# Cache/Memory Optimization

- Krishna Parthaje

# Hybrid Cache Architecture Replacing SRAM Cache with Future Memory Technology

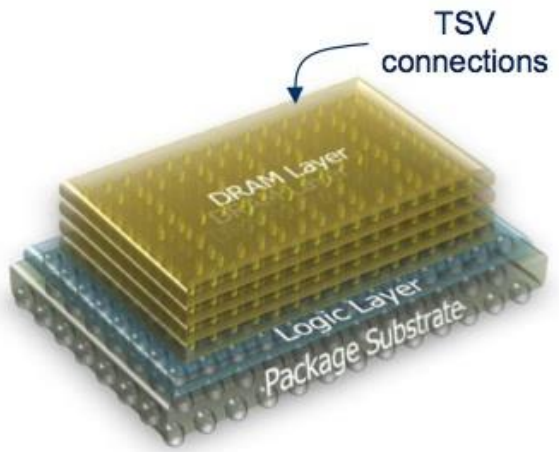
Suji Lee, Jongpil Jung, and Chong-Min Kyung
Department of Electrical Engineering,KAIST
Daejeon, Republic of Korea
ssooji555@kaist.ac.kr

# Introduction

- Motivation
- Memory Types and their working – PRAM, MRAM, DRAM, SRAM
- Overview
- Cache Model
  - ✓ Miss rate
  - ✓ Memory access time (MAT)
  - ✓ Average memory access time (AMAT)
  - ✓ Power consumption
- Results
  - Programs
  - Memory types
- Conclusion & Future Research

# Motivation

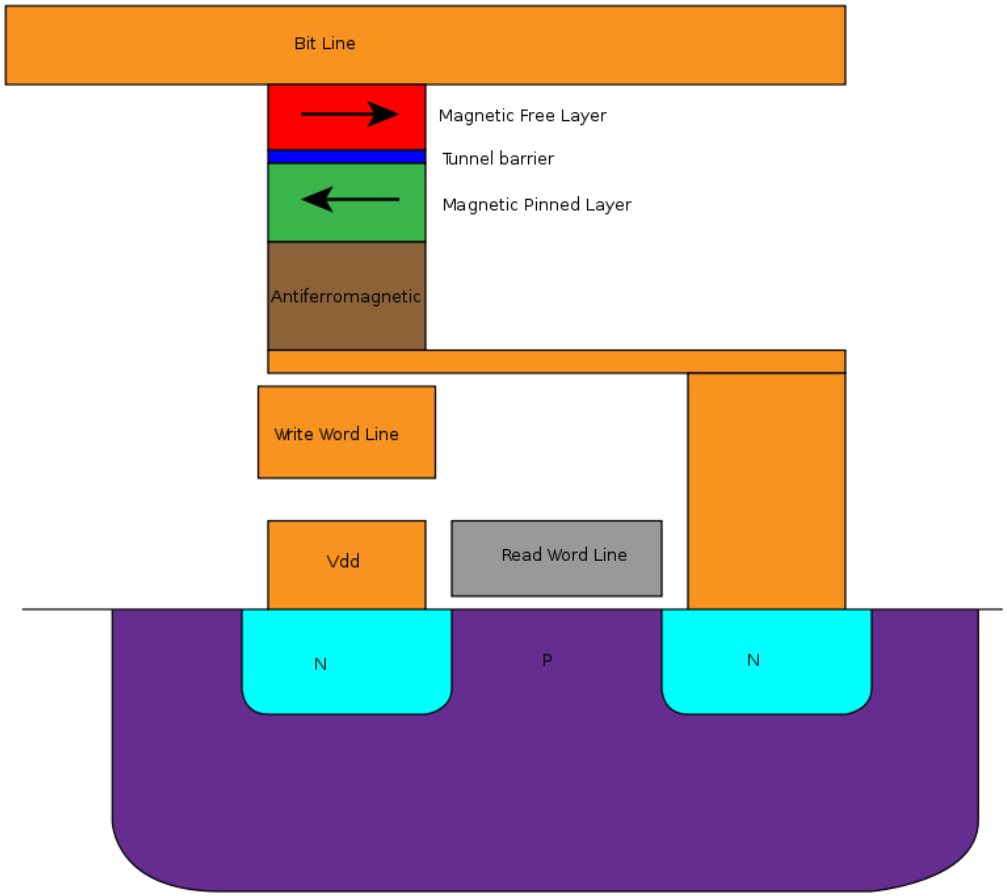
- Key Factors
  - ✓ Average Memory Access Time
  - ✓ Power Consumption
- SRAM disadvantages
  - – Low Density
  - – High Leakage
- Hybrid die stacking
- Optimized cache architecture for speed, power and area



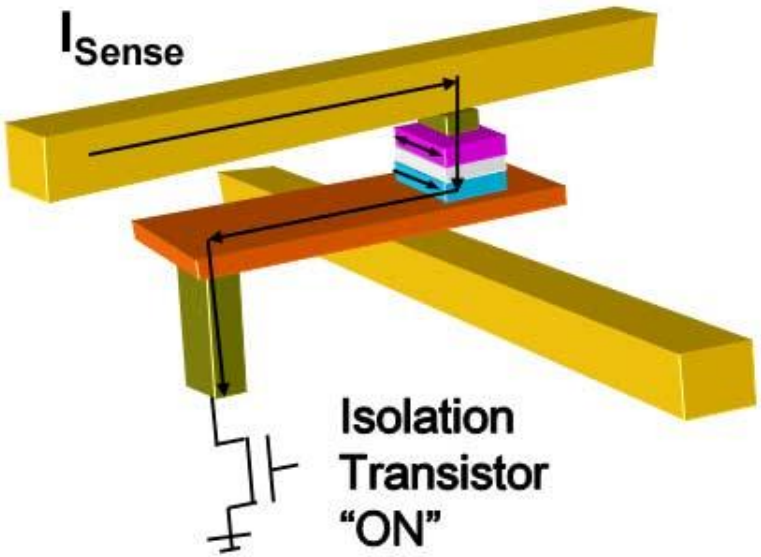

*Functioning prototypes in silicon TODAY*

# Memory Types

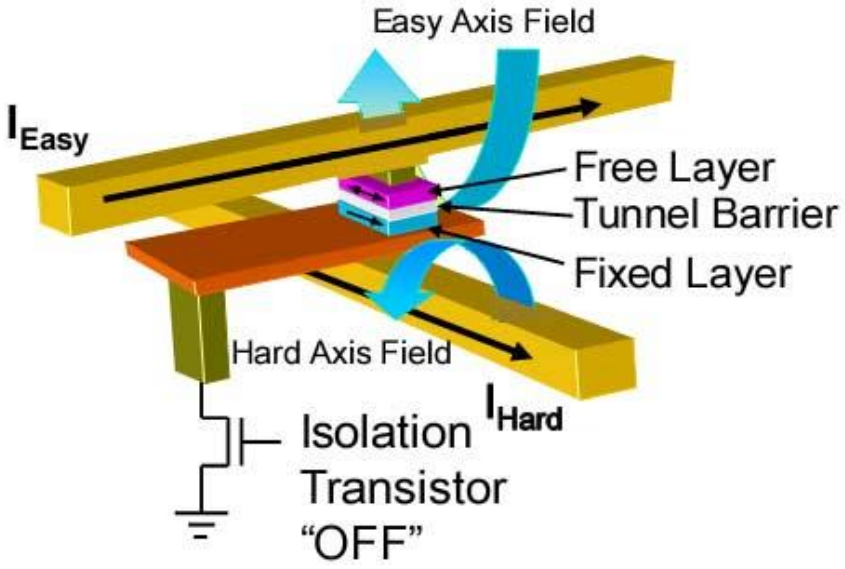
- MRAM – Magnetic RAM or Magneto resistive RAM



