# Designing Future Low-Power and Secure Processors with Non-Volatile Memory

**Xiang Pan**
**Computer Architecture Research Lab**

http://arch.cse.ohio-state.edu

THE OHIO STATE UNIVERSITY

# Outline

- Background
- Low-Power Processor Design with NVM in High-Voltage Domain (NVSleep)
- Low-Power Processor Design with NVM in Low-Voltage Domain (Respin)
- **Security Research on Processors Equipped with NVM Caches (NV-Insecure)**

# Universal Demand for Low Power

- Mobility
- Battery life

- Performance
- Power constraints

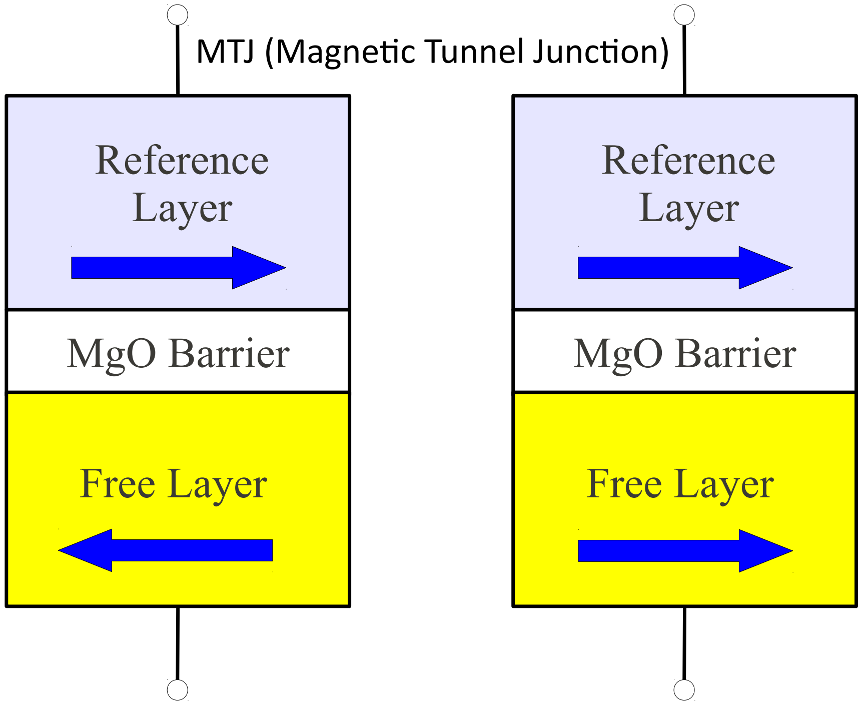
- Energy cost
- Environment

# Non-Volatile Memory Basics

- Non-Volatility – Resistance as data representation (e.g. PCM, STT-RAM, ReRAM, etc.)
- Near-Zero Leakage Power – Good fit for future power-constrained computing
- High Density – Great design candidate in the big data era
- Good Performance – Feasible for on-chip storage replacement

# STT-RAM

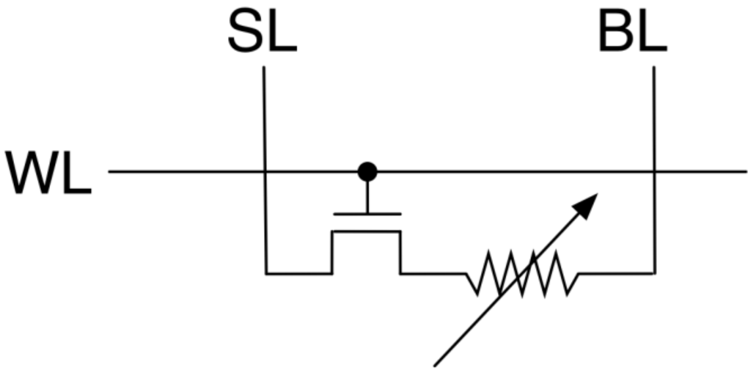
- Unique features of STT-RAM: fast read speed, low read energy, unlimited write endurance, and good compatibility with CMOS technology
    - Shortcomings: long write latency and high write energy



MTJ (Magnetic Tunnel Junction)

Anti-parallel state: high-resistance, representing a “1”

Parallel state: low-resistance, representing a “0”



STT-RAM Cell (1T-1MTJ)

# Outline

- Background
- **Low-Power Processor Design with NVM in High-Voltage Domain (NVSleep)**
- Low-Power Processor Design with NVM in Low-Voltage Domain (Respin)
- Security Research on Processors Equipped with NVM Caches (NV-Insecure)

# NVSleep: Using Non-Volatile Memory to Enable Fast Sleep/Wakeup of Idle Cores

- The first work to use non-volatility feature of STT-RAM to implement pipeline-level checkpointing
- A general and low overhead framework for reducing energy through exploiting short idle execution phases
- Achieved energy reduction of 17-34% with less than 3% performance and area overheads

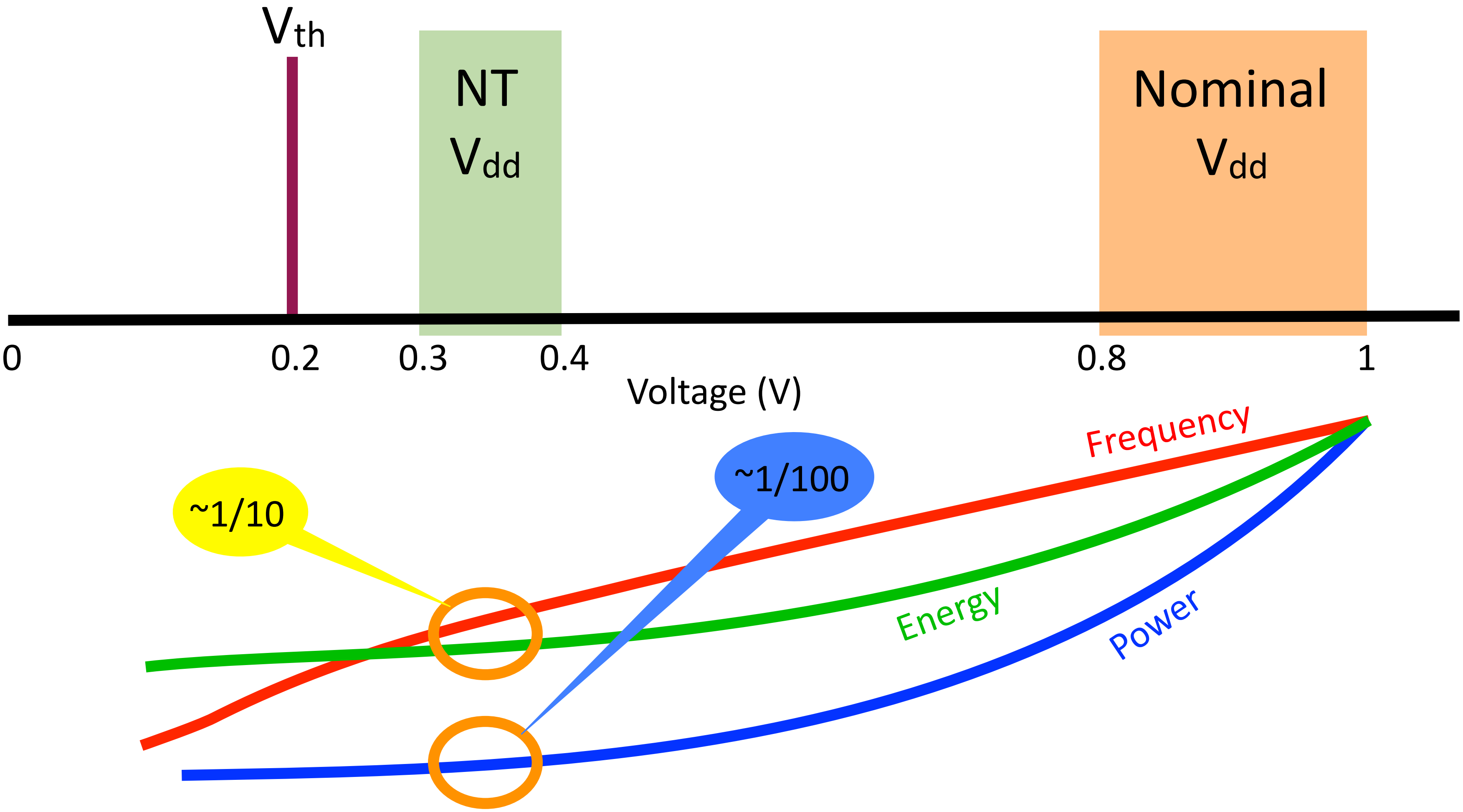
# Outline

- Background
- Low-Power Processor Design with NVM in High-Voltage Domain (NVSleep)
- **Low-Power Processor Design with NVM in Low-Voltage Domain (Respin)**
- Security Research on Processors Equipped with NVM Caches (NV-Insecure)

# Respin: Rethinking Near-Threshold Multiprocessor Design with Non-Volatile Memory

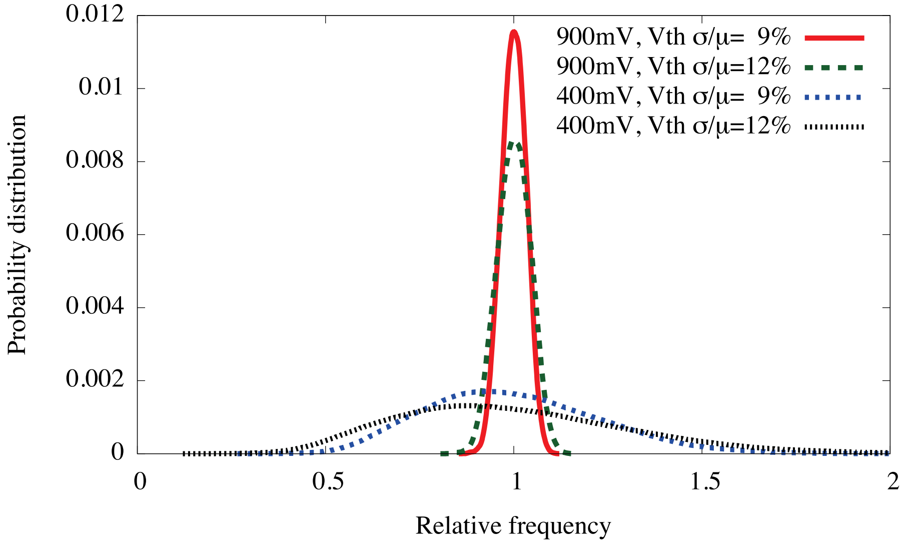
- The first work to explore the use of non-volatile caches in near-threshold chip multiprocessors
- A novel architecture designed to enhance NT-CMP performance and reduce energy consumption by sharing L1 caches and implementing dynamic core consolidation mechanism
- Achieved energy reduction by 33% and improved performance by 11%

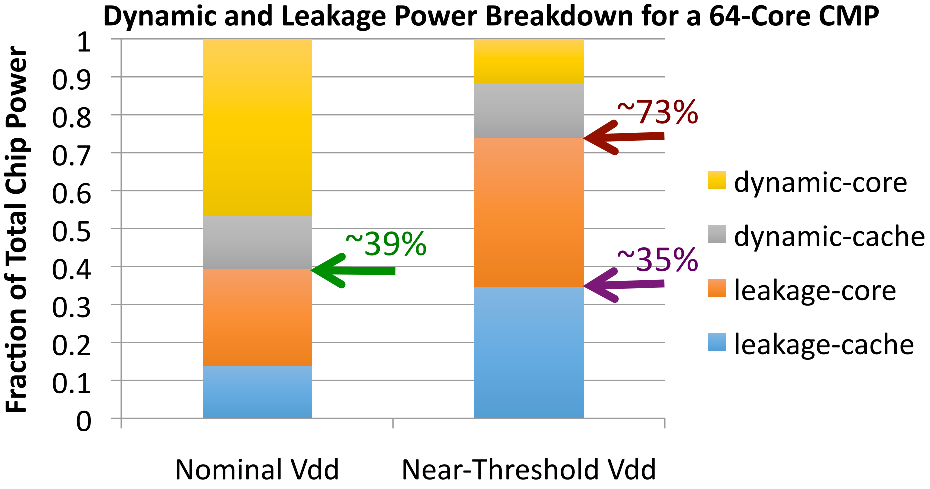
# Low Voltage Operation



# Challenges in Near-Threshold

- Performance degradation
- Function failure
- Amplified process variation
- Leakage power domination
- The initial idea of Respin – Build caches in NT-CMP with “leakage-free” non-volatile memories to reduce power consumption

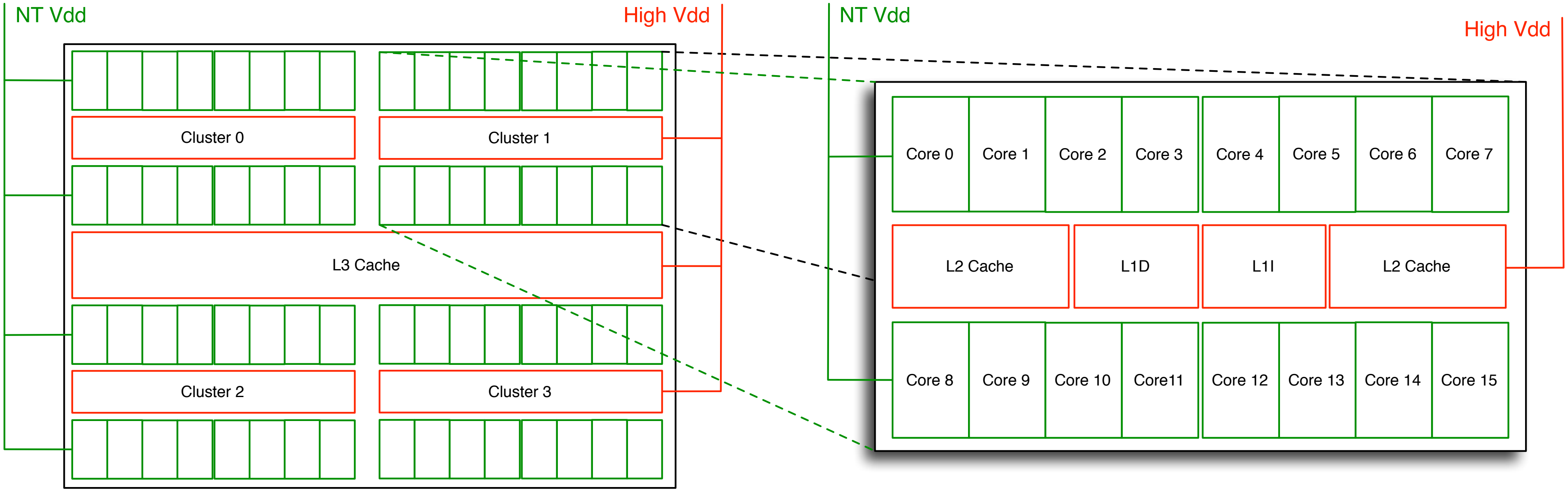




# STT-RAM is Good Fit for NT-CMP

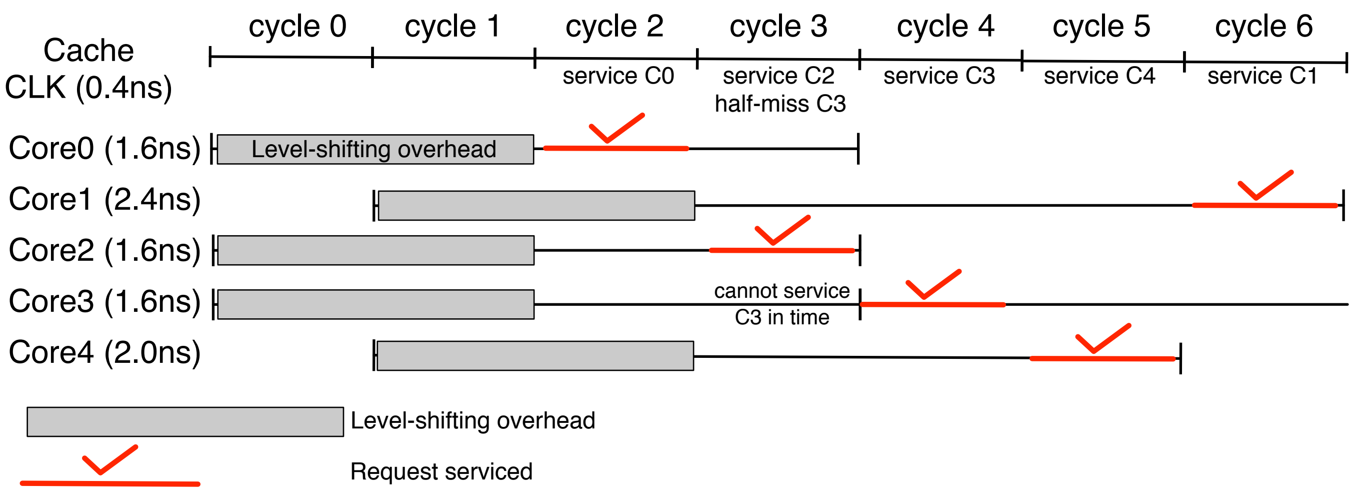
| NT-CMP | | STT-RAM |
|---|---|---|
| High leakage power | ☹ ⟷ ☺ | Near-zero leakage power |
| Low operating frequency | ☺ ⟷ ☹ | Long latency writes |
| Large numbers of cores requiring high cache capacity | ☹ ⟷ ☺ | High density (~4x denser than SRAM) |
| Unreliable functional units | ☹ ⟷ ☺ | Robust from soft-errors |

