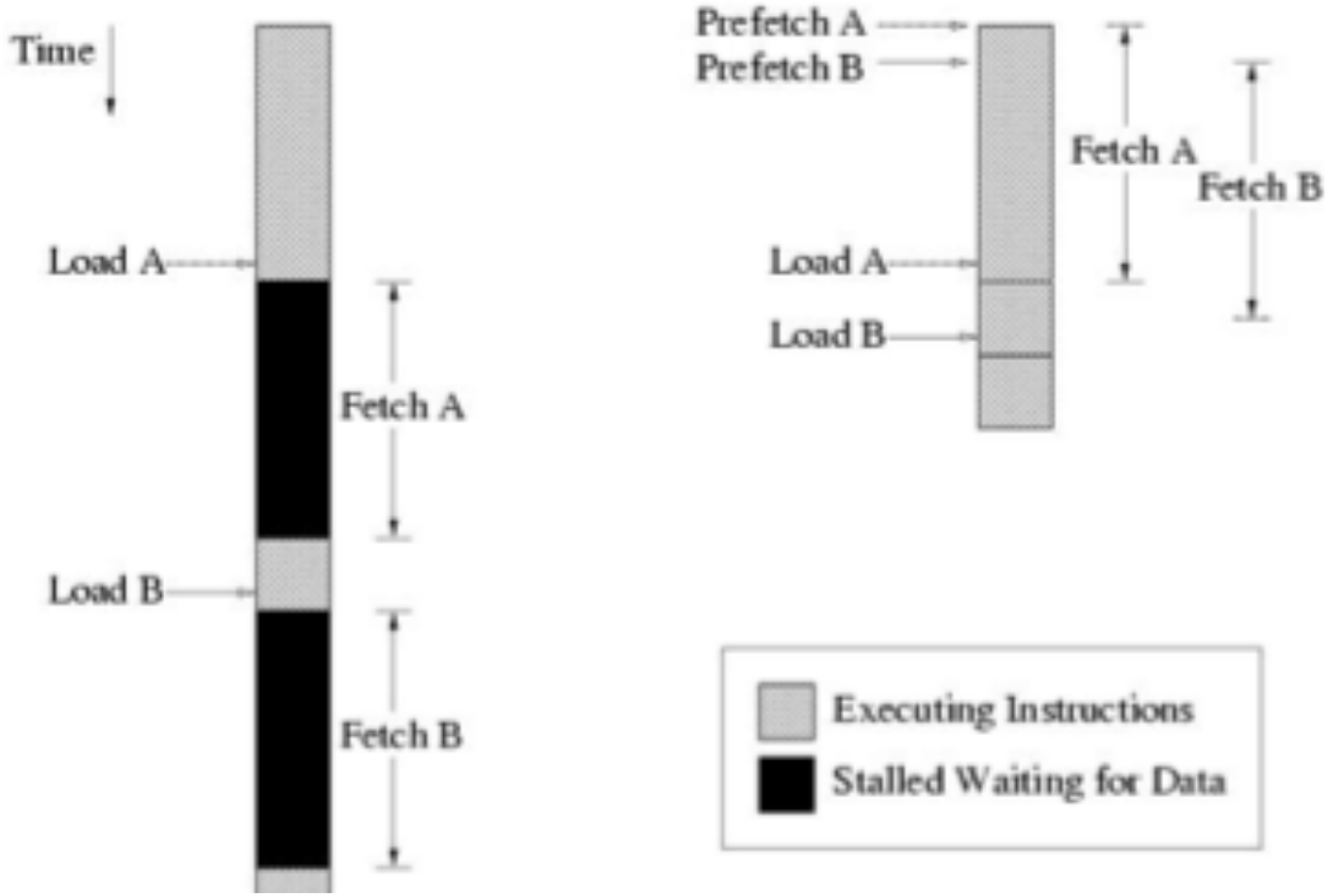
Prefetching techniques - Global History Buffers and Local History Buffers

Tham H., Shweta W.

Fall 2013

Why Prefetching?



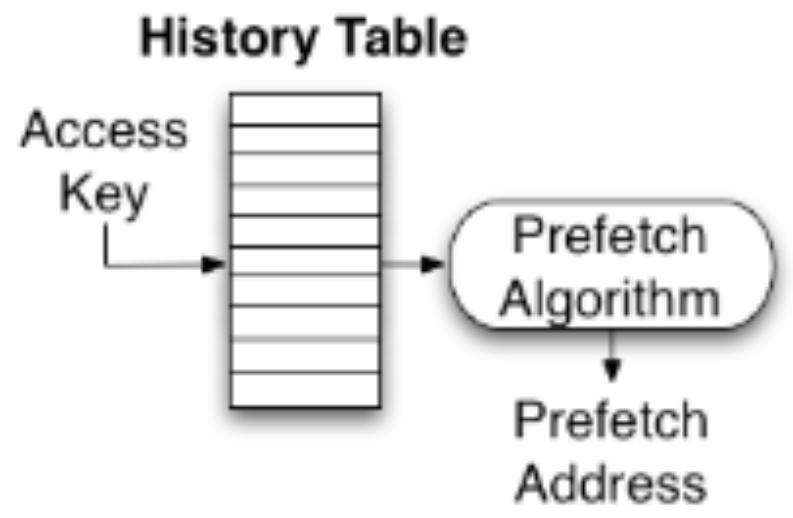
Global History Buffer

- Introduction: Global History Buffer that holds the most recent miss addresses in FIFO order.

- Purposes:
  - FIFO history buffer-> improve accuracy of correlation prefetching by eliminating stale data from the table
  - Cache miss history-> design more effective prefetching methods.

- Methods:
  - Stride Prefetching
  - Markov Prefetching
  - Distance Prefetching

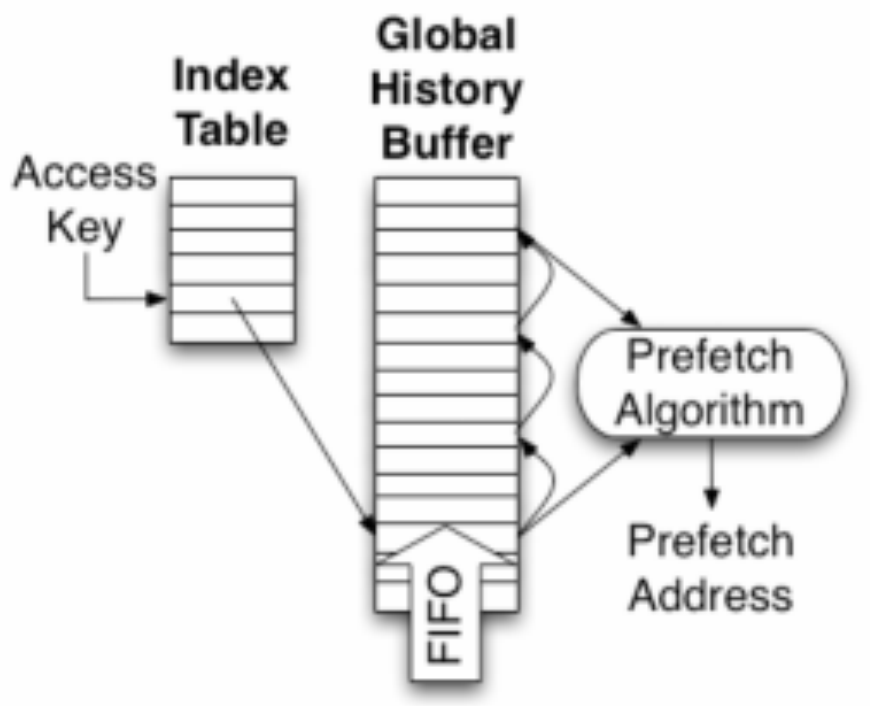
Basic Prefetch Table



Prefetch degree is maximum number of cache lines prefetched in response to a single prefetch request.

History table is accessed with a key e.g it represents history in the distant past which no longer reflects current conditions.

Global History Buffer Prefetch Structure

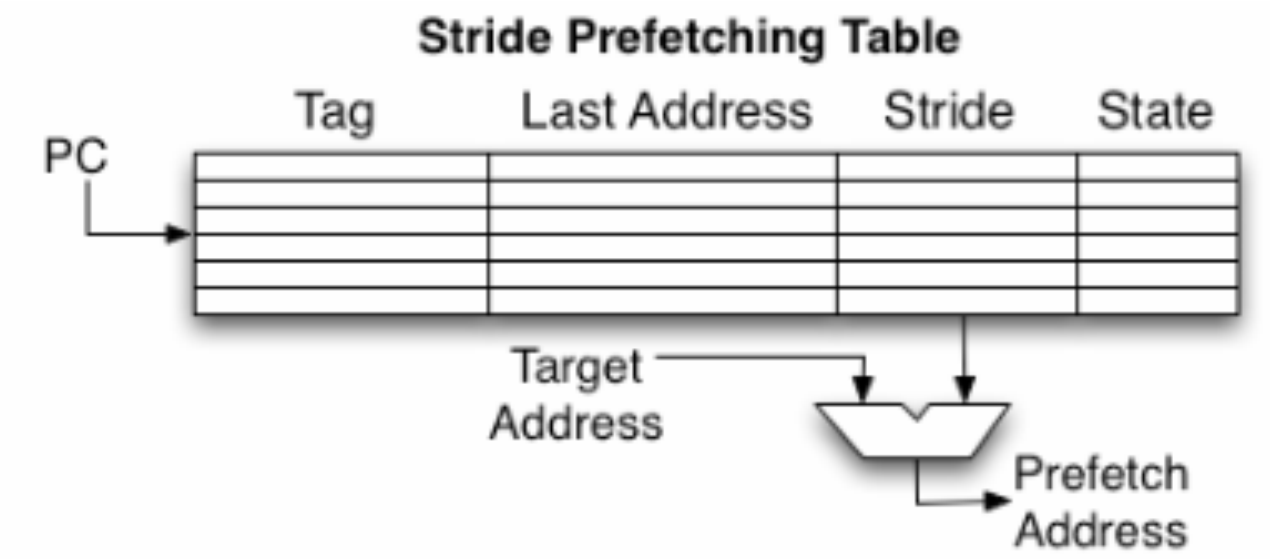


Global History Buffer holds all global miss addresses being placed in the table at the bottom and remove from the top.

GHB history information is maintained in linked list, which are accessed indirectly via a hash table.

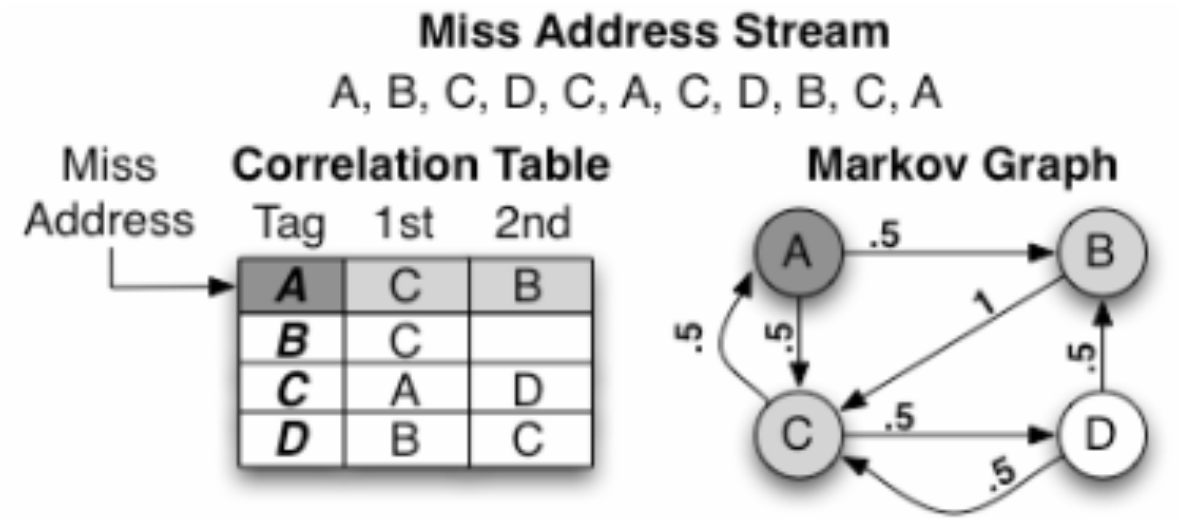
+ reduced stale history data
+ allows a more accurate re-construction of the history of access pattern

Arbitrary Stride Prefetching Table



PC (program counter) of a load instruction indexes the table.
Last address (to allow computation of the next local stride).
State: the stability of the load's recent stride behavior.

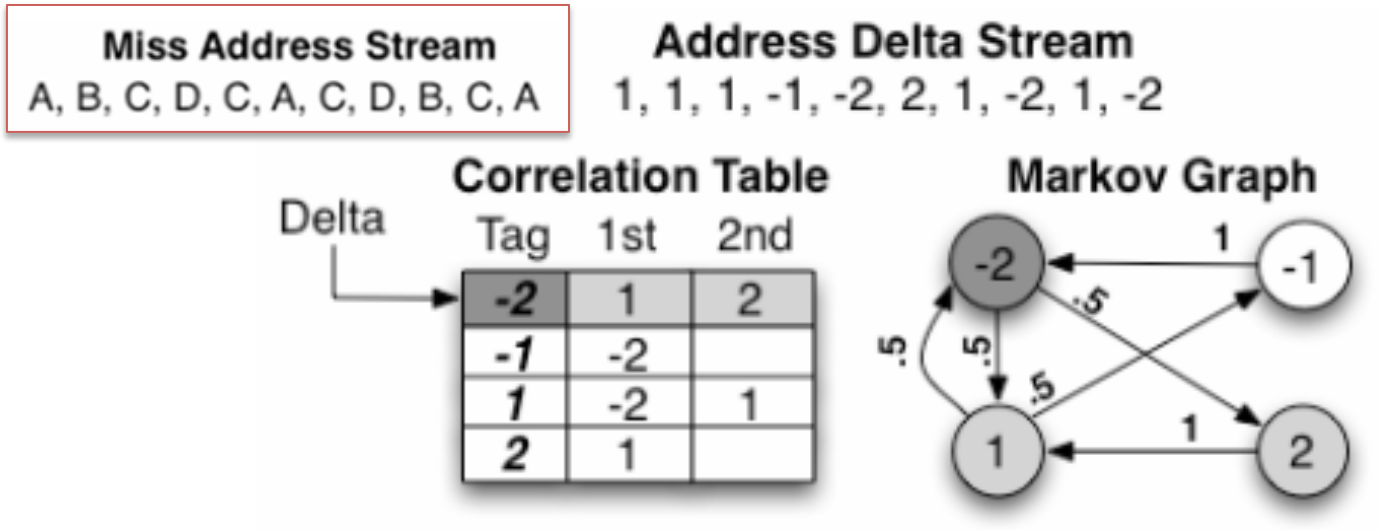
Markov Prefetching



Correlation Table: list of memory addresses represents arcs with the highest probability.

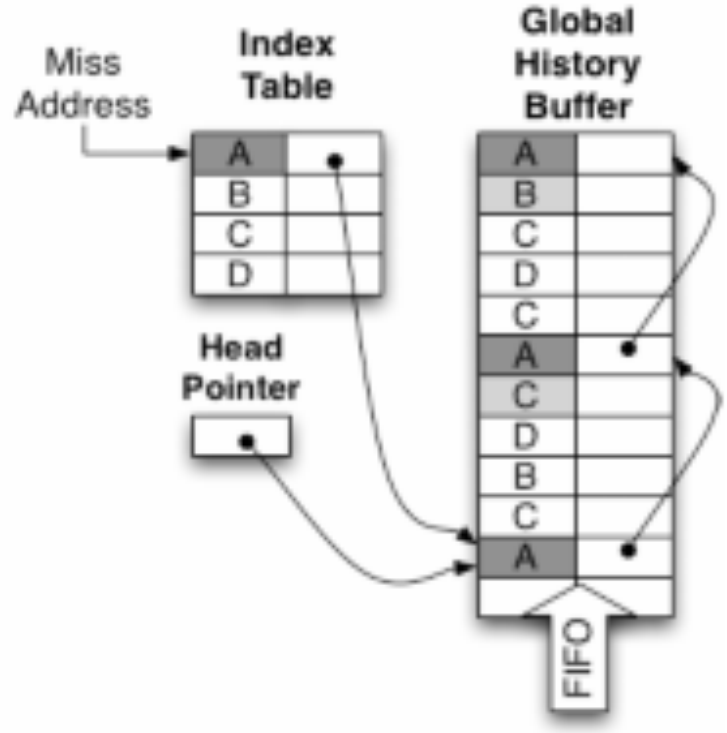
Markov Graph: addresses and the weights for edges

Distance Prefetching



DP uses the distance between two consecutive global miss addresses, address delta, to index correlation table.

