C-AMAT)**:

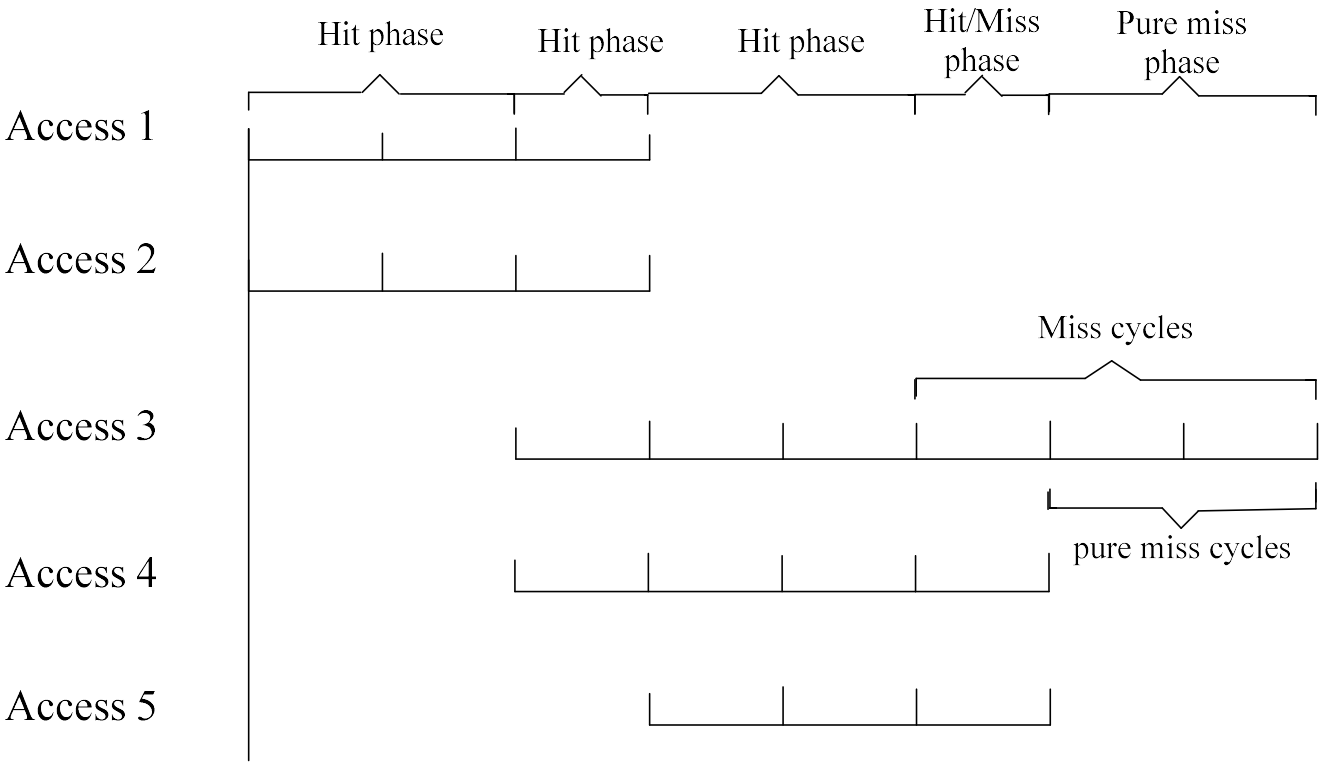
$$C\text{-}AMAT = HitCycle/C_H + pMR \times pAMP/C_M = 1/APC$$

- C_H is the hit concurrency; C_M is the *pure* miss concurrency
- *p*MR and *p*AMP are *pure* miss rate and average *pure* miss penalty
- A pure miss is a miss containing at least one cycle which does not have any hit activity

X.-H. Sun and D. Wang, "Concurrent Average Memory Access Time", in ***IEEE Computers***, vol. 47, no. 5, pp. 74-80,May 2014.(IIT Technical Report, IIT/CS-SCS-2012-05)

Concurrence & Pure-miss

- Miss is not important (Pure miss is)
- The penalty is due to pure miss



Conventional AMAT

- The traditional AMAT(Average Memory Access Time) :

$$AMAT = HitCycle + MR \times AMP$$

- MR is the miss rate of cache accesses; and AMP is the average miss penalty

AMAT is **Recursive**

- $AMAT = HitCycle + MR \times (H_2 + MR_2 \times AMP_2)$
- $AMAT = HitCycle + MR \times (H_2 + MR_2 \times (H_3 + MR_3 \times AMP_3))$
- Etc.