# Respin Architecture



- Cores operate at NT-Vdd rail with low frequencies
- Caches are built with STT-RAM and operate at high-Vdd rail making read speed extremely fast
- Clustered-CMP with fast STT-RAM read enables within-cluster shared L1 cache design, removing coherence costs

# Shared Cache Hierarchy



Cache CLK (0.4ns)

cycle 0, cycle 1, cycle 2, cycle 3, cycle 4, cycle 5, cycle 6

service C0, service C2 half-miss C3, service C3, service C4, service C1

Core0 (1.6ns)

Core1 (2.4ns)

Core2 (1.6ns)

Core3 (1.6ns)

Core4 (2.0ns)

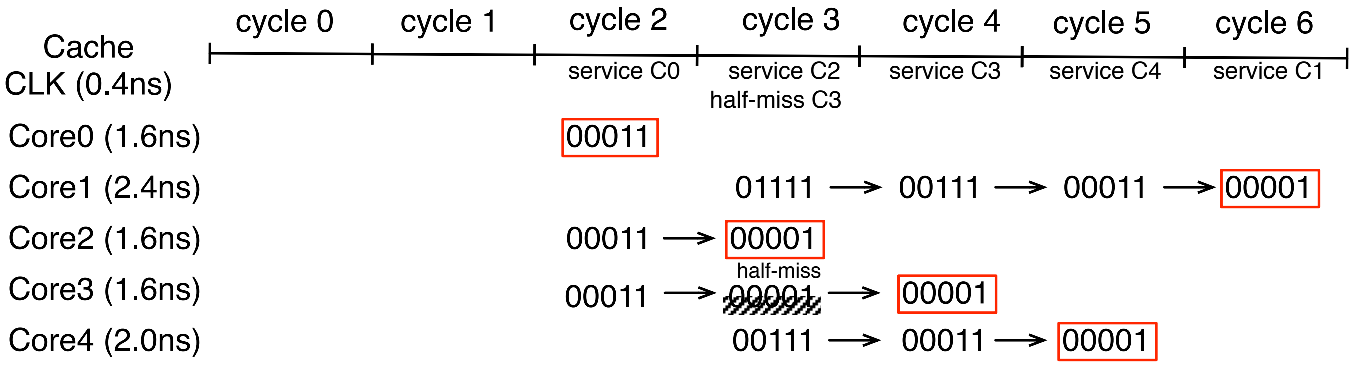
Level-shifting overhead

cannot service C3 in time

▭ Level-shifting overhead

✓ Request serviced

(a)

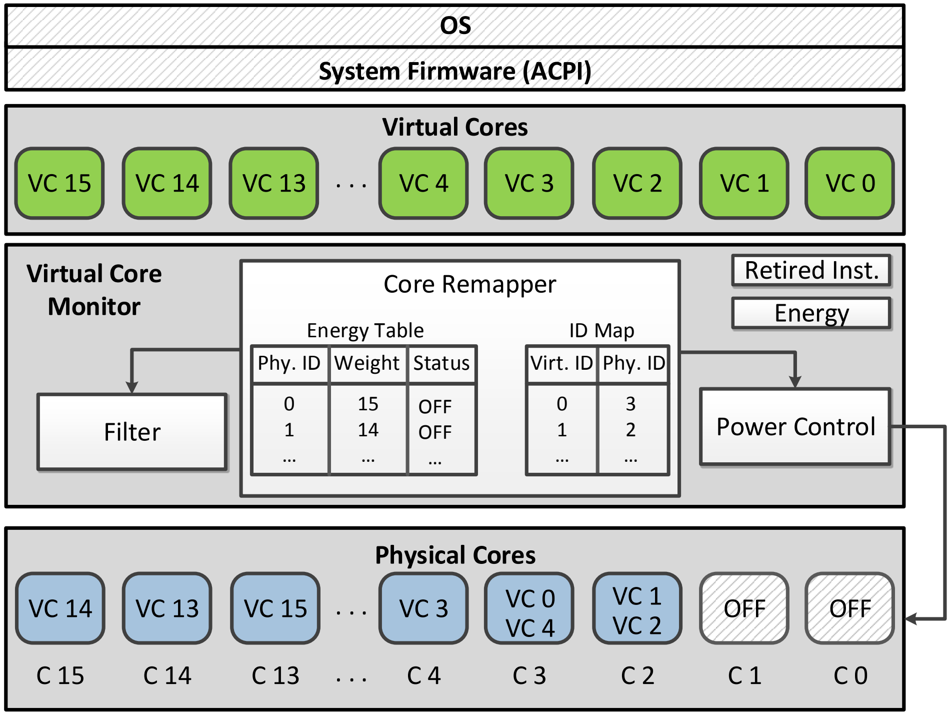


| Cache CLK (0.4ns) | cycle 0 | cycle 1 | cycle 2<br>service C0 | cycle 3<br>service C2<br>half-miss C3 | cycle 4<br>service C3 | cycle 5<br>service C4 | cycle 6<br>service C1 |
|---|---|---|---|---|---|---|---|
| Core0 (1.6ns) | | | 00011 | | | | |
| Core1 (2.4ns) | | | | 01111 → | 00111 → | 00011 → | 00001 |
| Core2 (1.6ns) | | | 00011 → | 00001 | | | |
| Core3 (1.6ns) | | | 00011 → | half-miss<br>~~00001~~ → | 00001 | | |
| Core4 (2.0ns) | | | | 00111 → | 00011 → | 00001 | |

(b)

# Dynamic Core Consolidation

- High process variation and leakage in NT-CMP lead to fast cores more energy-efficient than slow ones
- Upon shared cache design, dynamically consolidate threads onto efficient cores with greedy search can further save energy
- Energy-per-instruction used as greedy selection metric and instruction count used as selection interval

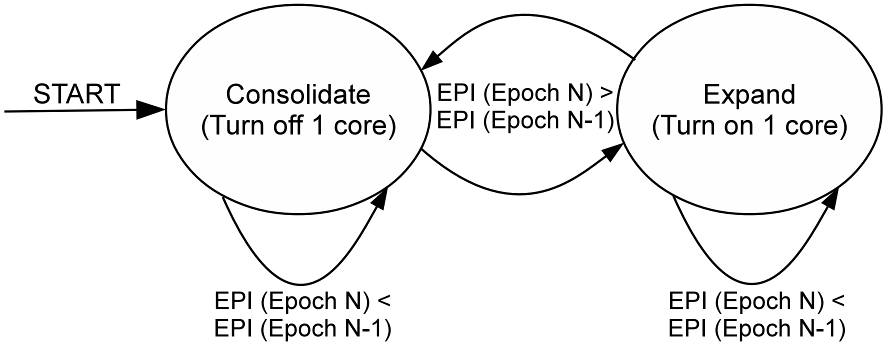


Energy Table

| Phy. ID | Weight | Status |
|---|---|---|
| 0 | 15 | OFF |
| 1 | 14 | OFF |
| … | … | … |

ID Map

| Virt. ID | Phy. ID |
|---|---|
| 0 | 3 |
| 1 | 2 |
| … | … |



# Methodology

| Level | Size ($mm^2$) | Block Size | Associativity | Read/Write Ports |
|---|---|---|---|---|
| L1I (Private/Shared within Cluster) | 16KB (Private)/256KB (Shared within Cluster) | 32B | 2-way | 1/1 |
| L1D (Private/Shared within Cluster) | | | 4-way | |
| L2 (Shared within Cluster) | 8MB (Small)/16MB (Medium)/32MB (Large) | 64B | 8-way | |
| L3 (Shared within Chip) | 24MB (Small)/48MB (Medium)/ 96MB (Large) | 128B | 16-way | |

Table 1. Summary of Cache Parameters.

| | Vdd Rail | Area ($mm^2$) | Read/Write Latency (ns) | Read/Write Energy (nJ) | Leakage Power (mW) |
|---|---|---|---|---|---|
| SRAM (16KB × 16) | Low (0.65V) | 0.9176 | 1.337 | 0.002578 | 573 |
| SRAM (256KB) | High (1.0V) | | 0.5336 | 0.04241 | 881 |
| STT-RAM (256KB) | | 0.2451 | 0.3774/5.208 | 0.02932/0.2093 | 114 |

Table 2. Comparison of SRAM vs. STT-RAM Technology Parameters.

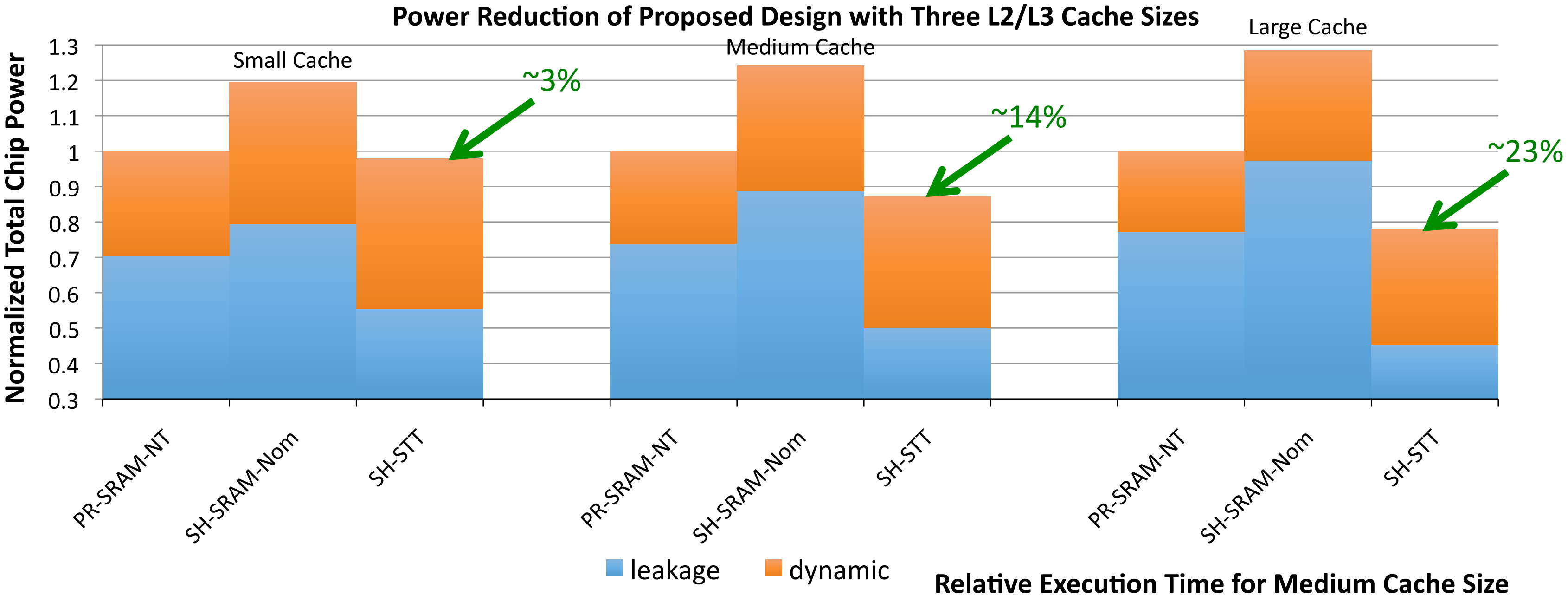
# Methodology

- Simulation Framework:
  - SESC for architecture simulation
  - CACTI, McPAT, and NVSim for latency, power, energy, and area simulation
- Benchmarks:
  - SPLASH2 and PARSEC
- Main Evaluated Configuration:
  - 64-core CMP with four 16-core clusters
  - Medium size L2 and L3 caches
  - 0.4ns shared L1 cache read latency

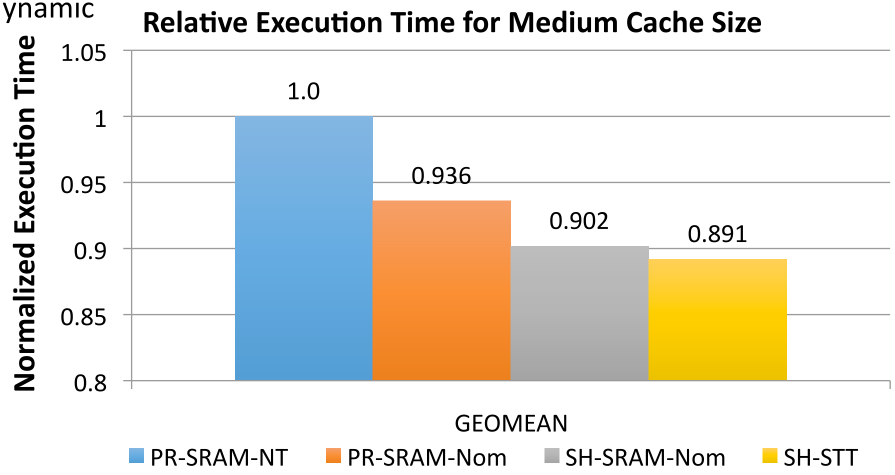
| CMP Architecture | |
|---|---|
| Cores | 64 out-of-order |
| Fetch/Issue/Commit Width | 2/2/2 |
| Register File Size | 76 int, 56 fp |
| Instruction Window Size | 56 int, 24 fp |
| Reorder Buffer Size | 80 entries |
| Load/Store Queue Size | 38 entries |
| NoC Interconnect | 2D Torus |
| Coherence Protocol | MESI |
| Consistency Model | Release Consistency |
| Technology | 22nm |
| NT-Vdd | 0.4V (Core), 0.65V (Cache) |
| Nominal-Vdd | 1.0V |
| Core Frequency Range | 375MHz – 725MHz |
| Median Core Frequency | 500MHz |
| Variation Parameters | |
| Vth std. dev./mean (σ/μ) | 12% (Chip), 10% (Cluster) |

Table 3. Summary of Experimental Parameters.