# Working

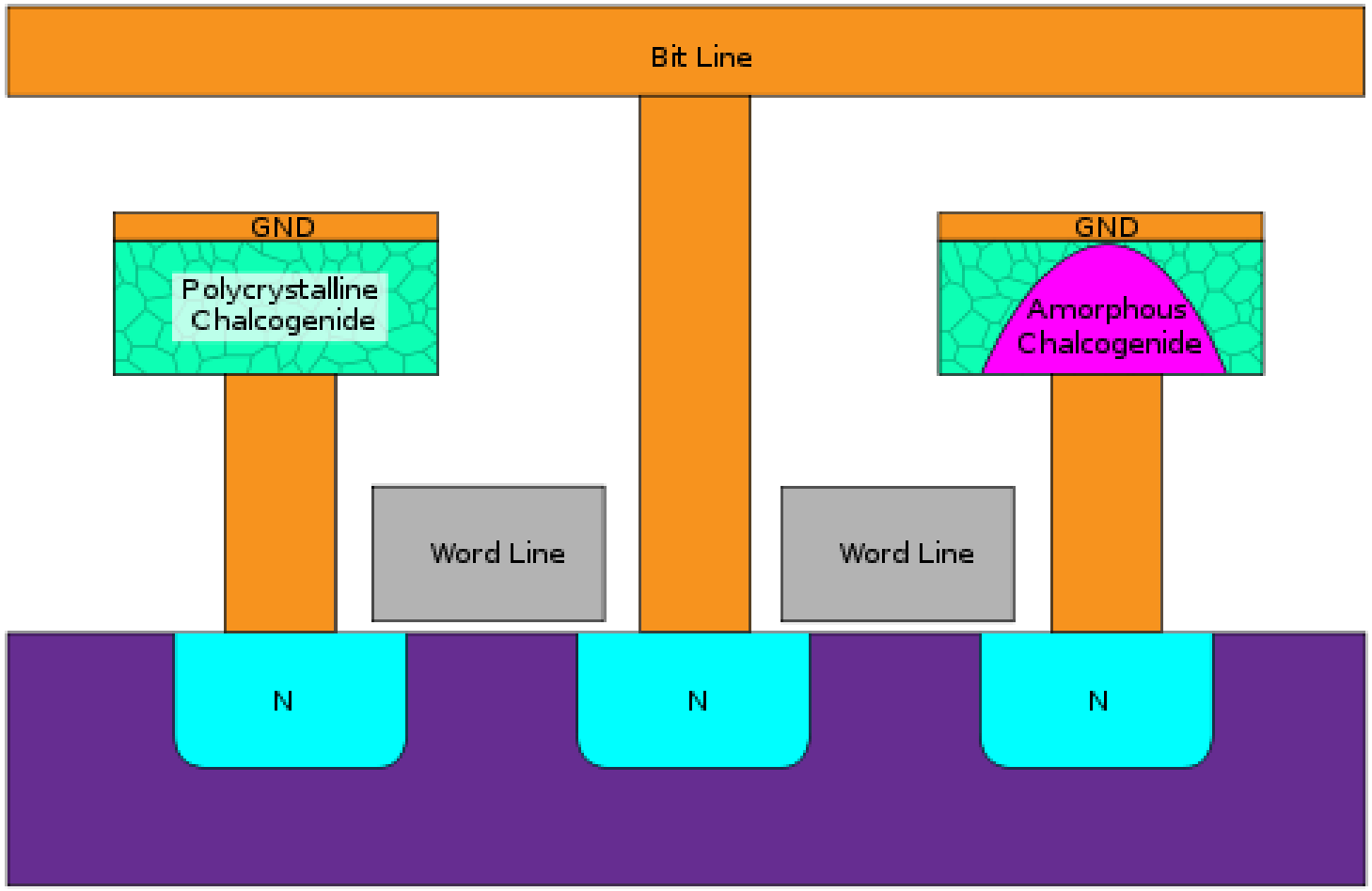



MRAM Read

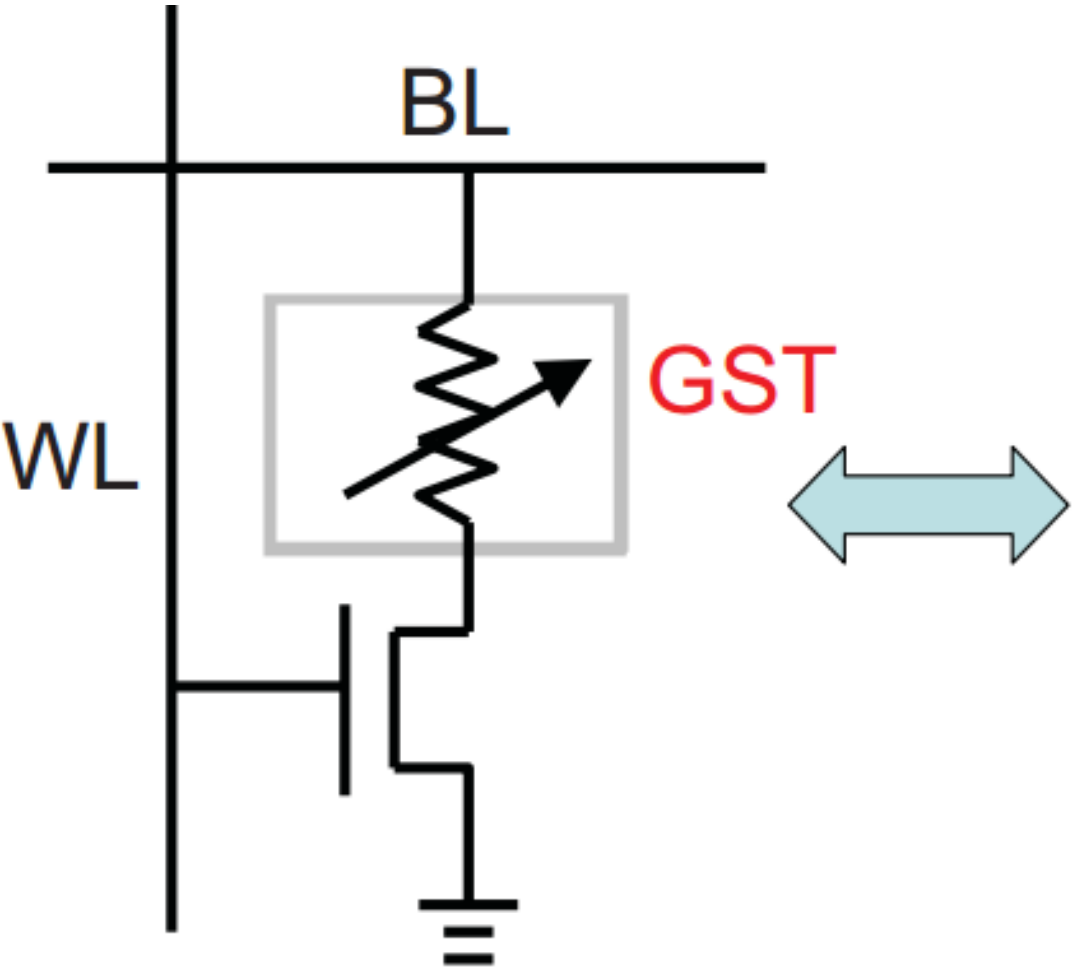


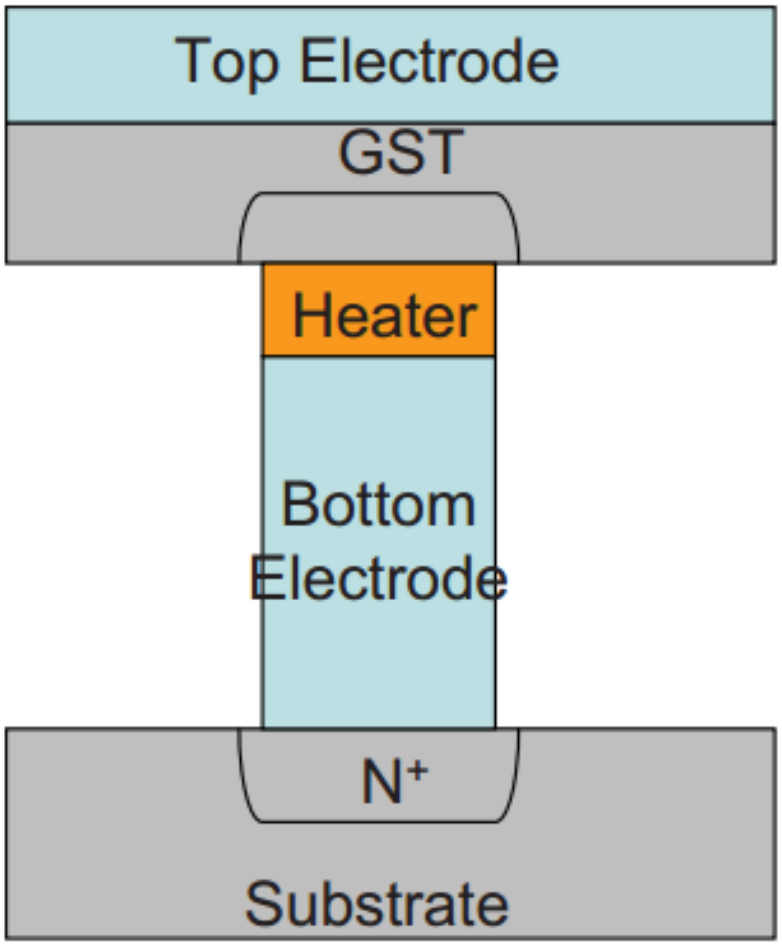

MRAM Write

- PRAM – Phase change RAM



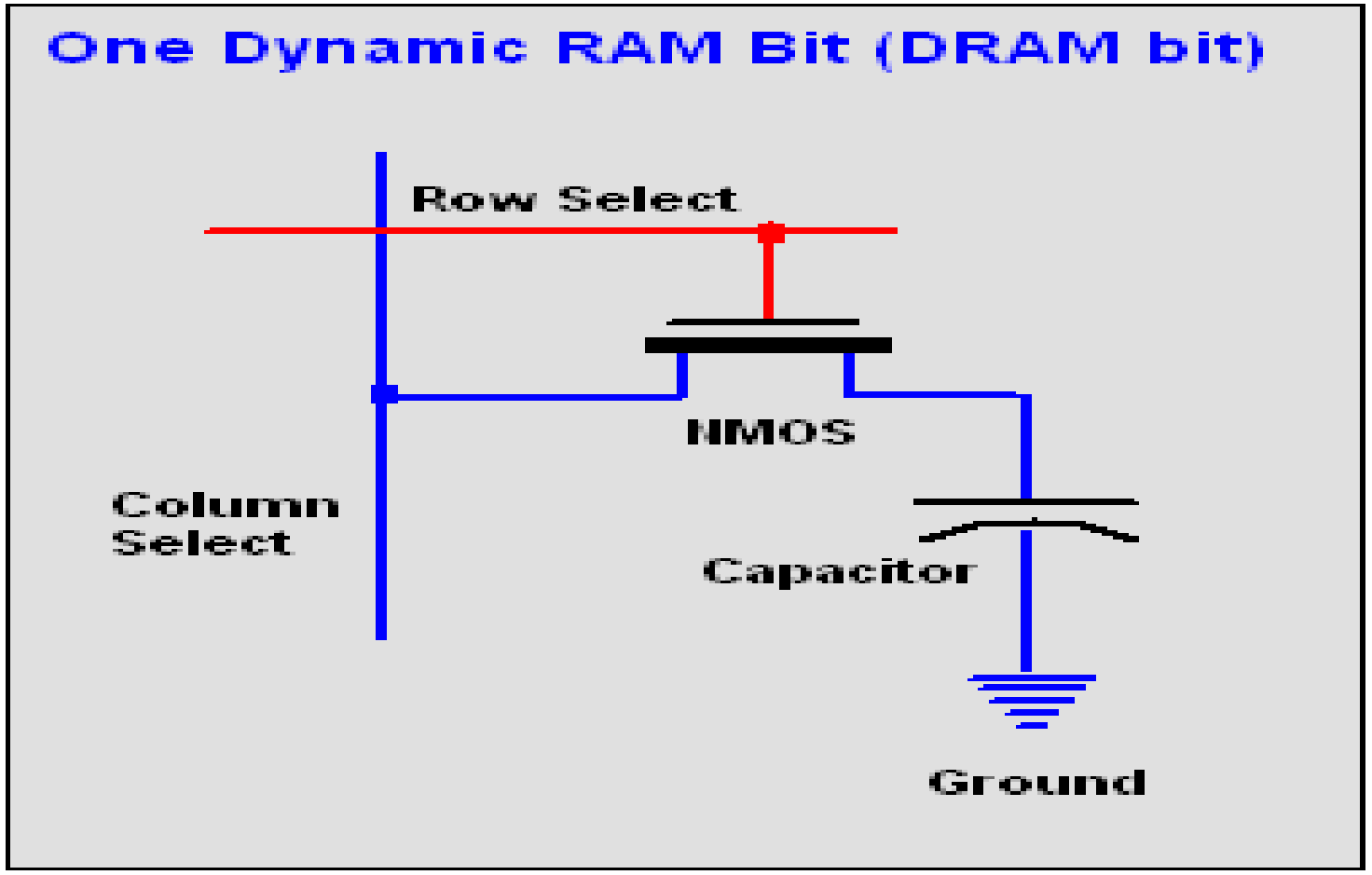
# Working


BL
WL
GST

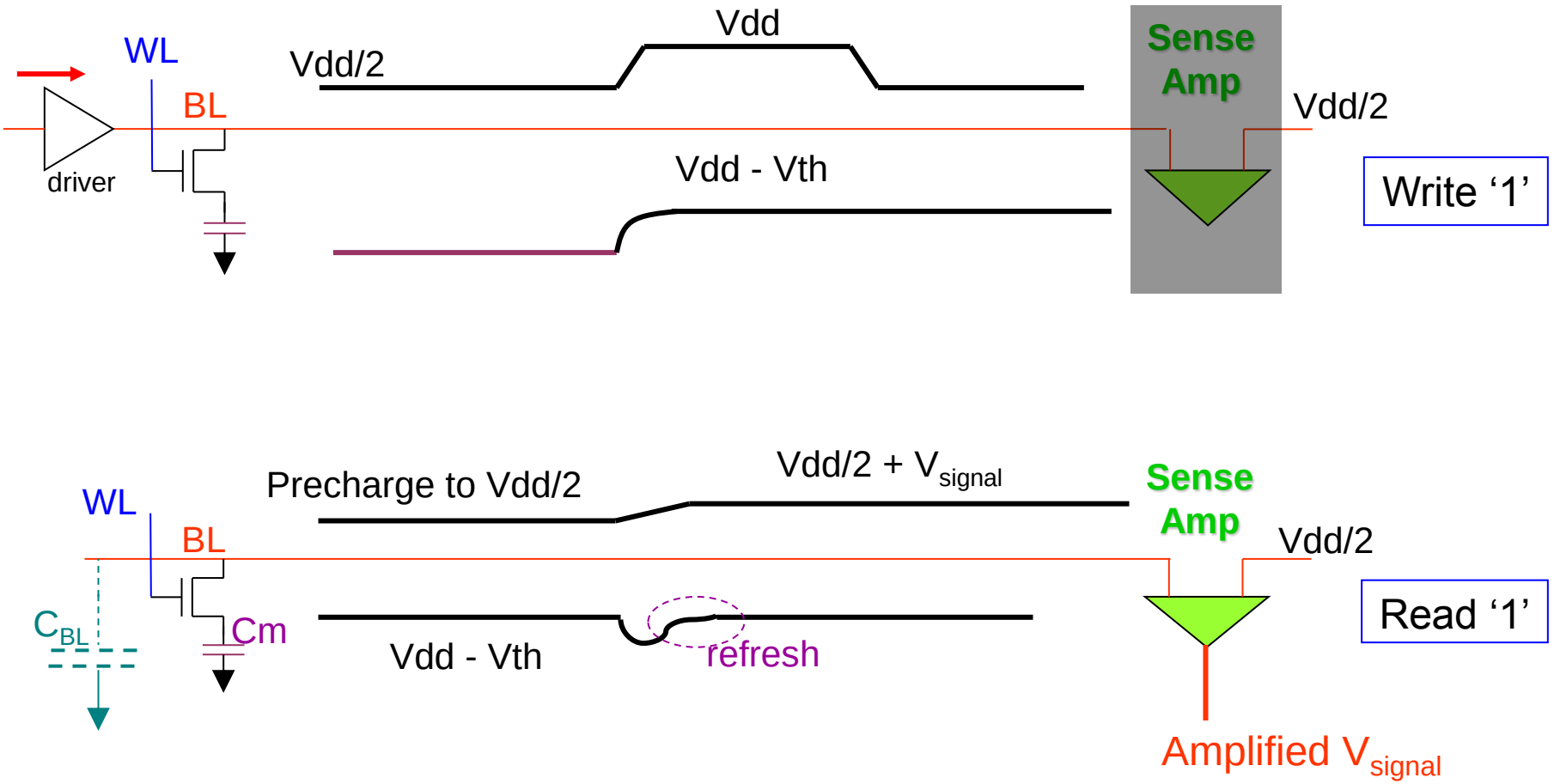

Top Electrode
GST
Heater
Bottom Electrode
N+
Substrate

- DRAM – Dynamic RAM

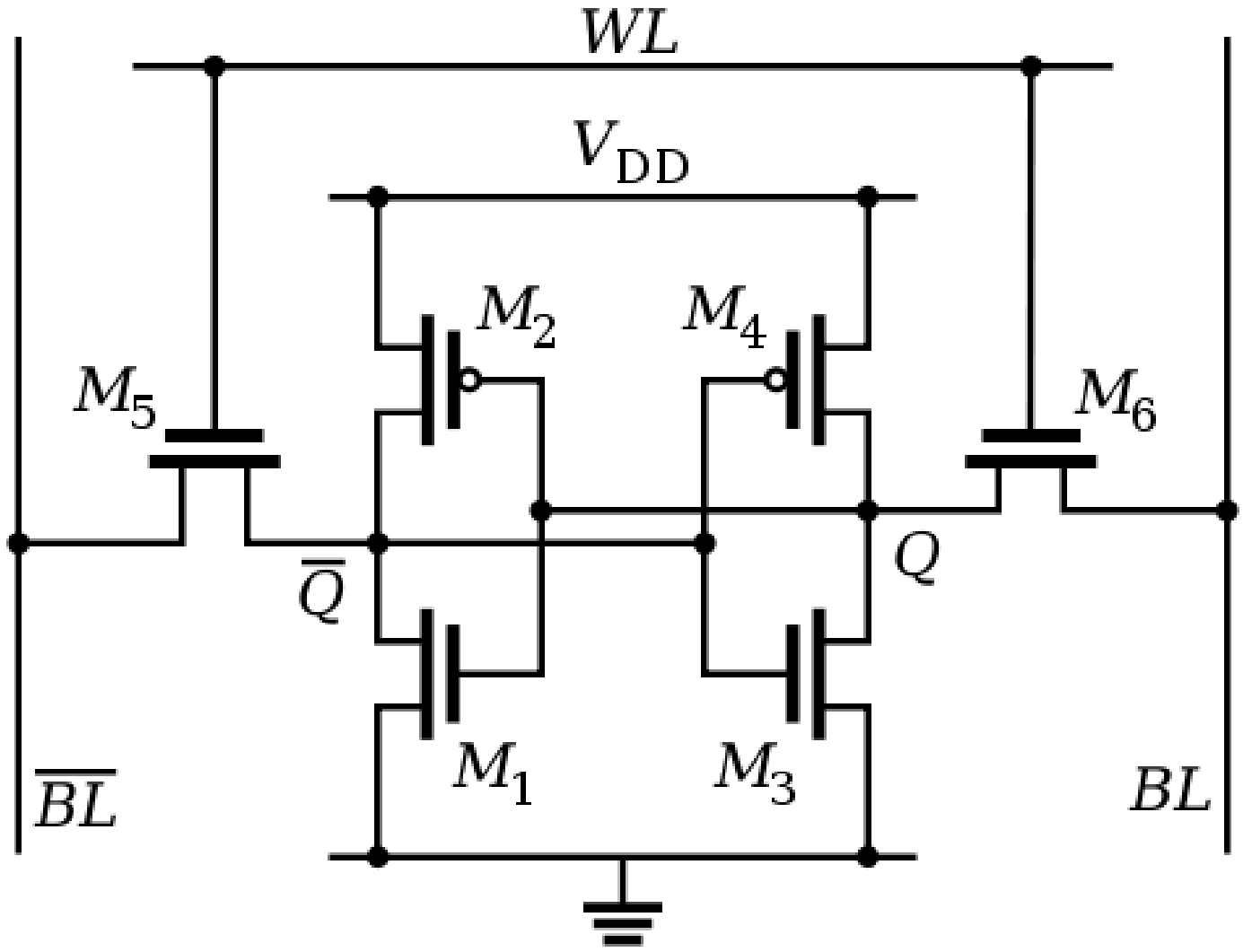
From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.