GHB Global / Address Correlation



Index table (IT) that is accessed with a key (load instruction's PC, a cache miss address, or some combination)

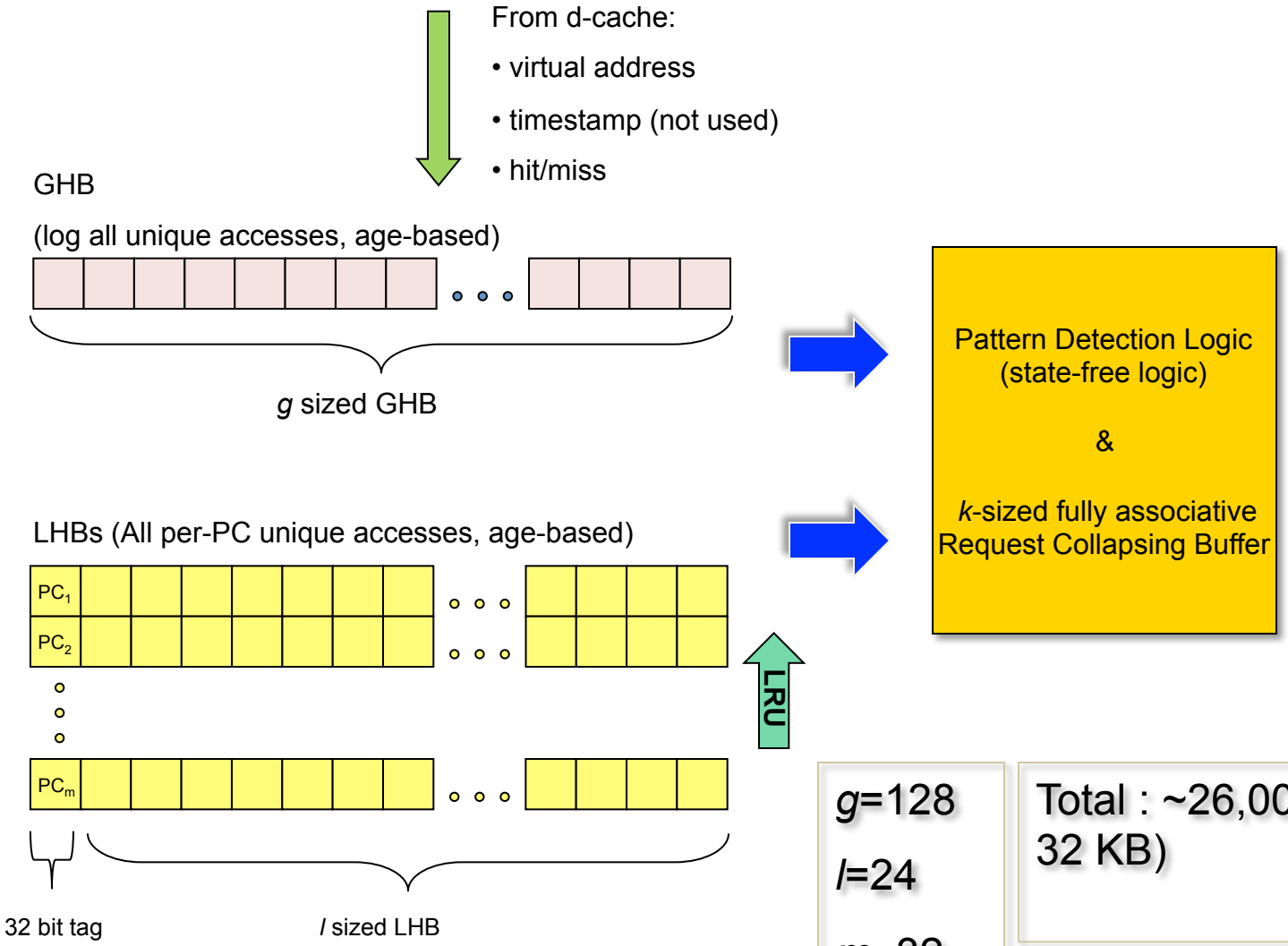
GHB entry = a global miss address + a link pointer.

Local History Buffer

Objective and Approach

- To propose a new hardware prefetcher that is a combination of the Global and Local History Buffer.
- This Local history Buffer keeps the memory access information for selective program counters.
- These buffers can then be queried on cache accesses to predict future memory accesses and enable data prefetching.

Our Data Prefetcher Organization



g=128

l=24

m=32

k=32

Total : ~26,000 bits (82% of 32 KB)

Rest dedicated to "temporaries"

Implementation

- Uses both a Global History Buffer (GHB) and multiple Local History Buffers (LHB).
- GHB tracks the most recent ‘n’ memory accesses in a program while LHBs track ‘m’ accesses performed at a particular Program Counter.
- When an address request is generated, the processor looks up the GHB and the LHB.
- An access address is only added if it is not found in these buffers.

Logic

- Inspects the deltas of accesses including both hits and misses in the recent past in a 64KB region around the latest access to see if there is a repeatable pattern.
- Inspects the deltas in the L2 misses in the recent past (if the buffer contains any) in a 64KB region around the latest access.
- Inspects the deltas of accesses in a region close to the latest access and orders them with respect to address space.

- Inspects the deltas of accesses in a region close to the latest access to see if a multiplicative increasing stride is found.

Conclusion

- Using approximately a 4KB storage budget, an average performance improvement of 20% can be obtained by exploiting and understanding a variety of memory access patterns.

References

- *The approach to prefetching data using history buffer*

Kyle J. Nesbit and James E. Smith Department of Electrical and Computer Engineering University of Wisconsin - Madison.

- *Data Prefetching Mechanism by Exploiting Global and Local Access Patterns*

Ahmad Sharif and Hsien-Hsin S. Lee School of Electrical and Computer Engineering Georgia Institute of Technology Atlanta, Georgia

VLIW Processors

Fahim Rahman
Ujjwal Guin

ECE Department
University of Connecticut

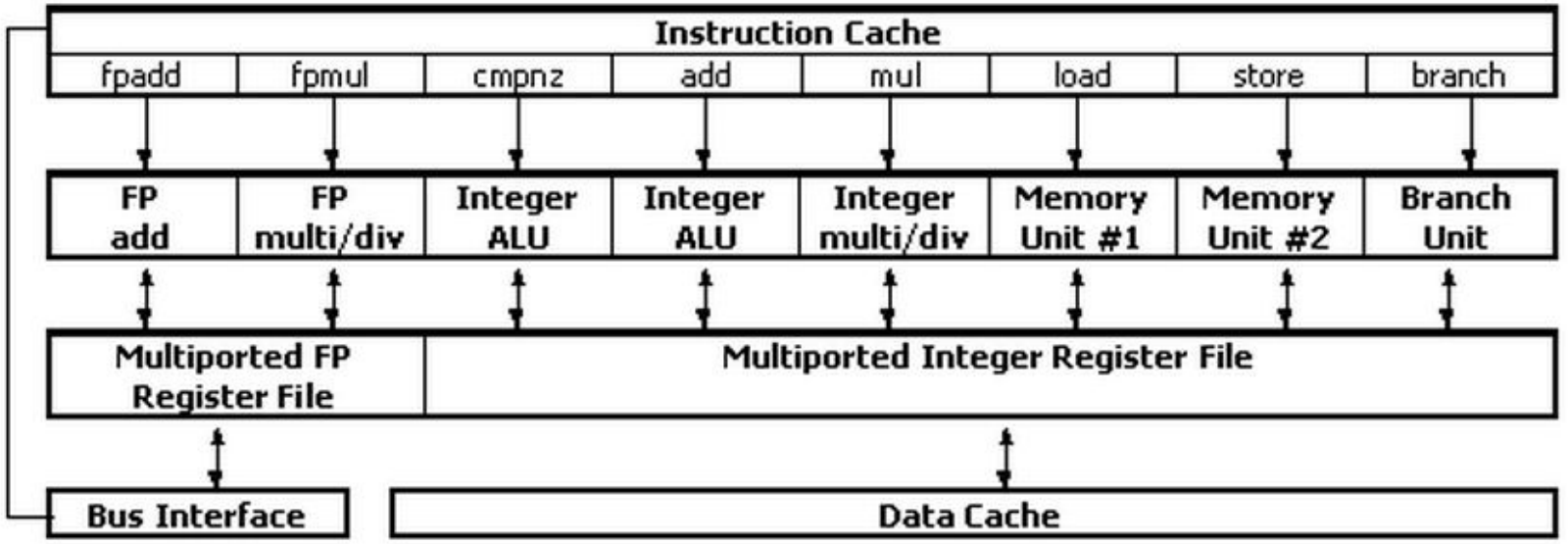
Outline

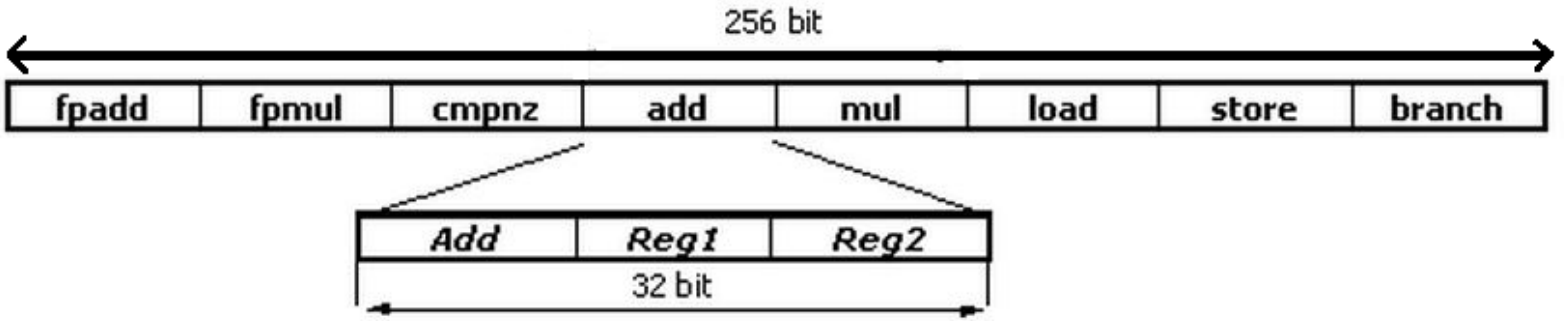
- VLIW Architecture
- VLIW – Logical Structure
- Instruction Level Parallelism (ILP)
- VLIW vs. Superscalar
- Commercial VLIWs

What is VLIW?

- Multiple Instruction Multiple Data (MIMD) Arch.
  - Controlled by a **V**ery **L**ong **I**nstruction **W**ord (**VLIW**)
  - Primarily introduced by Fisher in early 80's
- Multiple RISC-level instructions execution per cycle
  - A single long instruction per cycle : 256-1024bit long.
  - Each long instruction consists of some tightly coupled independent operations (may be RISC instructions).
  - Looks like parallel horizontal microcode.
  - Scheduling done by "intelligent" compiler (Static)
  - Each operation requires a small, statistically predictable number of cycles to execute – proper hardware support!

VLIW – Logical Structure



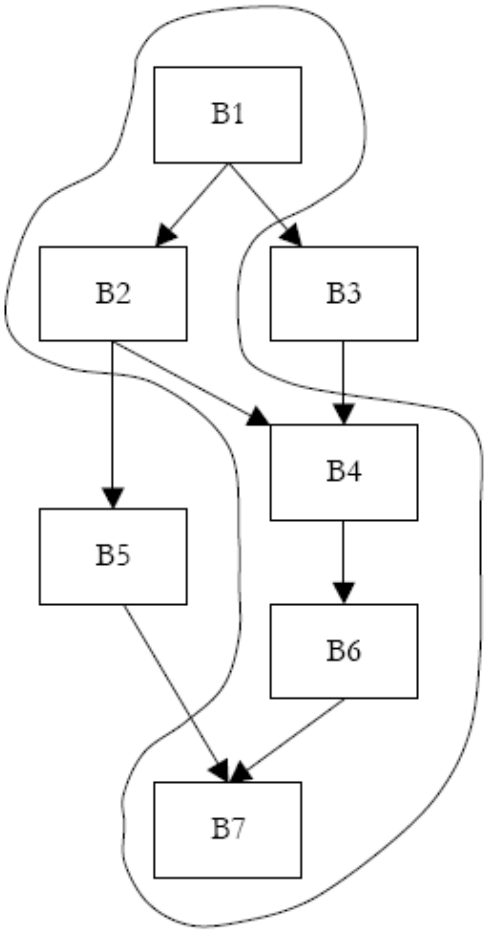


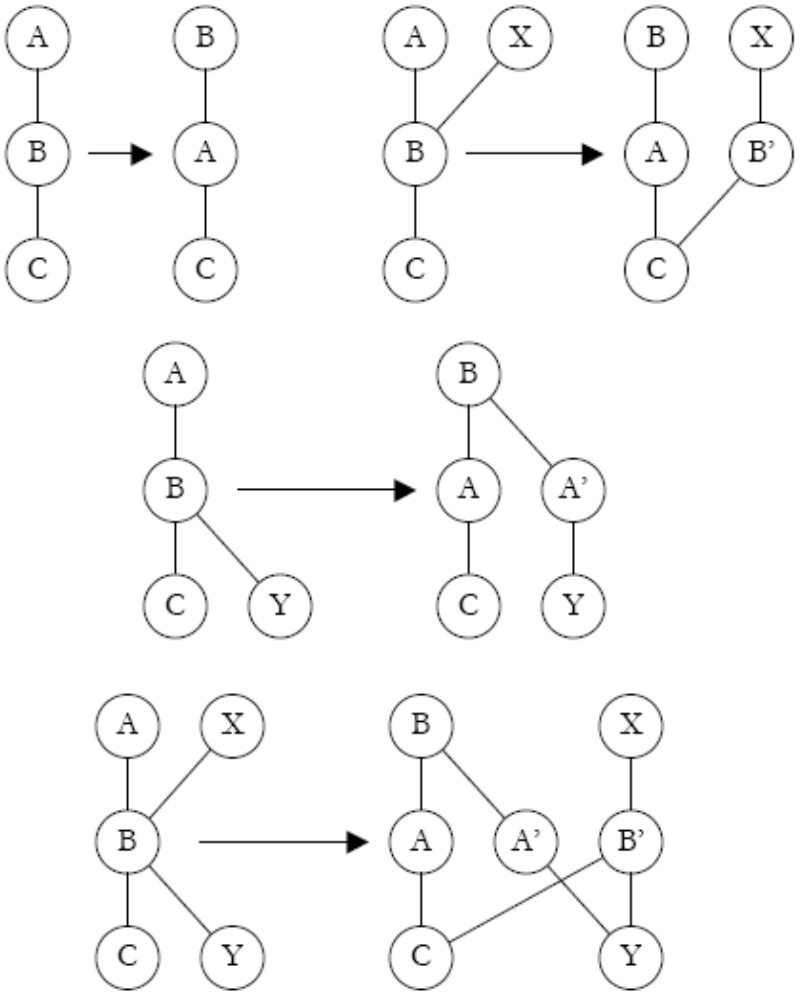
VLIW: Instruction Level Parallelism (ILP)

- Compiler Does the Hard Work
  - Compiler offers a strictly defined Plan Of Execution (POE) - Optimized Static Scheduling
  - Compiler has full idea on the processor/hardware
  - Compiler must be “Powerful” & “Intelligent” to extract ILP
    - Increases compiler complexity
  - Simpler Hardware

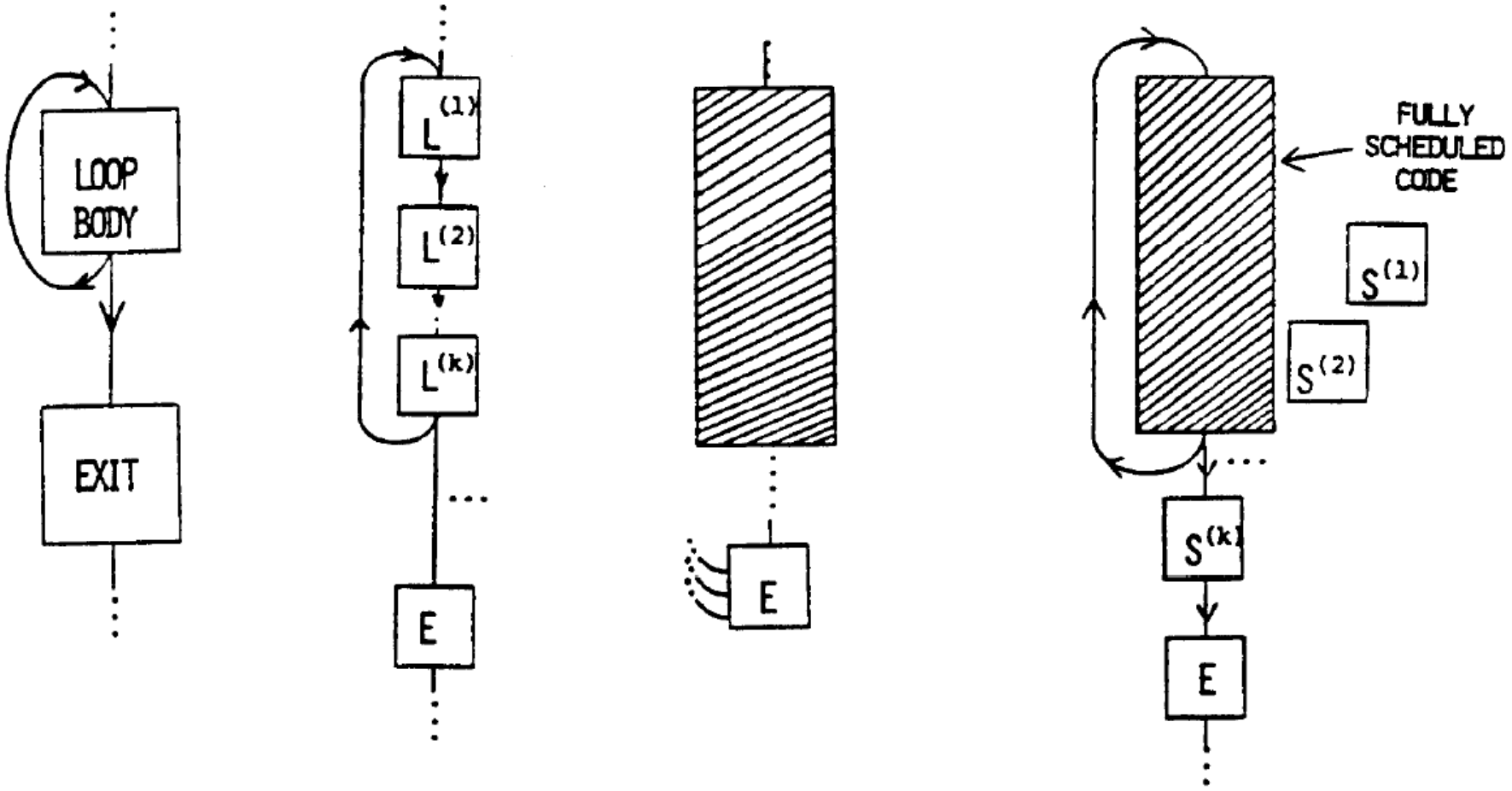
- ILP Extraction
  - Trace Scheduling
  - Software Pipelining
  - Speculative Execution

Trace Scheduling





Trace Scheduling – Cont.