# Power and Performance

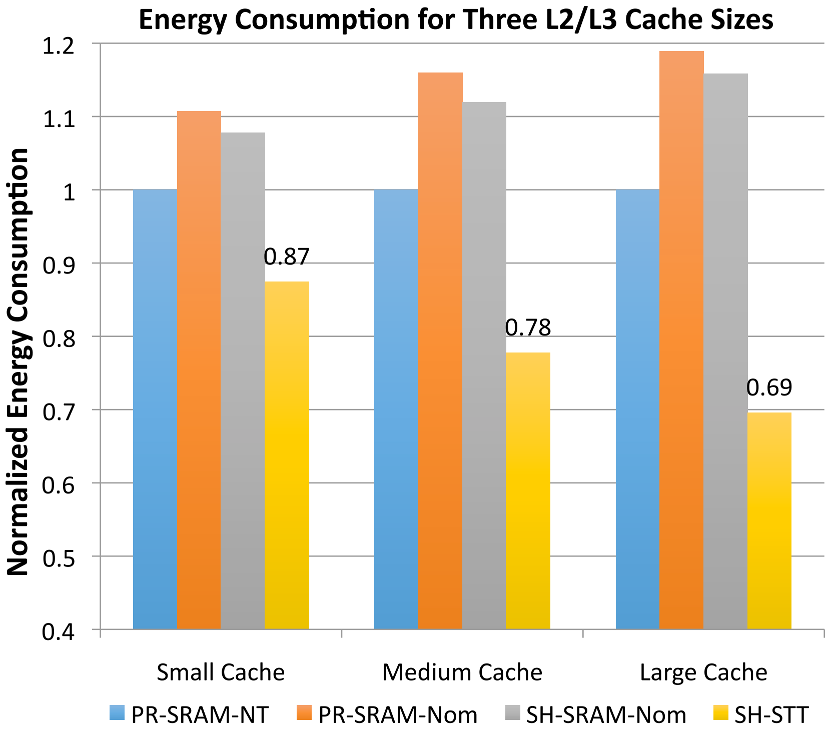


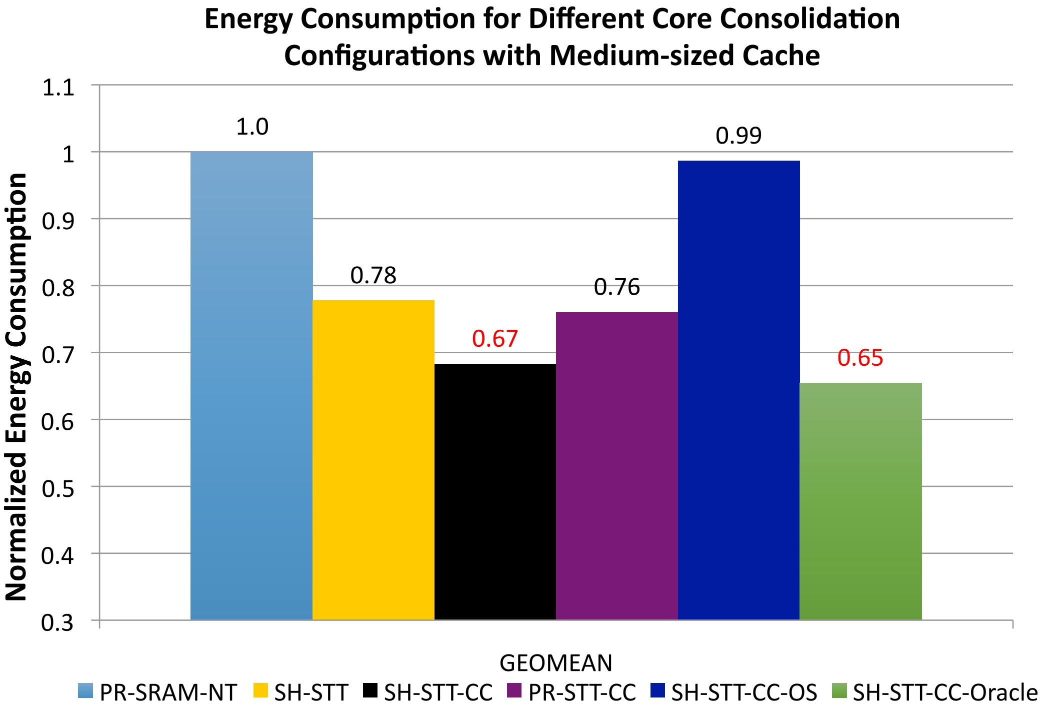
- Respin achieved 14% power reduction and 11% performance improvement with medium sized cache



# Energy Consumption

- For medium sized cache, Respin achieved 22% energy savings with the basic shared STT-RAM cache design plus additional 11% with core consolidation enabled

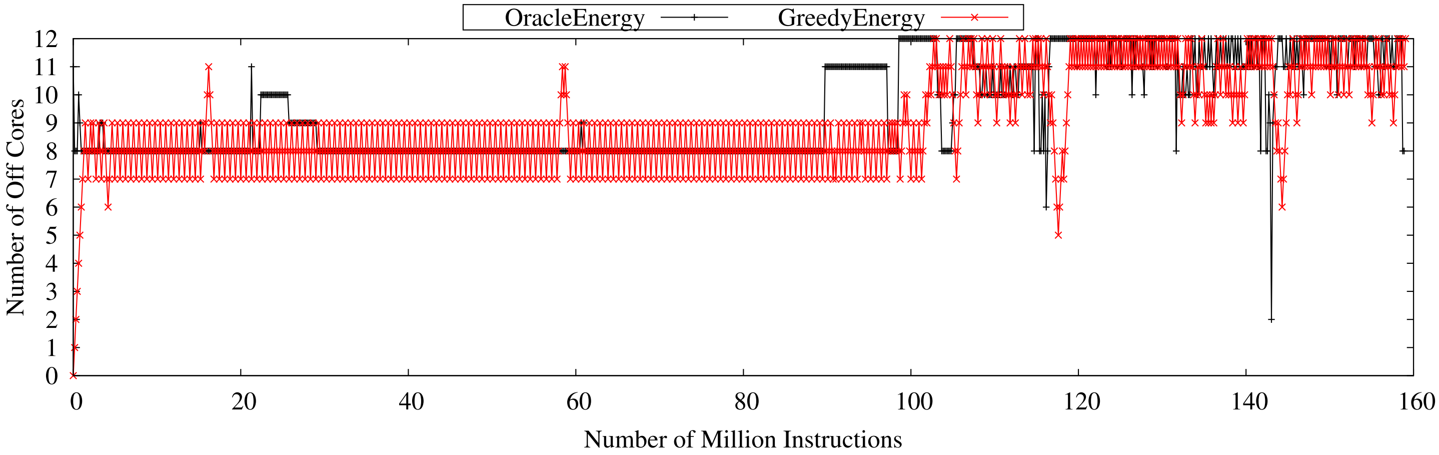




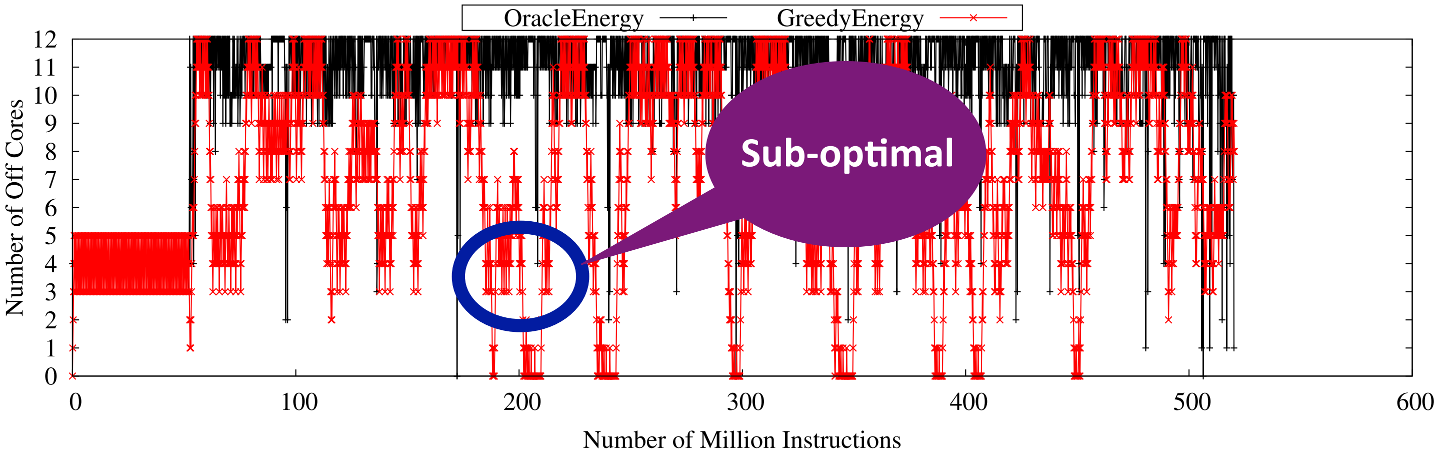
# Core Consolidation Analysis

- In most cases our greedy algorithm matches well with the oracle while in very few cases sub-optimal selection becomes the barrier to slow down the pace of our greedy mechanism

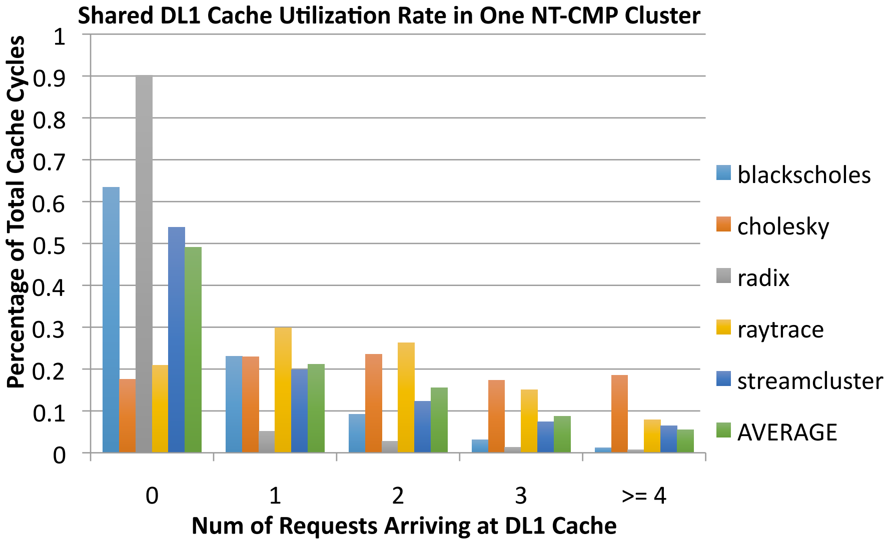
***radix*** (SPLASH2): greedy achieved 48% energy savings while oracle achieved 50%

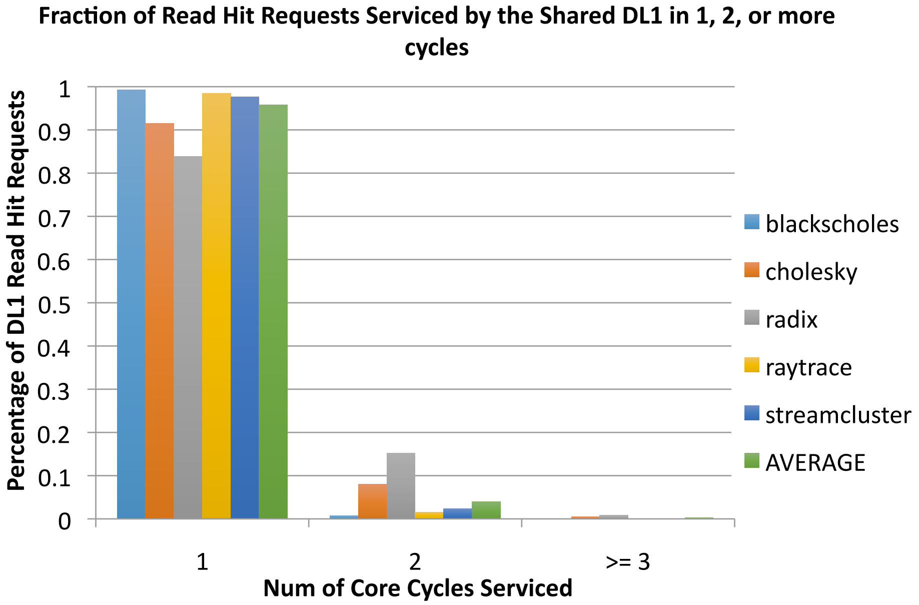


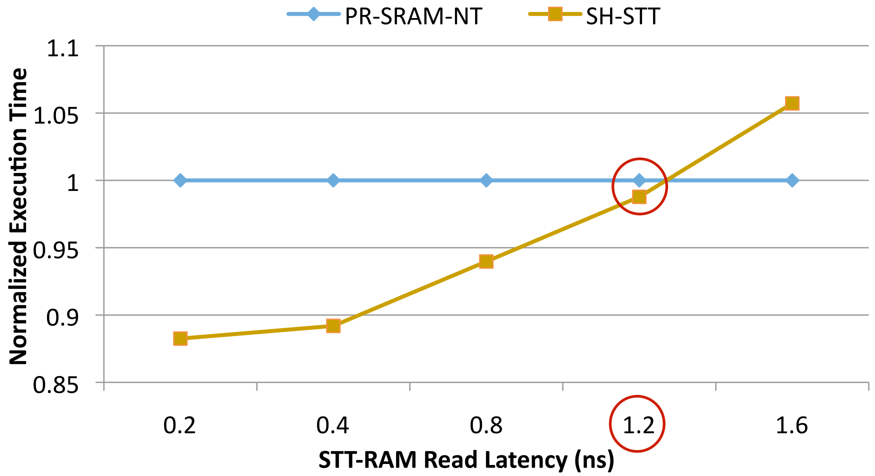
***lu*** (SPLASH2): greedy achieved 29% energy savings while oracle achieved 38%



# Sensitivity Studies







| Cluster Size (#cores) | Shared L1 (I/D) Size (KB) | Performance Gain (%) |
|---|---|---|
| 4 | 64 | 4.82 |
| 8 | 128 | 6.29 |
| 16 | 256 | 10.81 |
| 32 | 512 | 2.50 |

# Outline

- Background
- Low-Power Processor Design with NVM in High-Voltage Domain (NVSleep)
- Low-Power Processor Design with NVM in Low-Voltage Domain (Respin)
- **Security Research on Processors Equipped with NVM Caches (NV-Insecure)**

# NV-Insecure: When Non-Volatile Caches Meet Cold Boot Attacks

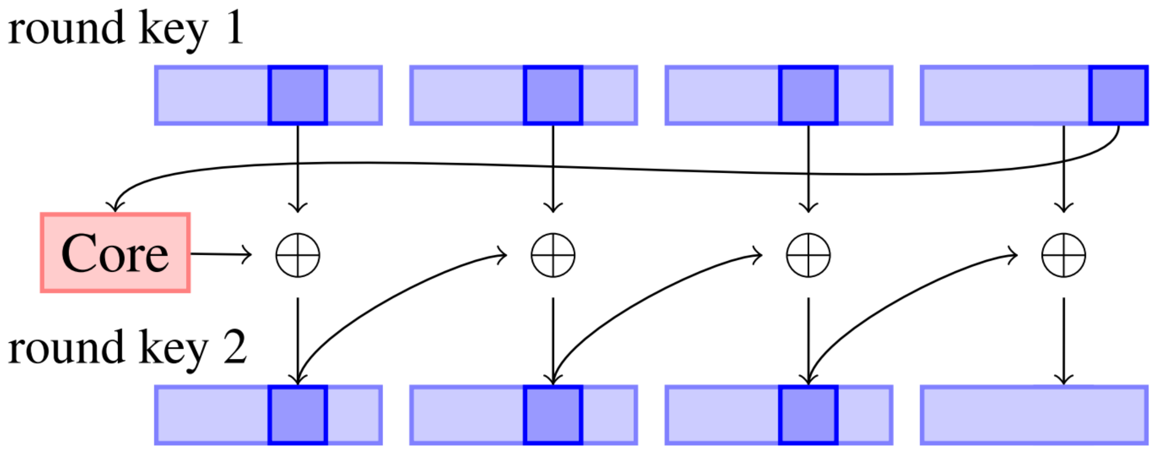
- The first work to examine cold boot attacks on non- volatile caches
- A comprehensive algorithm of finding AES keys in cache images has been developed
- Two types of cold boot attacks have been performed and shown to be effective on non-volatile caches
- A software-based countermeasure has been developed and proven to be effective with reasonable overhead