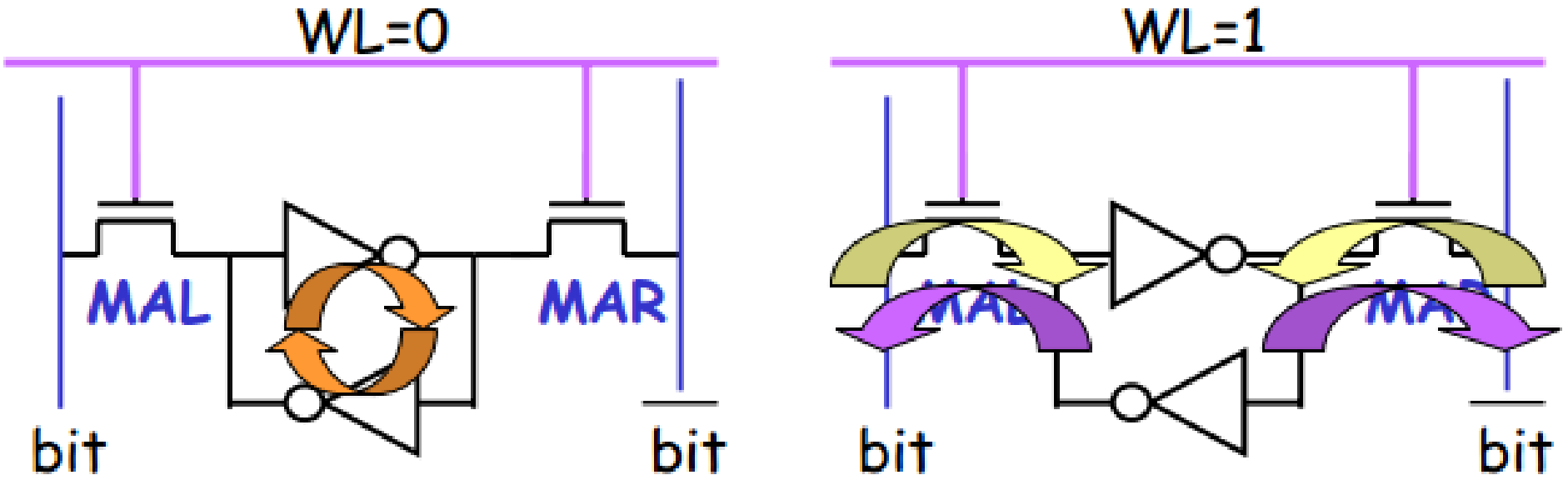
# Working


WL
BL
driver
Vdd/2
Vdd
Vdd - Vth
Sense Amp
Vdd/2
Write '1'
Precharge to Vdd/2
Vdd/2 + V signal
WL
BL
C BL
Cm
Vdd - Vth
refresh
Sense Amp
Vdd/2
Read '1'
Amplified V signal

- SRAM – Static RAM



# Working

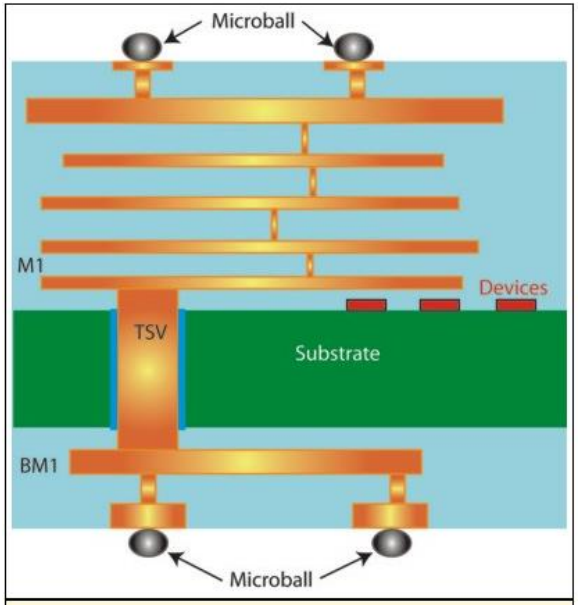


# Comparison of Different Memory Technologies

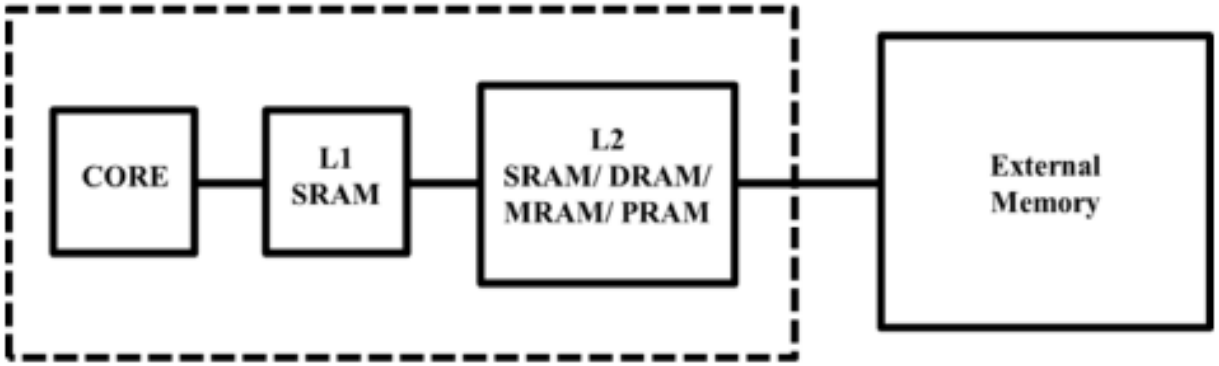
| Features | *SRAM* | *DRAM* | *MRAM* | *PRAM* |
|---|---|---|---|---|
| Density | Low | High | High | Very high |
| Speed | Very Fast | Fast | Fast read<br>Slow write | Slow read<br>Very low write |
| Dyn.Power | Low | Medium | Low read:<br>High write | Medium read<br>High write |
| Leak.Power | High | Medium | Low | Low |

# Developments

- Hybrid die stacking
  - Multiple layers of die are stacked with through-silicon-via (TSV)
  - Improves speed, power and performance with 3D integration
  - Reduces area size & wire length
  - Provides dense packaging
  - Efficient mixing of different process technologies
  - Improves routability



# Overview




3D hybrid cache architecture

- Assumption – 50 ns for external memory access and area limited to 100 $\mathrm{mm}^2$.
- 45nm technology node.
- To determine the best cache capacity for small power consumption and minimum average memory access time.

# Cache Model

- AMAT, power consumption and area is modeled to assess the performance

**A. Miss Rate**

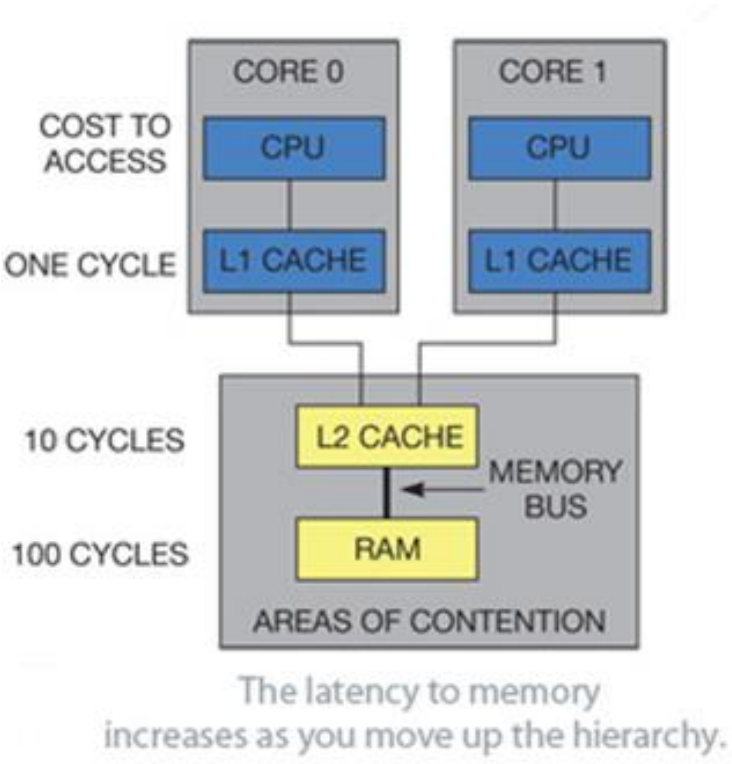
- Decreases with increase in cache capacity
- Depends on benchmark programs too
- Equation to express miss rate as a function of capacity

$$m(c) = \mu_0 \, . \, c^{-\mu_1}$$

- $\mu_0$ increases, overall miss rate also increases
  - Indicates more access of L2 than L1
- $\mu_1$gives a description of the relation between miss rate and cache capacity
  - When $\mu_1$ increases the impact of capacity increases
    - $0.3 < \mu_1 < 0.7$

**B. Memory Access Time (MAT)**

- CACTI 6.0 used to obtain data to model MAT

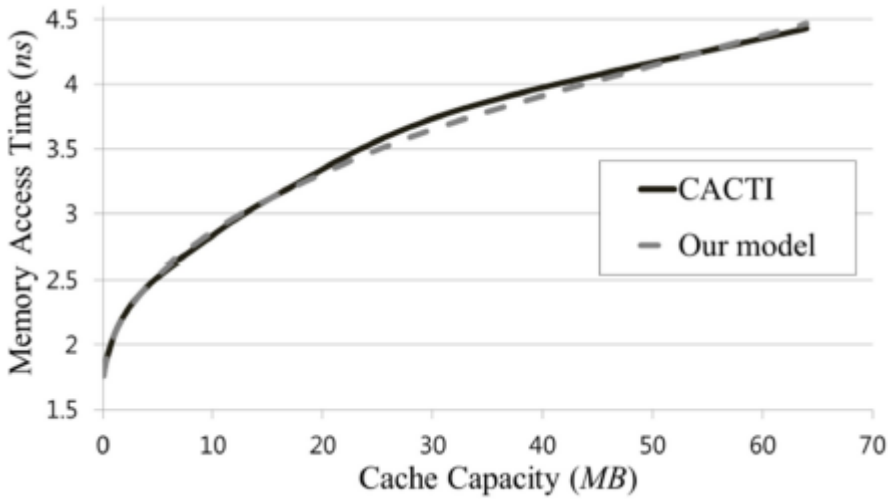



The latency to memory increases as you move up the hierarchy.