Software Pipelining

```
for (i = 0 ; i < N ; i++)
{
        y[i] = x[i] * x[i] + c;
}
```

(a) Simple loop

```
L0:     LOAD  a2,[a0]; PADD a0,a0,1
        FMUL  a3,a2,a2
        FADD  a4,a3,c
        STORE a4,[a1]; PADD a1,a1,1; LOOP L0
```

(b) Assembly code

memory	addition			multiplication			integer/pointer
LOAD	*FADD*		**FADD**		***FMUL***		PADD
STORE		*FADD*		FMUL		***FMUL***	**PADD**
LOAD	***FADD***		*FADD*		FMUL		**PADD**
STORE		***FADD***		**FMUL**		FMUL	*PADD*
LOAD	FADD		***FADD***		**FMUL**		*PADD*
STORE		FADD		*FMUL*		**FMUL**	***PADD***
LOAD	**FADD**		FADD		*FMUL*		***PADD***
STORE		**FADD**		***FMUL***		*FMUL*	PADD

VLIW vs. Superscalar

- ☺ Simpler hardware than Superscalar Architecture
  - Compiler based static scheduling
  - free of complex circuitry that Super Scalar chips must use to coordinate parallel execution at runtime
- ☺ Burns less power
- ☺ Less area – free area can be used for larger cache etc.
- ☺ Can achieve better performance

- ☹ Processor-Compiler unique pair – not versatile
- ☹ Highly “Compiler-performance” dependent
  - Complex and highly advanced compiler required
- ☹ Under utilized if not enough ILP (control dominated case)
- ☹ Idle FU (NOP) decreases efficiency

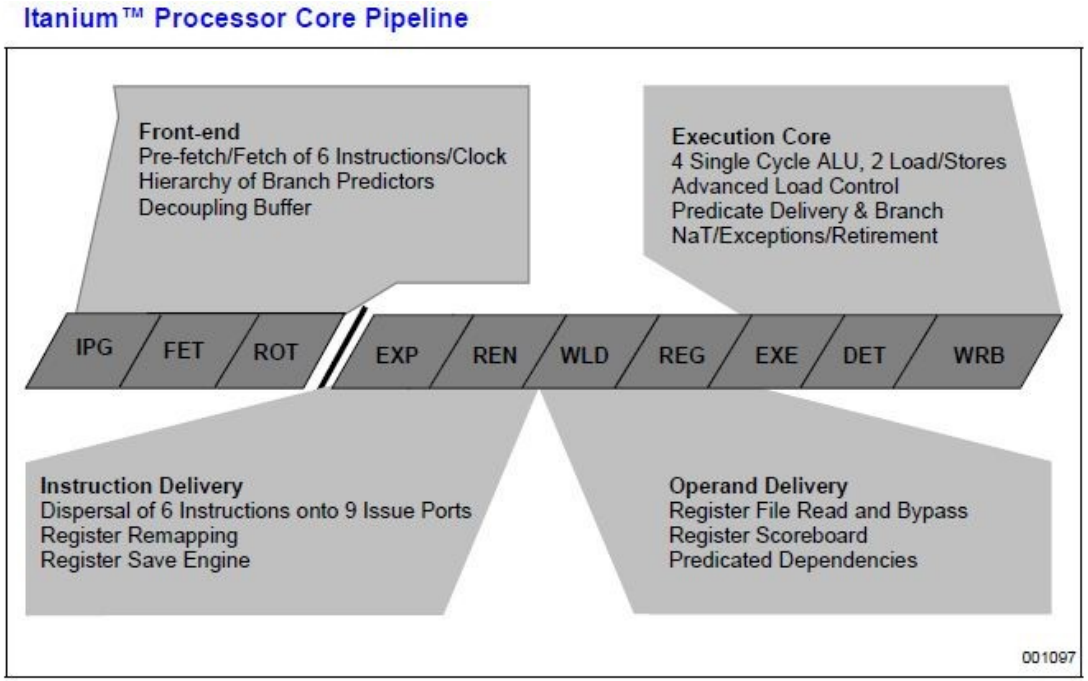
Commercial VLIWs

- AMD RADEON R700 (High Performance)
  - 5-way VLIW, 160 CORES, 750 MHZ, 150W
- TI TMS320/OMAP4430 (DSP Multicore)
  - 8-way VLIW, ARM, upto 1Ghz, 5W(max)
- Transmeta Crusoe (Embedded/Personal)
  - x86, 400-800Hz, 1.5-6Watt, Code-morphing

- Intel Itanium
  - EPIC (Explicitly Parallel Instruction Computing)
  - IA-64, 900MHz-2.53 GHz (2nd Gen), 6-wide EPIC

Intel Itanium

- Instruction Processing
  - Instruction fetch & prefetch, branch prediction, register stack
- Execution
  - Logic for integer, FPU, multimedia, branch prediction, register
- Control
  - Exception handler & pipeline control
- Memory Subsystem
  - L1,L2,L3, PIC, ALAT
- IA-32 Execution



References

[1] Joseph A. Fisher, Very Long Instruction Word architectures and the ELI-512, *Proc. of the 10th annual international symposium on Computer architecture, p.140-150, June 13-17, 1983*

[2] Theo Ungerer, Borut Robič and Jurij Šilc, “A survey of processors with explicit multithreading”, *ACM Computing Surveys (CSUR), v.35 n.1, p.29-63, March 2003*

[3] G. Blake, R. G. Dreslinski and T. Mudge "A survey of multicore architectures", IEEE Signal Process. Mag., vol. 26, no. 6, pp.26 -37 2009

[4] E. Ozer and T. M. Conte, "High-Performance and Low-Cost Dual Thread VLIW Processor Using Weld Architecture Paradigm", *IEEE Trans. Parallel and Distributed Systems, vol. 16, no. 12, 2005*

[5] Intel® Itanium™ Processor Hardware Developer’s Manual, August 2001

Thank you

DRAM Refresh Operation

Sara Tehranipoor
Sajad Mirzaei

December 2013

Contents

- Dynamic RAM Organization
- DRAM Refresh
- DRAM Refresh Operation Effects
- New method: Using DRAM Retention Time Distribution
- New mechanism details
- Retention Time Profiling
- Storing Retention Time Bins
- Results

DRAM Organization

- A DRAM structure is organized hierarchically
- The highest level is the channel
  - Each channel has command, address and data buses
  - A channel contains one or more ranks
- Each rank corresponds to an independent set of DRAM devices.
  - Within each rank is one or more banks.
- Each bank corresponds to a distinct DRAM cell array.
  - Each bank consists of a two-dimensional array of DRAM cells.
- The lowest level is DRAM cell that consists of a capacitor and an access transistor.

DRAM System Organization

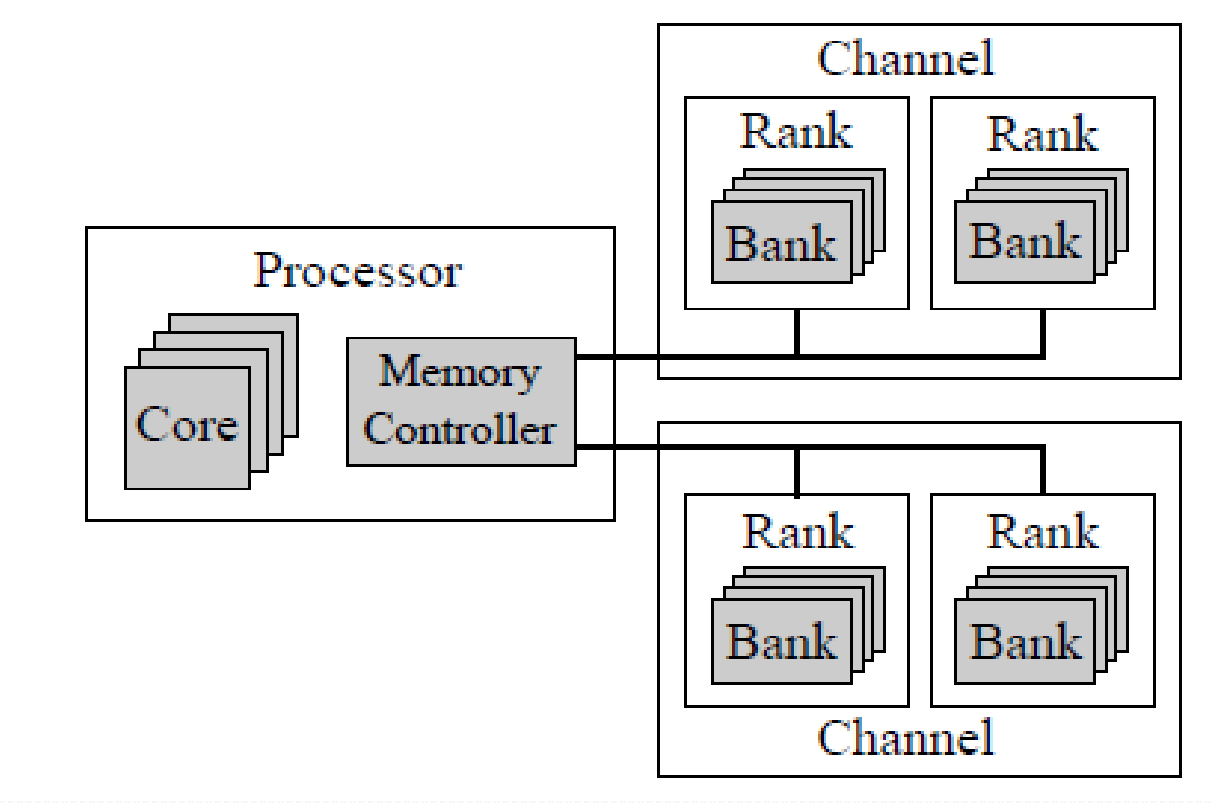


Figure 1: DRAM Hierarchy

DRAM System Organization

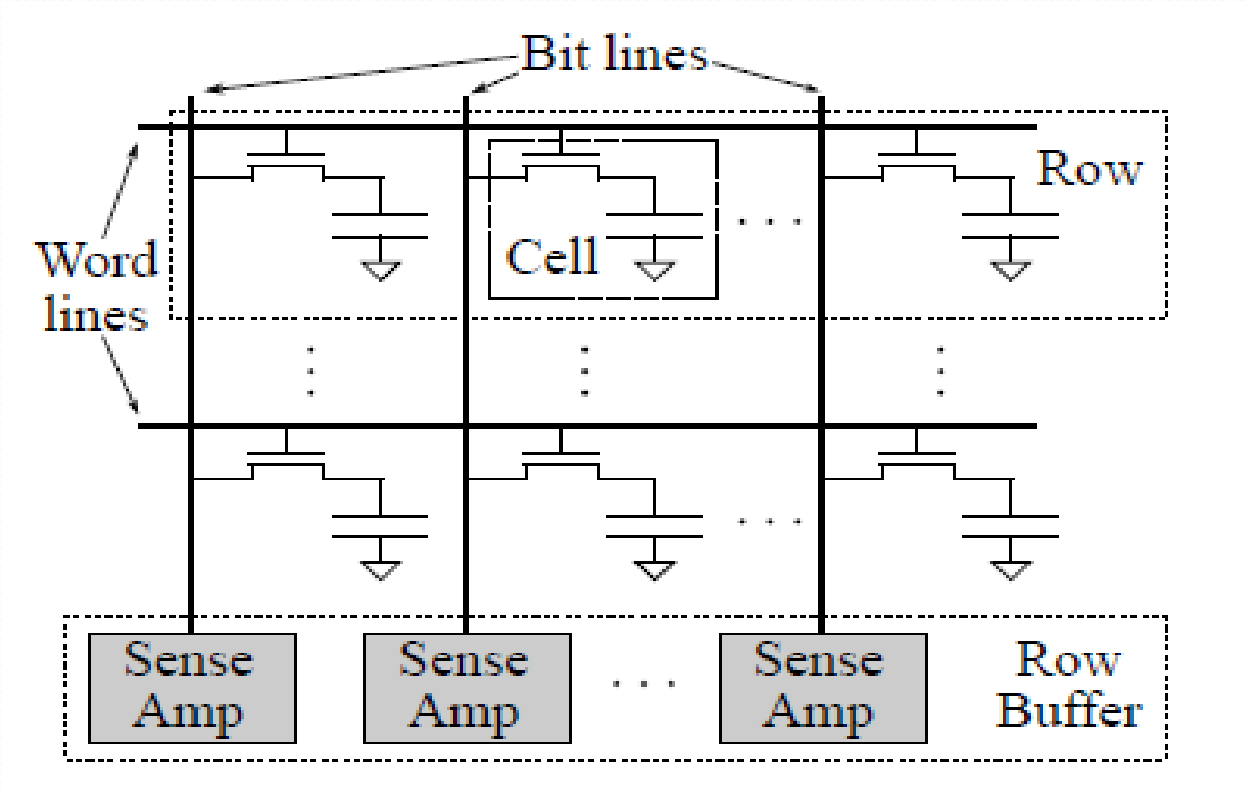


Figure 2: DRAM bank structure

DRAM Refresh

- DRAM cells lose data because capacitors leak charge over time.
- To preserve data integrity, the charge on each capacitor must be periodically restored or refreshed.
  - When a row is opened, sense amplifiers drive each bit line fully to either Vdd or 0V.
  - Hence, a row is refreshed by opening it.
- Refresh Intervel is a time between refreshed for a given cell that has remained a 64 ms for several DRAM generations.
- Refresh operations negatively impact both performance and energy efficiency.

DRAM Refresh Levels

- Memory Controller periodically issues an auto-refresh command to the DRAM.
- DRAM chooses which rows to refresh using an internal counter
- During normal temperature operation, the average time between auto-refresh Commands is 7.8 Microsecond and in the extended temperature is 3.9 Microsecond.

DRAM Refresh Operation Effects

- Refresh operations degrade performance in three ways:

- Loss of bank-level parallelism:

  - A DRAM bank cannot service requests whenever it is refreshing which result in decreased memory system throughput.

- Increased memory access latency:

  - Any access to a DRAM bank that is refreshing must wait for the refresh latency

- Decreased row hit rate:

  - A refresh operation causes all open rows at a rank to be closed, which causes a large number of row misses after each refresh operation.

Refresh Operation Effects

- Refresh Operation degrade energy efficiency:

- By consuming significant amounts of energy (opening a row is a high power operation)
- By reducing memory system performance

*** All of the problems are expected to worsen as DRAM device capacity increase.

New method: Using DRAM Retention Time Distribution

- What is Retention Time?
  - The time before a DRAM cell loses data that is depend on the leakage current for that cell's capacitor.
  - Retention Time varies between cells within a device.
- Retention time can be modeled by categorizing cells as either normal or leaky.
- The DRAM refresh interval is set by the DRAM cell with the lowest retention time.
- The goal of this paper is to design a mechanism to minimize waste of energy and time.

New mechanism details

- A set of bins is added to the memory controller, each associated with a range of retention times.
- Each bin contains all of the rows whose retention time falls into that bin's range.
- The shortest retention time covered by a given bin is the *bin's refresh interval*

Storing Retention Time Bins

- The memory controller must store the set of rows in each bin.
- A table of rows for each bin needed to store retention time.
- The exact number of rows in each bin will vary depending on the amount of DRAM in the system.
- If a table's capacity is inadequate to store all of the rows that fall into a bin, the implementation no longer provides correctness.
  - A row not in the table could be refreshed less frequently than needed.

- To overcome these difficulties, Bloom Filters can help.
  - It can provide a compact way of representing set membership.

Results

- In an 8-core system with 32 GB DRAM, new mechanism (RAIDR) achieves:
  - a 74.6% refresh reduction
  - an average DRAM power reduction of 16.1%
  - an average system performance improvement of 8.6%

Thank you

?