# Advanced Encryption Standard (AES)

- Symmetric block cipher with one master key for both encryption and decryption operations
- Key size ranges from 128-bit (AES-128), 192-bit (AES-192), and 256-bit (AES-256)
- An expanded key (aka. AES key schedule, 176-byte in AES-128) must be generated beforehand using the original key
- Byte substitution, shift row, mix column, and add round key operations will be performed during encryption/decryption

# Cold Boot Attacks

- Cooling DRAM to a certain low temperature can preserve its data for a short duration of time without power supply
- Examining data relationships in extracted memory image can identify AES keys used for disk encryption algorithms



Halderman et al., Lest We Remember: Cold Boot Attacks on Encryption Keys, citp.princeton.edu/research/memory

- Main memory based cold boot attacks have already been successfully conducted on desktop and mobile computers

# Current Countermeasures

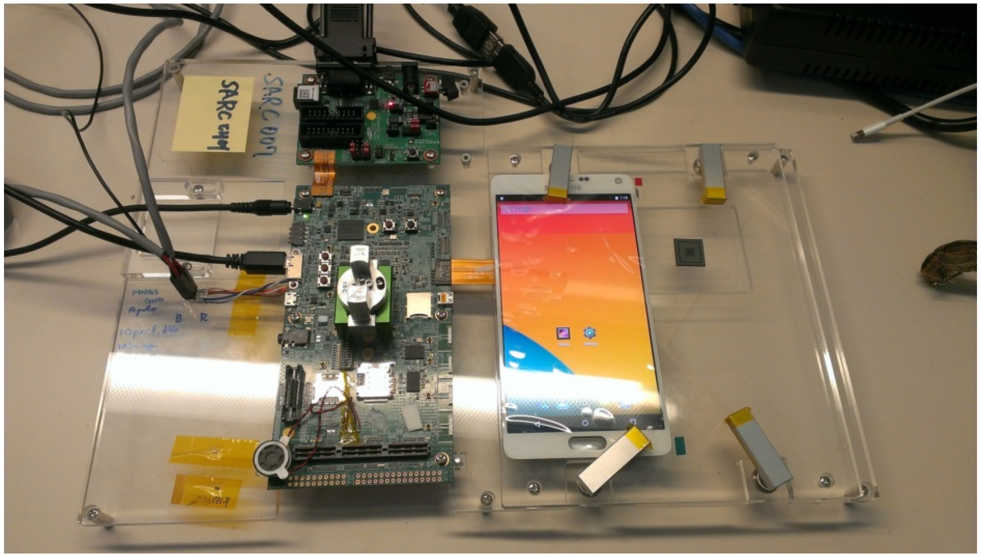
- Securing data at the destination side
  - Memory encryption technique
  - Acceptable at main memory level but can hardly be applied to caches because of its high performance overhead
- Protecting data from the source side
  - Keep secrets stored in CPU registers, caches, and other processor internal storage during system execution
  - Secret info can still be fetched into caches
  - A subset of this countermeasure even suggests keeping secrets in CPU caches

# Motivation

- Industry is pushing computers with NVMs into market soon
- After NVM replaces volatile-RAM in computers, cold boot attacks will be much easier to conduct
- Caches are highly likely to be replaced with NVM in the future but no previous work studied cold boot attacks on caches
- A few countermeasures even suggest keeping secrets in caches
- All in all, this will be the first work to study cold boot attacks on NVM caches

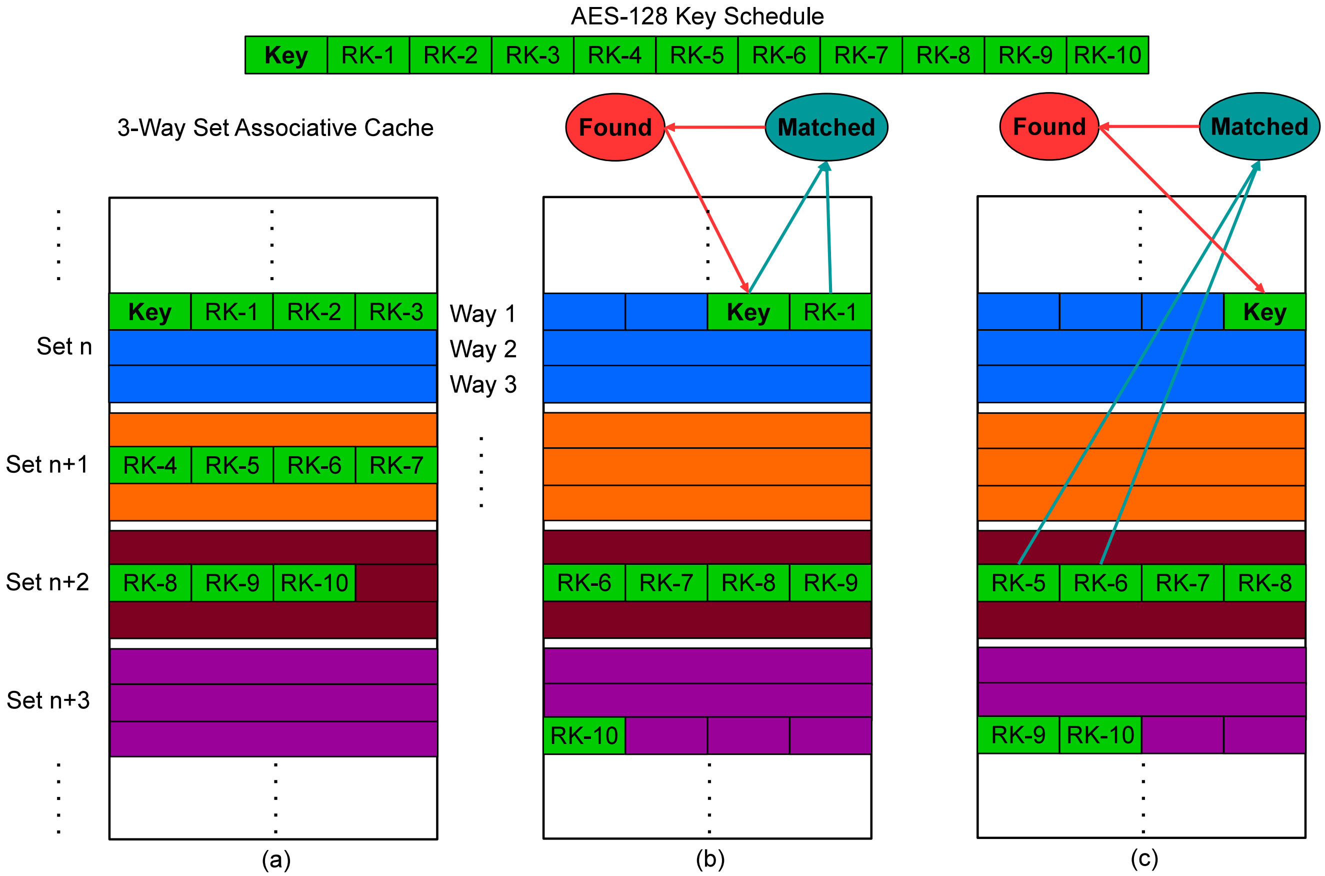
# Threat Model

- Attacker has physical access to the victim device
- Attacker has necessary equipments to extract data from CPU caches



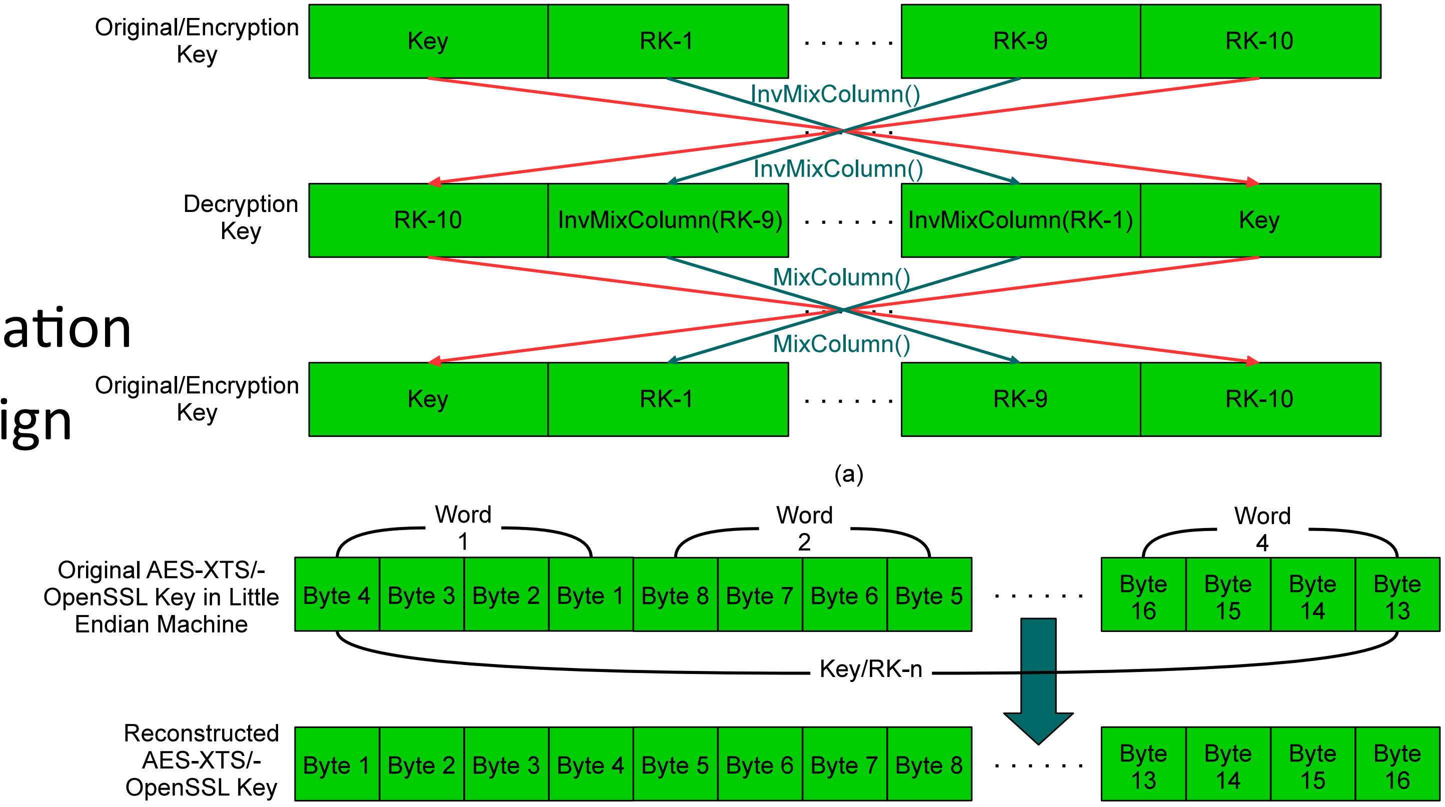
# Cache Aware AES Key Search Algorithm

- Non-contiguous memory space
- Incomplete key schedules



# Cache Aware AES Key Search Algorithm

- AES implementation dependent design



Original/Encryption Key: Key | RK-1 | . . . . . . | RK-9 | RK-10

InvMixColumn()

InvMixColumn()

Decryption Key: RK-10 | InvMixColumn(RK-9) | . . . . . . | InvMixColumn(RK-1) | Key

MixColumn()

MixColumn()

Original/Encryption Key: Key | RK-1 | . . . . . . | RK-9 | RK-10

(a)

Word 1 · Word 2 · Word 4

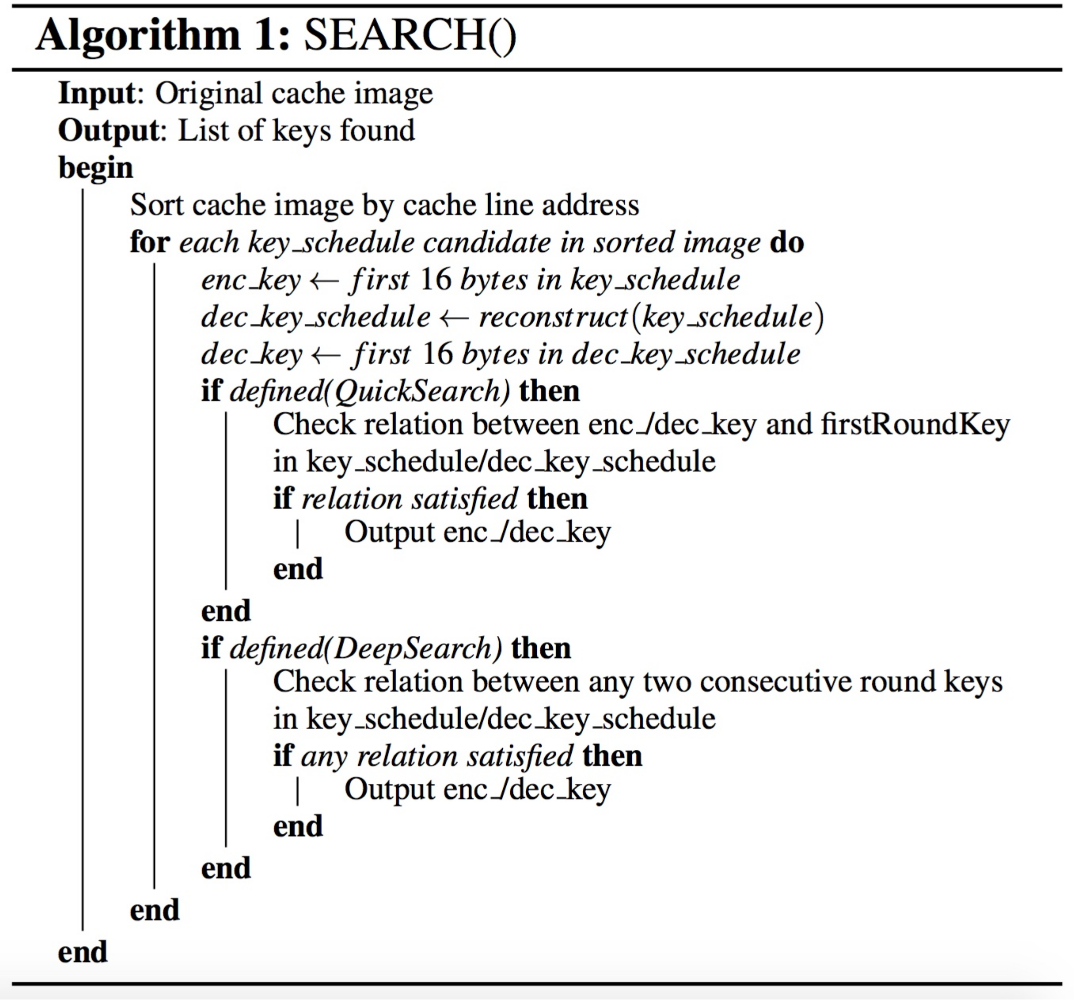
Original AES-XTS/-OpenSSL Key in Little Endian Machine: Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 8 | Byte 7 | Byte 6 | Byte 5 | . . . . . . | Byte 16 | Byte 15 | Byte 14 | Byte 13

Key/RK-n

Reconstructed AES-XTS/-OpenSSL Key: Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 | . . . . . . | Byte 13 | Byte 14 | Byte 15 | Byte 16