- Equation to represent the MAT according to cache capacity

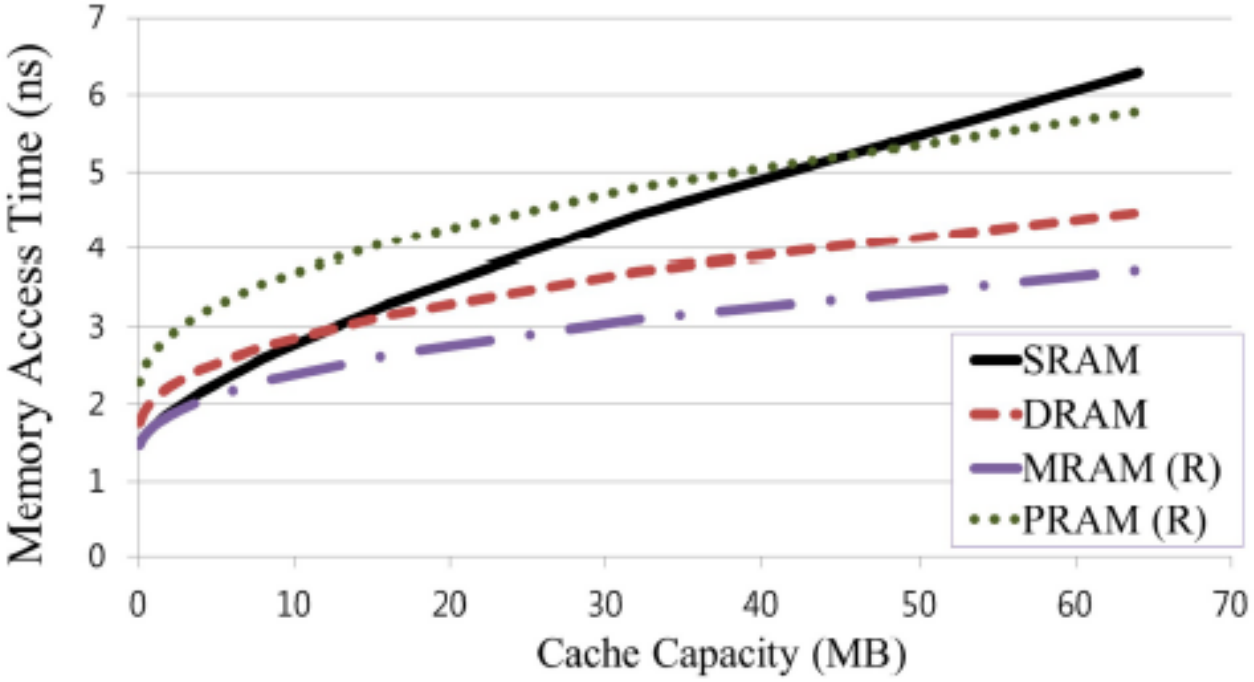
$$T(c) = \alpha . c^{\beta} + \gamma$$

- Error in this model is around 1.71% on average. The range would be from .08% to 8.36%



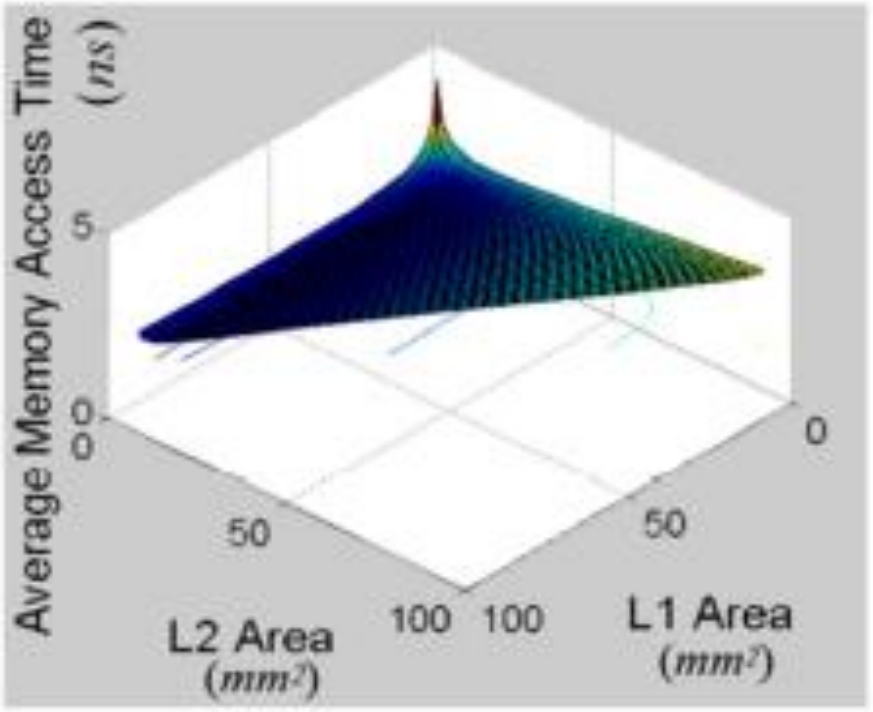

Memory access time using SRAM for L2 cache

- The figure shows the elasticity of the MAT compared to the capacity of each RAM type
- It increases when the capacity is over a few Mbytes
- Speed of MRAM/DRAM is comparable to SRAM due to the large cache size

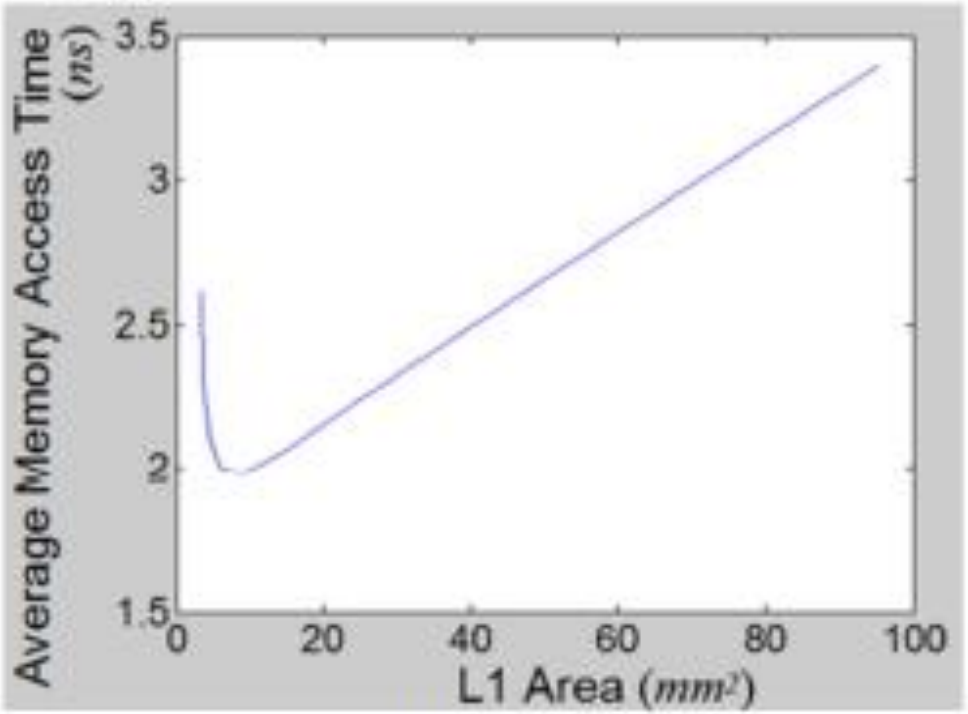



Memory access time of each memory type

# C. Average Memory Access Time




(a)




(b)

(a) overall 3D graph, (b) section of maximum area

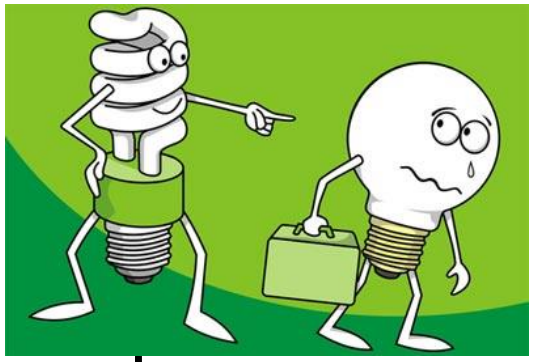
- Two equations used to model the AMAT, uses MAT and miss rate as shown in the equations

1. AMAT = h ($c_1$).$T_1$ ($c_1$) + m ($c_1$)*(h ($c_2$) $T_2$ ($c_2$) + m ($c_2$)
2. $T_i(c_i) = \rho.T_i^r(c_i) + (1 - \rho)T_i^w(c_i)$

- The curve shows a sharp reduction and then increases
- Low L1 range is accompanied with smaller AMAT as the area is increased
- On the other hand AMAT increases as L1 area increases in the high L1 range.

## D. Power Consumption

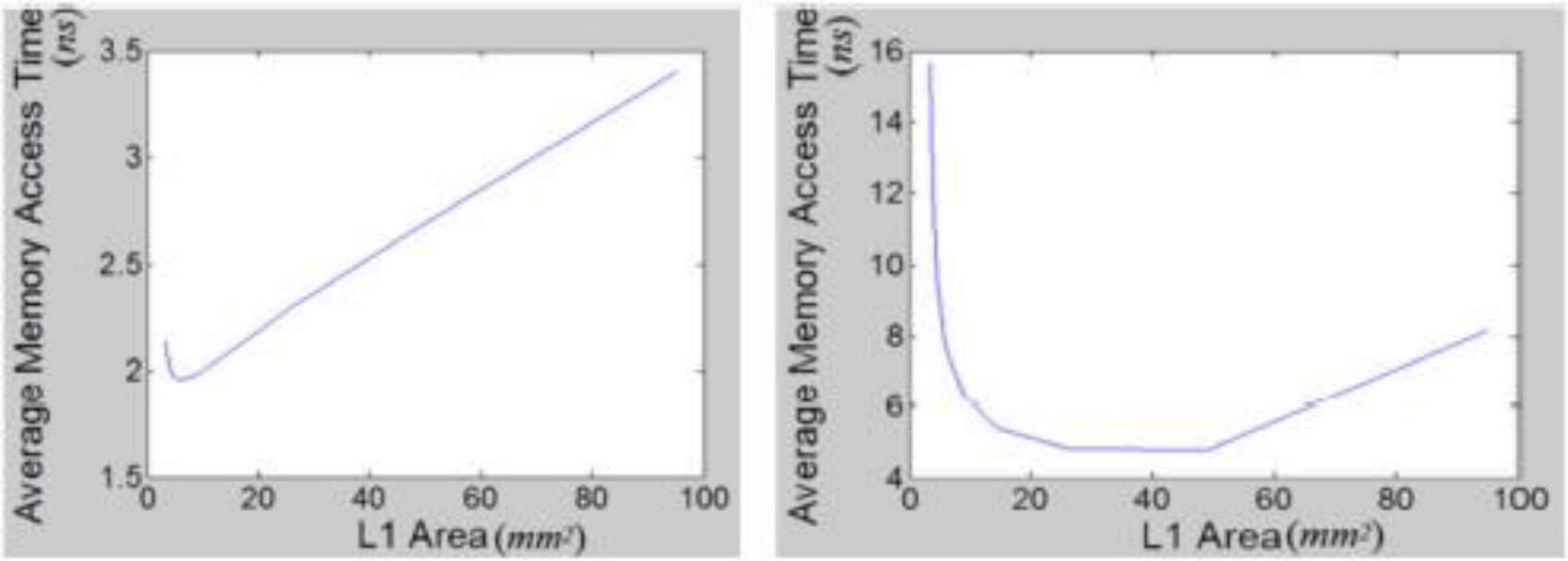
- According to linear equations the power consumption is modeled after the data extracted from the CACTI 6.0
  - $E_{dyn}(c) = \delta . c + \theta$
  - $E_{static}(\mathrm{c}) = \rho . \mathrm{c} + \sigma$



- Power consumption is formulated as
  1. $P(c_1, c_2) = P_1(c_1) + P_2(c_2)$
  2. $P_1(c_1) = N_{access} . \mathrm{h}(c_1) . E_{dyn1}(c_1) + P_{static1}(c_1)$
  3. $P_2(c_2) = N_{access} . m(c_1) . \mathrm{h}(c_2) . E_{dyn2}(c_2) + P_{static2}(c_2)$

