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

Xiang Pan and Radu Teodorescu
Computer Architecture Research Lab

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

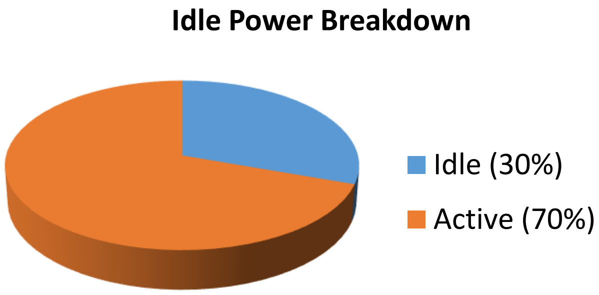
THE OHIO STATE UNIVERSITY

Outline

- **Motivation**
- NVSleep Framework Design
- NVSleep Implementation
- Evaluation
- Conclusion

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)



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) ☹

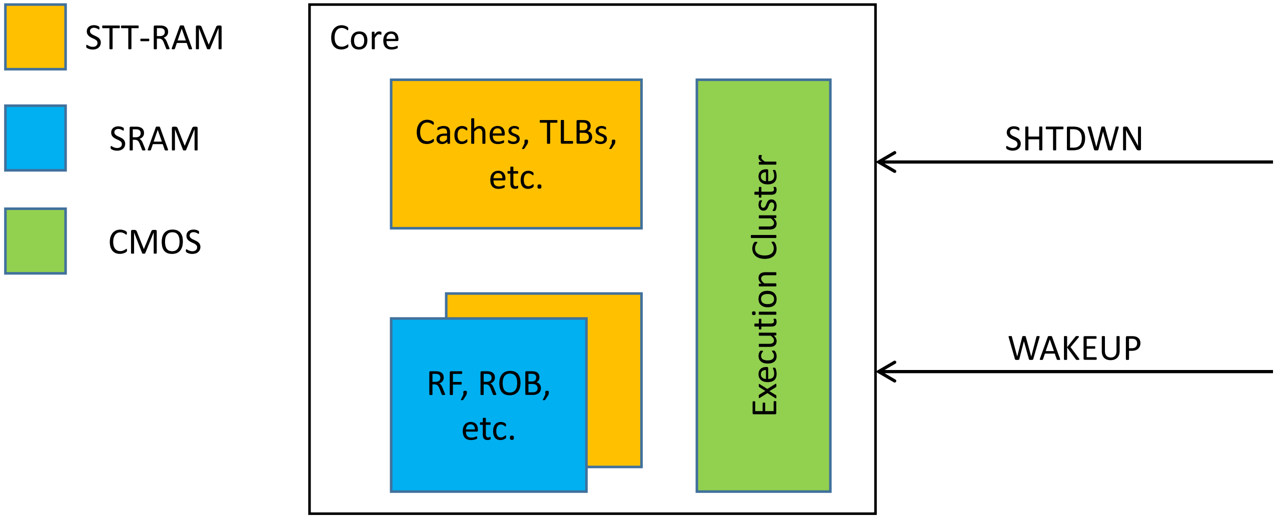
- STT-RAM has other good characteristics:
  - ~4X higher density than SRAM, better scalability
  - Infinite write endurance

Outline

- Motivation
- **NVSleep Framework Design**
- NVSleep Implementation
- Evaluation
- Conclusion