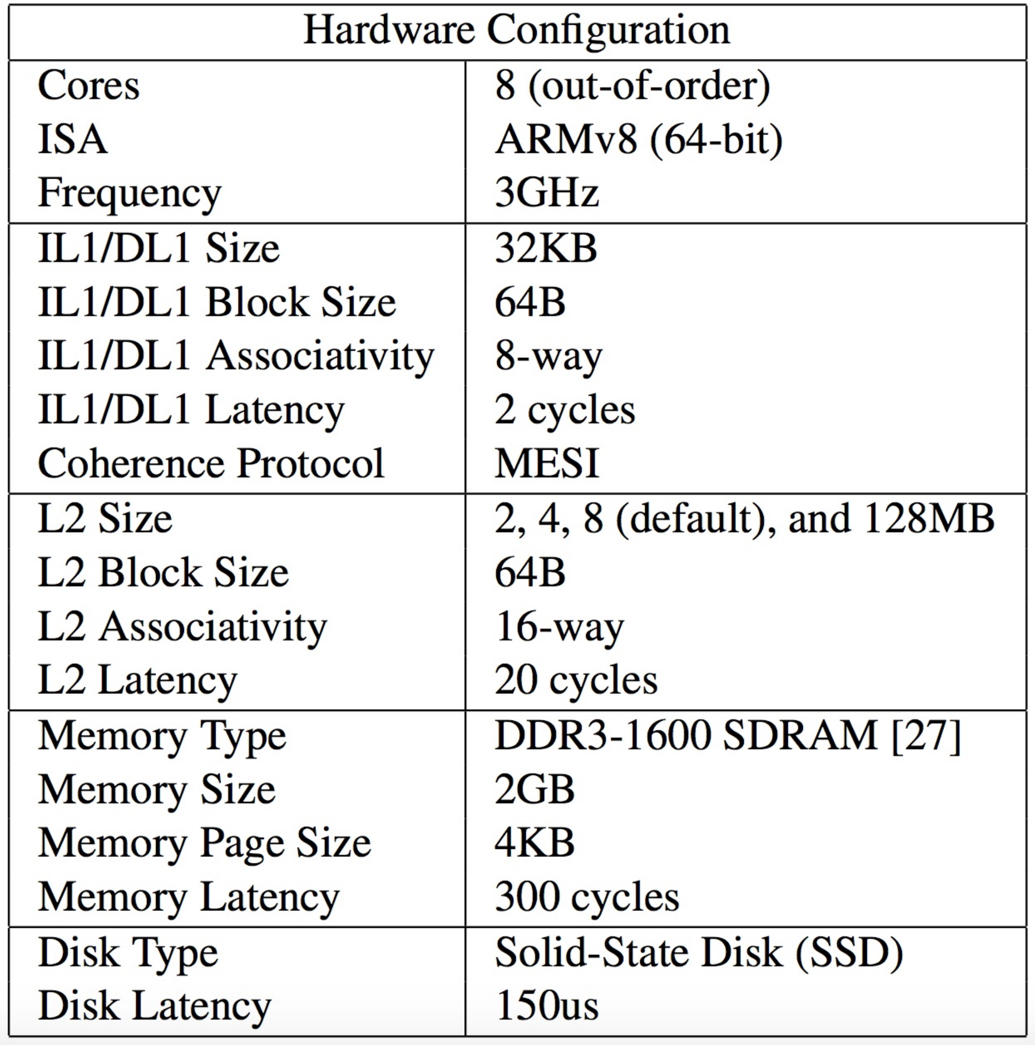
(b)

# Cache Aware AES Key Search Algorithm



```
Algorithm 1: SEARCH()
Input: Original cache image
Output: List of keys found
begin
    Sort cache image by cache line address
    for each key_schedule candidate in sorted image do
        enc_key ← first 16 bytes in key_schedule
        dec_key_schedule ← reconstruct(key_schedule)
        dec_key ← first 16 bytes in dec_key_schedule
        if defined(QuickSearch) then
            Check relation between enc_/dec_key and firstRoundKey
            in key_schedule/dec_key_schedule
            if relation satisfied then
                Output enc_/dec_key
            end
        end
        if defined(DeepSearch) then
            Check relation between any two consecutive round keys
            in key_schedule/dec_key_schedule
            if any relation satisfied then
                Output enc_/dec_key
            end
        end
    end
end
```

# Experimental Methodology



| Hardware Configuration | |
|---|---|
| Cores | 8 (out-of-order) |
| ISA | ARMv8 (64-bit) |
| Frequency | 3GHz |
| IL1/DL1 Size | 32KB |
| IL1/DL1 Block Size | 64B |
| IL1/DL1 Associativity | 8-way |
| IL1/DL1 Latency | 2 cycles |
| Coherence Protocol | MESI |
| L2 Size | 2, 4, 8 (default), and 128MB |
| L2 Block Size | 64B |
| L2 Associativity | 16-way |
| L2 Latency | 20 cycles |
| Memory Type | DDR3-1600 SDRAM [27] |
| Memory Size | 2GB |
| Memory Page Size | 4KB |
| Memory Latency | 300 cycles |
| Disk Type | Solid-State Disk (SSD) |
| Disk Latency | 150us |

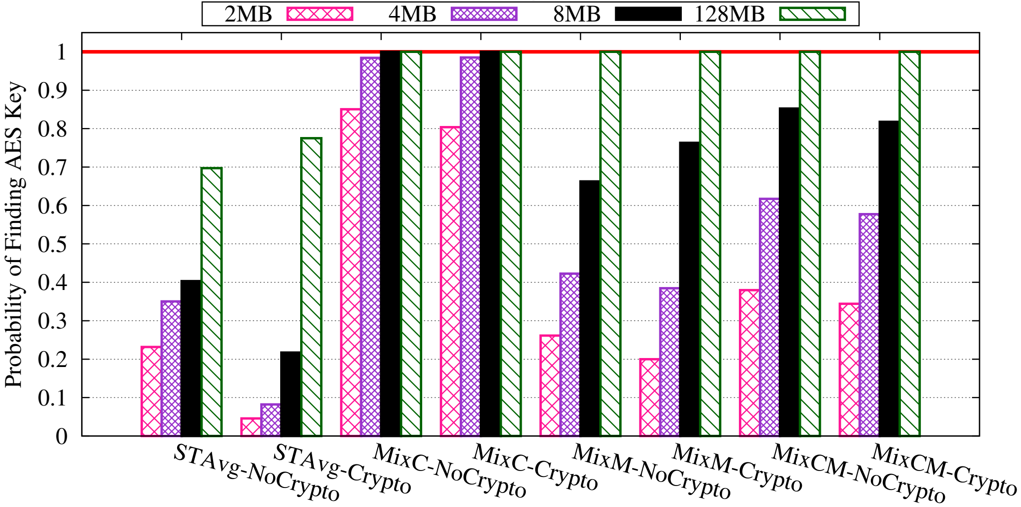
| Software Configuration | |
|---|---|
| Simulator | gem5 |
| OS | Ubuntu Trusty 14.04 64-bit |
| Disk Encryption Module | dm-crypt + LUKS |
| Encryption Algorithm | AES-XTS with 128-bit key |
| Application | SPEC CPU2006 |
| Execution | 1B insts to run |
| | 1M insts to sample |

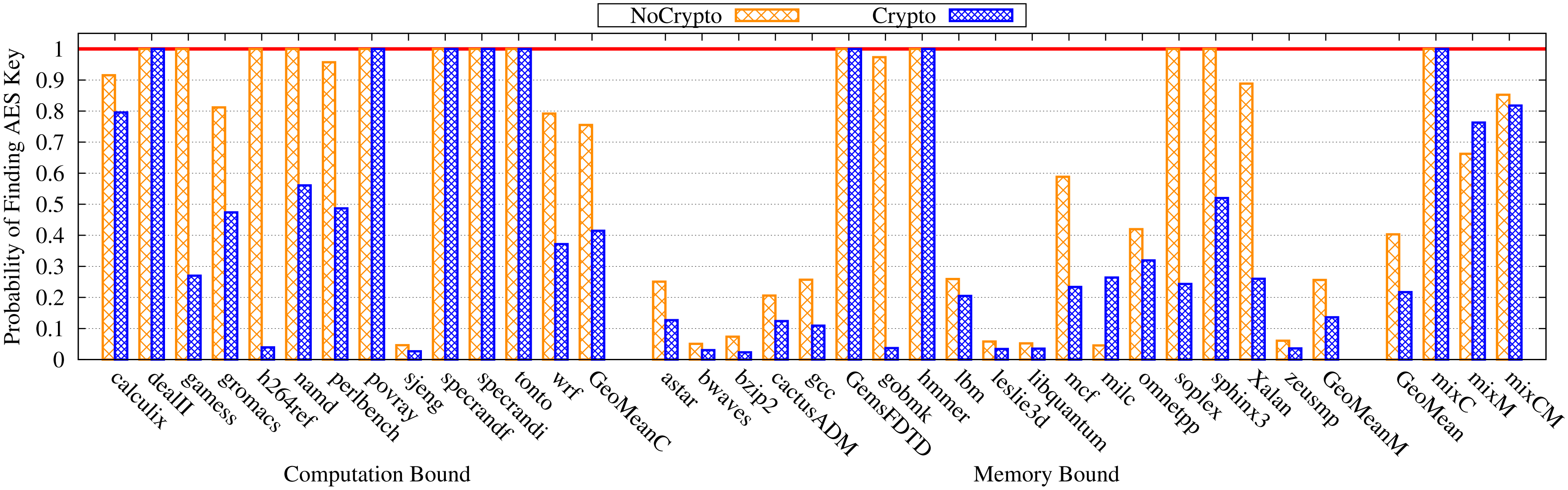
# Attack Scenarios

- Random Information Harvesting
  - Execution can be stopped at any given time to extract secrets from CPU caches
- Targeted Power-Off Attack
  - Conduct power-off operation on victim system and extract secrets from CPU caches
- Two Baselines for Evaluation
  - System without Cryptographic Acceleration Support (NoCrypto)
  - System with Cryptographic Acceleration Support (Crypto)