Emerging Memory Technology and Hierarchy

Mehdi Sadi

Md. Tauhidur Rahman

Bottleneck of Conventional 6-T SRAM Based Cache Memory

- The ever-increasing gap between processor speed and main memory latency has driven the demand for larger on-chip caches in processors
- Traditionally, on-chip caches in modern processors are implemented using 6 transistor static random access memories (6T-SRAM)
- Limited scalability, susceptibility to soft errors and high leakage power of SRAM pose challenges to high-density on-chip cache implementation

STT MRAM Based Cache Technology

- Among various candidates, Spin-Transfer Torque magnetic RAM (STT MRAM) is considered as a promising technology that can offer desirable memory attributes such as high endurance, non-volatility, soft error immunity, zero standby power and high integration capability

- More importantly, its compatibility with CMOS processes makes it an attractive

- However, higher write latency and write energy requirements, compared to the traditional embedded memories such as SRAM, are major issues with STT MRAM

STT Architecture

- A conventional STT MRAM cell comprises of a magnetic tunnel junction (MTJ) and an access transistor in series The MTJ contains a pinned layer and a free layer separated by a dielectric layer.
- The pinned layer has a fixed magnetization, and the free layer is programmable by changing its magnetic orientation. The resistance of the MTJ depends on the relative magnetization of the free layer with respect to the pinned layer.
- Parallel magnetization of the free layer with respect to the pinned layer leads to a lower resistance (R_P) compared to the resistance in the anti-parallel state (R_{AP}). The two resistances of the MTJ define the binary states of the memory cell
- A write operation is performed by passing a current (I_W) through the bit-cell that exceeds a critical current (I_C). The direction of (I_W) determines the final magnetization of the free layer

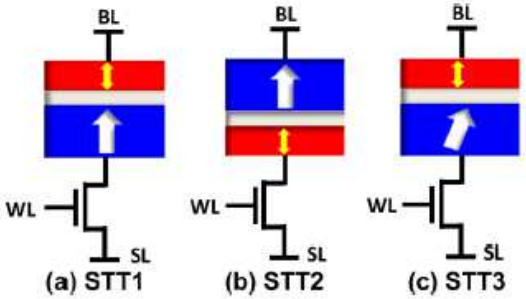


Figure 1: Schematics of an STT MRAM bitcell (a) in the standard-connected configuration (b) in the reverse-connected configuration and (c) with tilted magnetic anisotropy

Sang Phill Park; Gupta, S.; Mojumder, N.; Raghunathan, A.; Roy, K., "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* , vol., no., pp.492,497, 3-7 June 2012

Area and Energy Profile

- The energy dissipated in read operations in STT is higher than that of SRAM due to power dissipation in the analog read circuits, despite 4X smaller total cache area. However, for larger capacity (above 1MB), the energy dissipation due to interconnects becomes dominant in 6T-SRAM. Therefore, read dynamic energy is significantly lower in STT MRAM caches.
- STT MRAM cache using TMA bit-cells (STT3) shows significantly lower energy dissipation due to the lower write current requirement of the bit-cell
- Therefore, an STT MRAM cache can achieve high energy-efficiency along with high capacity in comparison to an SRAM cache, especially in lower levels of the cache hierarchy due to the low cache utilization

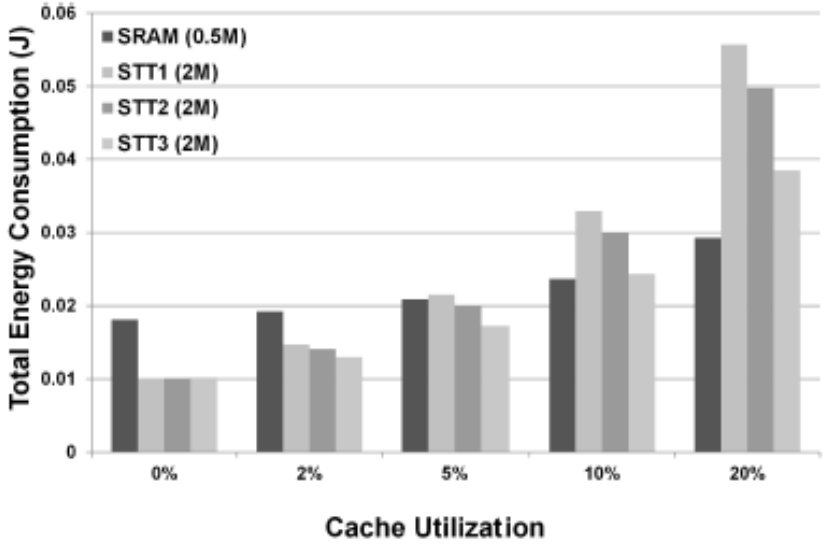


Figure 3: Total energy consumption of L2 caches at iso-area (0.5MB SRAM vs. 2MB STT MRAM)

Write Energy Reduction

- Each cache line is partitioned into n partial lines in-order to utilize the energy efficient column selection of STT MRAM arrays
- During write back from the SRAM L1 cache, only the partitions in the cache line that have been updated by the processor are written to the STT MRAM L2 data array

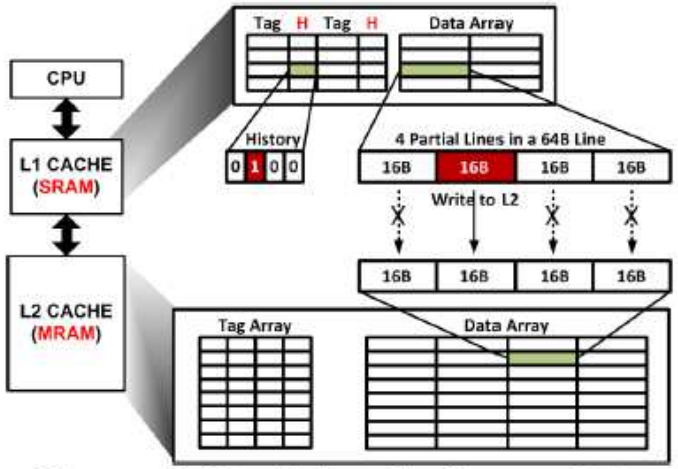


Figure 6: Partial cache line update

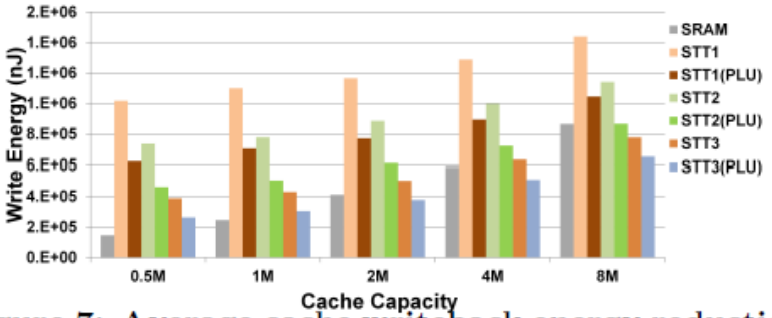


Figure 7: Average cache writeback energy reduction using PLU for 8 SPEC2000 integer benchmarks.

Sang Phill Park; Gupta, S.; Mojumder, N.; Raghunathan, A.; Roy, K., "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* , vol., no., pp.492,497, 3-7 June 2012

Read Energy Reduction

- In sequential tag-data access, a cache probes the tag array first, and identifies a hit or miss. Access to the data array occurs only when there is a cache hit, and only the sub-array storing the corresponding cache line in the data array is accessed.
- All cache lines in the row of the SRAM sub-array dissipate dynamic energy during read operations. On the other hand, in a sub-array of an STT MRAM based cache, only the bit-columns storing a single cache line consume energy

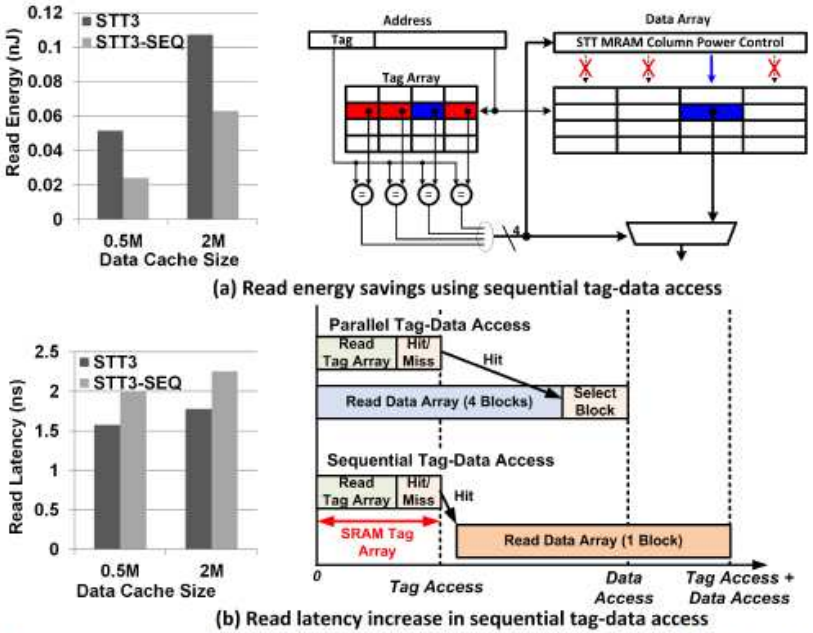


Figure 5: (a) Read energy savings and (b) Read latency increase in sequential tag-data access

Sang Phill Park; Gupta, S.; Mojumder, N.; Raghunathan, A.; Roy, K., "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, vol., no., pp.492,497, 3-7 June 2012

Sang Phill Park; Gupta, S.; Mojumder, N.; Raghunathan, A.; Roy, K., "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, vol., no., pp.492,497, 3-7 June 2012

STT Vs. 6T-SRAM

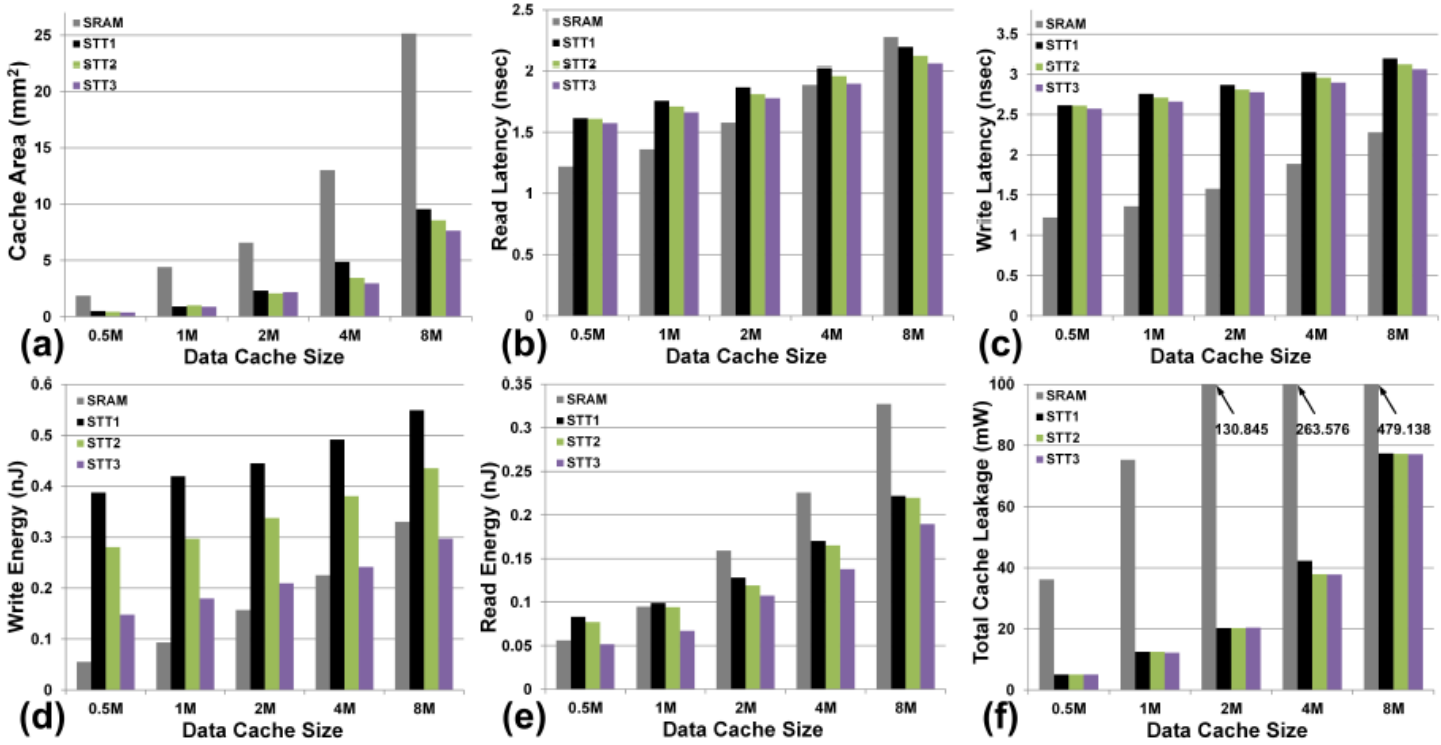


Figure 2: (a) Area requirement of SRAM and STT MRAM based caches (4-way, 64B cache line, B=Byte, M=Mega Byte) in mm^2, (b) read latency and (c) write latency in ns, (d) read energy and (e) write energy in nJ per operation, and (f) leakage power in mW

Sang Phill Park; Gupta, S.; Mojumder, N.; Raghunathan, A.; Roy, K., "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* , vol., no., pp.492,497, 3-7 June 2012

Traditional multi-level SRAM-based cache

- Limited size with CMP: cache-core balance
- Leakage power
- More cache levels: design overhead, coherence

Emerging Memory Technologies

- Improve cache power-performance under the same chip area/footprint
  - Embedded DRAM (EDRAM)
  - Magnetic RAM (MRAM)
  - Phase Change RAM (PRAM)

	SRAM	eDRAM	MRAM	PRAM
Density(ratio)	Low(1)	**High(4)**	**High(4)**	**High(16)**
Dynamic Power	Low	Medium	**Low for read High for write**	**Medium for write ; High for write**
Leakage Power	High	Medium	**Low**	**Low**
Speed	Very Fast	Fast	**Fast for read; Slow for write**	**Slow for read; Very slow for write**
Non-volatile	No	No	Yes	Yes
Scalability	No	Yes	Yes	Yes

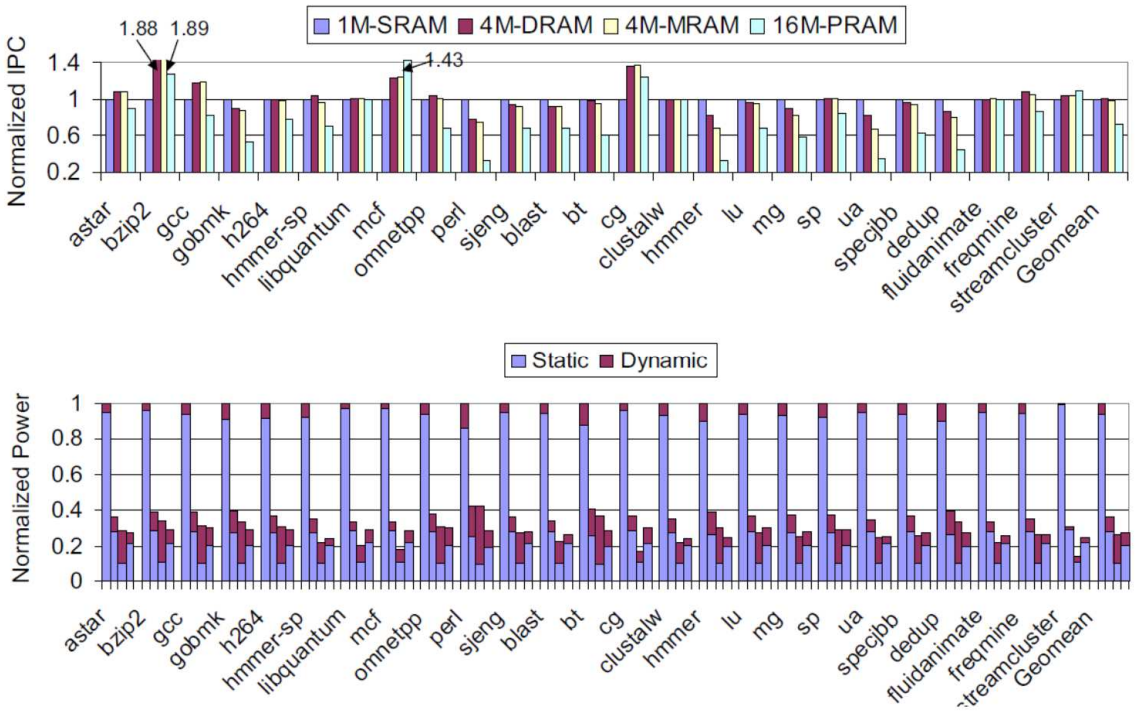
Reduce Cache miss rate

Increase hit latency

Low leakage power

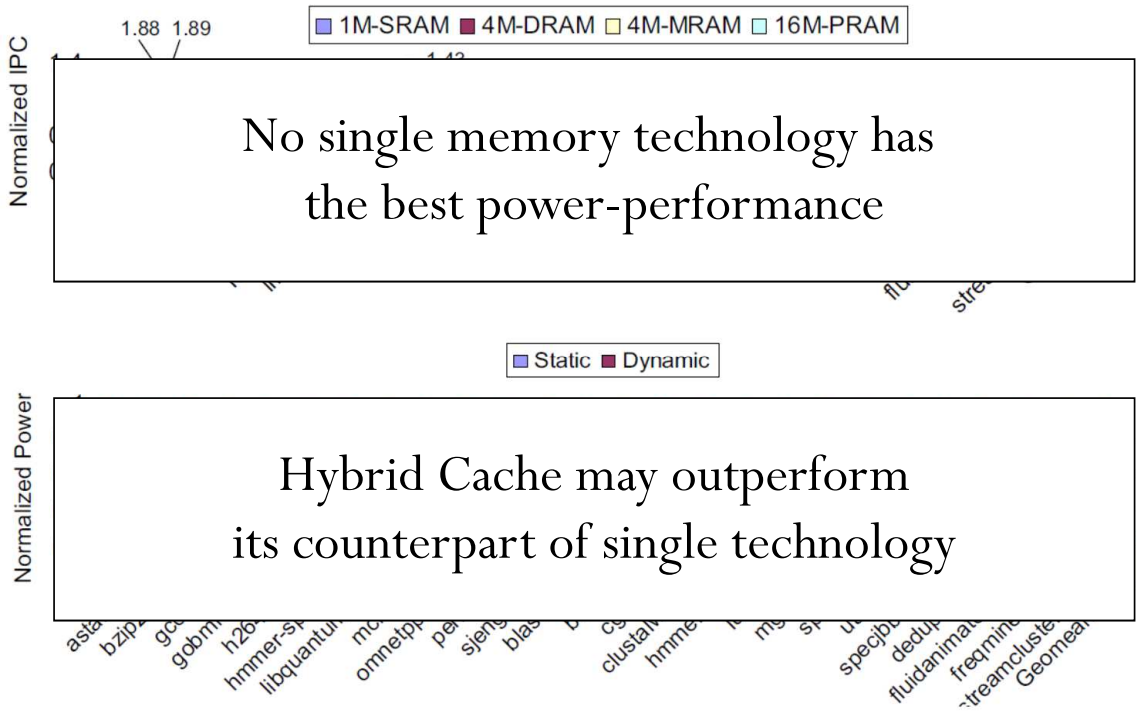
High dynamic power

Motivation



Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

Motivation

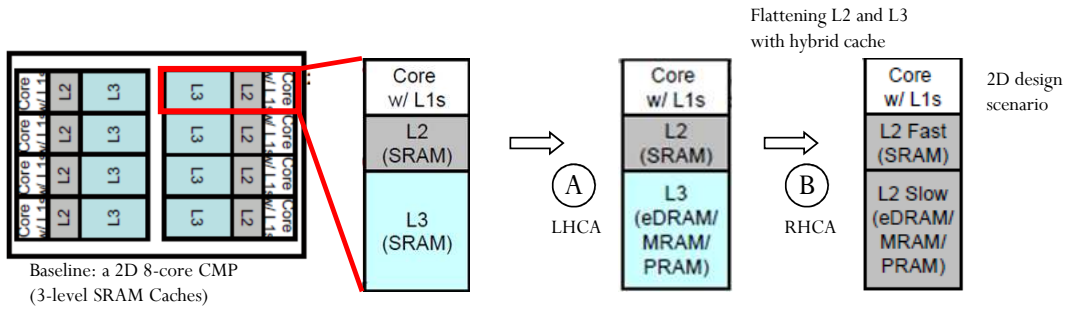


Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

Hybrid Cache Arch. (HCA)