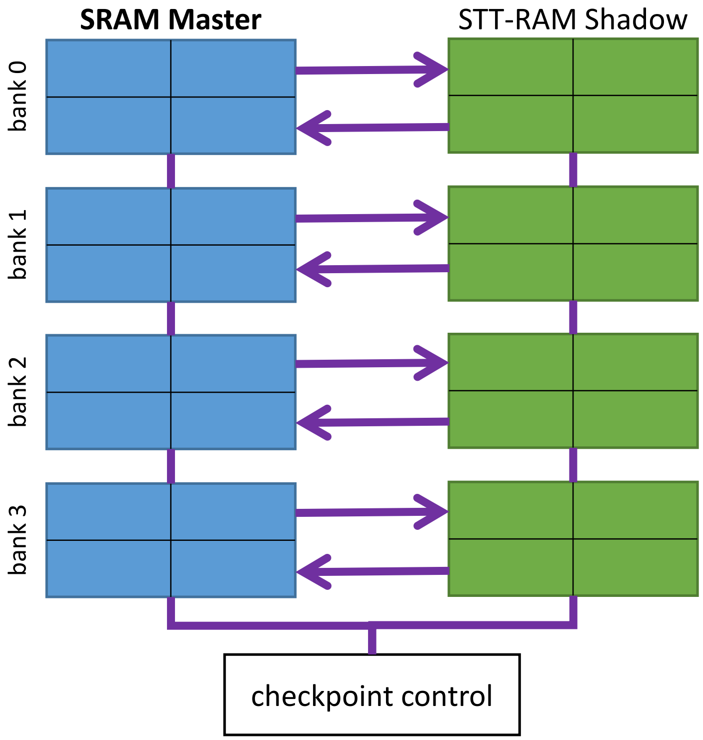
NVSleep Framework

- 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



Outline

- Motivation
- NVSleep Framework Design
- **NVSleep Implementation**
- Evaluation
- Conclusion