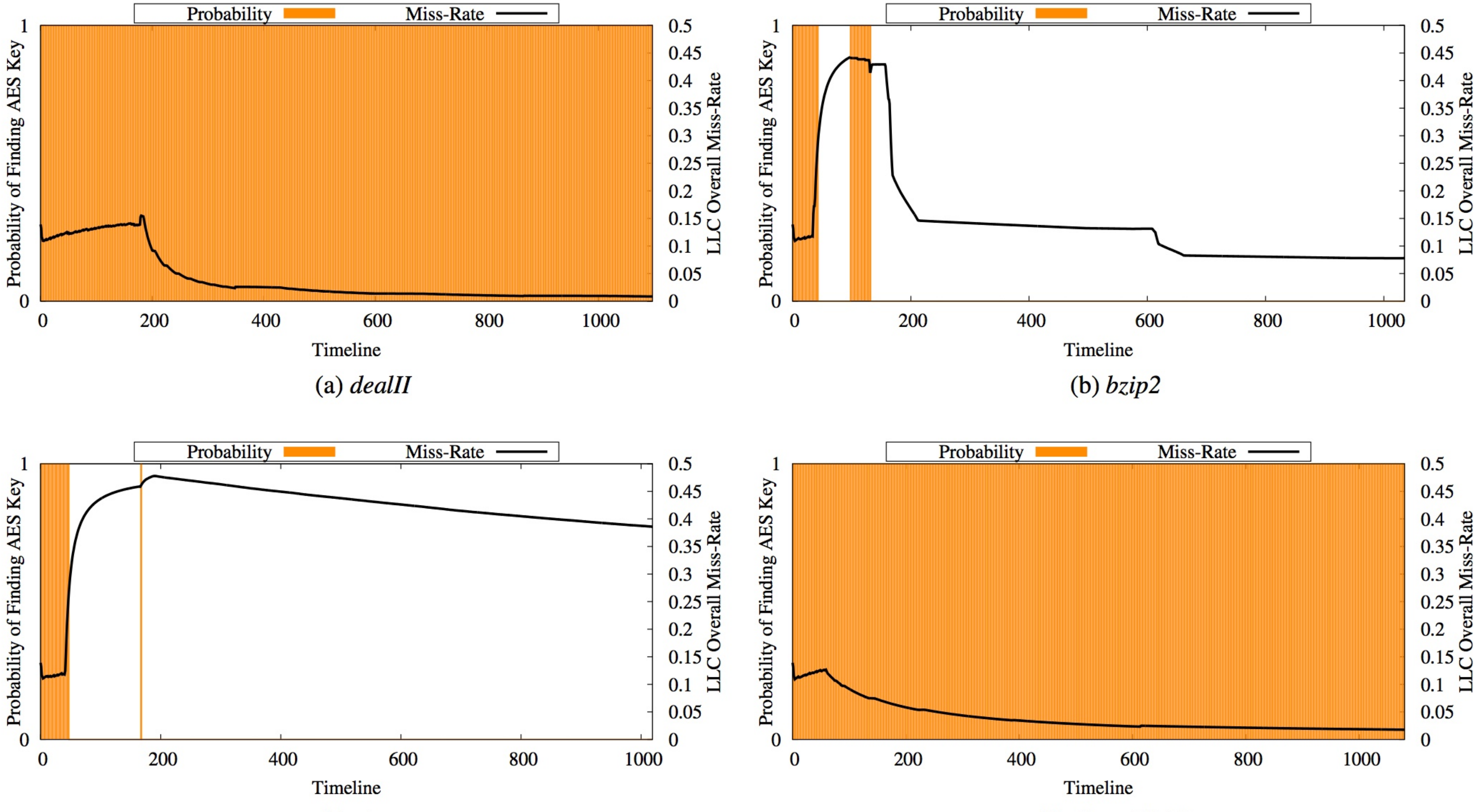
# Random Attack Analysis

- Two factors:
  - Encrypted Disk Accesses
  - Cache Evictions





# Random Attack Analysis



(a) *dealII*

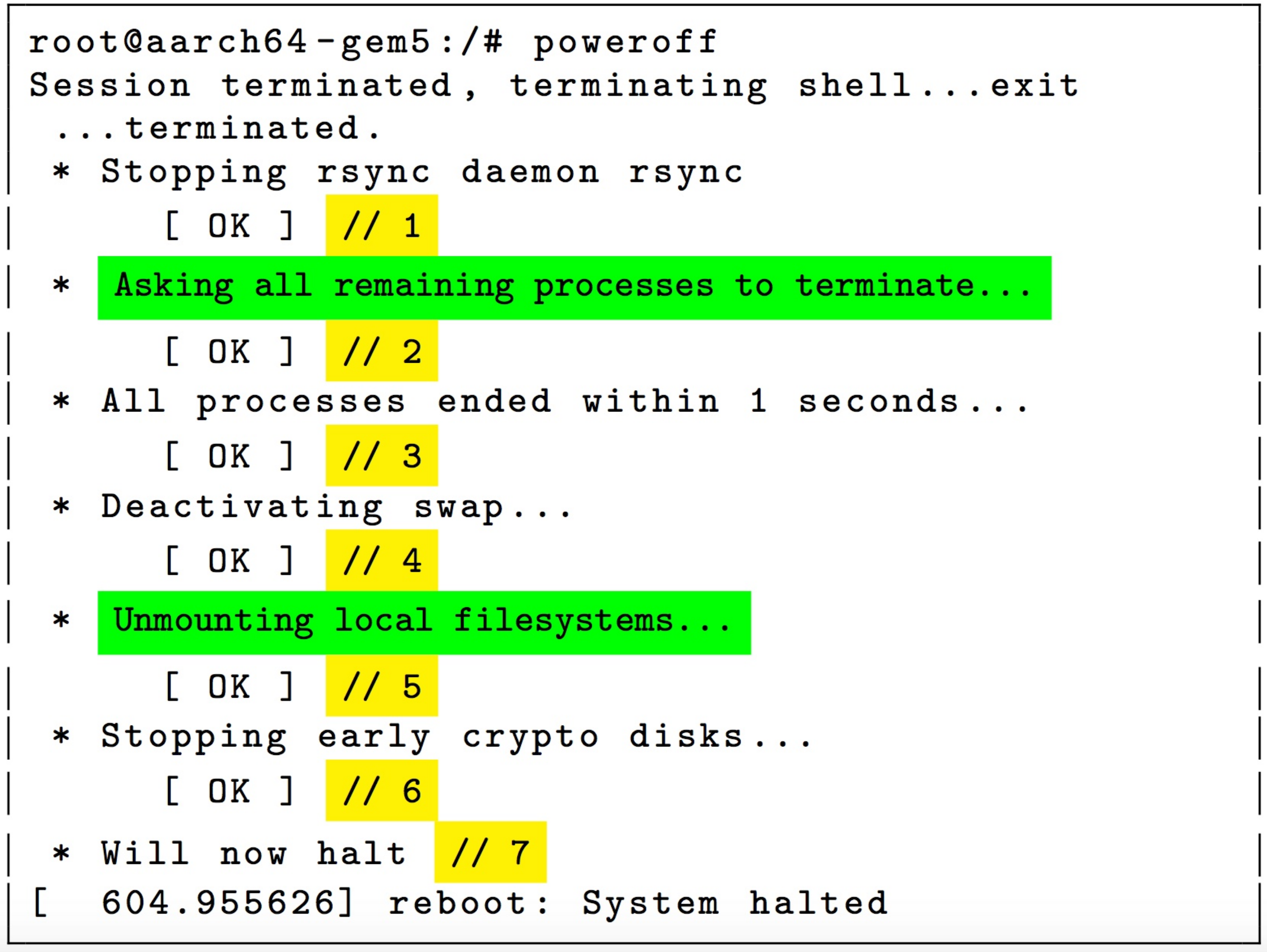
(b) *bzip2*

(c) *sjeng*

(d) *GemsFDTD*

# Power-Off Attack Analysis

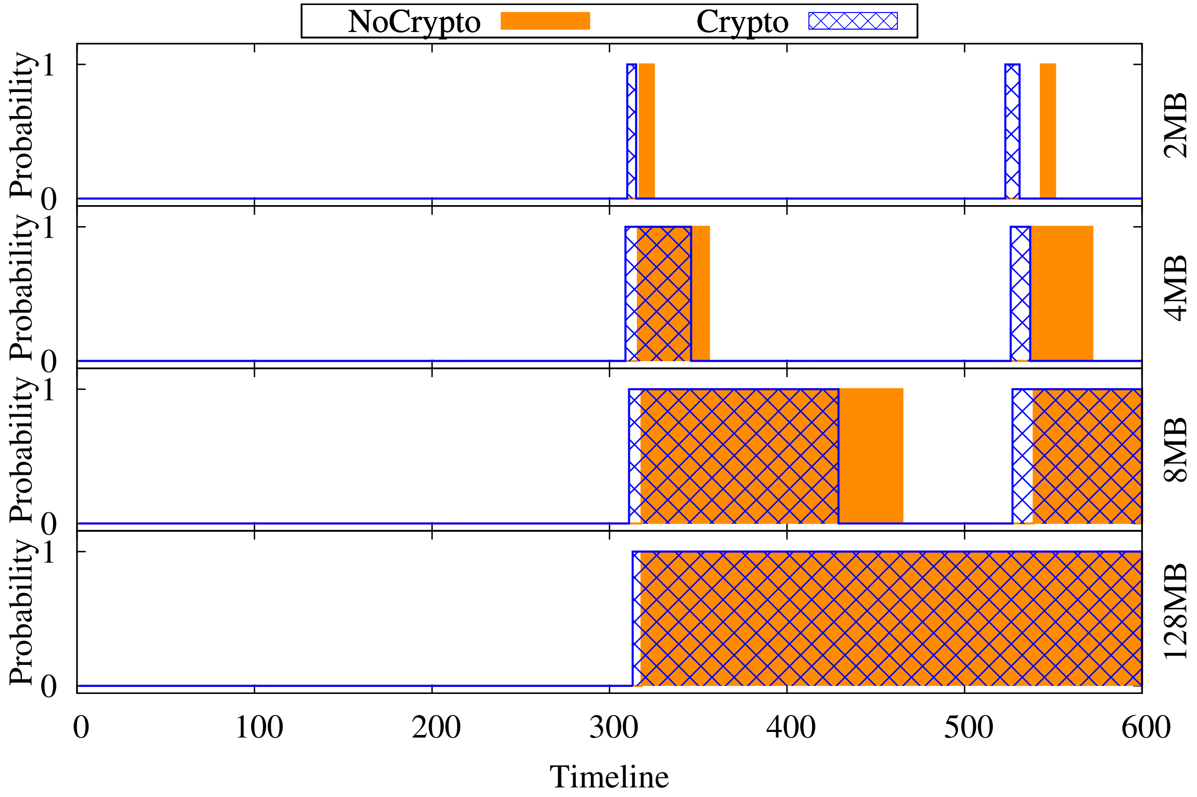
- Two modes:
  - Normal Power-Off: poweroff (-p)
  - Force Power-Off: poweroff -f



```
root@aarch64-gem5:/# poweroff
Session terminated, terminating shell...exit
 ...terminated.
 * Stopping rsync daemon rsync
     [ OK ] // 1
 * Asking all remaining processes to terminate...
     [ OK ] // 2
 * All processes ended within 1 seconds...
     [ OK ] // 3
 * Deactivating swap...
     [ OK ] // 4
 * Unmounting local filesystems...
     [ OK ] // 5
 * Stopping early crypto disks...
     [ OK ] // 6
 * Will now halt // 7
[  604.955626] reboot: System halted
```

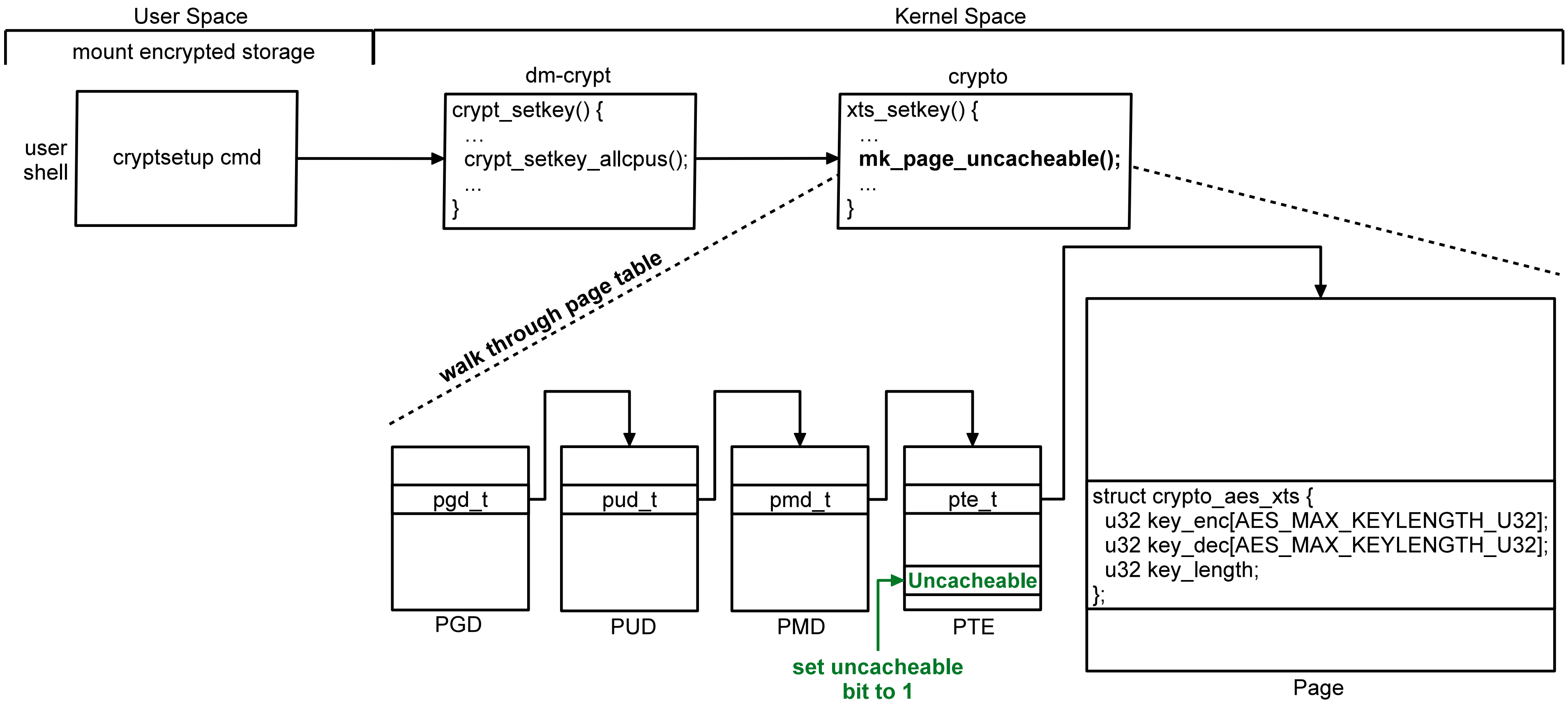
# Power-Off Attack Analysis

| Mode | Command | Keys exist in cache after power-off? | | | |
|---|---|---|---|---|---|
| | | 2MB | 4MB | 8MB | 128MB |
| Normal Power-off | poweroff (-p) | N | N | Y | Y |
| Forced Power-off | poweroff -f | Y | Y | Y | Y |



# Software-based Countermeasure

- Key idea: Marking secret information as uncacheable



User Space

mount encrypted storage

Kernel Space

user shell

cryptsetup cmd

dm-crypt

```
crypt_setkey() {
  …
  crypt_setkey_allcpus();
  …
}
```

crypto

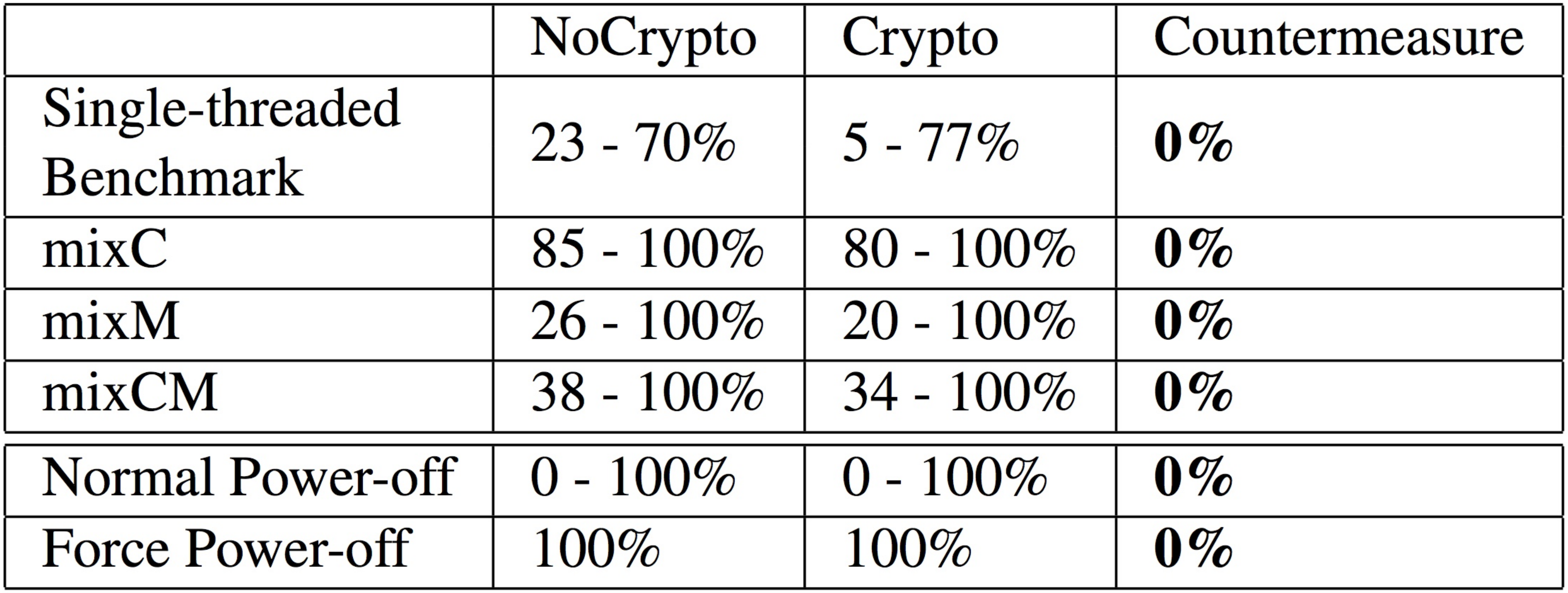
```
xts_setkey() {
  …
  mk_page_uncacheable();
  …
}
```

walk through page table

```
struct crypto_aes_xts {
  u32 key_enc[AES_MAX_KEYLENGTH_U32];
  u32 key_dec[AES_MAX_KEYLENGTH_U32];
  u32 key_length;
};
```

# Countermeasure Analysis

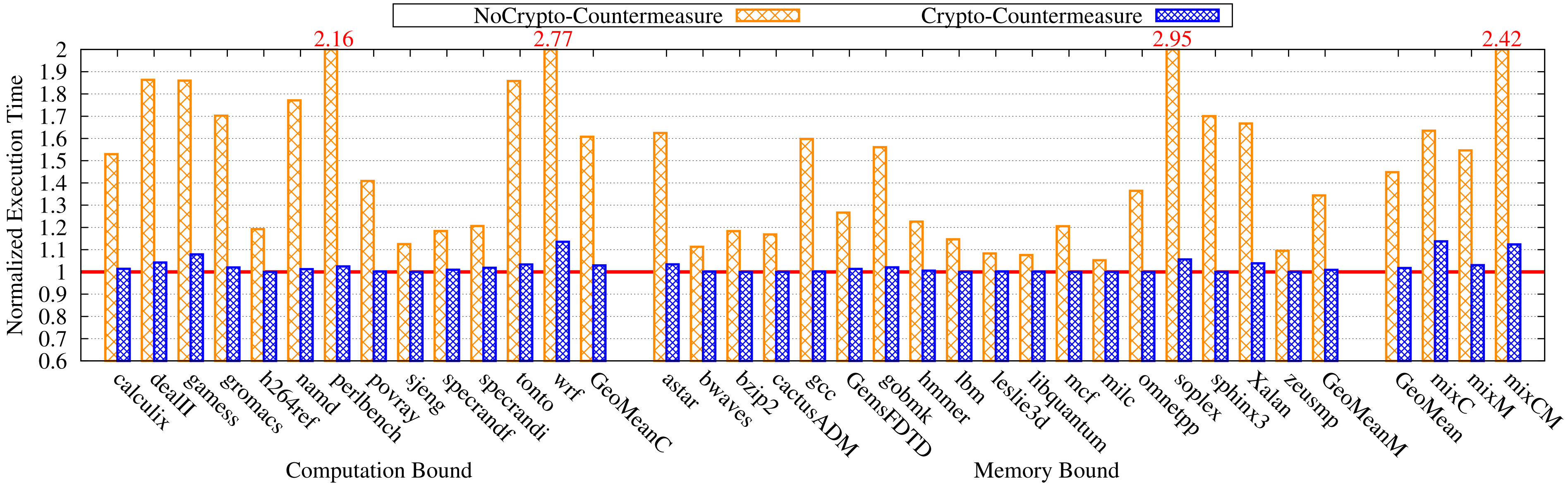
- Effectiveness



| | NoCrypto | Crypto | Countermeasure |
|---|---|---|---|
| Single-threaded Benchmark | 23 - 70% | 5 - 77% | **0%** |
| mixC | 85 - 100% | 80 - 100% | **0%** |
| mixM | 26 - 100% | 20 - 100% | **0%** |
| mixCM | 38 - 100% | 34 - 100% | **0%** |
| Normal Power-off | 0 - 100% | 0 - 100% | **0%** |
| Force Power-off | 100% | 100% | **0%** |

# Countermeasure Analysis

- Performance Overhead



# Future Work in Reducing Countermeasure Overhead

- Taking advantage of volatile SRAM write buffers equipped with every NVM device to store secret information
  - Hardware-based solution
  - Require hardware-software co-design (changing ISA, adding software interface, …)
  - Ideally will exhibit zero performance overhead

# Research Publication List

- **X. Pan**, A. Bacha, S. Rudolph, L. Zhou, Y. Zhang, R. Teodorescu "NV-Insecure: When Non-Volatile Caches Meet Cold Boot Attacks", **USENIX Security-2017** (In Submission)
- M. Samavatian, A. Bacha, I. Gururajaprasad, L. Zhou, **X. Pan**, R. Teodorescu "RNNFast: Accelerator for Recurrent Neural Networks using Domain Wall Memory", **ISCA-2017** (In Submission)
- **X. Pan**, A. Bacha, R. Teodorescu "Respin: Rethinking Near-Threshold Multiprocessor Design with Non-Volatile Memory", **IPDPS-2017**
- **X. Pan** and R. Teodorescu "NVSleep: Using Non-Volatile Memory to Enable Fast Sleep/Wakeup of Idle Cores", **ICCD-2014**
- **X. Pan** and R. Teodorescu "Using STT-RAM to Enable Energy-Efficient Near-Threshold Chip Multiprocessors", **PACT-2014** (Short Paper)
- T. Miller, R. Thomas, **X. Pan**, R. Teodorescu "VRSync: Characterizing and Eliminating Synchronization Induced Voltage Emergencies in Many-Core Processors", **ISCA-2012**
- T. Miller, **X. Pan**, R. Thomas, N. Sedaghati, R. Teodorescu "Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips", **HPCA-2012**

# Questions?

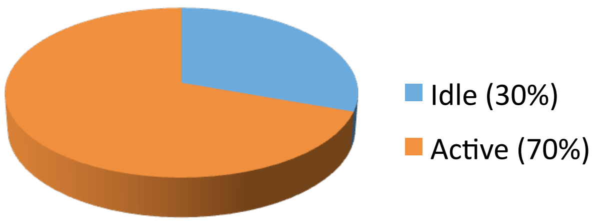
Thank you!