- inter cache Level HCA (LHCA)
  - levels in a cache hierarchy can be made of disparate memory technologies

- Region based HCA (RHCA)
  - a single level of cache can be partitioned into multiple regions, each of a different memory technology

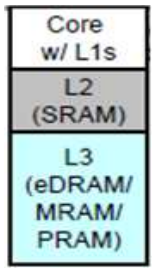


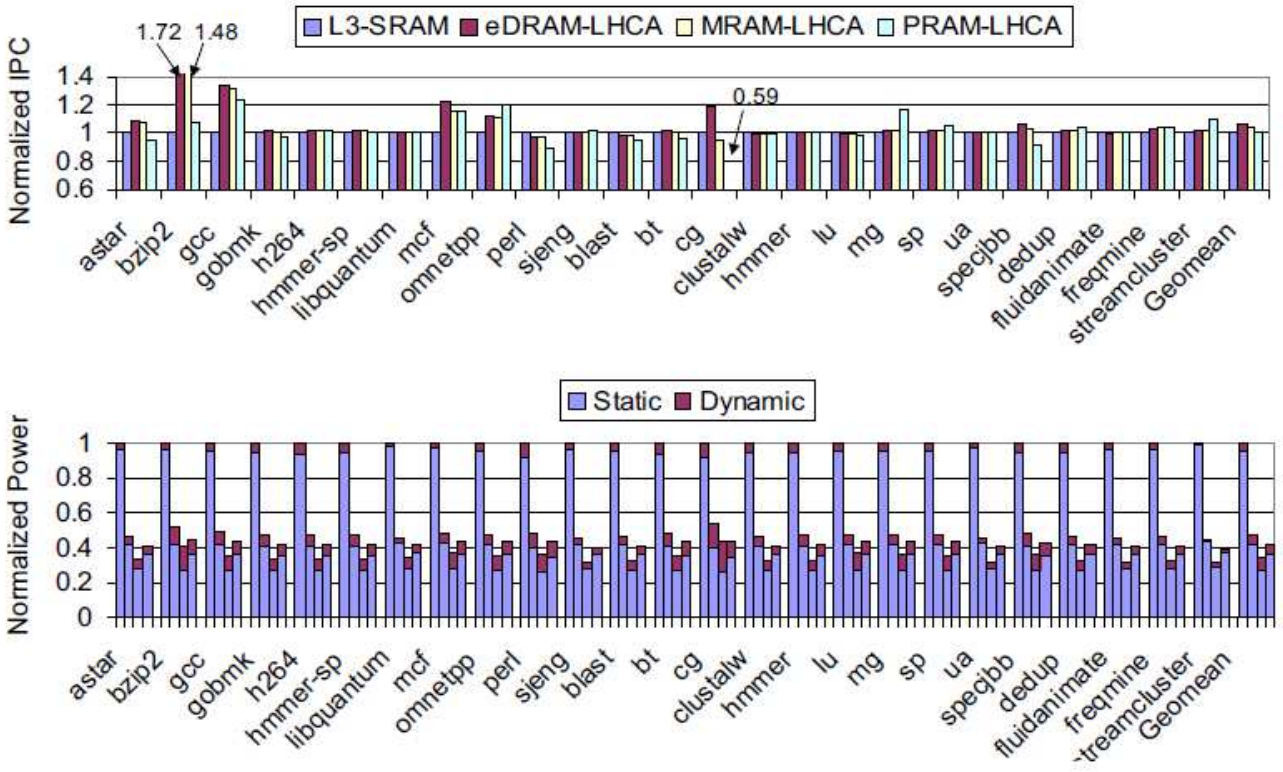
Benefits

- Effective cache size can increase significantly under the same chip area constraint
- Applying non-volatile memory technologies can reduce cache power significantly
- By merging multiple cache levels into one, the multiple cache regions can be checked in parallel
- By reducing the number of cache levels, the coherence traffic between levels is reduced.
- Performance can be improved by placing faster cache regions closer to the cache controller
- When the fast region returns valid data, the search signal to the slow region can be canceled

Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

LHCA: Performance and Power





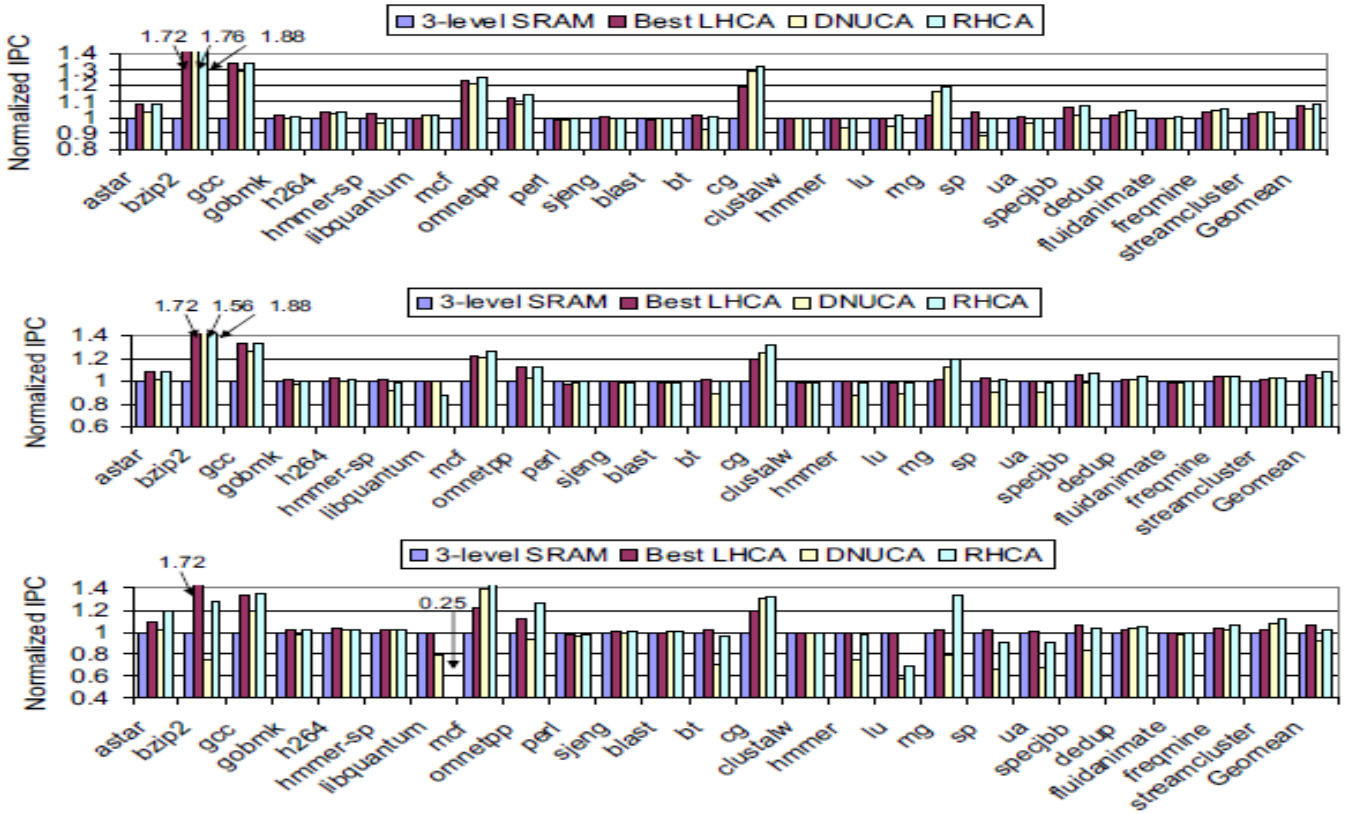
Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

RHCA Features

- Flatten the cache hierarchy by merging the eDRAM or MRAM or PRAM L3 into the SRAM L2
- Fast and slow regions in one cache level
  - Fast region made of SRAM
  - Slow region made of eDRAM, MRAM or PRAM
- The large hybrid L2 cache has the potential of providing fast-region access time and large-region capacity simultaneously
- Intra-cache data movement policy: Move frequently used data to the fast region

Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

RHCA Results



Xiaoxia Wu et al., " Hybrid Cache Architecture with Disparate Memory Technologies." In Proc. of the 36th annual int. symp. on Computer architecture, pages 34-45, 2009.

Questions?

Energy Efficient Cache Design

Presented by

Menglong Guan & Junlin Chen

Part I:

Way Guard: A Segmented Counting Bloom Filter Approach to Reducing Energy for Set-Associative Caches

------from *ISLPED* 2009

Presented by

Junlin Chen

UCONN

Drawback of Conventional Cache

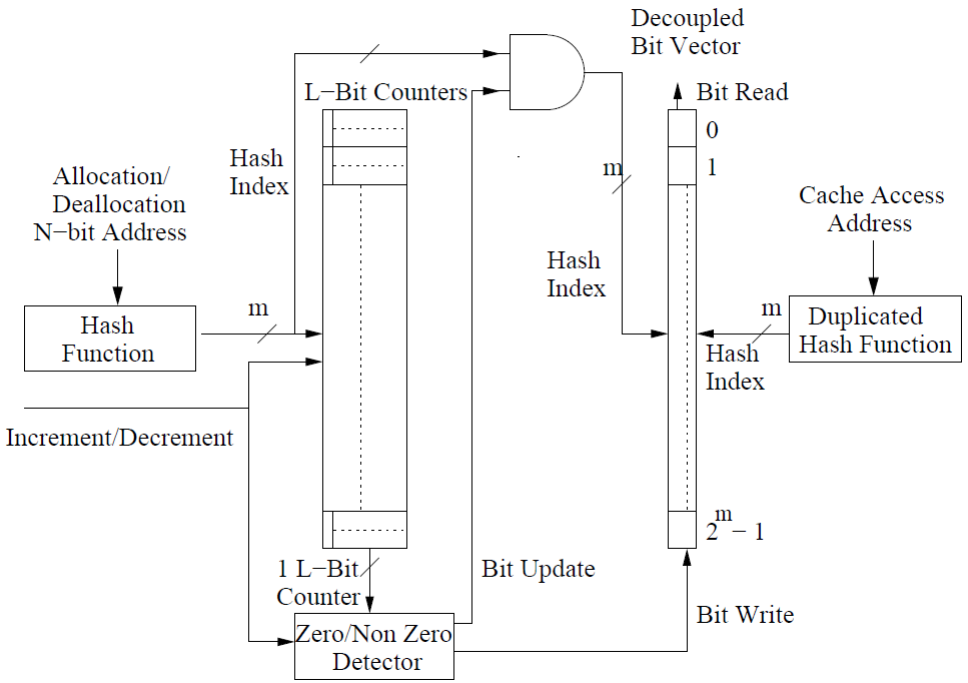
- For processors employing highly associative caches (e.g., 32 way), the energy consumption gets even worse as ***N-tag comparisons*** are needed for each parallel lookup of an N-way cache.
- In fact, most of the energy consumed for such lookups is redundant as the requested data can only be present in one particular way.
- This ***redundancy*** provides a good opportunity for saving dynamic energy.

Main Idea of Way Guard

- The ***Way Guard*** scheme uses counting Bloom filters to efficiently skip the lookup of cache lines that do not contain the requested data to save significant energy in cache accesses.
- They can replace the expensive set-associative *tag matching* with a simple bit vector that precisely identifies addresses that have not been observed before.

Segmented Bloom Filter

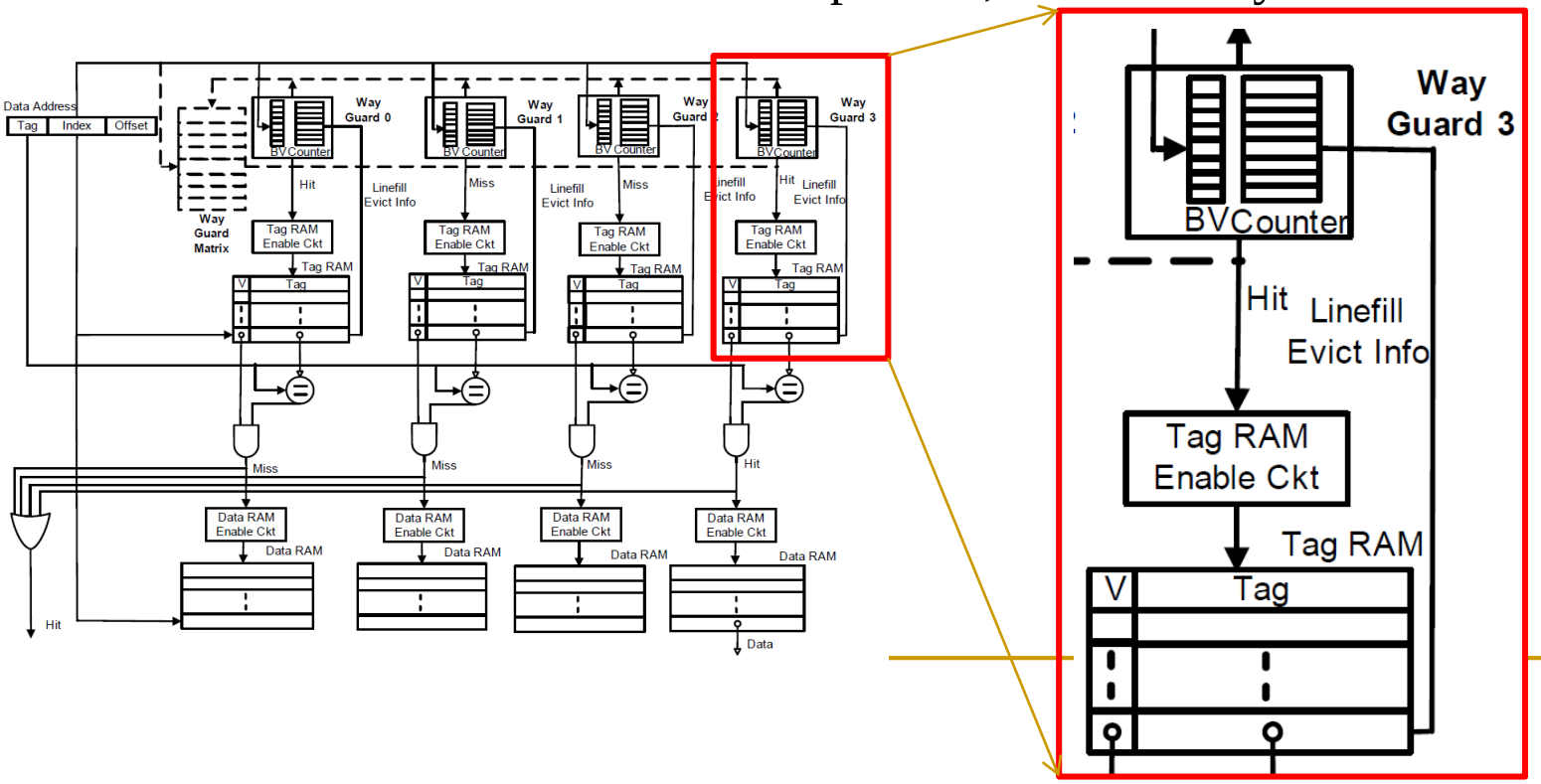
- Main parts: L-Bit Counters; Bit Vectors (BV)
- Simple and fast access, penalty is negligible.



Way Guard Mechanism

---Filter out Unnecessary Cache Way Lookup

- Better performance for cache miss:
  - If the filter indicates the address is not present in the way it is guarding, the data is certainly not in the way.
  - If the filter indicates the address is present, the data may be in the way.



Experimental Analysis-I

- Way Guard filter size: 4 times the number of lines in cache way
- For higher set associativity cache, 25% ways are checked.

Less ways looked up for cache miss!