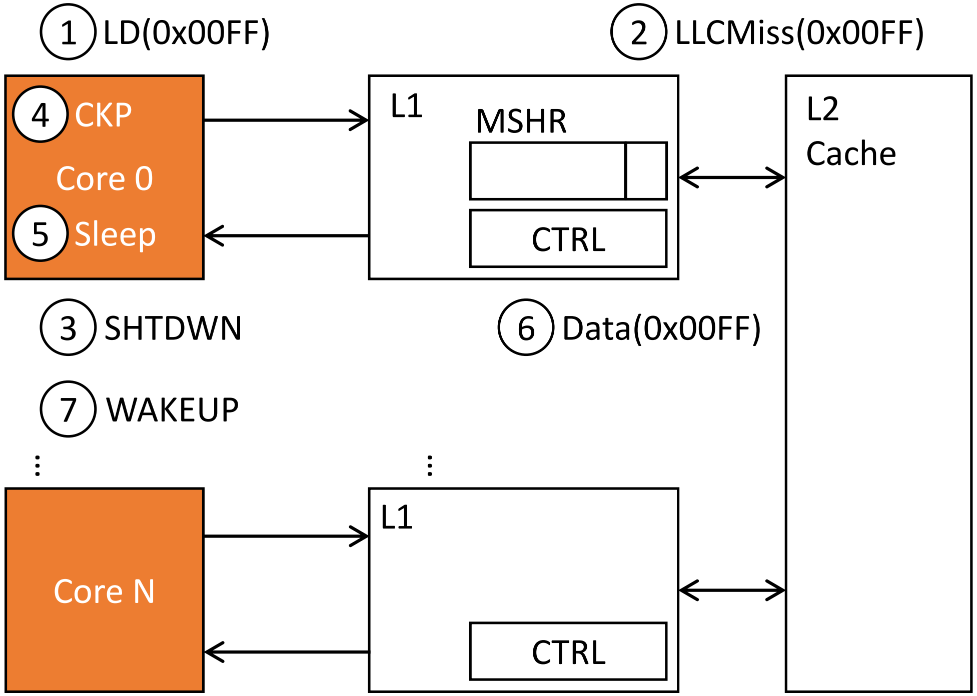
NVSleep Implementation

- 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

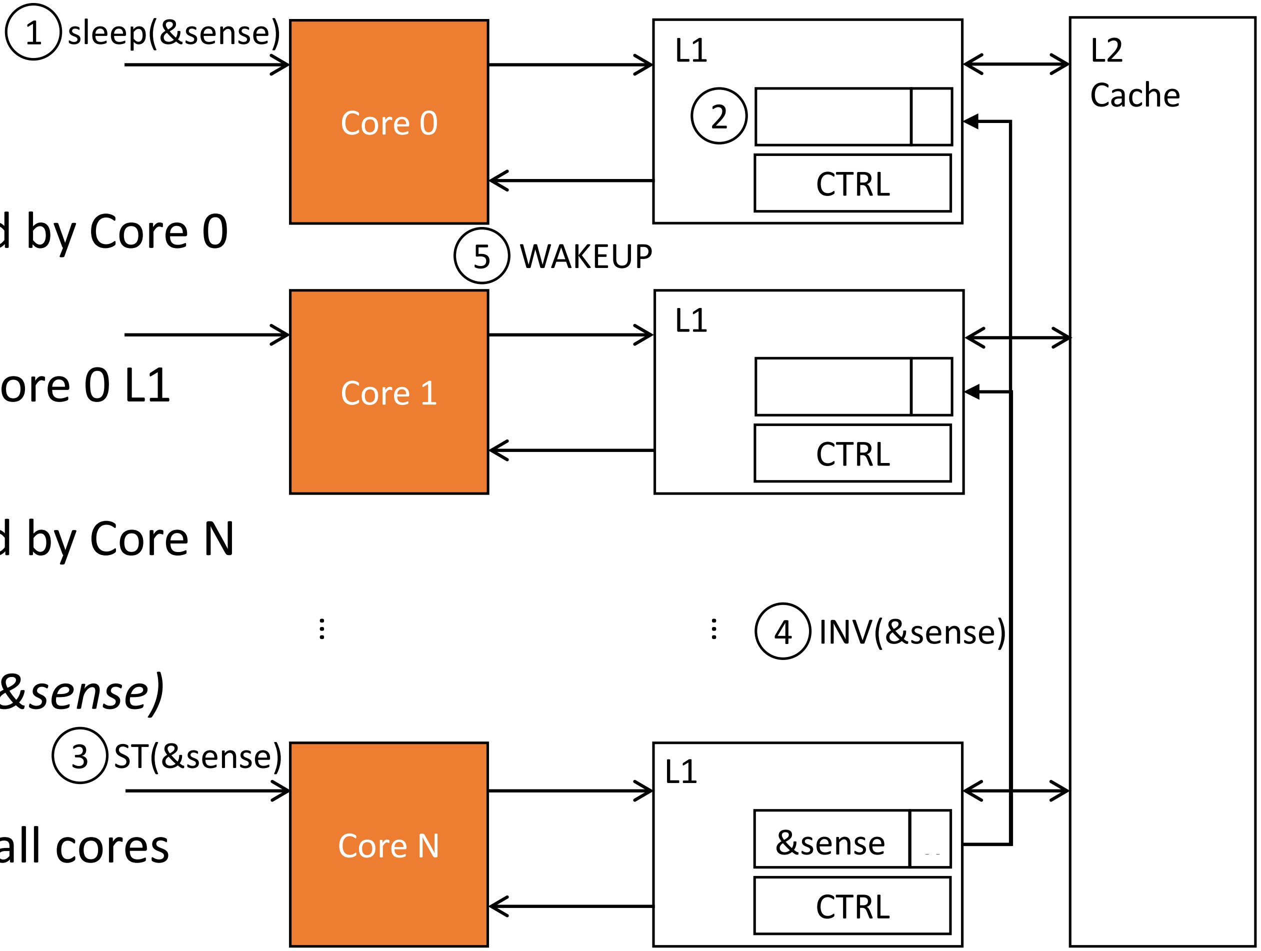
```
void barrier(int count, int sense, int num_threads)
{
        int local_sense;
        local_sense = !sense;

        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

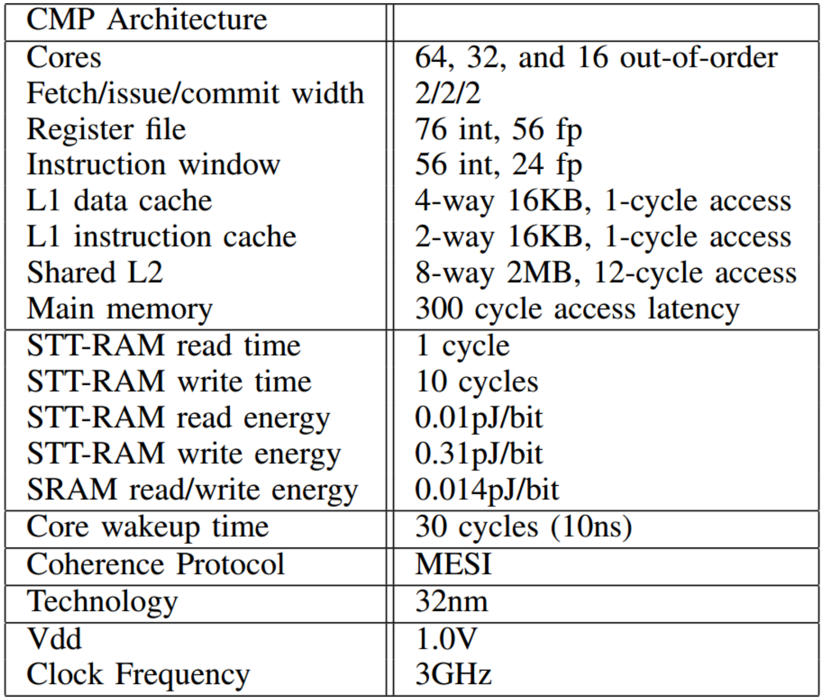


Outline