# Results

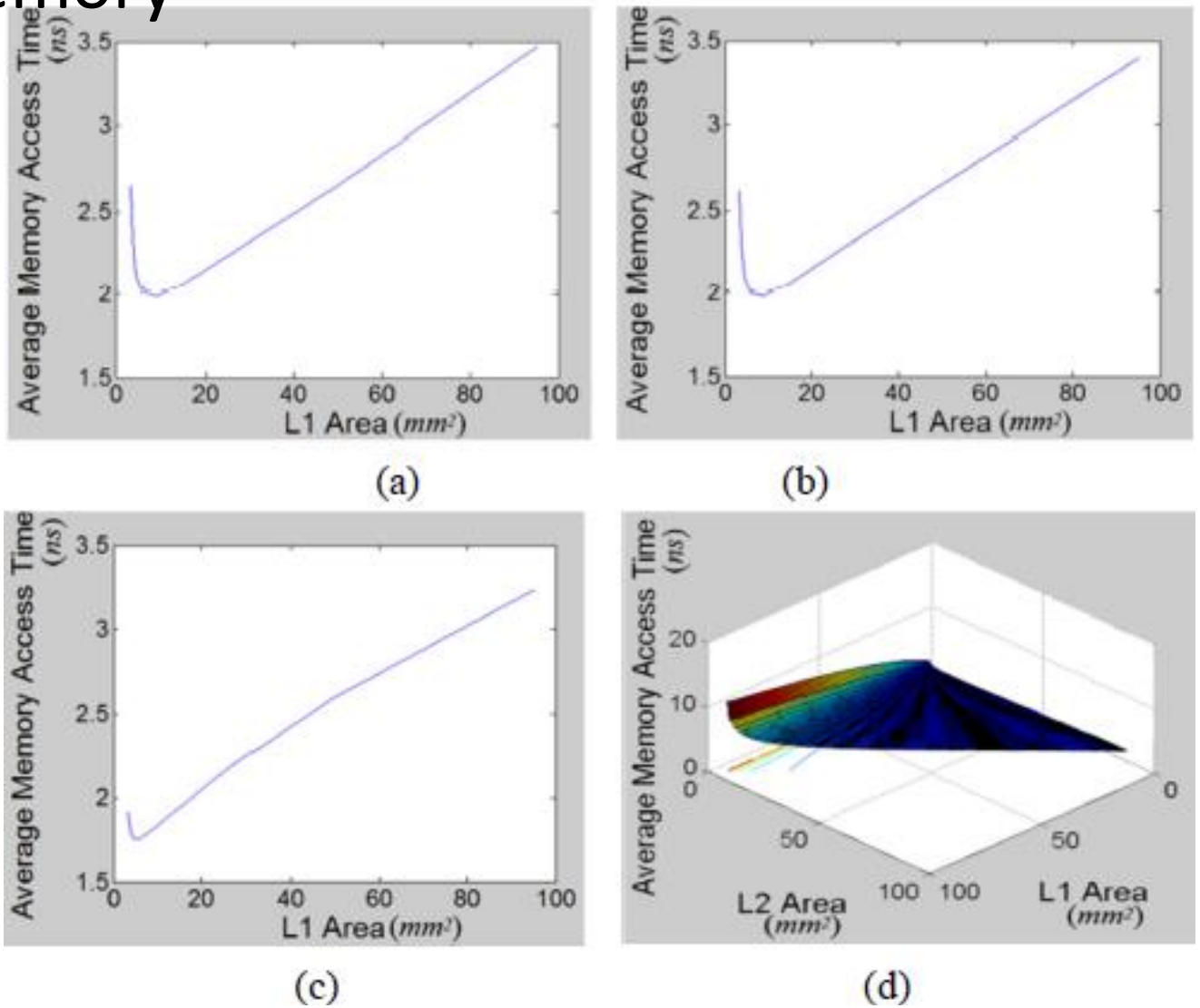
- Program




Average memory access time for each program:
(a) equake, (b) face_rec

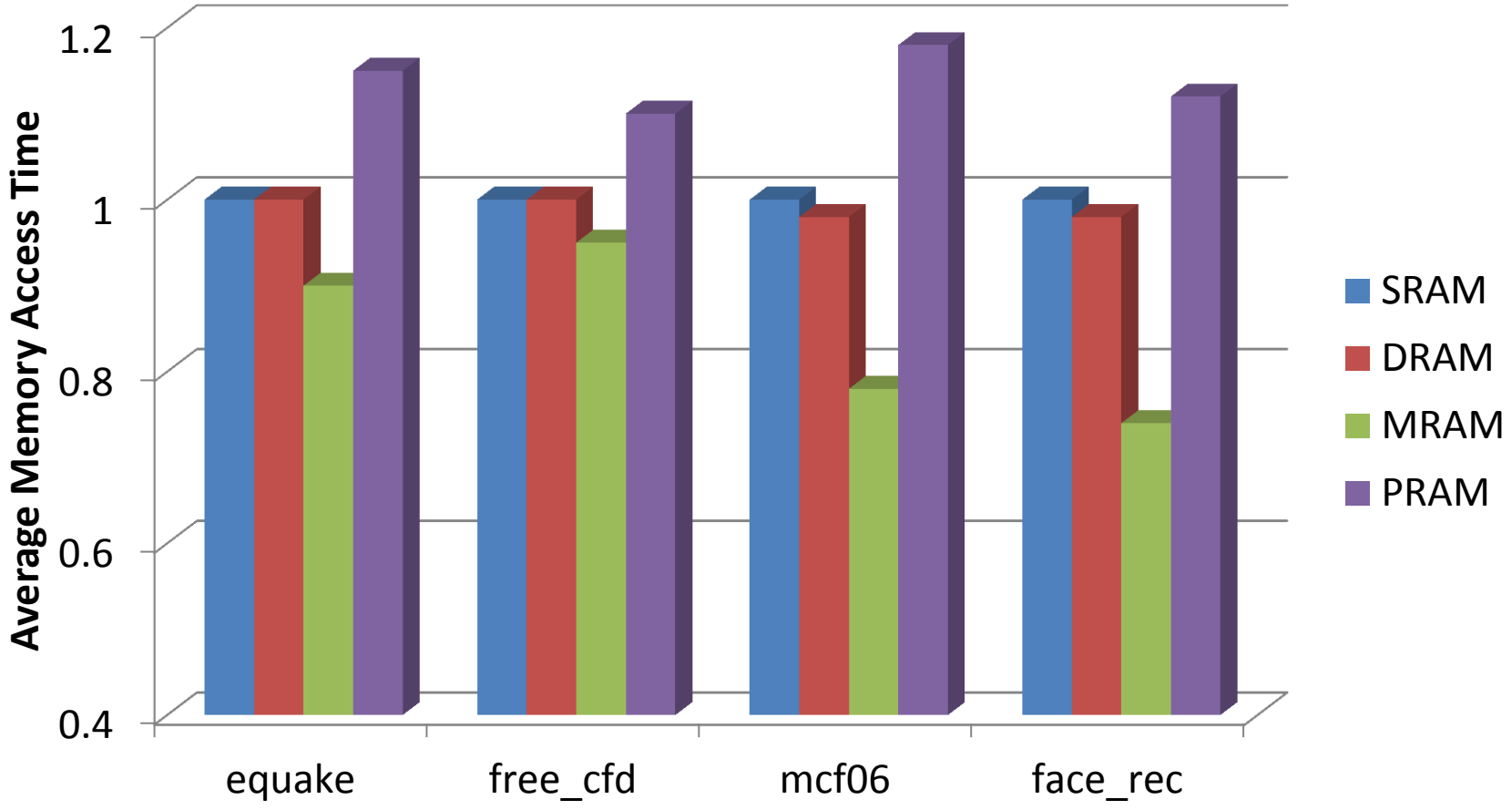
- If miss rate is low, access hits to L1 is high
- AMAT increases with L1 capacity
- More data hits slow L2 if miss rate is high
- Possible to make reconfigurable hardware
- Useful for building 3D multi-core microprocessor designs
- Each cache level could be separated into multiple partitions and each partition adjusted dynamically, thus reducing overall AMAT

- Memory




Average memory access time for L2 cache using (a) SRAM, (b) DRAM, (c) MRAM, and (d) PRAM

# Average Memory Access Time



- From simulation least AMAT is by MRAM when used as L2 cache
- There is 16.9% reduction in AMAT and 15.2% reduction in power consumption.
- DRAM is rational alternatives for lowest power consumption which is  33.0% on average and AMAT is reduced by only 2%.
- For PRAM, minimum AMAT corresponds with minimum area of PRAM.
- PRAM is too slow for a cache memory
- Therefore not suitable for L2 cache but can be used for large storage

# Conclusions

- Different types of hybrid cache architectures have been compared
- SRAM is selected for L1 and other types of memory selected for L2 cache
- Several benchmark programs used to test AMAT and power consumption
- L1:SRAM, L2: MRAM offers 16.9% AMAT reduction and
- 15.2% power saving
- L1: SRAM, L2: DRAM offers 33.0% power saving than homogenous SRAM architecture
- PRAM as L2 cache is not suitable

# Future Improvements

- PCRAM has got an access time in the order less than few ns
- Could replace SRAM in L1 itself
- MRAM speeds coming close to SRAM could be a paradigm shift if SRAM got replaced
- Multithreaded and multi programmed applications need to be tested on these memory technologies

# Bandwidth-Aware Reconfigurable Cache Design with Hybrid Memory Technologies

Jishen Zhao, Cong Xu, Yuan Xie

Computer Science and Engineering Department, Pennsylvania State University

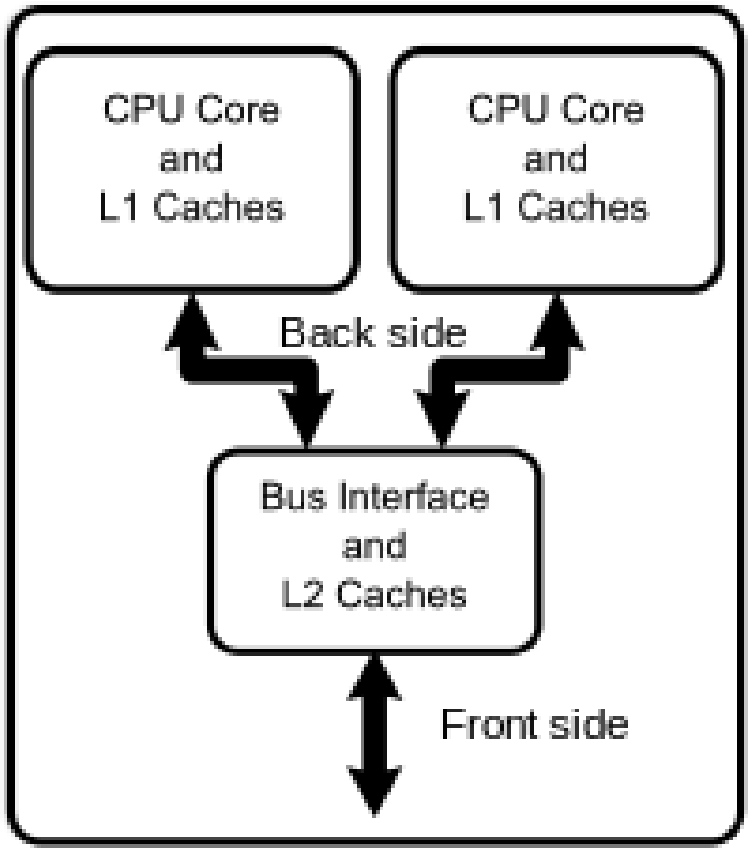
{juz138,czx102,yuanxie}@cse.psu.edu

# Introduction

- Motivation
- BARCH – **B**andwidth **A**ware **R**econfigurable **C**ache **H**ierarchy
- Design Methodology
  - Hybrid Cache Hierarchy
  - Reconfiguration
  - Prediction Engine
- Experimental Setup
- Results
- Conclusion & Future Improvements

# Motivation

- CMP scaling inefficient due to memory bottleneck
- Additional cycles is wasted in accessing off-chip memories
- A good cache hierarchy design helps to ease the pressure off the external memory and reduces the latency
- 3D integration technology helps to stack memories to provide high bandwidth to the cores on the chip



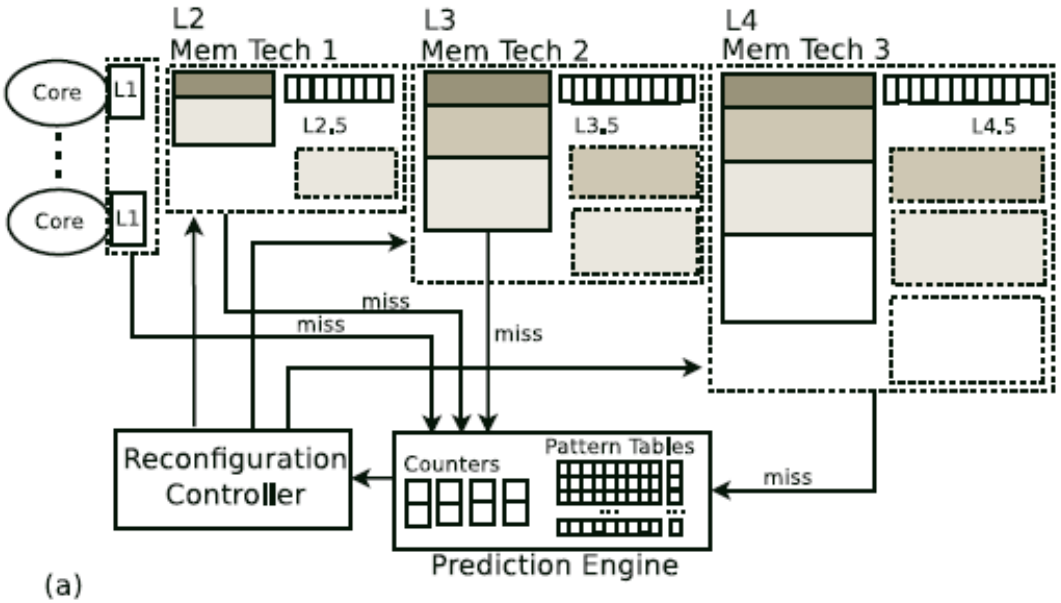
# Background

- LLC cache partitioning according to different tasks
- Research on new memory technologies to reduce latency, power and improve bandwidth
- Minimizing memory access latency with reconfigurable caches
- Application behavior predictions with predictor engines for reconfigurable architectures

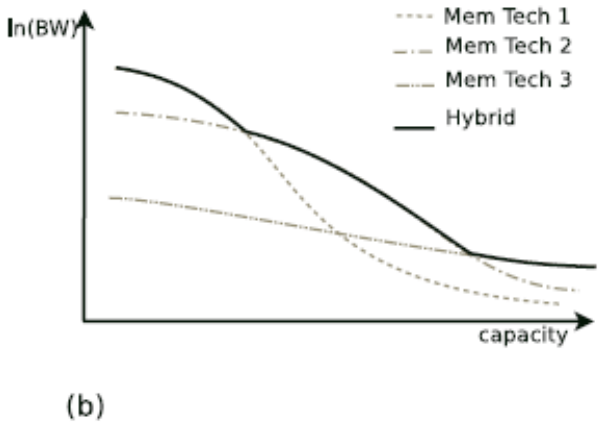
# Aim

- Using different memory technologies to improve the bandwidth of caches with large capacities
- A bandwidth-aware reconfigurable hybrid cache hierarchy to provide an optimized overall bandwidth
- A run-time cache reconfiguration mechanism that dynamically adapts the cache space of each level according to the bandwidth-demanding variations of applications
- A probability-based prediction engine that facilitates the reconfiguration mechanism

# BARCH




Configuration of reconfigurable hybrid cache hierarchy




The overall bandwidth-capacity curve of the hybrid cache hierarchy

- Different memory technologies are explored for their read/write latencies, dynamic energy and bandwidth