# Back-up Slides

# Motivation

- **Idle/leakage power:** source of inefficiency in CMPs
  - Expected to increase in future technologies
- Cores are often idle, wasting power
- Power gating can help
  - Functional units with little or no states (ALUs) – power gating OK
  - Most FUs have significant states (RF, ROB, …) – power gating expensive
- NVSleep Idea: non-volatile memory can enable fast micro-checkpointing
  - Reduce the performance overhead of power gating
  - Enable power gating during short idle intervals (e.g. stalls on LLC misses)

Idle Power Breakdown



Shimpi et al., The Haswell Review: Intel Core i7-4770K & i5-4670K Tested, www.anandtech.com

# STT-RAM in NVSleep

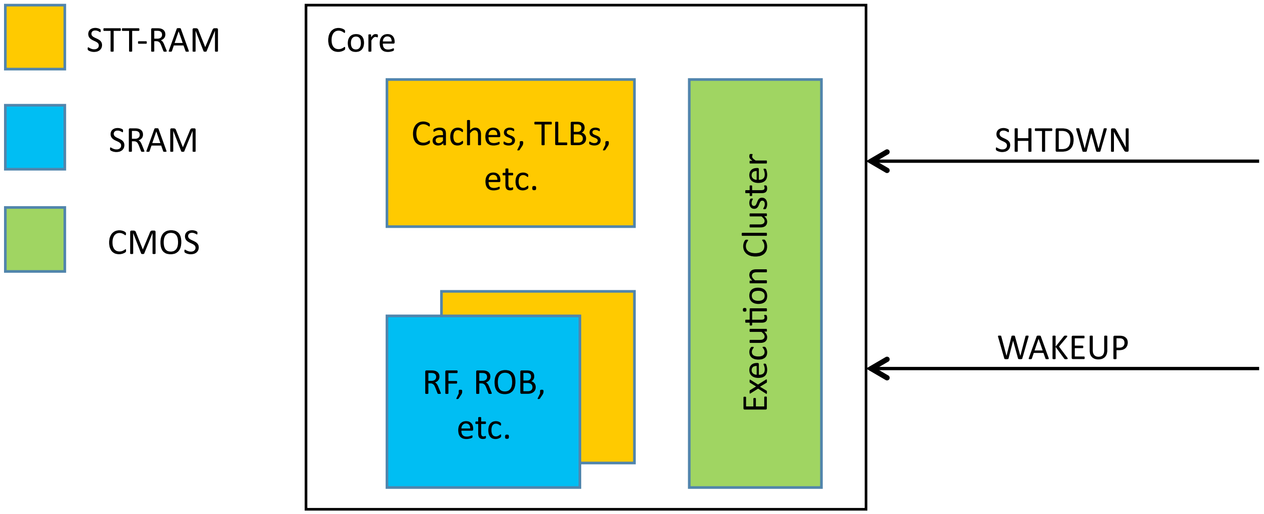
- We use Spin Transfer Torque RAM, a new type of magnetic memory
- STT-RAM can be a good candidate for NVSleep checkpointing

| **NVSleep checkpointing** | | **STT-RAM** |
|---|---|---|
| Non-volatility | ✓ | Long data retention time (as long as 10 years) |
| Low latency access | ✓ | Fast read (~0.9X SRAM) and slow write (~ 20X SRAM) ☹ |
| Low energy | ✓ | Low energy read (~0.7X SRAM) and high energy write (~20X SRAM) ☹ |

- STT-RAM has other good characteristics:
  - ~4X higher density than SRAM, better scalability
  - Infinite write endurance

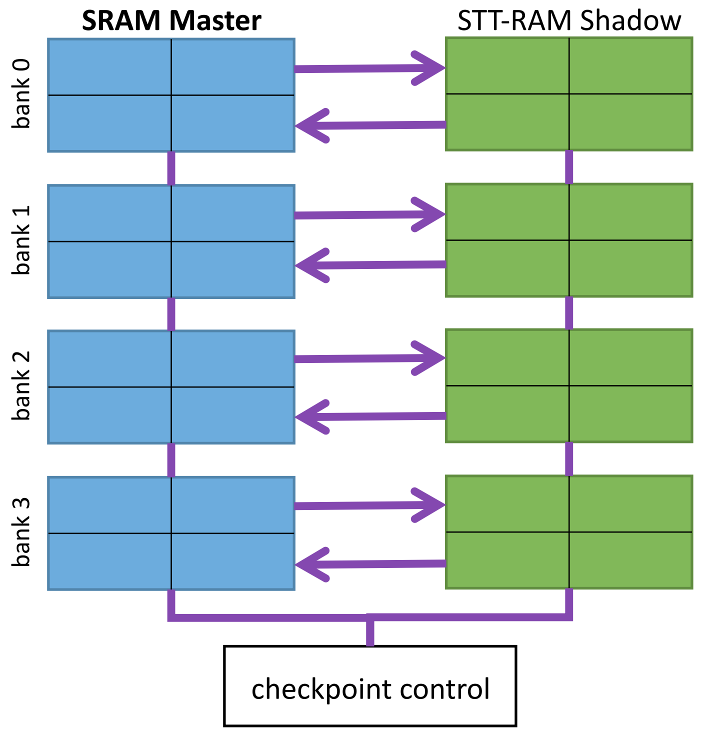
# NVSleep Framework

- NVSleep leverages STT-RAM for on-chip storage structures
  - Write latency-tolerant units (caches, TLBs, etc.) are implemented with STT-RAM (combined with SRAM write buffers to help hide long latency writes)
  - Write latency-sensitive units (RF, ROB, etc.) are implemented with hybrid SRAM/ STT-RAM design



# SRAM/STT-RAM hybrid Design

- SRAM for primary storage
- STT-RAM shadow of identical size used for micro-checkpointing
- Banked design to parallelize checkpointing process



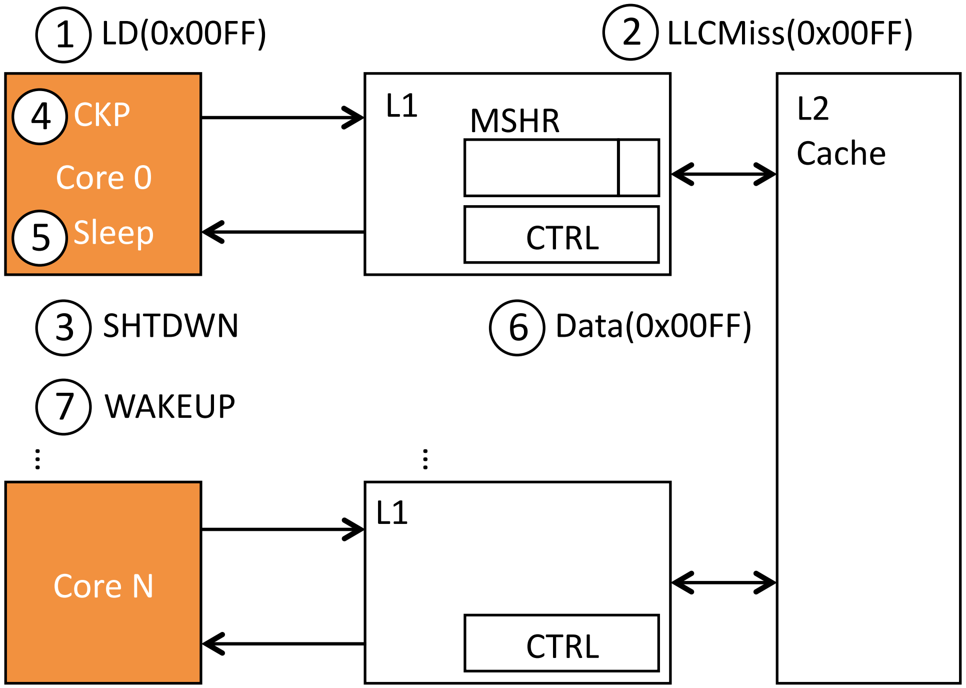
# NVSleep Implementation

- NVSleepMiss: Hardware-driven
  - Ideal for short idle events, detected by hardware
  - Our implementation: cores sleep on LLC misses
- NVSleepBarrier: Software API
  - Exposes NVSleep to the system software
  - Can be used by the OS or applications to "suspend" cores quickly
  - Ideal for software observable idle events such as blocking on synchronization (e.g. barriers, locks, etc.)
  - Our implementation: cores sleep when blocked on barriers

# NVSleepMiss: Hardware-driven

- Checkpointing and wakeup of cores are coordinated by the L1 cache controller of each core
- Hardware-driven checkpointing/wakeup sequence:

1. LD issued, missed in L1
2. LLC miss reported by LLC
3. Sleep signal sent to Core 0
4. Checkpointing starts
5. Core 0 goes to sleep after stalls
6. Missing data returns
7. Wakeup signal sent to Core 0



# NVSleepBarrier: Software API

- Expose micro-checkpointing system to software through API
  - Dedicated *sleep(0xADDR)* instruction
  - When executed on a core – it will shut down
  - Wakeup triggered by another core through write operation to *0xADDR*
- Example application in barrier:
  - All but last thread – *sleep(&sense)*
  - Last thread writes to *sense*, wakes-up all other threads

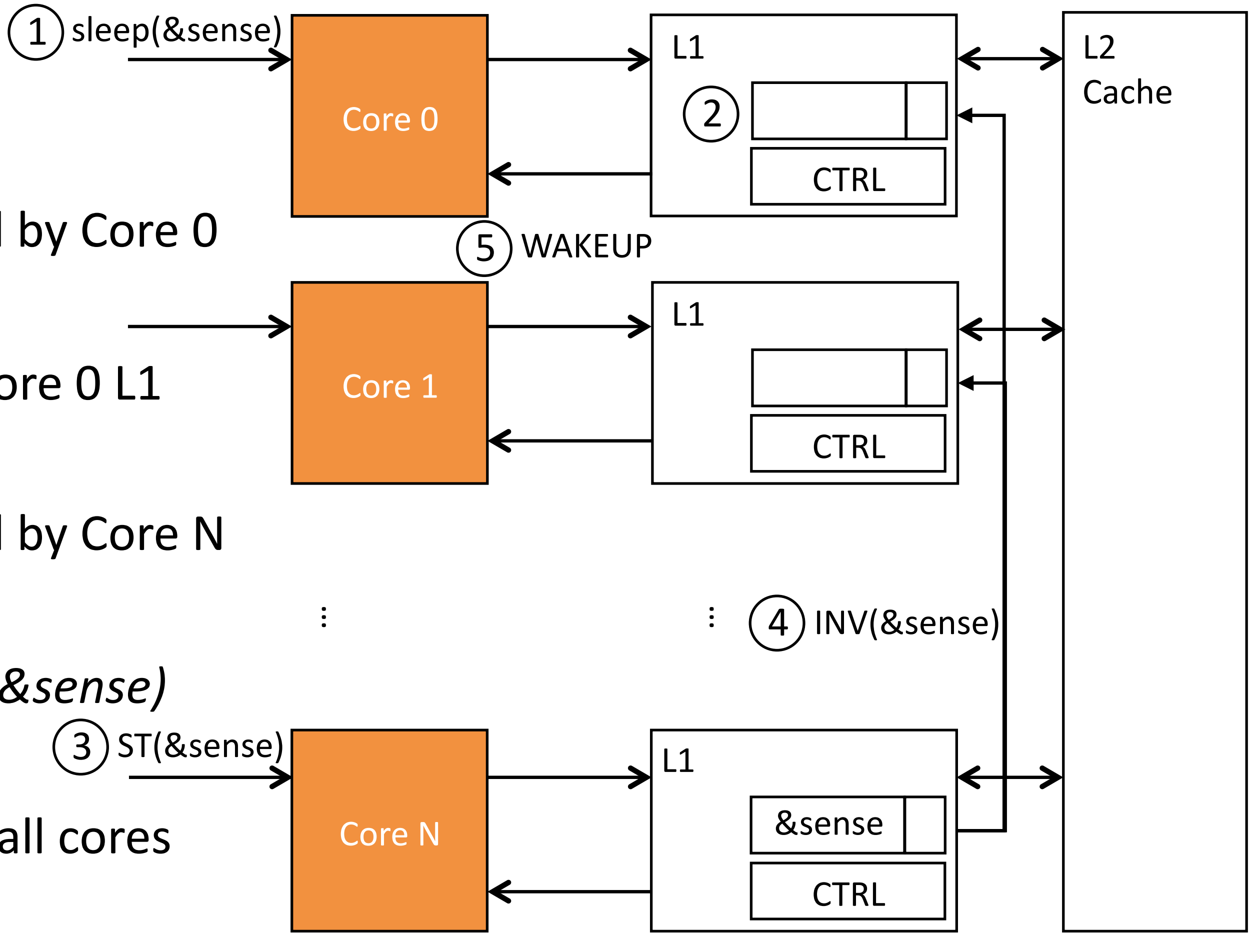
```
void barrier(int count, int sense, int num_threads)
{
        int local_sense;
        local_sense = !sense;

        if (count != (num_threads - 1)) {
                while (local_sense != sense) {
                        sleep(&sense);
                }
        }
        else {
                count = 0;
                sense = local_sense;
        }
}
```