- Motivation
- NVSleep Framework Design
- NVSleep Implementation
- **Evaluation**
- Conclusion

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

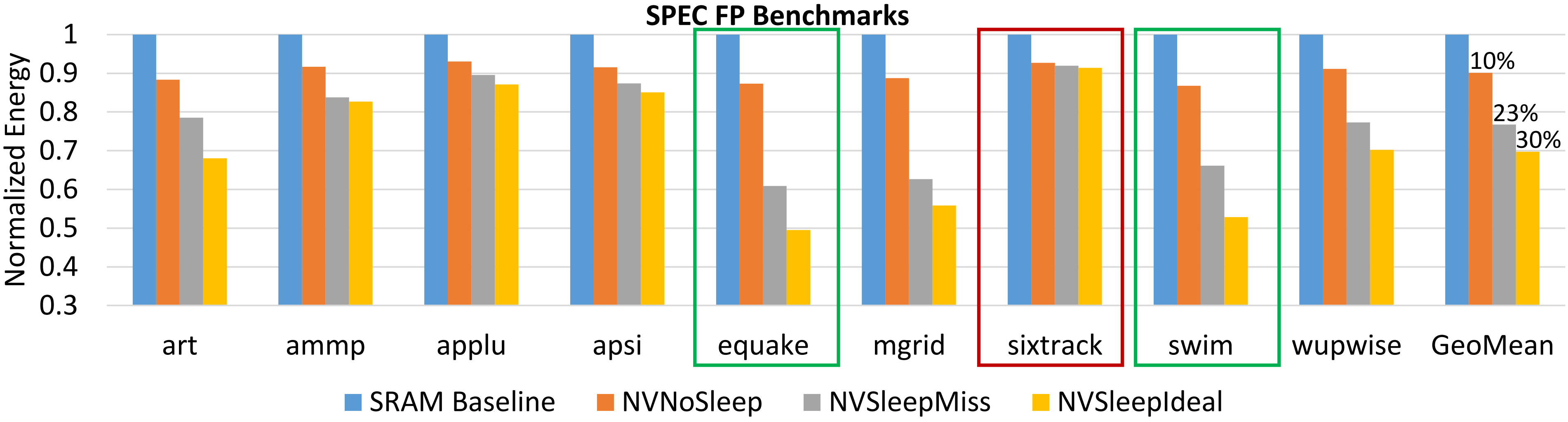


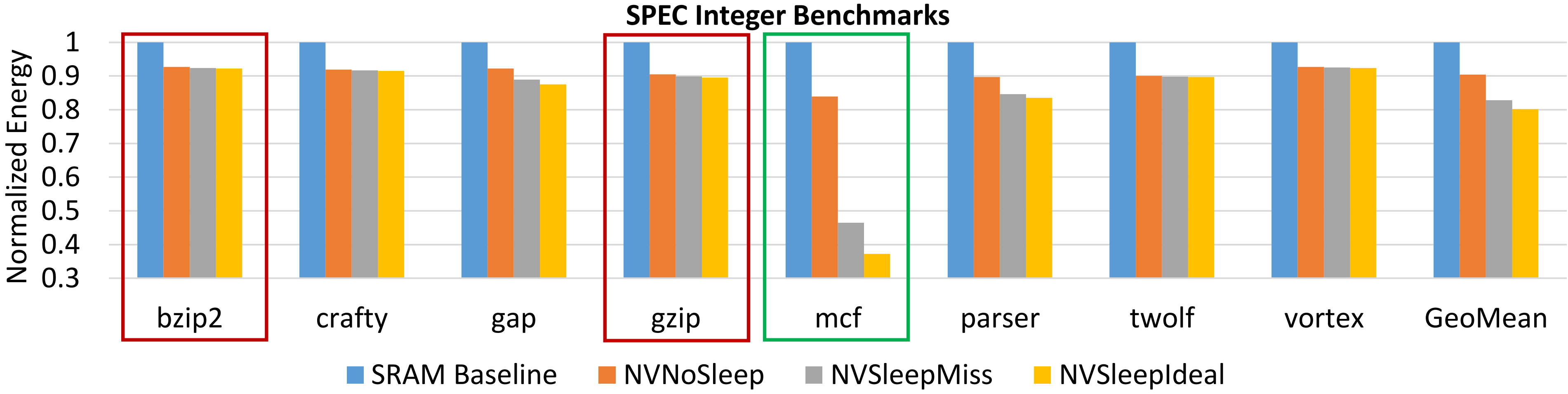
CMP Architecture	
Cores	64, 32, and 16 out-of-order
Fetch/issue/commit width	2/2/2
Register file	76 int, 56 fp
Instruction window	56 int, 24 fp
L1 data cache	4-way 16KB, 1-cycle access