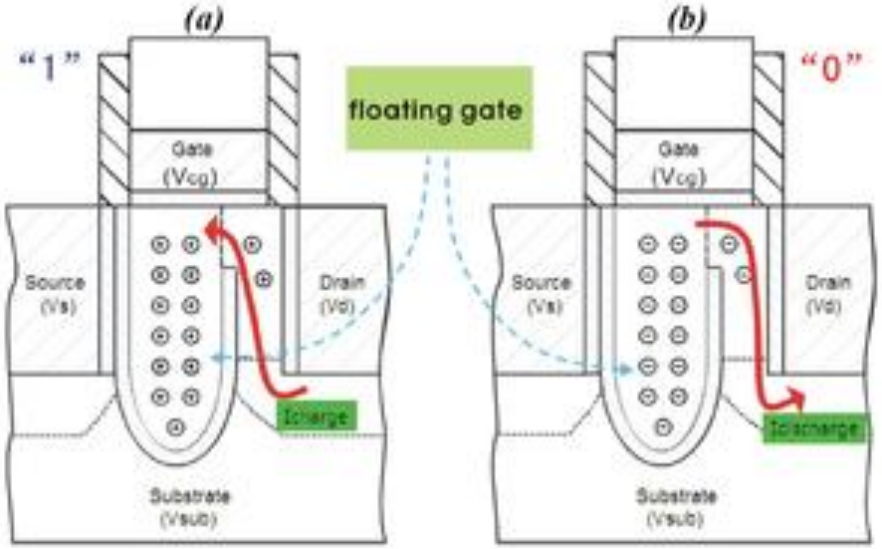
- It is difficult to a find a single memory which provides a high bandwidth across different range of capacities
- At each level different memories activated such that overall bandwidth is improved
- The total cache space at each level is partitioned to a set of fast ways and slow ways
- The cache space is adjusted according to the applications running
- The predictor core is based on a probabilistic model for achieving high accuracy with small overhead

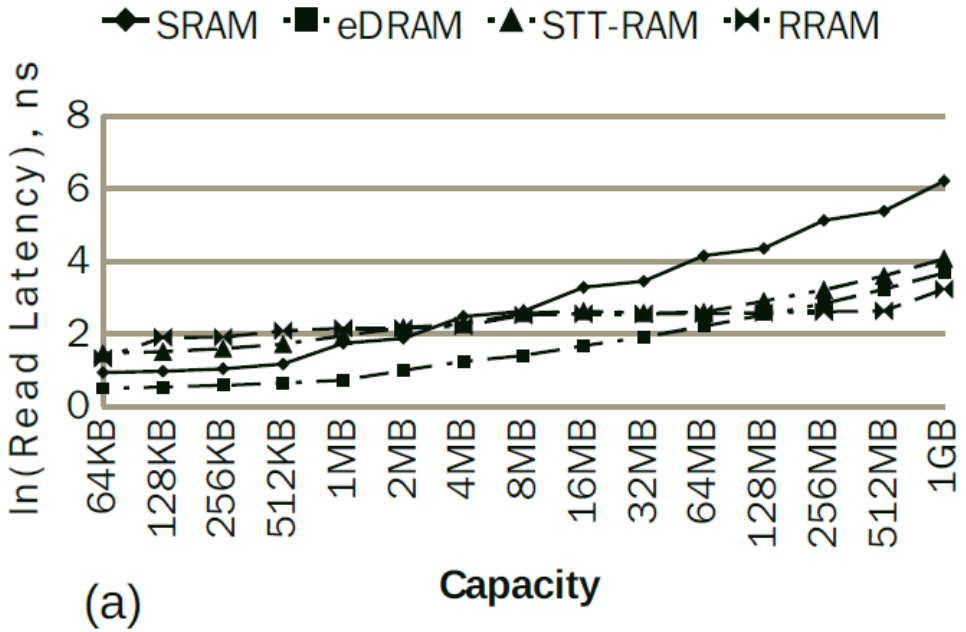

# Design Methodology

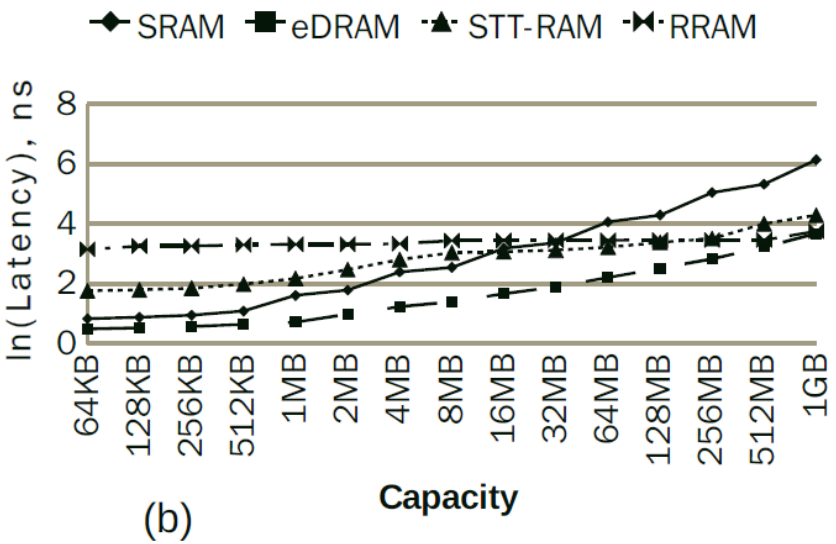
## A. Hybrid Cache Hierarchy

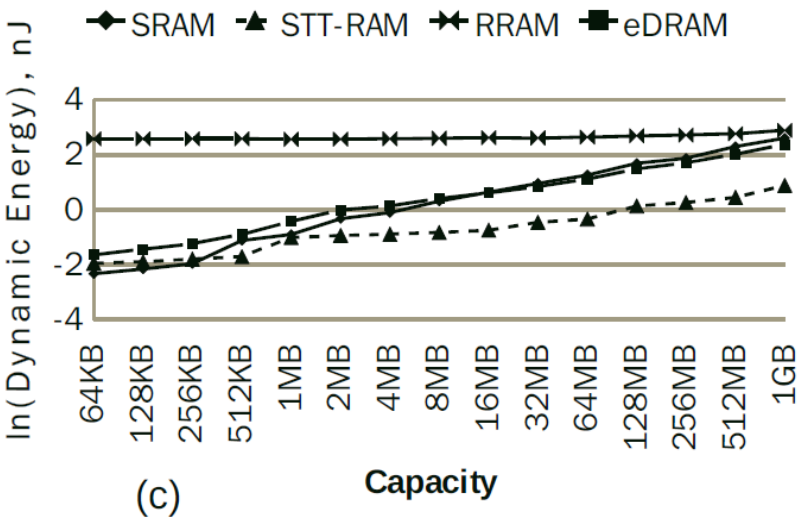
- Memory technologies used – SRAM, eDRAM, RRAM, STT-RAM
- Nvsim used for evaluation of latency, BW & energy
- Equation for read latency **dr = dHti + dwl + dbl + dcomp + dHto**

eDRAM

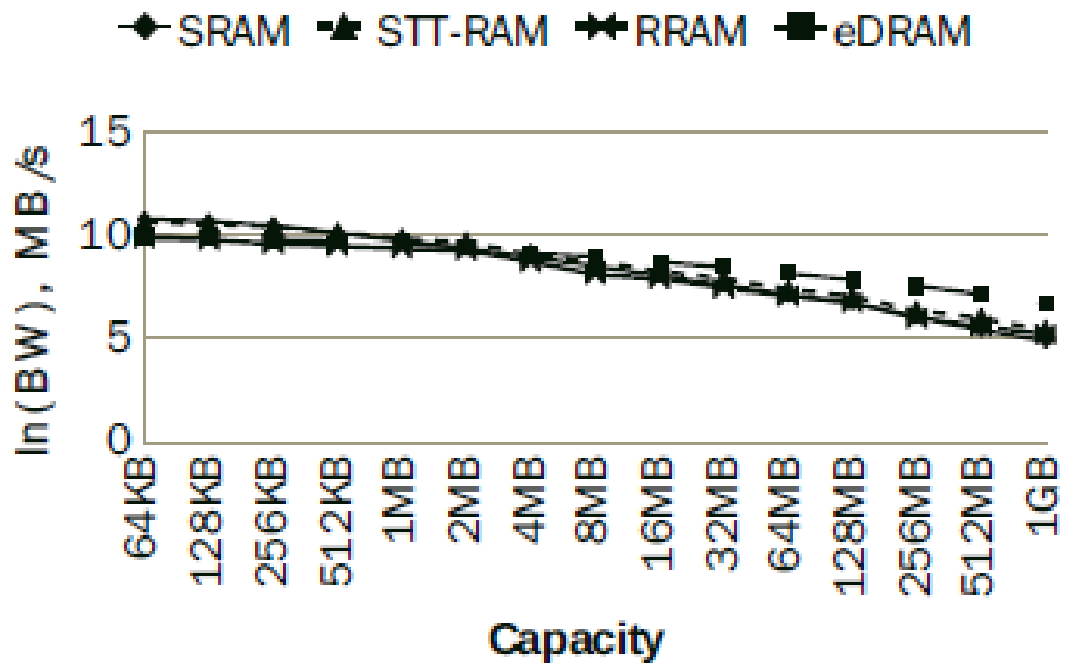




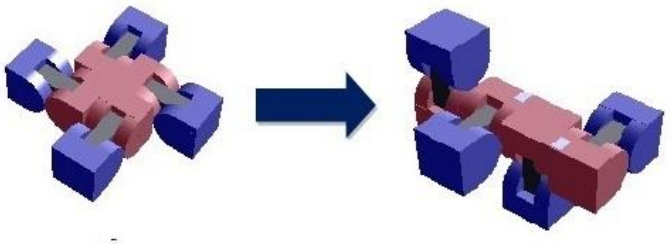

Latency and dynamic energy of different memory technologies




Bandwidth-capacity curves of different memory technologies under dynamic energy constraint (with 40% of write intensity)

- Access power of cache = $\mathrm{BW} \times \sqrt{\mathrm{capacity}}$
- Evaluating latency, energy, and bandwidth of different memory technologies SRAM, STT-RAM, and eDRAM selected
- RRAM has high dynamic energy and low endurance

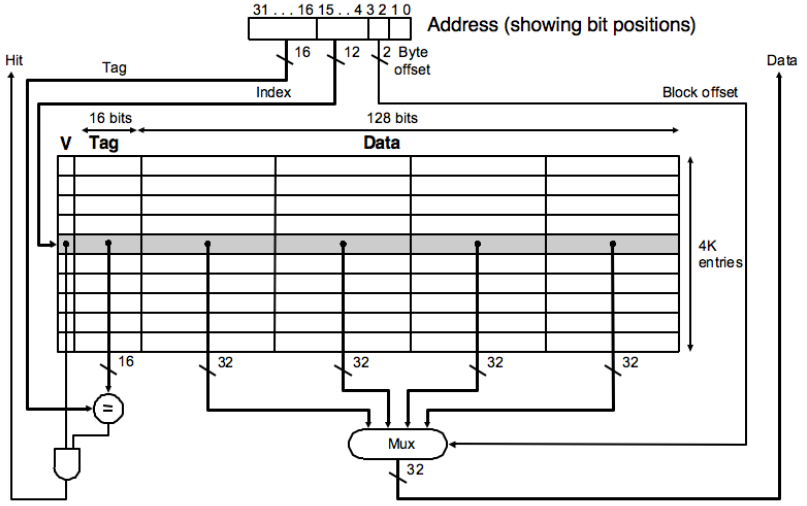
- Configuration of cache hierarchy
  - Number of levels
  - Memory technology of each level
  - Capacity of each level



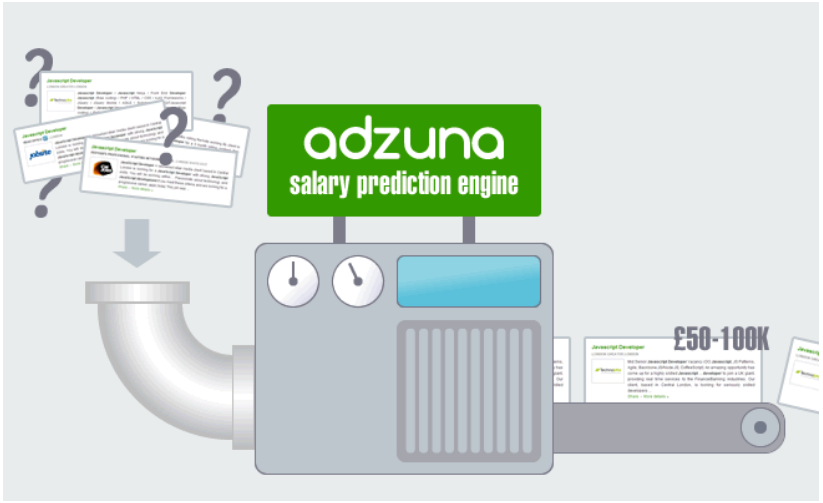
B. Reconfiguration

- Reconfigure each cache level at run time to adapt to the various bandwidth demands of various applications and also adjust the cache capacity accordingly
- Fast and slow way partitions at each level
- Faster partitions -> higher BW but small capacity
- Slower partitions -> lower BW but large capacity

- Applied at the end of each time interval
- The capacity of cache level-i is selected in the range $s_i^l \leq s_i \leq s_i^u$ where
  - $s_i^u = f^{-1}(DBW_i)$
  - $s_i^l = f^{-1}(DBW_i * (1 + \sigma))$
- $DBW_i$ generated using the prediction engine which is $C_m B_l$/t.
- Exploits set associativity. Therefore easy modification to cache architecture since division of ways already present
- Divide each level into granularity of k ways where k is determined by the capacity range. Reconfiguration will not affect the bits of the address fields that are used as tag, index