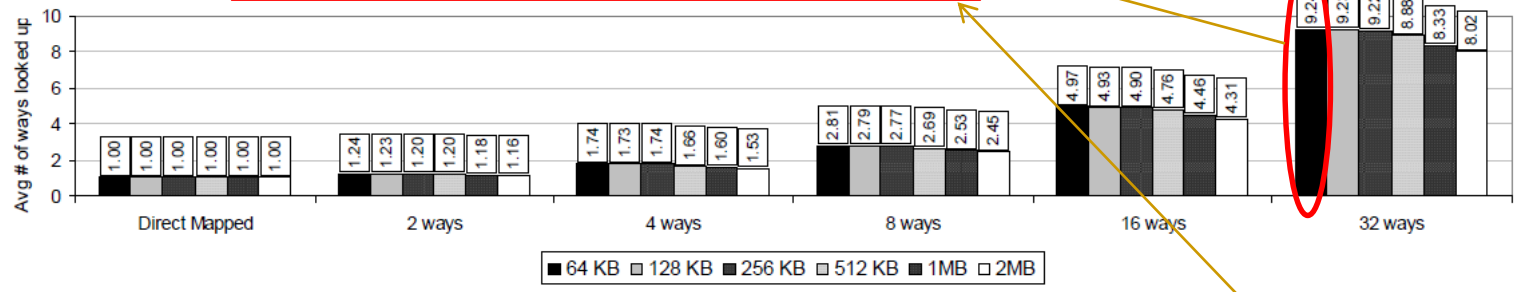


Figure 4: Average Number of Ways Looked Up for Hits in an L2 Cache

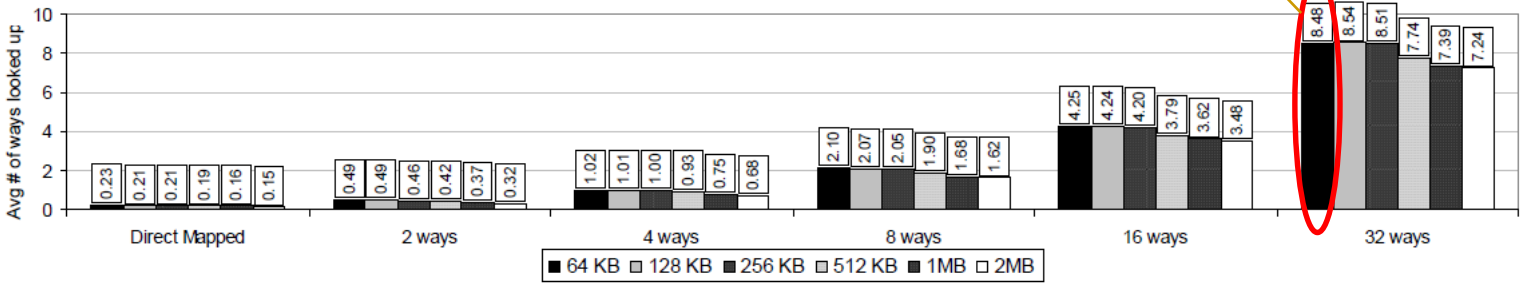


Figure 5: Average Number of Ways Looked Up for Misses in an L2 Cache

Experimental Analysis-II

- Way Guard filter is more effective at indicating absence of data

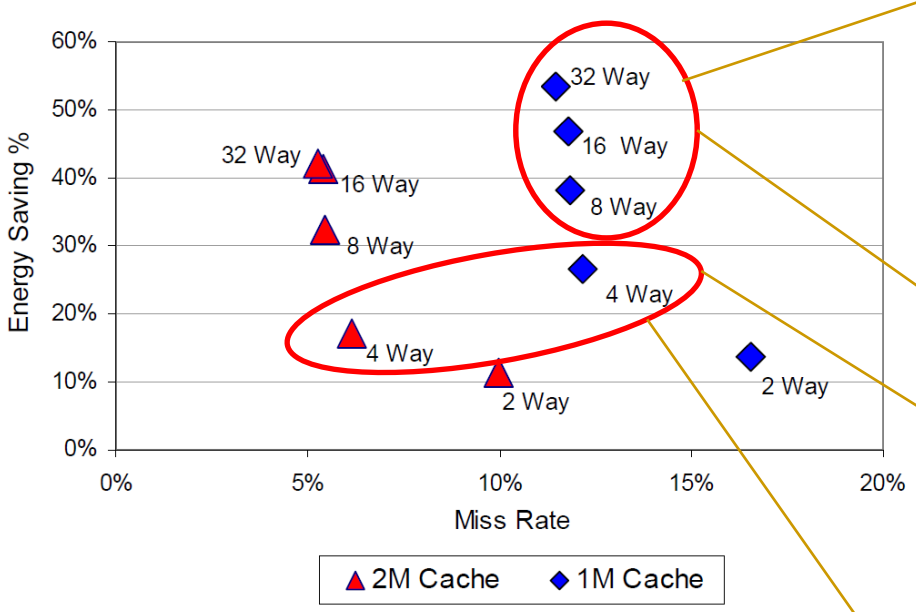


Figure 6: Energy Savings with respect to Miss Rate

- For the same cache size: higher associativity gives a greater chance for indicating absence, leading to larger energy savings.
- For the same associativity:
  - Larger cache has larger overhead of bloom filter;
  - Larger cache means lower miss rate.

Part Ⅱ :

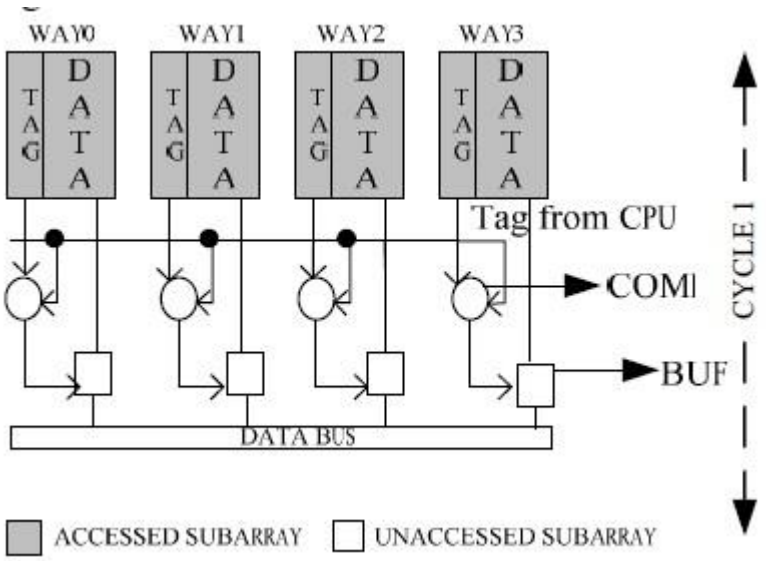
Phased Set Associative Cache Design For Reduced Power Consumption

------from ICCSIT 2009

Presented by

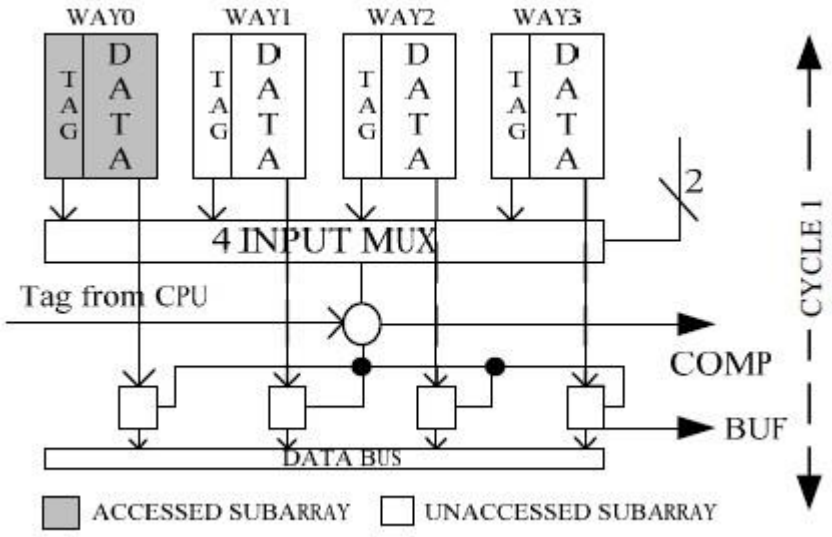
Menglong Guan

Drawback of Conventional Cache



Sequential: The tag is selected and compared with the tag generated by the CPU, at the same time the data of the corresponding way will be fed to the input of the buffer and the buffer is enabled only if there is a hit.

Parallel: In conventional parallel cache the tag comparison is done in parallel and hence it will take only one clock cycle to complete the comparison, but will consume much more power by making all the comparators working at the same time and by accessing unwanted data and tags.



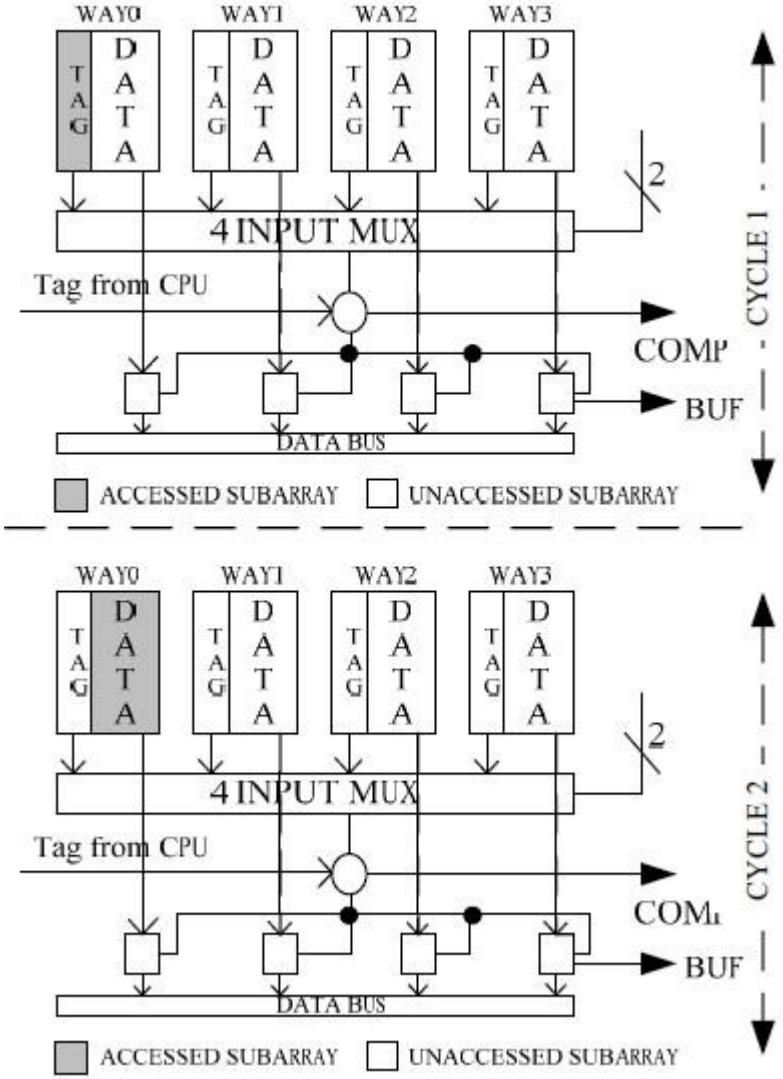
Main Idea of phased cache

- The usage of phased cache will reduce the power consumption by avoid accessing the unwanted data subarrays.

- First phase: In phased cache the tags of all the ways are compared sequentially or in parallel in the first phase
- Second phase: If there is a tag match then the corresponding data is accessed and fed to the input of the buffer. The corresponding buffer will be enabled in the same clock cycle and the data will be available in the data bus.

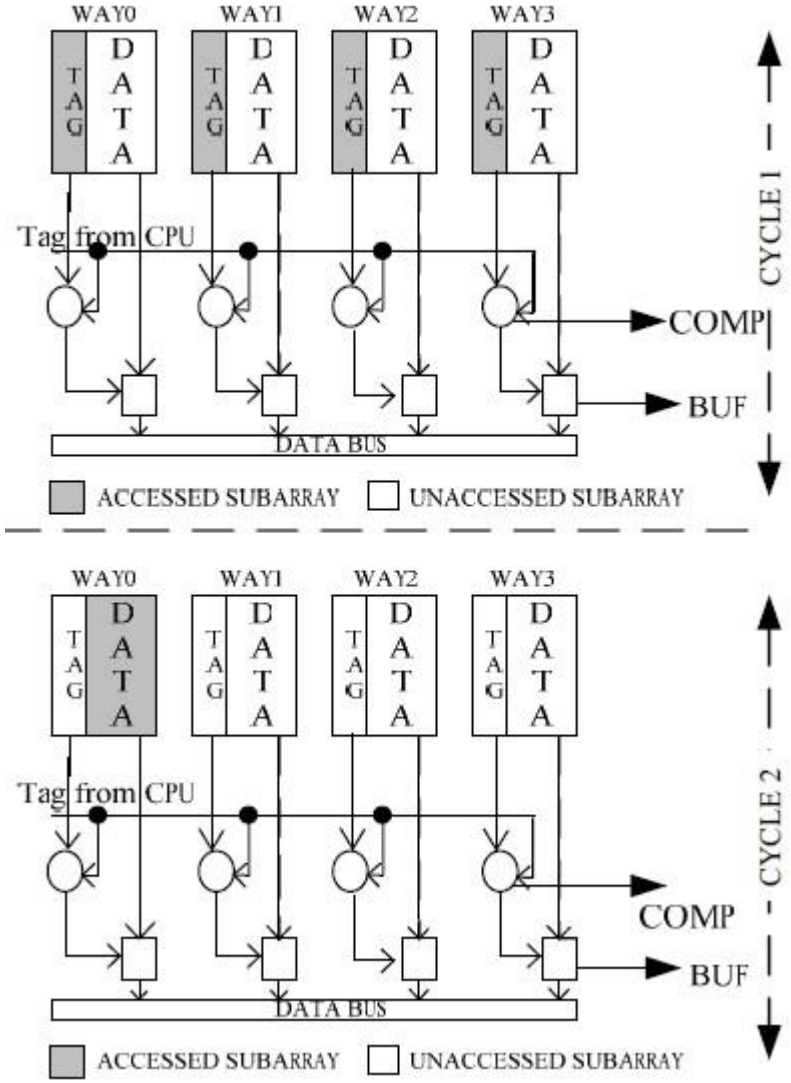
Phased Sequential cache

- In phased sequential cache architecture there are physically separated tag subarray and data subarray.
- The tag subarray of way0 is accessed at first and then compared with the tag generated by the CPU.
- If the tag matches only then the data subarray is accessed. If there is no match, then the tag of the next way is accessed and compared. This is repeated until a hit occurs or after comparing all the ways in a particular set.

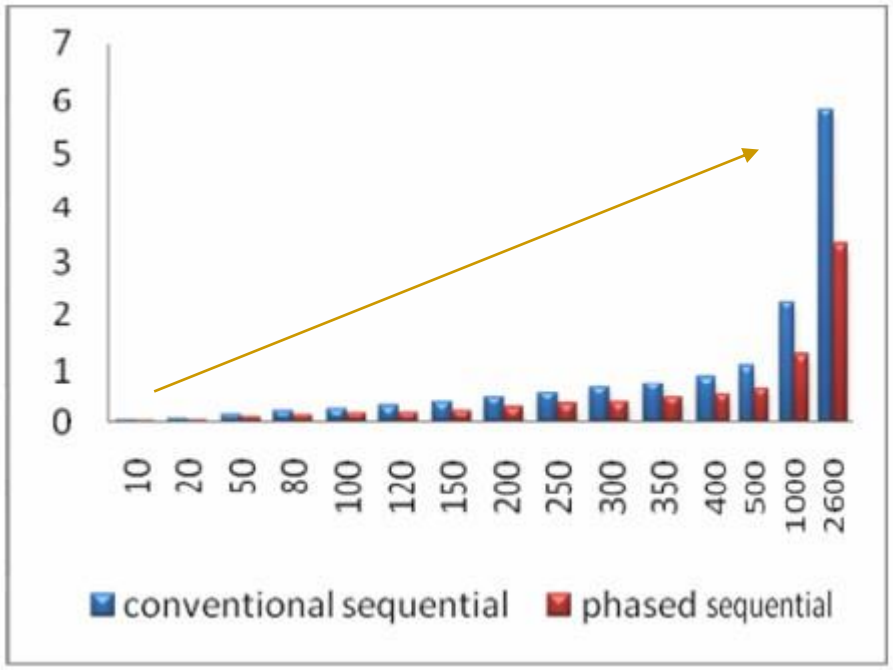


Phased Parallel cache

- In phased parallel cache there are physically separated tag and data subarrays.
- The tag subarrays of all the ways are accessed simultaneously and compared with the tag from CPU.
- If there is a match then the data subarray of the matched way is accessed and is fed to the input of the buffer. The corresponding buffer is enabled and the requested data will be available in the data bus in two clock cycles provided it is not a miss.

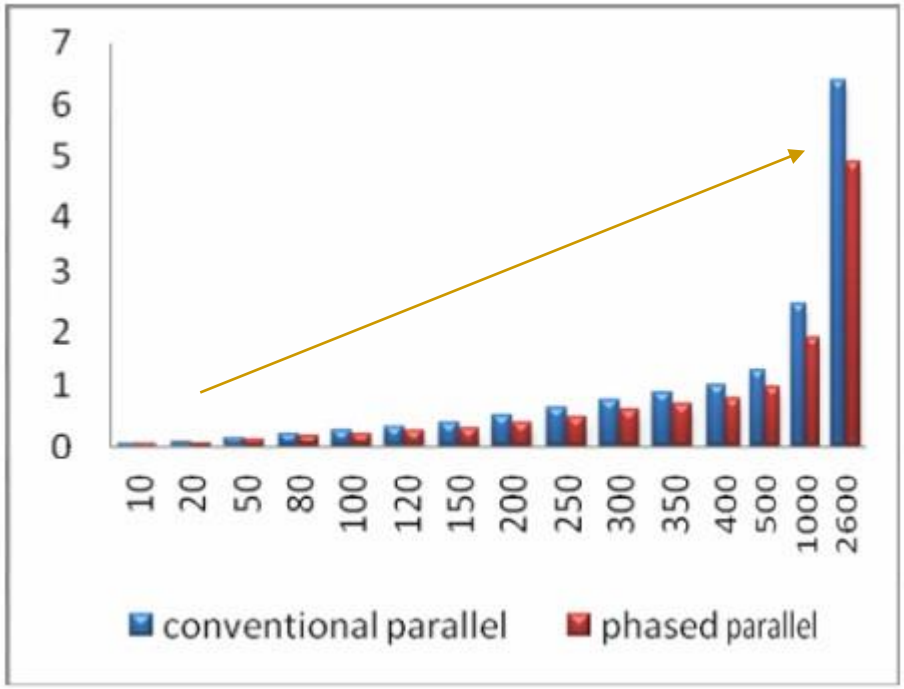


Experimental Results for Sequential



Clock Frequency (in MHz)	*Power Consumed (in W)*		% Reduction in Power Consumption
	Conventional Sequential	*Phased Sequential*	
10	0.036	*0.029*	19.4
20	0.058	*0.039*	32.7
50	0.124	*0.077*	37.9
80	0.19	*0.116*	38.9
100	0.235	*0.142*	39.5
120	0.279	*0.167*	40.1
150	0.346	*0.206*	40.5
200	0.457	*0.27*	40.9
250	0.567	*0.334*	41.1
300	0.678	*0.339*	50
350	0.789	*0.463*	41.3
400	0.9	*0.528*	413
500	1.121	*0.655*	41.5
1000	2.229	*1.292*	42
2600	5.835	*3.332*	42.9

Experimental Results for Parallel



Clock Frequency (in MHz)	*Power Consumed (in W)*		% Reduction in Power Consumption
	Conventional Parallel	*Phased Parallel*	
10	0.035	*0.033*	5.7
20	0.063	*0.053*	15.8
50	0.136	*0.113*	16.9
80	0.209	*0.171*	18.18
100	0.259	*0.208*	19.69
120	0.307	*0.245*	20.19
150	0.381	*0.302*	20.73
200	0.503	*0.396*	21.27
250	0.626	*0.491*	21.56
300	0.748	*0.585*	21.79
350	0.871	*0.678*	22.15
400	0.993	*0.772*	22.25
500	1.239	*0.96*	22.51
1000	2.462	*1.899*	22.86
2600	6.386	*4.906*	23.21

Conclusion

Way Guard

- This paper presents an efficient use of the counting segmented Bloom Filter called ***Way Guard*** to reduce significant dynamic cache energy by filtering out unnecessary way lookup in a set associative cache.