L1 instruction cache	2-way 16KB, 1-cycle access
Shared L2	8-way 2MB, 12-cycle access
Main memory	300 cycle access latency
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

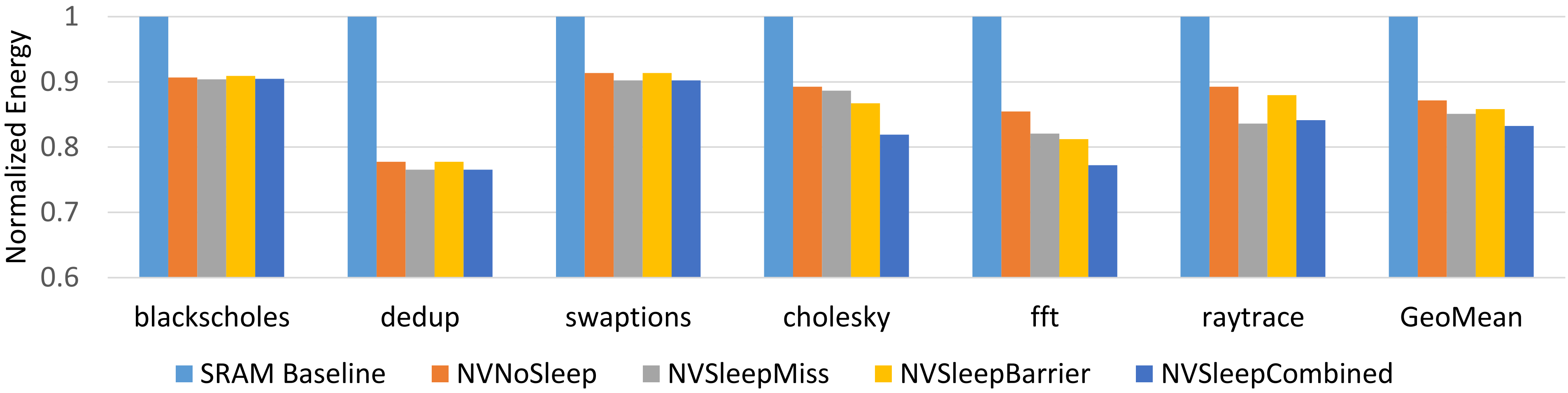




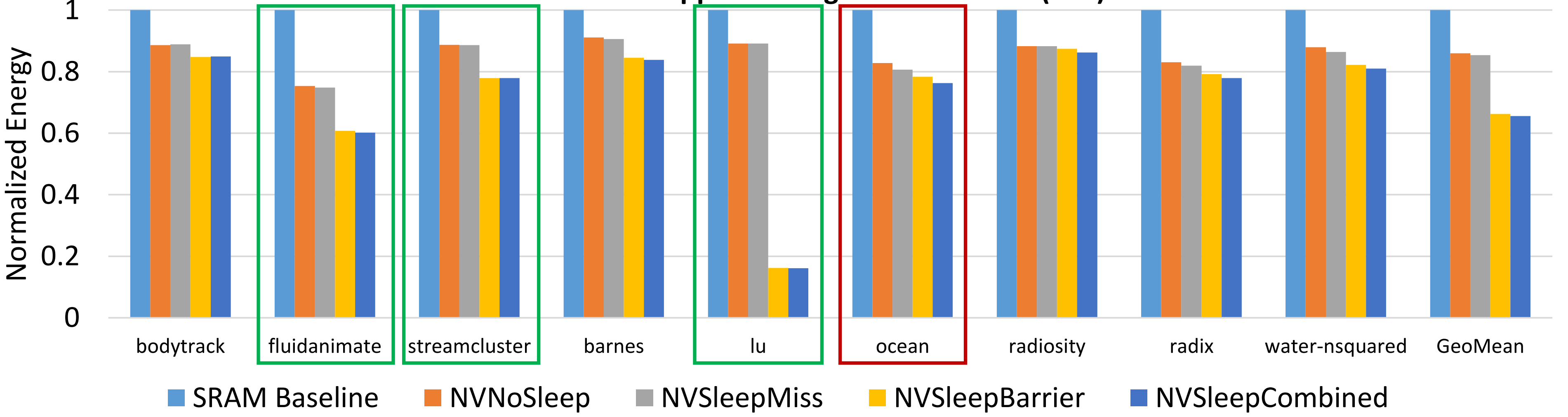
NVSleepBarrier Energy Reduction

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

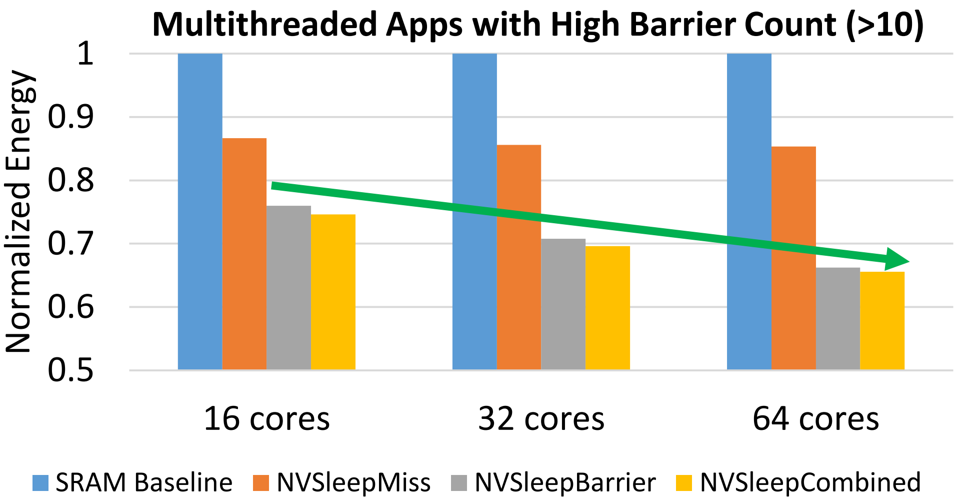