Recursive in Memory Hierarchy

- AMAT is recursive
  - $AMAT = HitCycle_1 + MR_1 \times AMP_1$

  Where $AMP_1 = (HitCycle_2 + MR_2 \times AMP_2)$

- C-AMAT is also recursive

$$C\text{-}AMAT_1 = \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times C\text{-}AMAT_2$$

Where

$$C\text{-}AMAT_2 = \frac{H_2}{C_{H_2}} + pMR_2 \times \frac{pAMP_2}{C_{M_2}}$$

$$\kappa_1 = \frac{pMR_1}{MR_1} \times \frac{pAMP_1}{AMP_1} \times \frac{C_{m_1}}{C_{M_1}}$$

With Clear Physical Meaning

X.-H. Sun, "Concurrent-AMAT: a mathematical model for Big Data access," HPC-Magazine, May 12, 2014

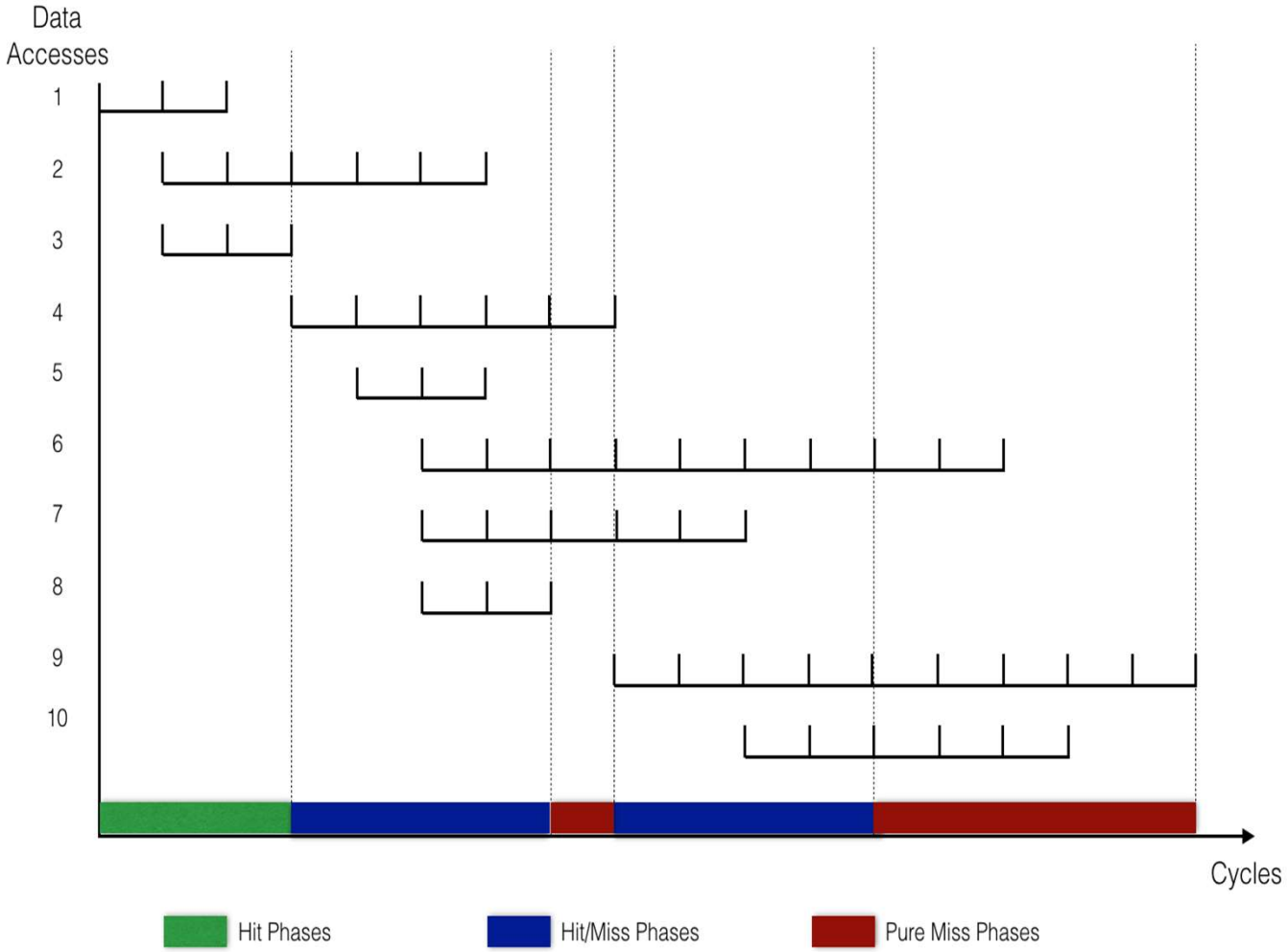
Physical meaning of κ in C – AMAT

- **κ Example:**

In the right figure, the red bar is pure miss cycles, so $T_{C-pM} = 6$;

the red bar and the blue bar is the miss cycles, so $T_{C-m} = 14$;

then $\kappa = 6/14$。



Physical meaning of κ in C – AMAT

- ***κ* has clear and exact physical meaning**
  - ***κ*** is the ratio of pure (memory active) miss cycles and (memory active) miss cycles
  - Recall pure miss cause CPU stall, so ***κ*** smaller is better
  - ***κ*** smaller, C-AMAT is smaller