Phased Cache

- **Advantage**: Here we have proposed and compared two techniques which will effectively reduce the logical hardware and thus save a significant amount of power.
- **Disadvantage**: This implementation has the disadvantage of slower execution and increase in access time. This design may be implemented especially in embedded systems where speed is not a major concern but reduction in power consumption is appreciated.

Computer Architecture Research Project
Topic: Low Power Cache Design

Department of Electrical and computer Engineering

Advisor: Prof. Omer Khan

Students: Miao He, Yang Zhao

Agenda

I. Motivation

II. Introduction

III. Background: SRAM Cell Design

IV. Improved Design for low power SRAM

V. Conclusion

I. Motivation

- Why do we choose the topic for low power cache
  - Power consumption is a pressing issue due to the scaling down of silicon CMOS technology
  - In modern CPU, cache accounts for 30% to 60% of the whole power consumption.

II. Introduction

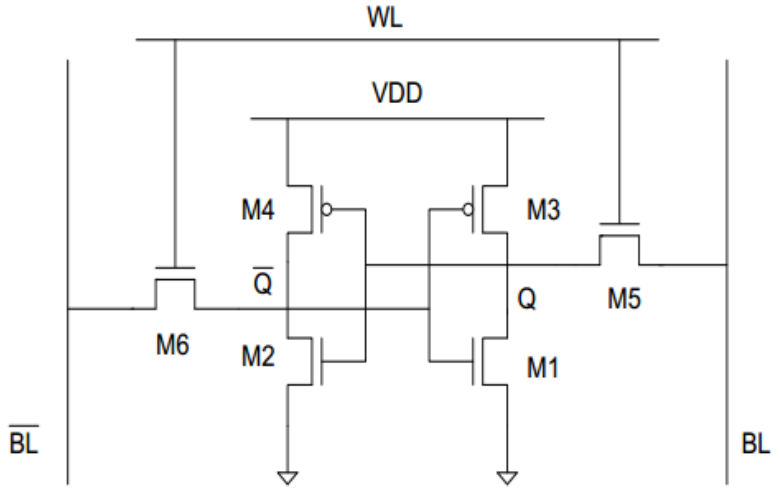
- Cache Power Consumption
  - Cache memories normally occupy more than half the processor die in current multicore process.
  - Lower power cache has been a hot issue in the design
  - SRAM is a main part of cache, so its power consumption has always been researched

- SRAM
  - Static random-access memory: Use bistable latching circuitry to store each bit
  - Not like dynamic RAM which must be refreshed periodically
  - SRAM is more expensive and less dense and therefore is not used for high-capacity, low-cost applications

III. Background: SRAM Cell Design

- SRAM cell
  - The 6-transistor SRAM cell is the most frequently cell using on chip memory.

The two cross-coupled inverter



M1, M2 - driver

M3, M4 - load

M5, M6 - Access

6T SRAM cell schematic

- The 6T cell design involves complex balance among many factors:
  - Low leakage current
  - Minimize area to achieve high density, reduce power and cost
  - Cell stability
  - Minimum word line pulse width to save power

- Write and read operation
  - To write a "1", initially at a "0"

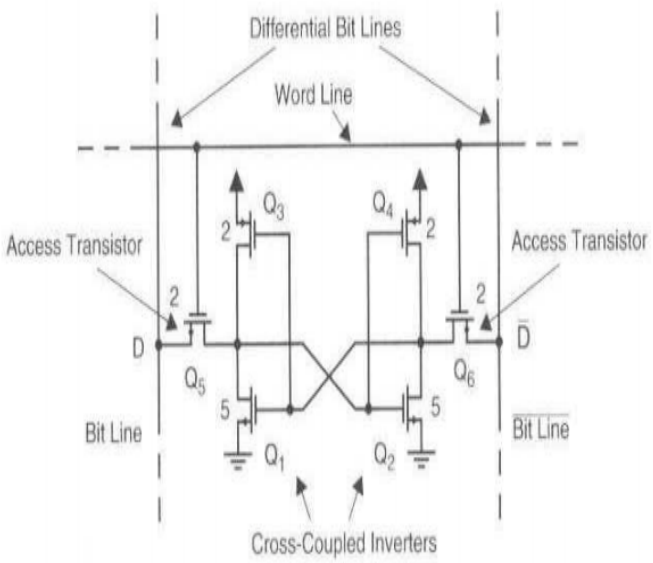
•WL at 0V, both access transistors are off

•Pre-charge one bit line high (D=B=VDD), the other low (D=B=0V)

•Correct WL will go high

•Source (B) of Q5 goes to 0->(VDD-Vt), and drain (B) of Q6 goes to VDD->0V

•Positive feedback takes over, and cell stores a "1" on D



- To read a "D=1"

•WL at 0V, both access transistors are off

•Pre-charge both bit lines high (either VDD or VDD/2)

•Correct W/L will go high

•If a "D=1 D=0" is stored, then W/L=H causes Q5 to pass VDD to Q2/Q4, 0V to Q1/Q3

•Charge flows Q6->Q2, thus discharging the B bit line voltage

•Charge flows Q3->Q5, thus charging the B bit line voltage

•Differential B to B voltage of 50-100mV is sensed at the sense amp; must be small so FF doesn't flop

IV. Improved SRAM Design

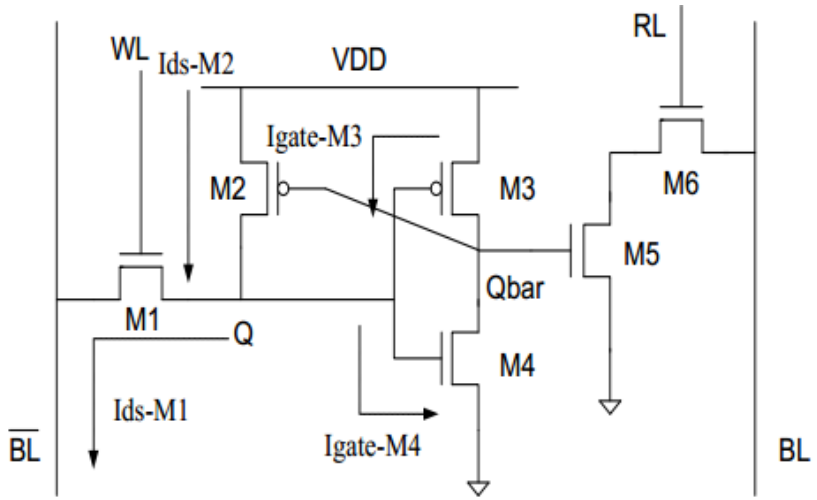
- Why focus on new approaches to robust SRAM and cache design?
  - Less leakage current ---> lower power consumption
  - High static noise margin ---> better stability
- Many different designs are proposed by researchers.
  - Separate read and write operation
  - Use voltage island to decouple the memory supply from logic supply
  - CNFET-based single-ended 6T SRAM cell

  …….

- 6T SRAM cell structure using separate write and read operation
  - Why to separate write and read operation?
    - With the technology scaling down, the conventional 6T cell stability is degraded and data is more easily interfered.
  - The schematic of new 6T cell SRAM

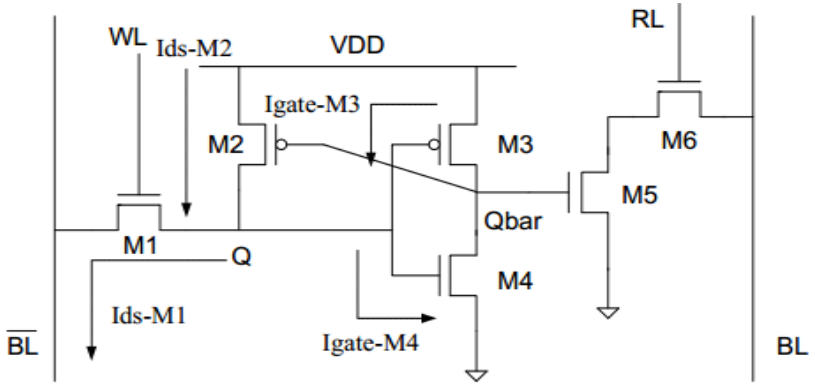
M5,M6 - read

M1~M4 - writ

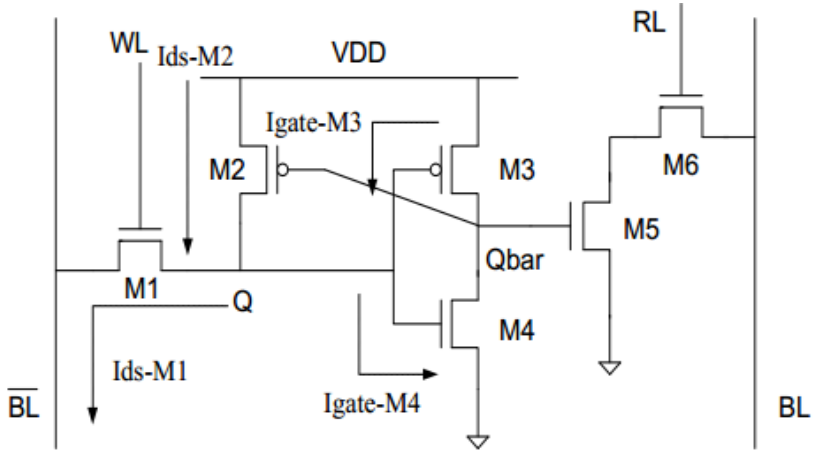


- Advantages of separate mechanism
  - No data interference between bit-line.
  - Only one bit-line works in each process, which reduces power consumption

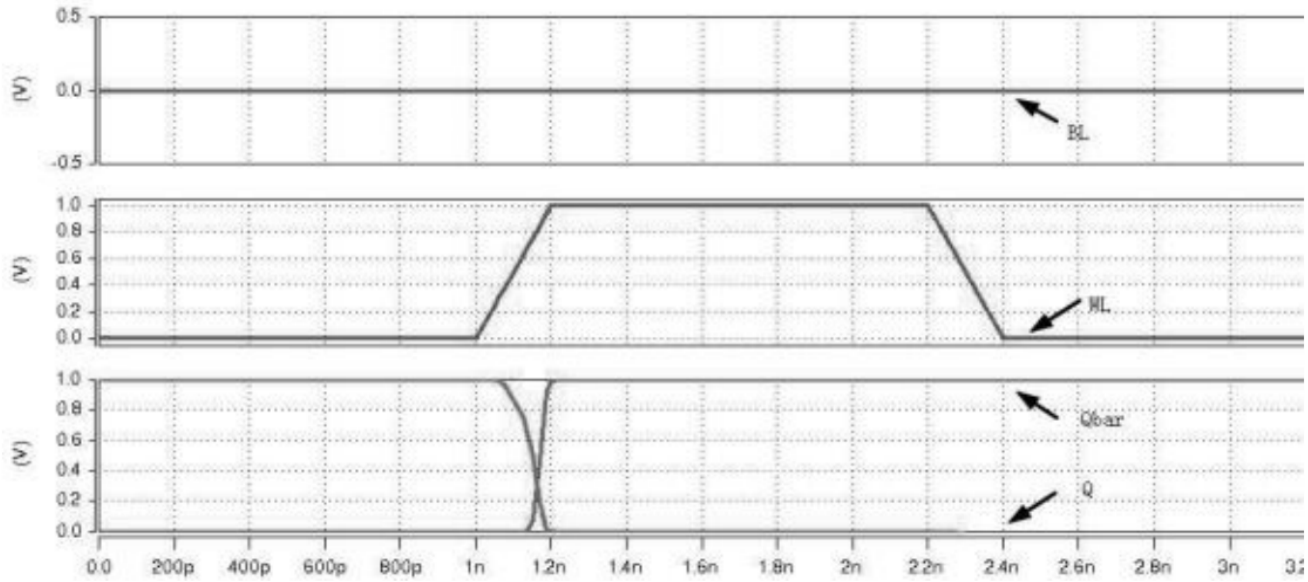
- Write and read operation
  - Write operation: word-line is asserted to Vdd and read-line maintained at GND
    - Data is 1: BLB-line is charged to VDD and Q node is pulled up to VDD-Vth by NMOS access transistor (M1),
    - therefore M4 will be on and Qbar node will be pulled down to GND, thus M2 will be on and then positive feedback is created by M2 and M4.



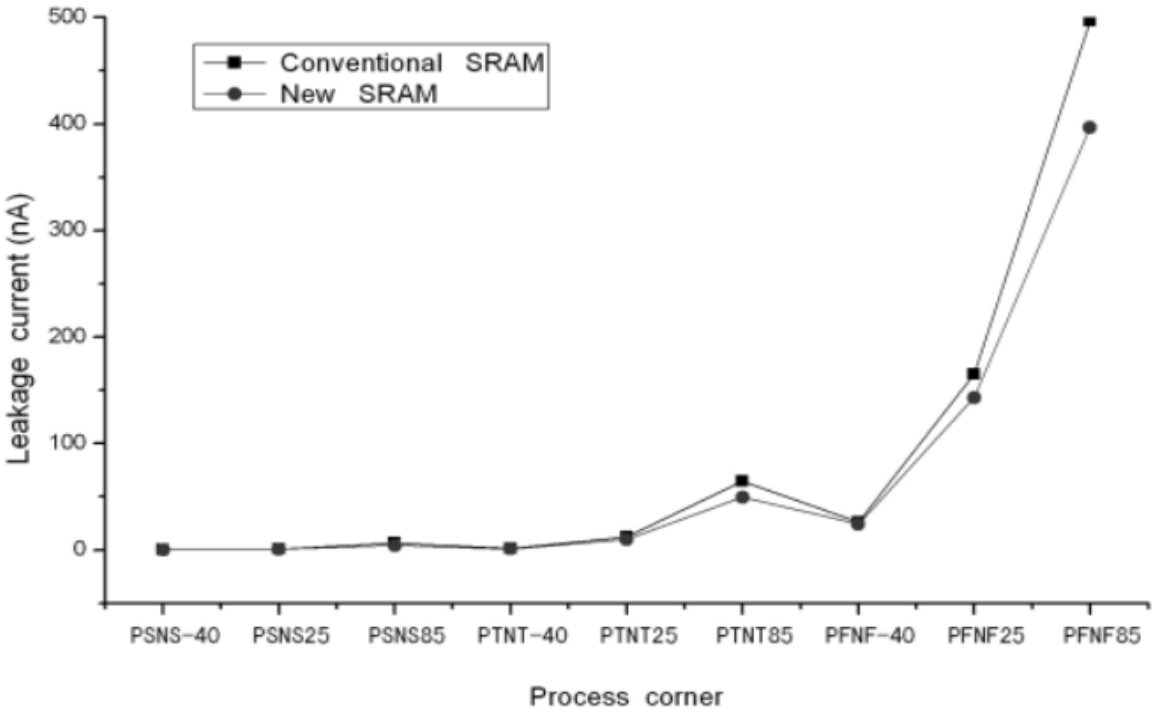
- Read operation: M6 is on, word-line is discharged to GND.
  - Bit-line is charged to Vdd before reading.
  - When the voltage of Q node is high, bit-line will still retain high level.
  - When Q node store '0', bitline will be discharged to GND. Then we can read data through sense amplifier which connects to the bit-line.



- Simulation and result
  - SRAM cell HSpice simulation
    - Standard 65nm CMOS technology with 1.0 v supply voltage for the case of write '0'.



- Leakage current



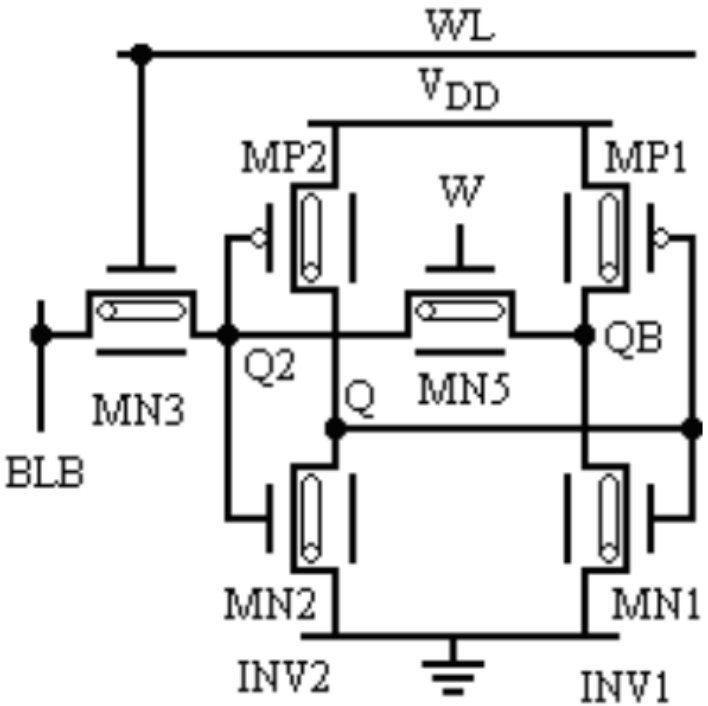
- Power consumption

State	Conventional cell （nw）	New cell （nw）
Write 0	12.0624	9.3542
Write 1	12.0624	12.0624
Read 0	9.3542	9.3542
Read 1	9.3542	9.3542

- Conclusion
  - New structure with separate write and read mode
  - Only one bit-line works in each operation
  - Simulation shows power consumption is reduced about 22.45%
  - New structure improves the static noise margin

- CNFET-based single-ended 6T SRAM cell
  - As CMOS reaching the scaling limits, the need for alternative technologies are necessary.
  - Nanotechnology-based fabrication is expected to offer the extra density and better performances.
  - Nanoscale electronic devices are demonstrated: carbon nanotube-based field effect transistor

- Schematic of CNFET-base 6T SRAM cell



- Simulation results and discussion

THE SRAM cells are simulated in HSpice using CNFET model and the 32nm PTM.