Multithreaded Apps with Low Barrier Count (<10)

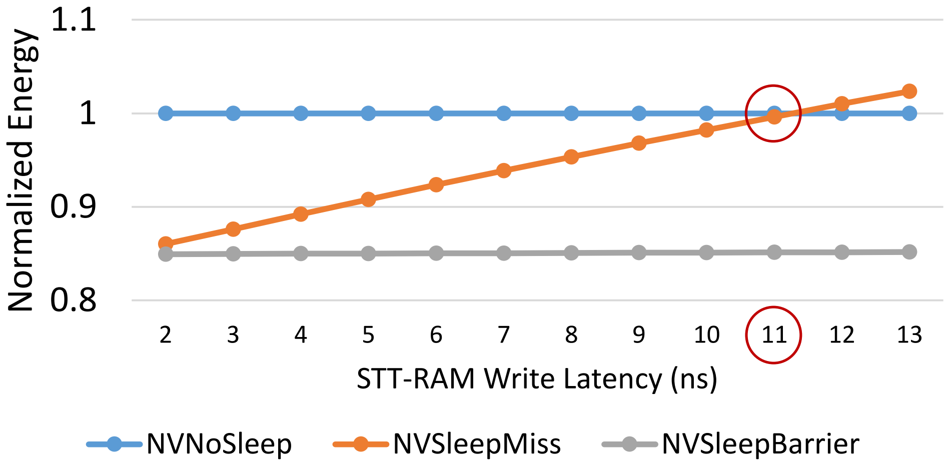


Multithreaded Apps with High Barrier Count (>10)

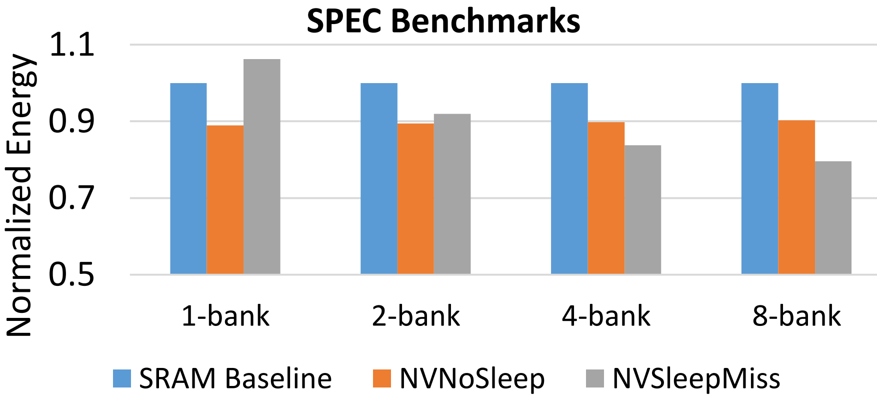


Sensitivity Studies





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



More details can be found in the paper!

Conclusion

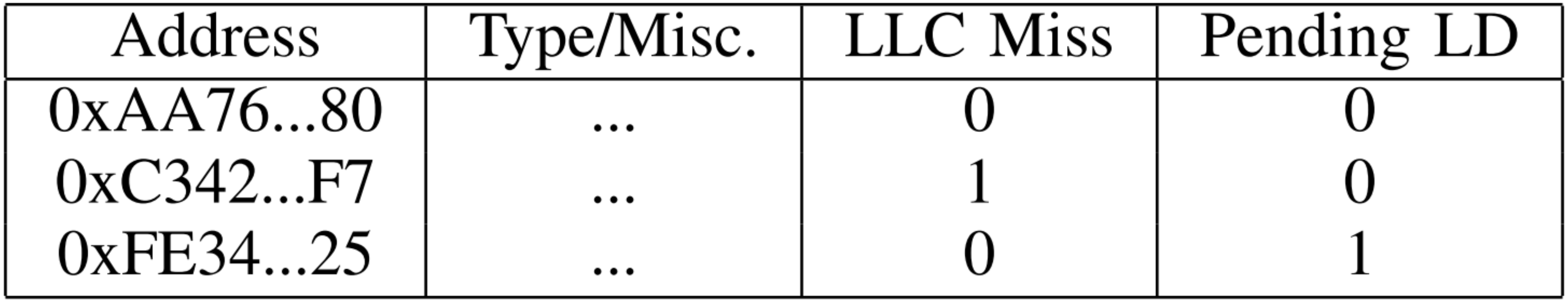