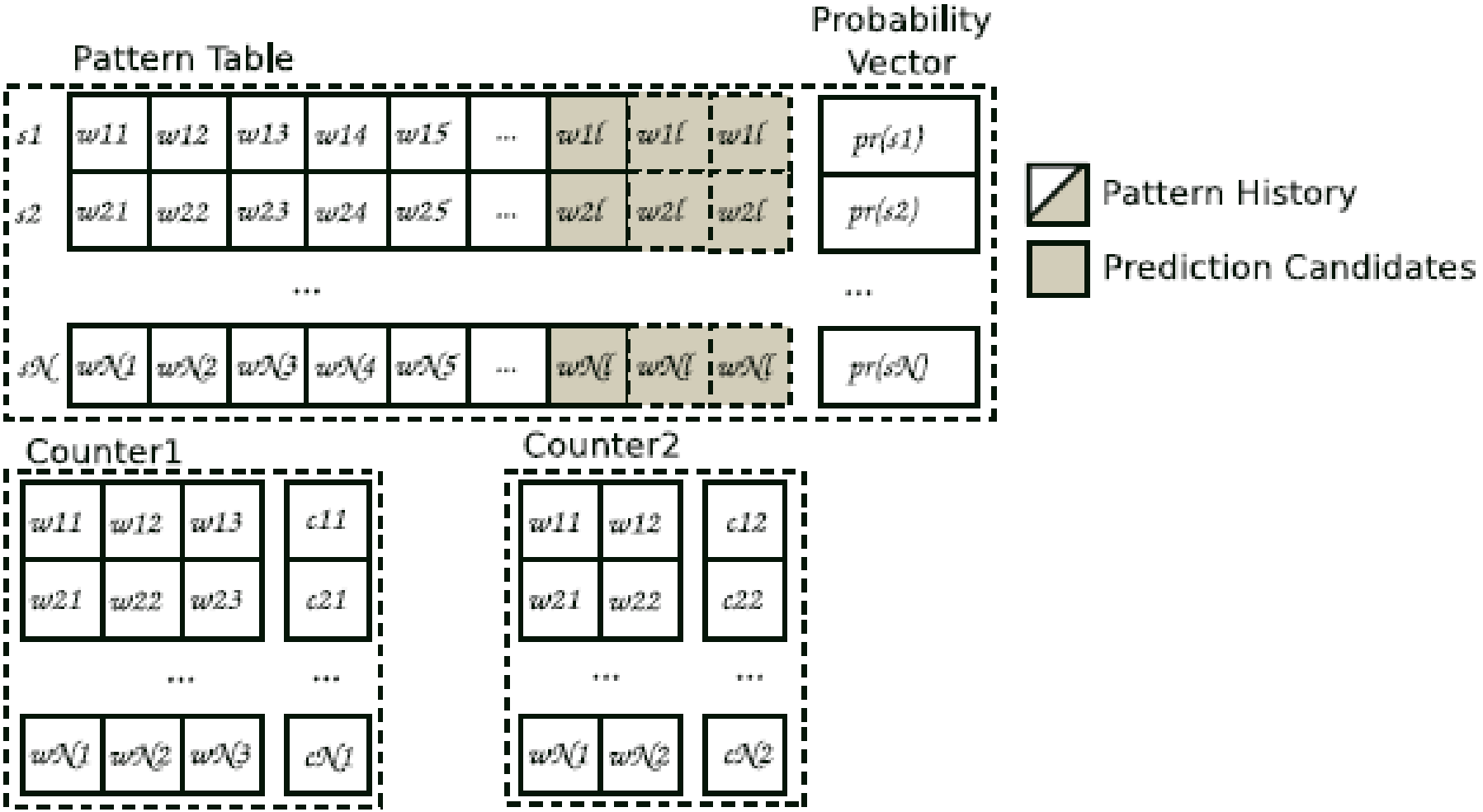
- Modifications
  - Memory status vector
  - I/O Paths
  - Additional Multiplexers




C. Prediction Engine

- Table based predictors cannot be efficient with long range patterns of an application nor patterns with variable lengths
- Instead a statistical predictor used which is similar to n- Gram model used in natural language processing
- At each time interval, the prediction engine will update the pattern table with the new DBW sample, and calculate probability of the updated pattern

- Chain rule is used to calculate the conditional probability for length ‘l’ of string ‘s’
  - **p(s) = p($w_1$)p($w_2$ | $w_1$)…p ($w_l$ | $w_1$… $w_{l-1}$)**
- The expression can be reduced
  - **p(s) =** $\prod_{i=1}^{l} p(w_i \mid w_1 \ldots w_{i-1})$
- In n-gram models an approximation is made for conditional probability using the preceding n-1 samples
  - **p(s) =** $\prod_{i=1}^{l} p(w_i \mid w_{i-n+1}^{i-1})$
- An estimation of the above equation is derived using the maximum likelihood function
- n = 3 gives a reasonable accuracy (called trigram model)
- Each conditional probability is calculated by
  - $\hat{p}(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-2}^{i})}{c(w_{i-2}^{i-1})}$



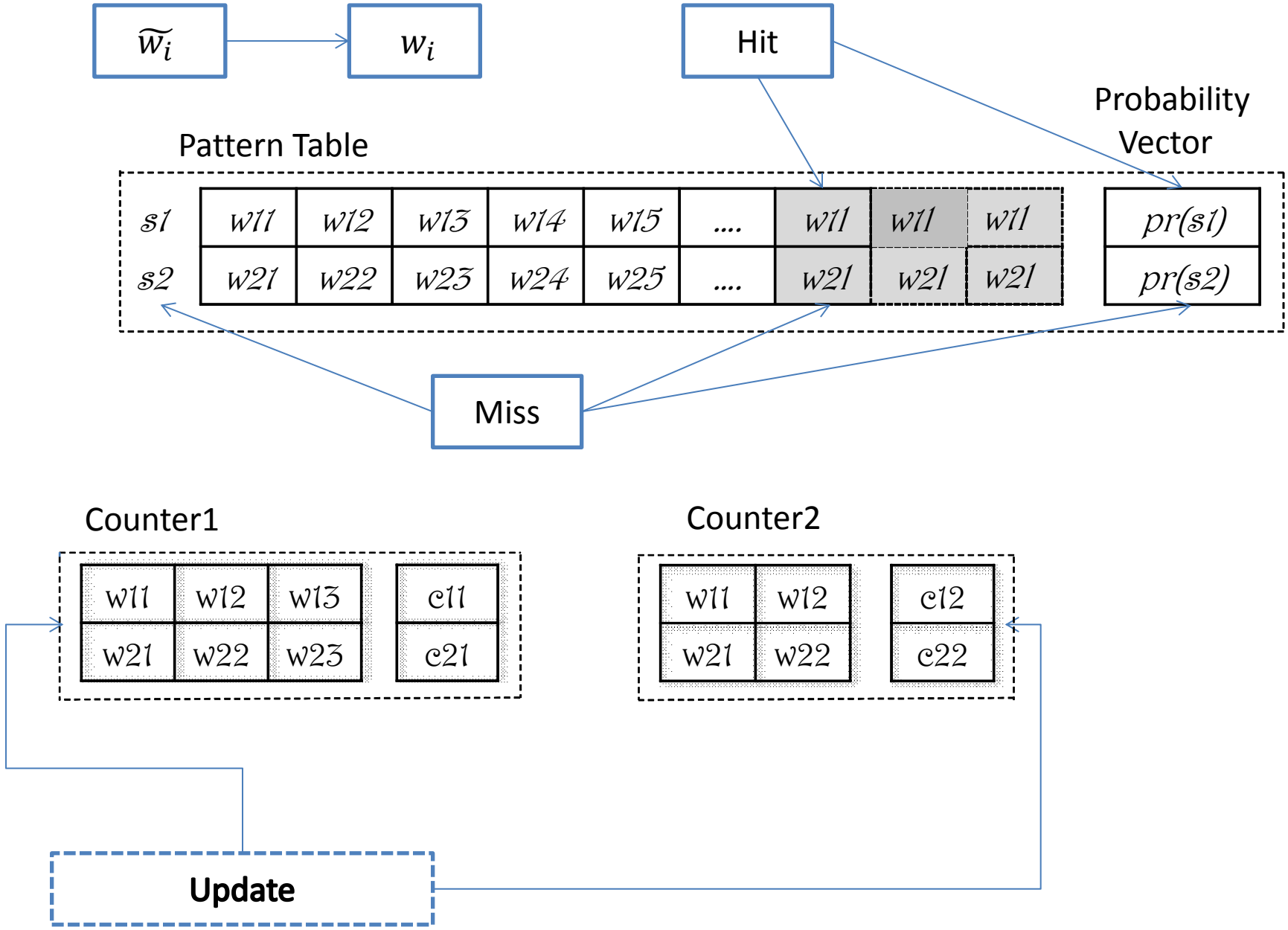

Components of the prediction engine include the pattern table, the probability vector, and an array of counters.

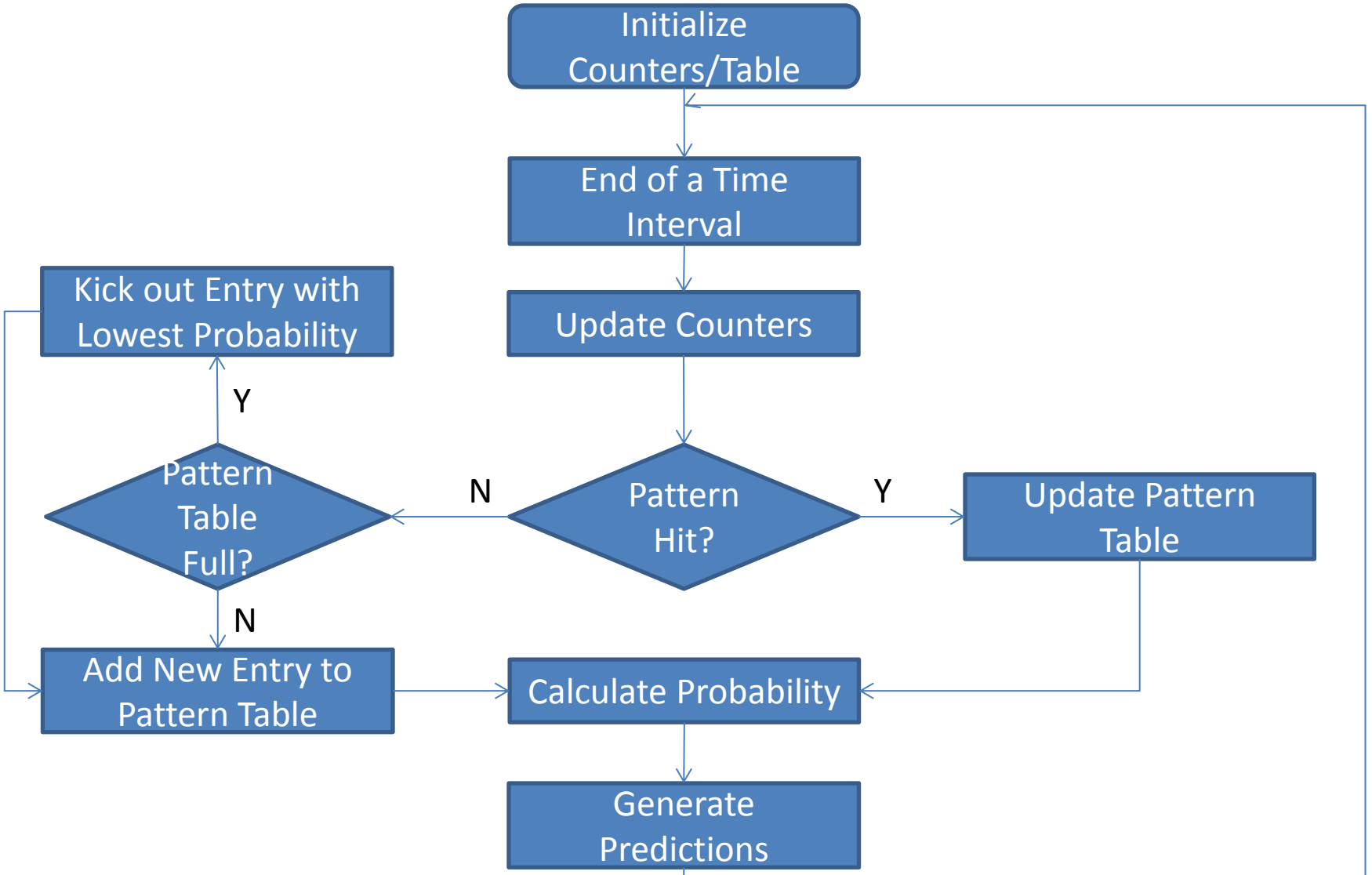
**Algorithm 1** Statistical prediction algorithm

**Input:** The new demand bandwidth sample $\tilde{w}_i$ in time interval $i$.

**Output:** The prediction of demand bandwidth $w_{i+1}$ in the next time interval $i+1$.

1: Normalize the new sample to one of quantization bins as $w_i$;
2: Update the two counters $c(w_{i-2}^{i})$ and $c(w_{i-2}^{i-1})$ with $w_i$;
3: **if** !hitPattern($s \leftarrow w_{i-l+1}..w_i$) **then**
4: Add an new entry $s$ into pattern table;
5: $p(s) \leftarrow calcProbability(s, c(w_{i-2}^{i}), c(w_{i-2}^{i-1})$;
6: **else**
7: $p(s) \leftarrow calcProbability(s, c(w_{i-2}^{i}), c(w_{i-2}^{i-1})$;
8: **end if**
9: $k \leftarrow indexOfMaxProbability(p)$
10: $w_{i+1} \leftarrow patternTable[k][l]$