1. Read Delay

SRAM	TRA with QB storing "0" (ps)	V_{DD} (V)
Dif-6T	2.308	1
SE- 6T	0.5236	1

Read delay comparison while reading with QB storing "0"

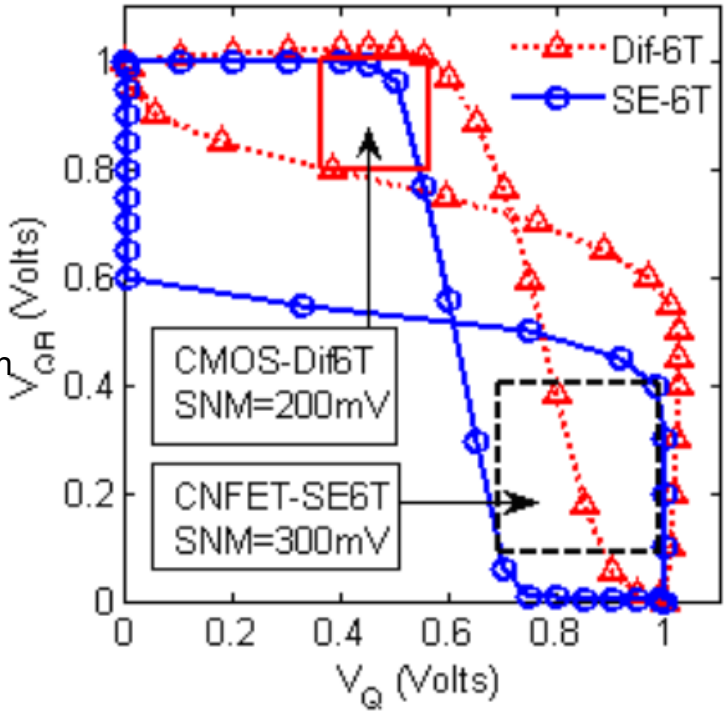
2. Write Delay

SRAM	T_{WA} while writing "0" to QB (ps)	T_{WA} while writing "1" to QB (ps)	V_{DD} (V)
Dif-6T	21.88	53.80	1
SE- 6T	24.96	103.9	1

3. Hold Stability

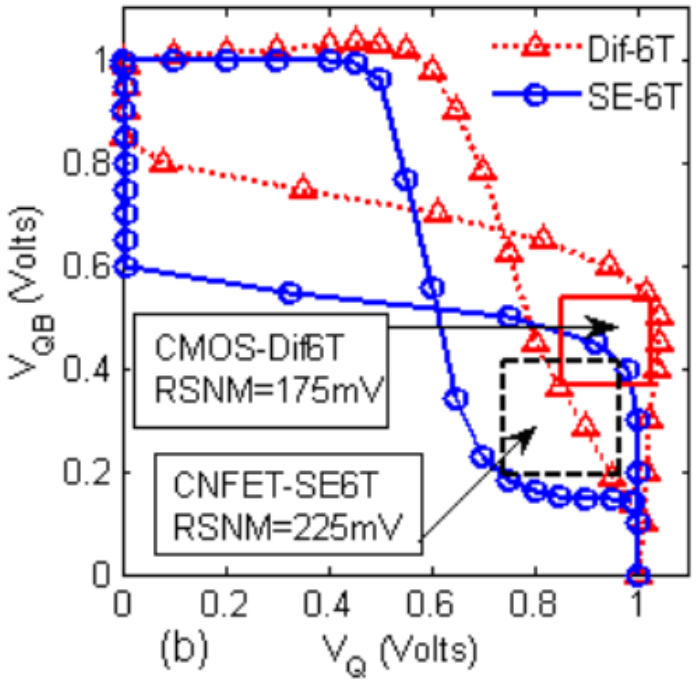
The hold stability of SRAM is determined by static noise margin.

50% improvement in proposed design



4. Read Stability

28.6% improvement in proposed design



➢ Conclusion

- Presents a CNFET-based single-ended 6T SRAM cell which help to save dynamic power
- The proposed design provides higher stability compared with the conventional SRAM, which shows 50% improvement in SNM and 28.6% improvement in RSNM

IV. Conclusion

- Research reading on low power cache design
- Understanding on the basic structure of conventional SRAM
- Learning the principles and operations of the SRAM
- Studying on different techniques for improving the SRAM to get lower power consumption and better stability

Reference:

[1] B. Mohammad et al, "Cache Design for Low Power and High Yield," IEEE 9th International Symposium on Quality Electronic Design, 2008

[2] Y. Wang et al, "A New 6-Transistor SRAM Cell for Low Power Cache Design" IEEE Solid-State and Integrated Circuit Technology (ICSICT), 2012

[3] A. Islam et al, "Low Active Power High Speed Cache Design," IEEE International Symposium on Electronic System Design, 2011

Thank you!

PRECISE INTERRUPT AND PIPELINING

PRESENTED BY:
AREEJ ALTHUBAITY

INTRODUCTION

- Interrupt handler: a piece of code that suppose to process the interrupt
  - Recognize the interrupting instruction
  - What actions should be taken
  - What are the system resources that could be needed to process the interrupted instruction
- precise interrupt
  - All the instructions before the interrupt are completed
  - The instructions after the interrupt are not issued yet.
  - PC is pointing to the interrupt instruction

PIPELINING AND PRECISE INTERRUPTS

- Handling interrupt is easier on sequential architecture
- Modern processors are using pipelining
  - complicates the implementation of precise interrupts

IN-ORDER INSTRUCTION COMPLETION (IIC)

- Instructions should be completed in the same order that they issued
- Handling interrupting instruction can be done by:
  1. Allowing the instruction to reach its last pipeline stage
  2. squashing the completion of all subsequent instructions from the program.
- Limitation:
  - Forcing in-order completion can result in performance degradation

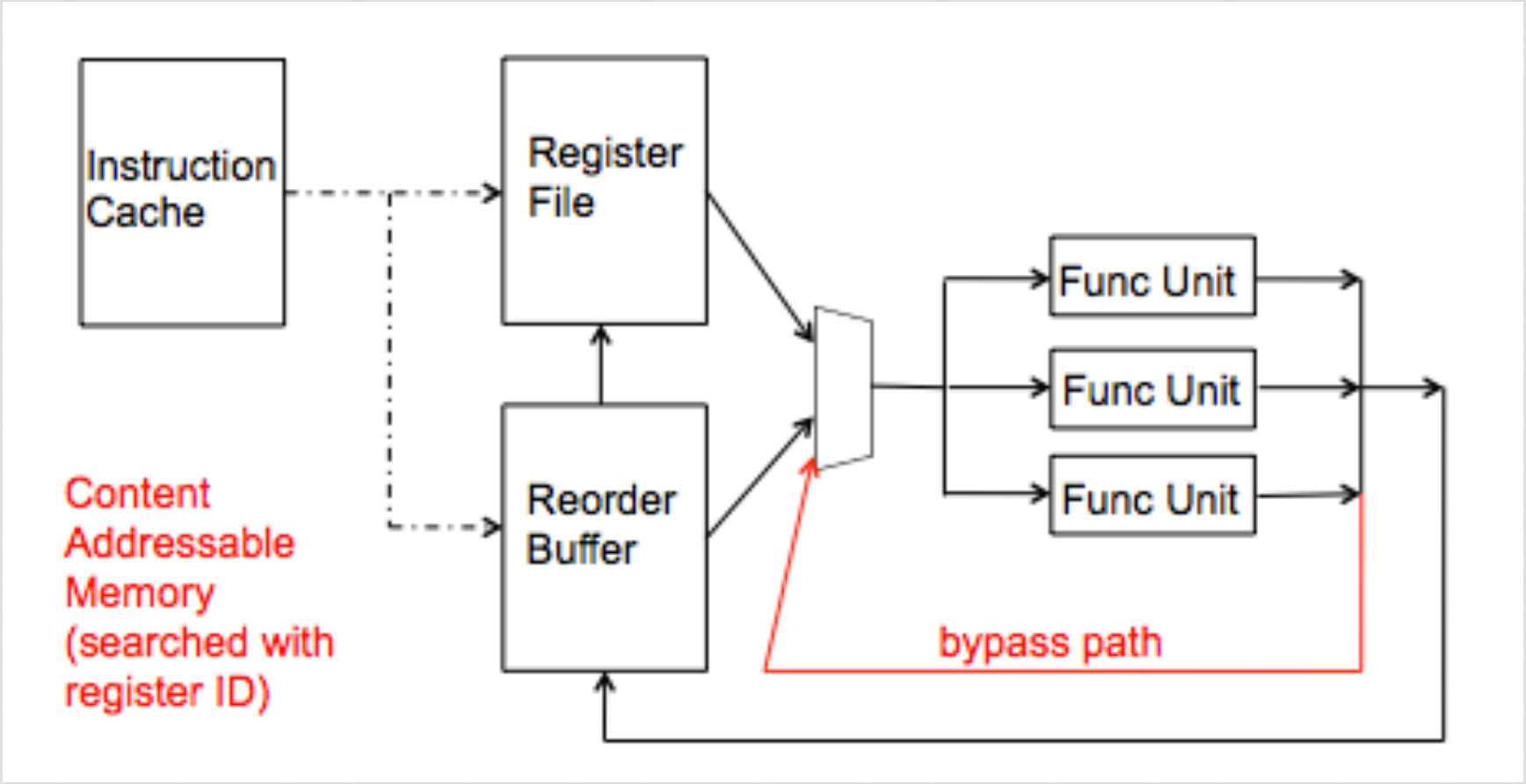
OUT-OF-ORDER COMPLETION

- The Reorder Buffer (ROB)
- History Buffer (HB)
- Future File (FF)
- The Instruction Window (IW)

THE REORDER BUFFER (ROB)

- Complete instructions out-of-order, but reorder them before making results visible to architectural state
- When instruction is decoded it reserves an entry in the ROB
- When instruction completes, it writes result into ROB entry
- When instruction oldest in ROB and it has completed, its result moved to register file or memory
- A register value can be in the register file, reorder buffer, (or bypass paths)

ROB CONT.


Instruction Cache
Register File
Reorder Buffer
Content Addressable Memory (searched with register ID)
Func Unit
Func Unit
Func Unit
bypass path

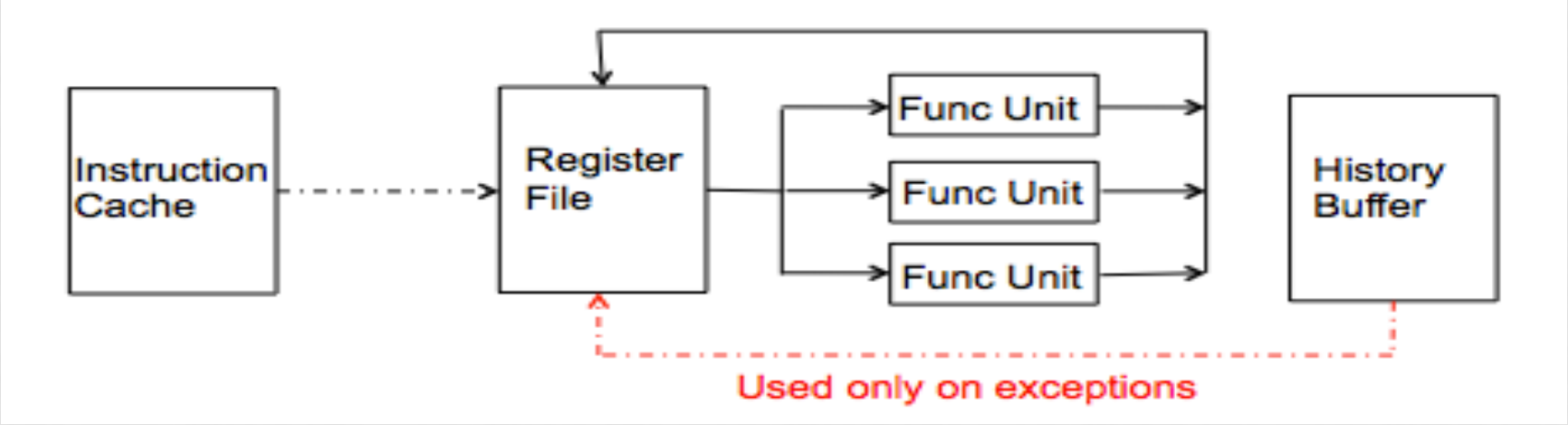
ROB PROS AND CONS

- Pros:
  - Simple for supporting precise exceptions
- Cons:
  - Results computed out of order held in ROB until all previously issued instructions have written to the register file
  - Instructions dependent on these results cannot be issued until these results are written

HISTORY BUFFER (HB)

- Update architectural state when instruction completes, but UNDO UPDATES when an exception occurs
- When instruction is decoded, it reserves an HB entry
- When the instruction completes, it stores the old value of its destination in the HB
- When instruction is oldest and no exceptions/interrupts, the HB entry discarded
- When instruction is oldest and an exception needs to be handled, old values in the HB are written back into the architectural state from tail to head

HISTORY BUFFER CONT.



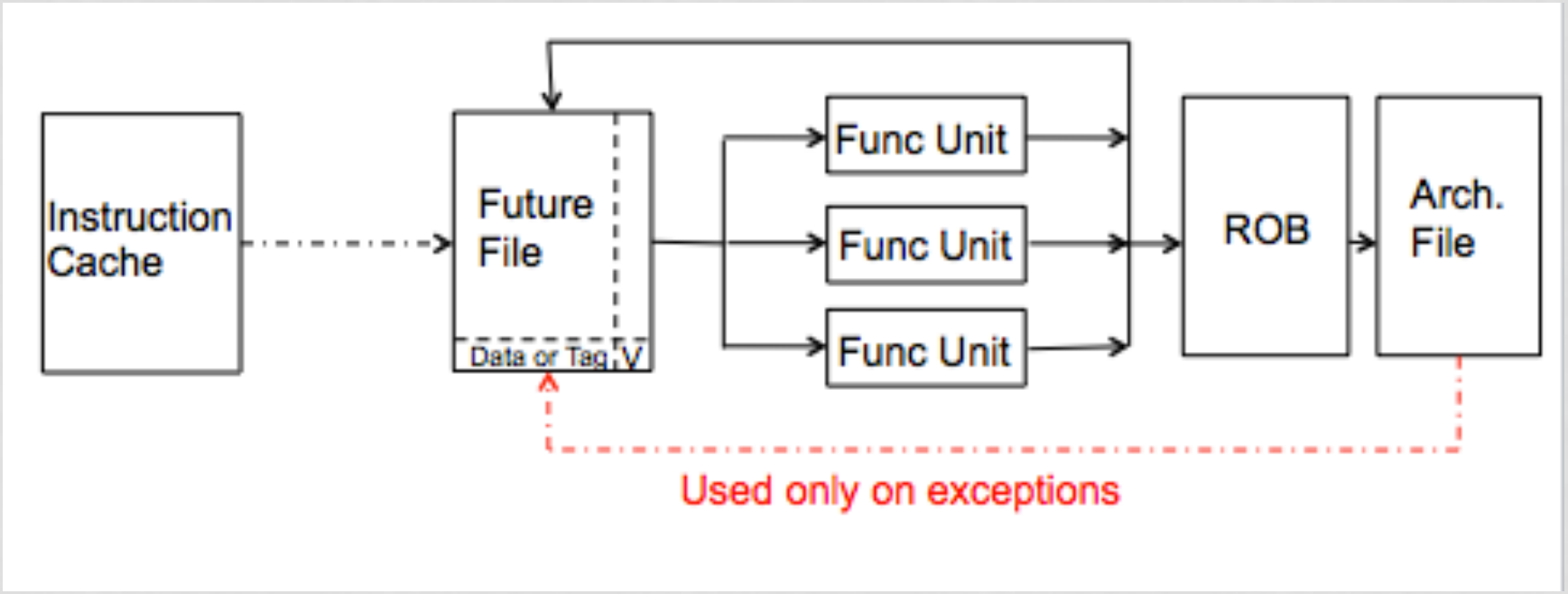
HB PROS AND CONS

- Pros:
  - Register file contains up-to-date values.
- Cons:
  - Increased interrupt handling latency

FUTURE FILE (FF)

- Keep two register files:
  - Arch register file: Updated in program order for precise exceptions
  - Future register file: Updated as soon as an instruction completes (if the instruction is the youngest one to write to a register)
- Future file is used for fast access to latest register values
- Architectural file is used for recovery on exceptions

FUTURE FILE (FF) CONT.



FUTURE FILE (FF) PROS AND CONS

- Pros:
  - No sequential scanning of history buffer: Upon exception, simply copy architecture file to future file
  - No need for extra read of destination value
- Cons:
  - Multiple register files + reorder buffer

THE INSTRUCTION WINDOW (IW)

- Allows out-of-order execution without requiring results to be committed in-order.
- Keeps track of the completion state of instructions in an instruction window which is saved at the time of the exception.
- After servicing the exception, the processor re-executes only the instructions that are still incomplete.

INSTRUCTION WINDOW PROS AND CONS

- Pros:
  - Only Saved and restored the contents of the instruction window and is not distributed across the processor.
- Cons:
  - more difficult exceptions would require the original instruction window to be saved and restored.