- The first work to use non-volatility feature of STT-RAM to implement pipeline-level checkpointing
- A general and low overhead framework for reducing energy through exploiting short idle execution phases
- Achieved energy reduction of 17-34% with less than 3% performance and area overheads

Questions?

Thank you!

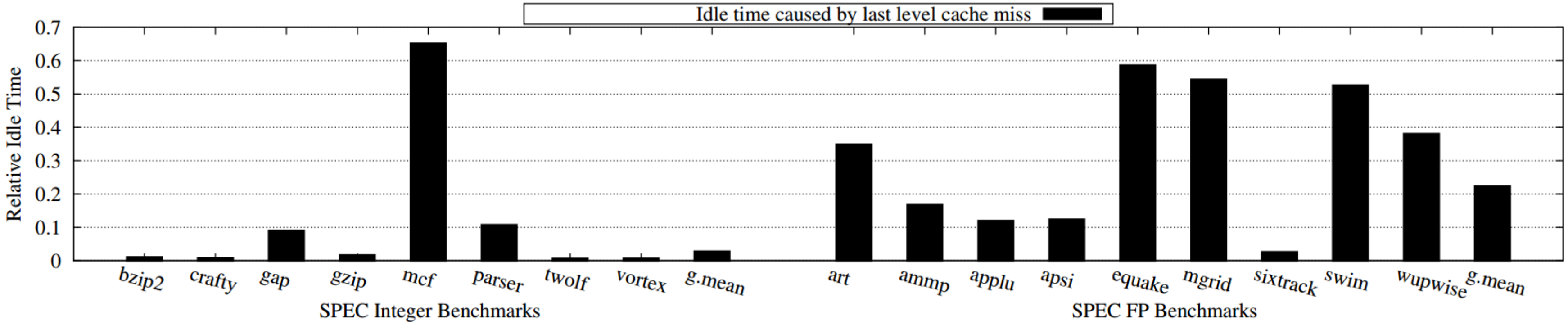
Backup Slides

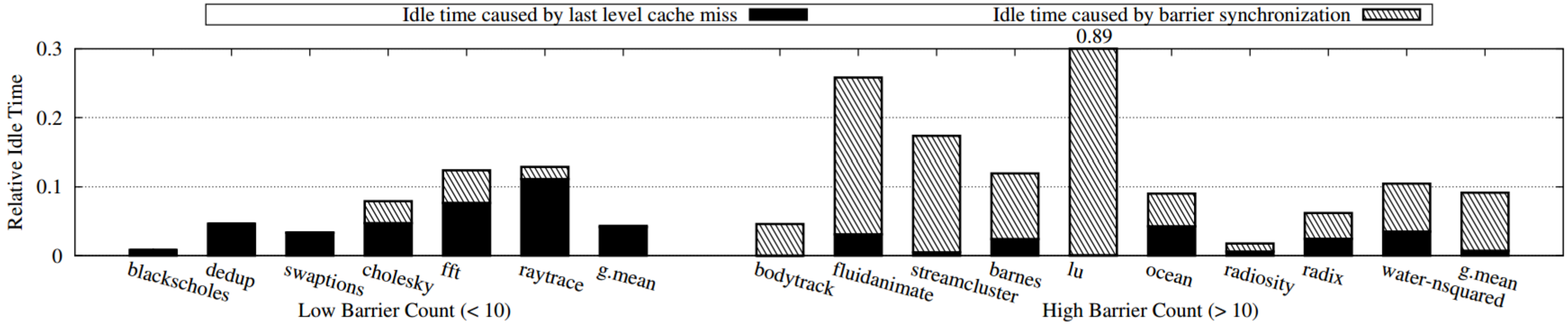