$\widetilde{w_i}$
$w_i$
Hit
Probability Vector
Pattern Table
s1
w11 w12 w13 w14 w15 .... w11 w11 w11
pr(s1)
s2
w21 w22 w23 w24 w25 .... w21 w21 w21
pr(s2)
Miss
Counter1
w11 w12 w13
c11
w21 w22 w23
c21
Counter2
w11 w12
c12
w21 w22
c22
Update




**Control Flow of Prediction Engine**

- Prediction Accuracy

PREDICTION ACCURACY FOR DIFFERENT WIDTHS OF THE PATTERN TABLE

| Benchmark | Width of Pattern Table | | | | |
|---|---|---|---|---|---|
| | 12 | 10 | 9 | 8 | 7 |
| canneal | 100% | 34% | 34% | 33% | 31% |
| facesim | 98% | 30% | 21% | 19% | 15% |
| streamcluster | 100% | 44% | 42% | 40% | 33% |
| astar | 100% | 100% | 100% | 37% | 31% |
| bwaves | 100% | 100% | 100% | 31% | 35% |
| gamess | 100% | 100% | 100% | 100% | 100% |
| GemsFDTD | 100% | 100% | 100% | 100% | 100% |
| lbm | 100% | 100% | 100% | 56% | 62% |
| mcf | 100% | 100% | 100% | 97% | 49% |
| perlbench | 100% | 100% | 100% | 100% | 100% |
| wrf | 100% | 100% | 100% | 100% | 100% |
| zeusmp | 100% | 100% | 100% | 100% | 100% |

- Storage Overhead

STORAGE OVERHEAD OF THE PREDICTION ENGINE.

| Component | Width | Length | Storage |
|---|---|---|---|
| Pattern Table | 12-byte | 240 | 3KB |
| Probability Vector | 8-byte | 240 | 2KB |
| Counter Vector | 3-byte | 240 | 1KB |

- Computational Overhead
  - Computational complexity of the prediction algorithm O(ql)
  - Bounded by the size of the pattern table and the limited quantization bins (in $\mu$s)

# Experimental Setup

- Simics used to model a four-core CMP.
- Similar to UltraSPARC III
- Multithreaded and Multiprogrammed workloads used as benchmarks

BASELINE CMP CONFIGURATION.

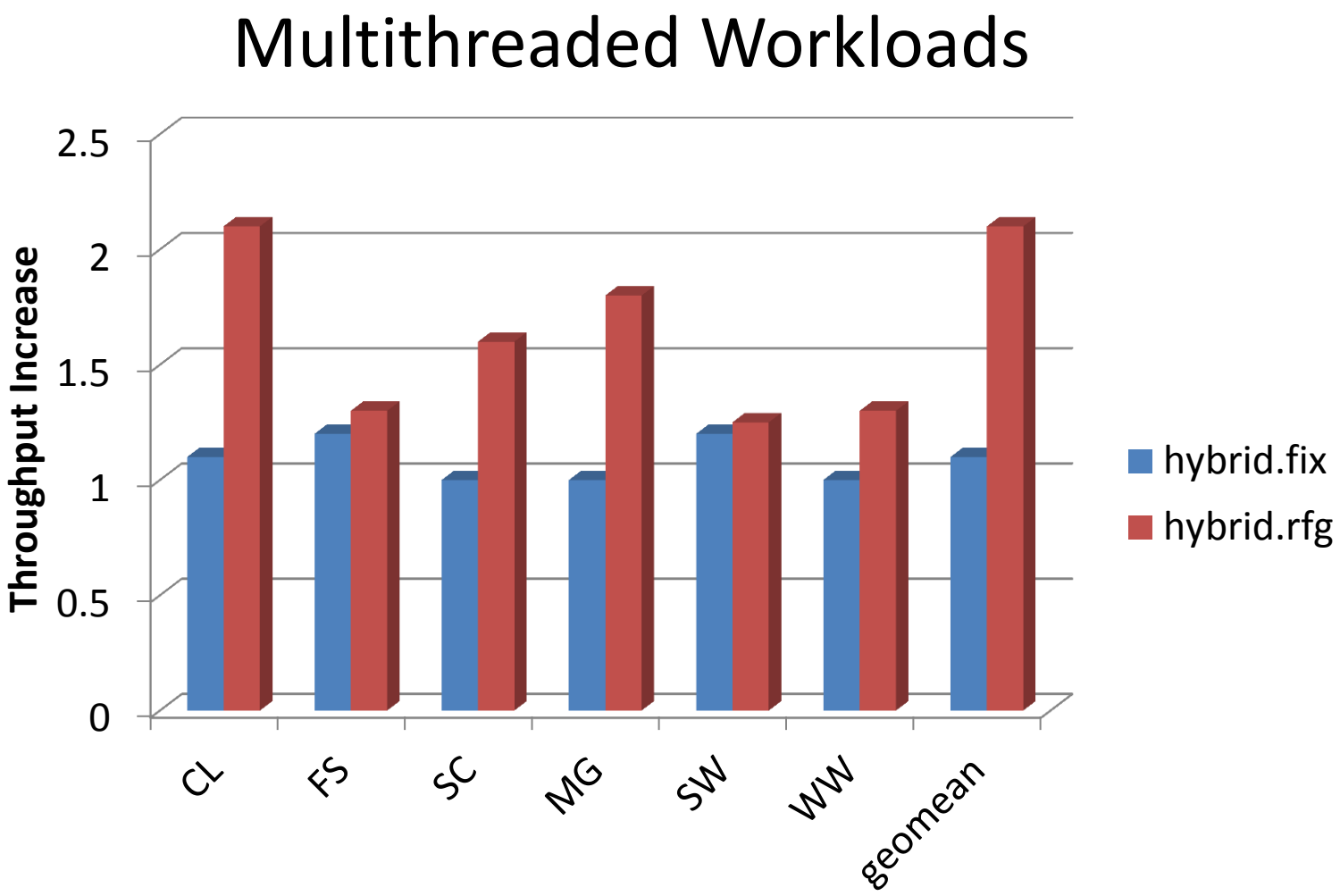
| | |
|---|---|
| No. of cores | 4 |
| Configuration | 1GHz, in-order, 14-stage pipeline |
| Private L1 | SRAM, 64B line, size 64KB |
| Shared caches | SRAM/STT-RAM/eDRAM/RRAM, 64B line, 1 to 3 levels, size of 512KB to 64MB |
| Main memory | 4GB |

- Shared cache hierarchy tested in four different cases
  - Pure SRAM-based L2 cache with fixed capacity(SRAM.fix)
  - Hybrid L2/L3/L4 caches with fixed maximum available capacity at each level (hybrid.fix)
  - Hybrid reconfigurable caches (hybrid.rfg)
  - Hybrid reconfigurable caches with workload partition (hybrid.par)
- Workloads that vary in the L2 cache write intensity (Write%) and peak demand bandwidth (PDBW) selected listed in the following table

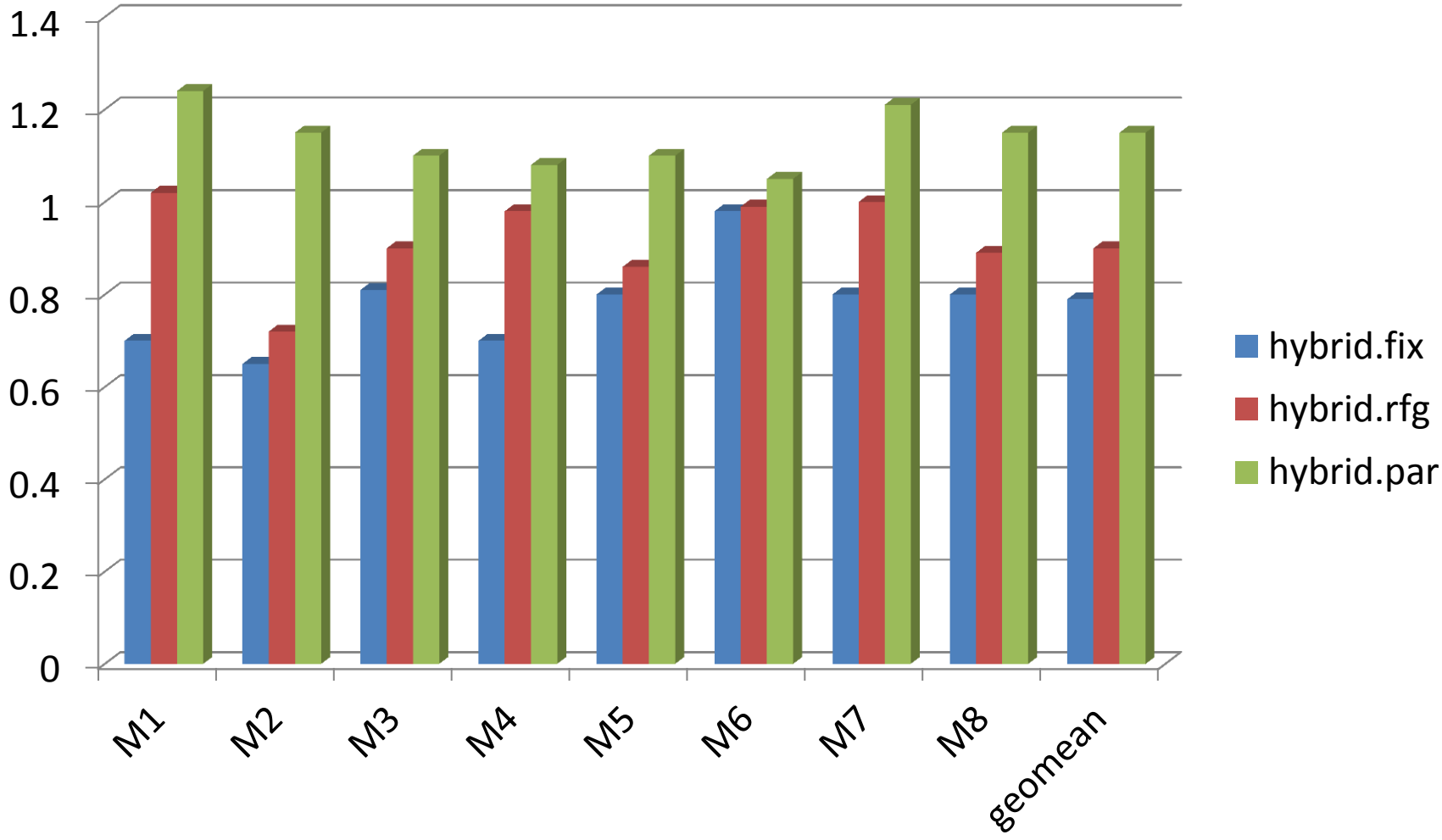
CHARACTERISTICS OF SELECTED BENCHMARKS. I'06 AND F'06 REPRESENT THE SPEC CPU2006 INTEGER AND FLOATING POINT BENCHMARKS RESPECTIVELY.

| Benchmarks | Benchmark Suite | Write% | PDBW | Abbreviation | Application Sets |
|---|---|---|---|---|---|
| canneal | PARSEC | 31.4% | 791 MB/s | CL | canneal |
| facesim | PARSEC | 30% | 572 MB/s | FS | facesim |
| streamcluster | PARSEC | 0.6% | 552 MB/s | SC | streamcluster |
| mgrid | SPEC OMP2001 | 3.6% | 562 MB/s | MG | mgrid |
| swim | SPEC OMP2001 | 3.6% | 643 MB/s | SW | swim |
| wupwise | SPEC OMP2001 | 4% | 536 MB/s | WW | wupwise |
| astar | I'06 | 38% | 4.1 GB/s | M1 | sphinx3+astar+lbm+zeusmp |
| bwaves | F'06 | 24.5% | 2.5 GB/s | M2 | wrf+GemsFDTD+bwaves+mcf |
| gamess | I'06 | 28.4% | 1.1 GB/s | M3 | perlbench+milc+gamess+sphinx3 |
| GemsFDTD | F'06 | 30.5% | 2.6 GB/s | M4 | sphinx3+wrf+perlbench+astar |
| lbm | F'06 | 42.2% | 3.9 GB/s | M5 | gamess+milc+perlbench+mcf |
| mcf | I'06 | 26.2% | 1.8 GB/s | M6 | mcf+milc+lbm+gamess |
| wrf | F'06 | 25.1% | 2.6 GB/s | M7 | perlbench+lbm+astar+milc |
| zeusmp | F'06 | 5.5% | 3 GB/s | M8 | zeusmp+bwaves+wrf+mcf |

# Results


Multithreaded Workloads
Throughput Increase
2.5
2
1.5
1
0.5
0
CL
FS
SC
MG
SW
WW
geomean
hybrid.fix
hybrid.rfg

# Multiprogrammed Workloads



# Conclusion

- Proposed a bandwidth aware reconfigurable hierarchy method
- Hybrid cache hierarchy leverages different memory technologies to provide an optimized bandwidth-capacity curve
- Dynamically reconfigure the cache space at each level adaptive to the demands of different applications
- Prediction engine provided for this reconfiguration
- Experimental results show that reconfigurable hybrid cache leads to 58% and 14% performance improvements to multithreaded and multiprogrammed applications, respectively

# Future Improvements

- Could use PCRAM
- Limited benchmarks used
- Need to think about scalability
- Depends lot on technology advancements in VLSI