# NVSleepBarrier: Software API

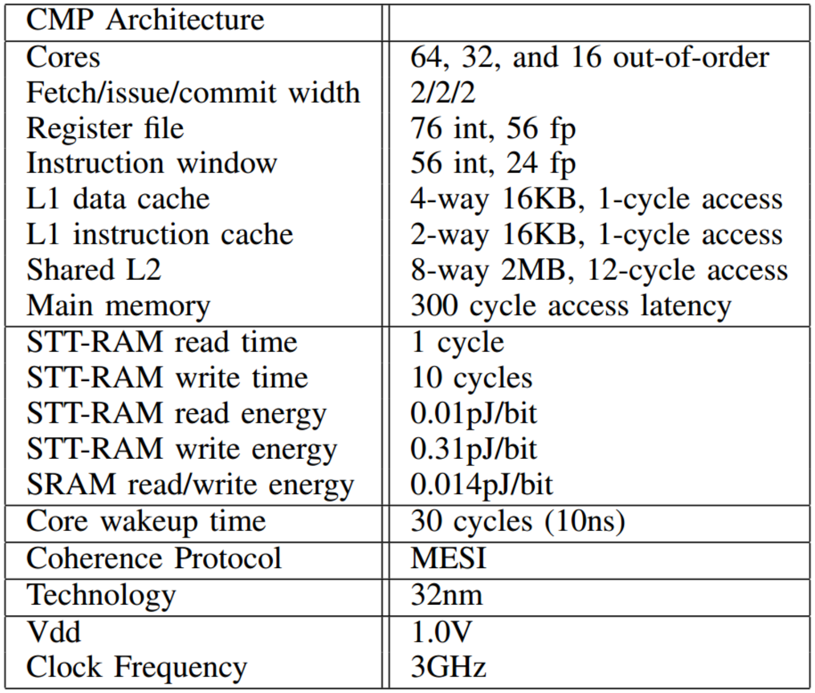
- Software-driven checkpointing/wakeup sequence:

1. *sleep(&sense)* executed by Core 0
2. *“&sense”* reserved in Core 0 L1
3. *write(&sense)* executed by Core N
4. Coherence: *invalidate(&sense)*
5. Wakeup signal sent to all cores



# Methodology

- Simulation Framework:
  - SESC for architecture simulation
  - CACTI, McPAT, and NVSim for power, energy, and area simulation
- Benchmarks:
  - Single-threaded: SPEC CPU2000
  - Multi-threaded: SPLASH2 and PARSEC

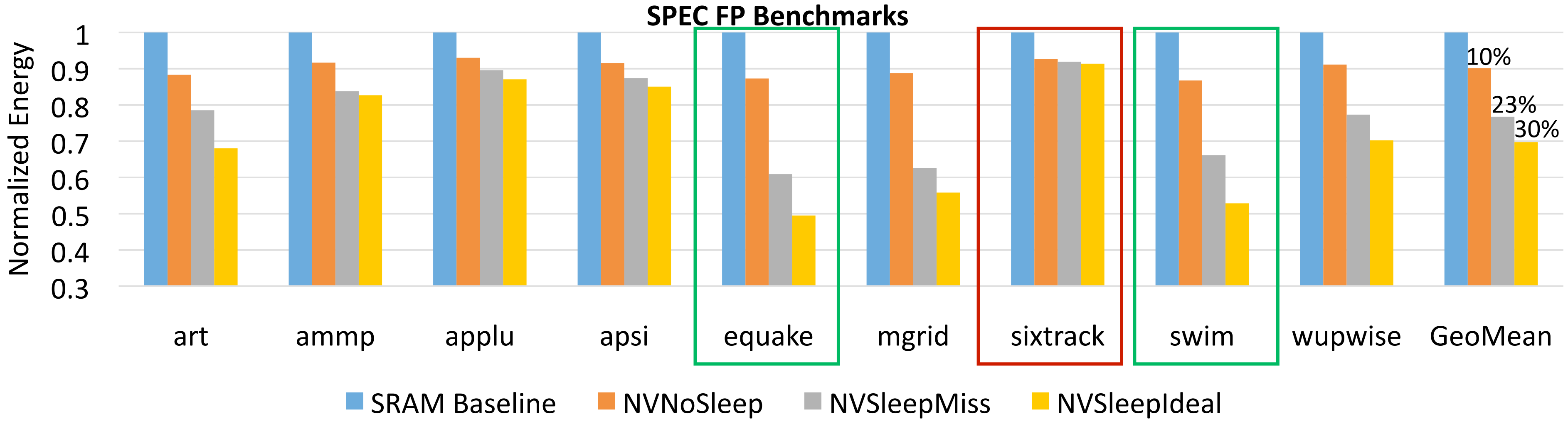


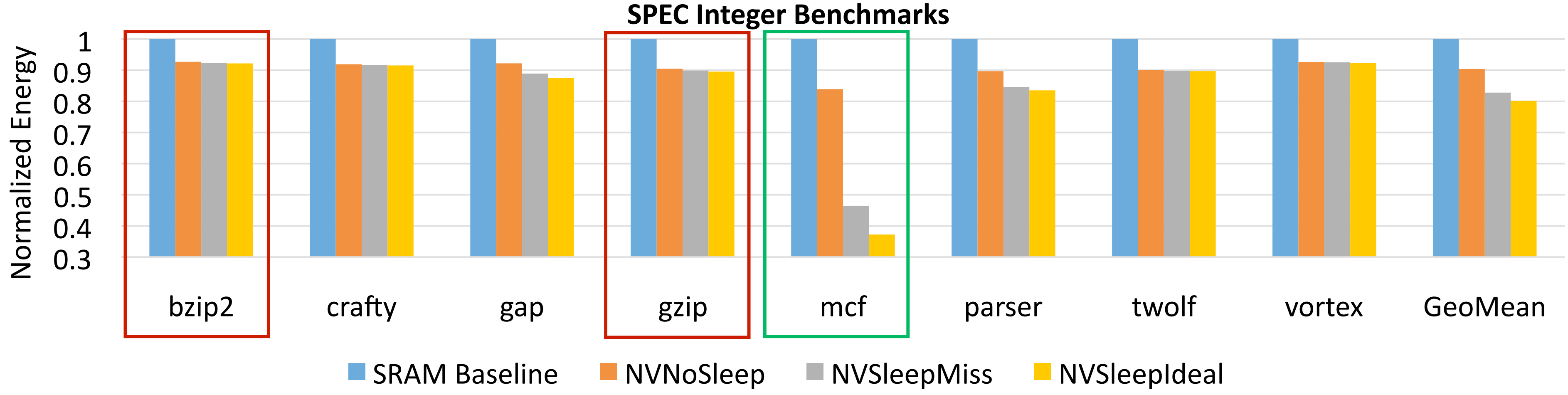
| CMP Architecture | |
|---|---|
| Cores | 64, 32, and 16 out-of-order |
| Fetch/issue/commit width | 2/2/2 |
| Register file | 76 int, 56 fp |
| Instruction window | 56 int, 24 fp |
| L1 data cache | 4-way 16KB, 1-cycle access |
| L1 instruction cache | 2-way 16KB, 1-cycle access |
| Shared L2 | 8-way 2MB, 12-cycle access |
| Main memory | 300 cycle access latency |
| STT-RAM read time | 1 cycle |
| STT-RAM write time | 10 cycles |
| STT-RAM read energy | 0.01pJ/bit |
| STT-RAM write energy | 0.31pJ/bit |
| SRAM read/write energy | 0.014pJ/bit |
| Core wakeup time | 30 cycles (10ns) |
| Coherence Protocol | MESI |
| Technology | 32nm |
| Vdd | 1.0V |
| Clock Frequency | 3GHz |

- Main Evaluated Configuration:
  - CMP with 64 out of order cores
  - 8-bank design SRAM/STT-RAM hybrid structures
  - 3.3ns STT-RAM write latency for checkpointing

# NVSleepMiss Energy Reduction

NVSleepMiss: 23% (SPECFP) and 17% (SPECINT) energy reduction

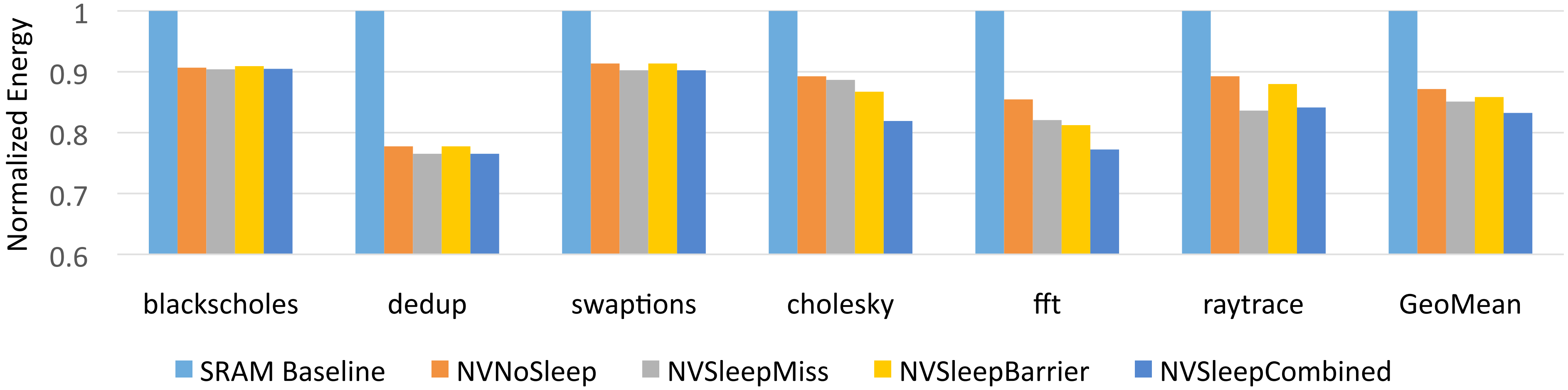

SPEC FP Benchmarks


SPEC Integer Benchmarks

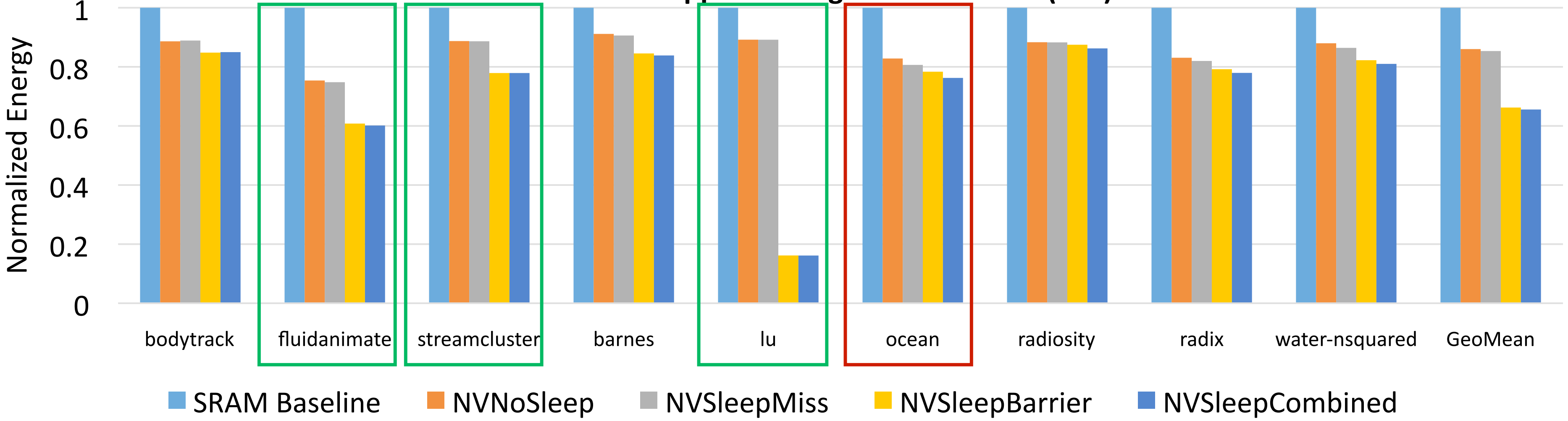
# NVSleepBarrier Energy Reduction

NVSleepBarrier: 34% energy reduction for apps with >10 barriers

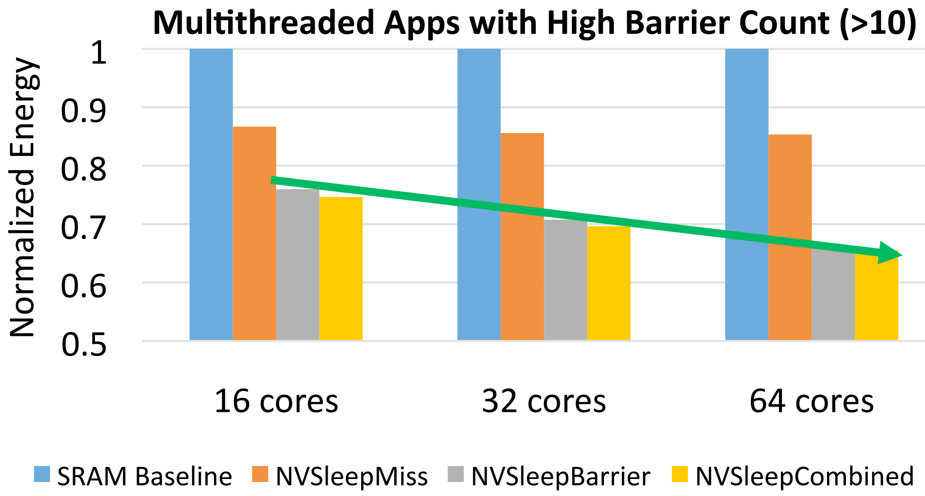
Multithreaded Apps with Low Barrier Count (<10)

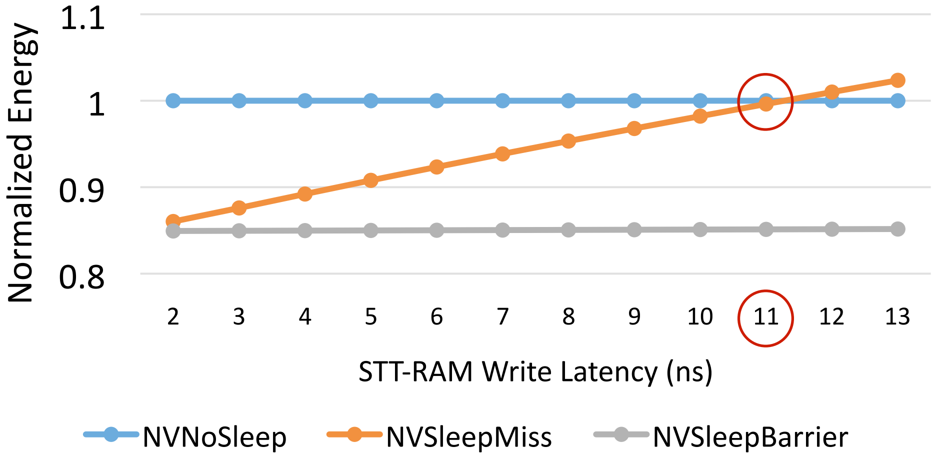


Multithreaded Apps with High Barrier Count (>10)

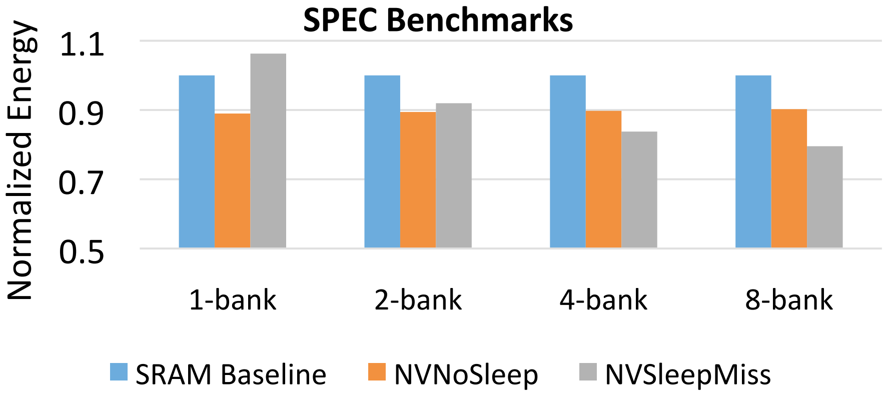


# Sensitivity Studies





| Num of Banks | energy/access (pJ) | area (mm2) |
|---|---|---|
| 1 | 0.448 | 0.007543 |
| 2 | 0.552 | 0.012091 |
| 4 | 0.628 | 0.018883 |
| 8 | 0.741 | 0.029032 |


SPEC Benchmarks
Normalized Energy
1-bank
2-bank
4-bank
8-bank
SRAM Baseline
NVNoSleep
NVSleepMiss