NVSleep MSHR Design



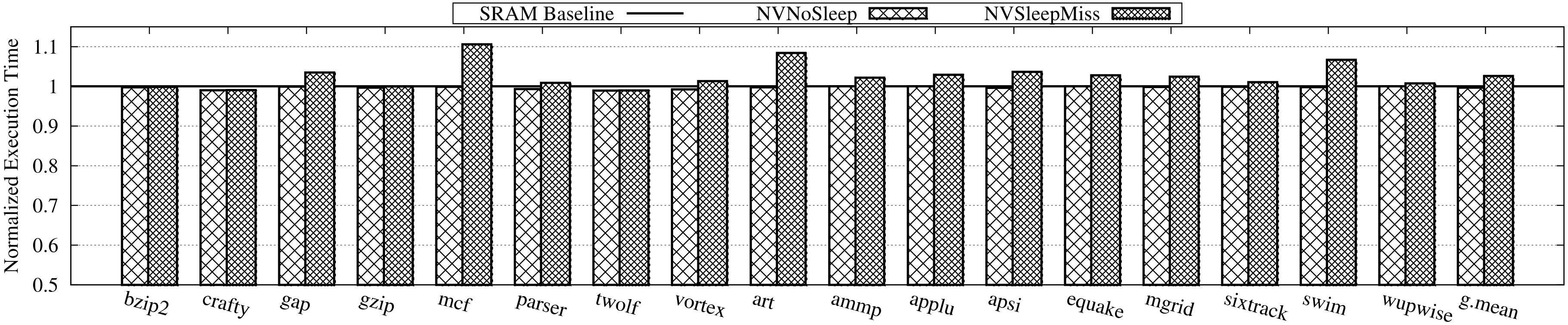
Address	Type/Misc.	LLC Miss	Pending LD
0xAA76...80	...	0	0
0xC342...F7	...	1	0
0xFE34...25	...	0	1

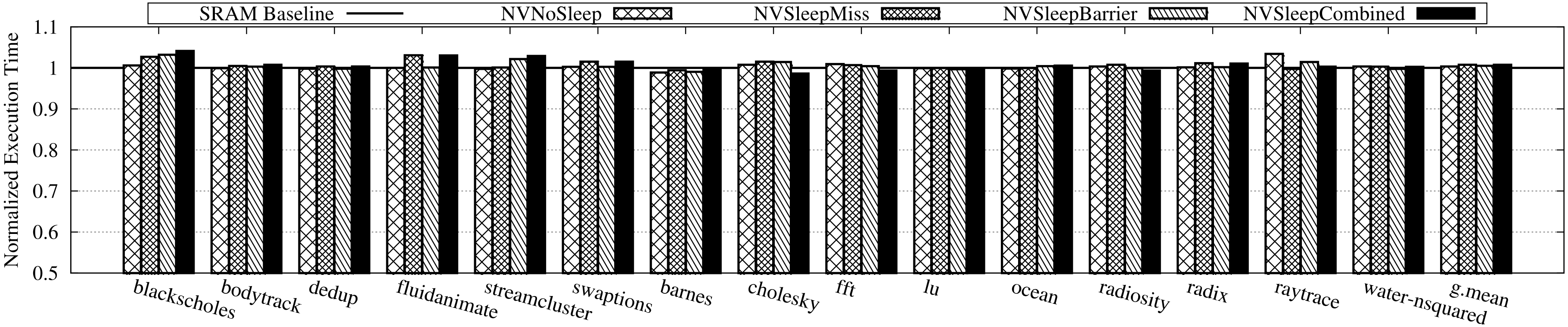
Core Idle Time Analysis





Performance Overhead





Area Overhead

- Area – less than 3% compared to SRAM baseline
  - STT-RAM shadow structures increase area
  - STT-RAM caches help decrease area