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

Xiang Pan
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH, USA
panxi@cse.ohio-state.edu

Radu Teodorescu
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH, USA
teodores@cse.ohio-state.edu

***Abstract*—Spin-transfer torque random access memory (STT-RAM) is an emerging memory technology with several attractive properties including non-volatility, high density, low leakage, and high endurance. These characteristics make it a potential candidate for replacing SRAM structures on processor chips. This paper presents NVSleep, a low-power microprocessor framework that leverages STT-RAM to implement fast checkpointing that enables near-instantaneous shutdown of cores without loss of the execution state. NVSleep stores almost all processor state in STT-RAM structures that do not lose content when power-gated. Memory structures that require low-latency access are implemented in SRAM and backed-up by "shadow" STT-RAM structures that are used to implement fast checkpointing. This enables rapid shutdown of cores and low-overhead resumption of execution, which allows cores to be turned off frequently and for short periods of time to take advantage of idle execution phases to save power. We present two implementations of NVSleep: NVSleepMiss which turns cores off when last level cache misses cause pipeline stalls and NVSleepBarrier which turns cores off when blocked on barriers. Evaluation of a simulated 64-core system shows average energy savings of 21% for NVSleepMiss in SPEC2000 benchmarks and 34% for NVSleepBarrier in high barrier count multi-threaded workloads from PARSEC and SPLASH2 benchmarks.**

I. Introduction

Power consumption is a first-class constraint in microprocessor design. The increasing core count in chip multiprocessors is rapidly driving chips towards a new "power wall" that will limit the number of compute units that can be simultaneously active. Traditional power management techniques such as dynamic voltage and frequency scaling (DVFS) are becoming less effective as technology scales. Supply voltage scaling has slowed significantly limiting the range and effectiveness of DVFS. Techniques like clock gating are not affected by voltage scaling but they cannot control leakage power which is expected to increase in future technologies [5]. Power gating, a technique that cuts power supply to functional units can be an effective mechanism for saving both leakage and dynamic power. Unfortunately power gating leads to the loss of data stored in the gated units. Stateless units such as ALUs can be restarted with little overhead. However, units with significant storage such as register files, caches, and other buffers and queues hold large amounts of data and state information. Restoring that information following a shutdown incurs a significant performance overhead.

This work was supported in part by the Defense Advanced Research Projects Agency under the PERFECT (DARPA-BAA-12-24) program and the National Science Foundation under grants CCF-1117799 and CCF-1253933.

Non-volatile memories such as Phase Change Memory (PCM), Spin-Transfer Torque RAM (STT-RAM), NAND Flash, etc. are emerging as promising alternatives to DRAM and SRAM. These new memory technologies have many promising characteristics, such as very low leakage, high-density, and scalability that is expected to exceed that of SRAM and DRAM. STT-RAM, a new generation of Magnetoresistive RAM is a particularly attractive class of non-volatile memory because it has infinite write endurance, good compatibility with CMOS technology, fast read speed, and low read energy [1], [3], [13], [14], [8]. A significant drawback of STT-RAM is higher write latency and energy compared to SRAM.

This paper proposes NVSleep, a low-power microprocessor framework that leverages STT-RAM to implement rapid shutdown of cores without loss of execution state. This allows cores to be turned off frequently and for short periods of time to take advantage of idle execution phases to save power. We present two implementations of NVSleep: NVSleepMiss which will turn cores off when last level cache (LLC) misses cause pipeline stalls and NVSleepBarrier which will turn cores off when blocked on barriers.

In both NVSleep implementations all memory-based functional units that are not write-latency sensitive (such as caches, TLBs, and branch predictor tables) are implemented using STT-RAM. These structures do not lose content when power-gated. Other on-chip structures that require low-latency writes are implemented using SRAM and backed-up by shadow STT-RAM structures. A fast checkpointing mechanism stores modified SRAM content into STT-RAM. After the content is saved, the entire structure can be power-gated. When power is restored, the content of the SRAM master is retrieved from the non-volatile shadow. This allows rapid shutdown of cores and low-overhead resumption of execution when cores are powered back up.

Evaluation using SPEC CPU2000, PARSEC, and SPLASH2 benchmarks running on a simulated 64-core system shows average energy savings of 21% for NVSleepMiss in SPEC2000 benchmarks and 34% for NVSleepBarrier in high barrier count multi-threaded workloads from PARSEC and SPLASH2 benchmarks. The energy savings are achieved with a very small performance overhead.

Overall, this paper makes the following contributions:

- To the best of our knowledge, NVSleep is the first work to take advantage of the non-volatility feature of STT-RAM to implement rapid pipeline-level checkpointing.
- NVSleep proposes a general and low overhead framework for reducing energy consumption by exploiting short idle execution phases.
- NVSleep is also the first checkpointing framework that is sufficiently fast to allow cores to be shutdown during LLC misses.

The rest of this paper is organized as follows: Sections II and III describe the design and implementations of the NVSleep framework. An experimental evaluation is presented in Sections IV and V. Finally, Section VI discusses related work, and Section VII concludes.

II. NVSleep Framework Design

NVSleep improves microprocessor energy efficiency by rapidly turning off cores during idle periods and quickly restoring them to full activity when work becomes available. NVSleep is designed to both checkpoint state very quickly and restore execution almost instantly after the core is turned on, without requiring the flushing and refilling of the pipeline. This allows NVSleep to take advantage of short idle execution phases such as those caused by misses in the last level cache.

The NVSleep framework uses two designs for on-chip memory structures. Storage units that are less sensitive to write latency – such as caches and branch predictor tables – are implemented with non-volatile STT-RAM equivalents. When a core is powered off, these units will not lose state. In order to improve the write performance of STT-RAM structures, especially in the presence of bursty activity, we add small SRAM write buffers. A similar optimization was introduced by prior work [3], [12].

Memory structures that are more sensitive to write latency and are frequently updated in the critical path of the execution (such as Register File, Reorder Buffer, etc.) are implemented using a hybrid SRAM/STT-RAM design. Figure 1 illustrates this design. The primary storage elements are implemented using SRAM. The SRAM "Master" banks are backed-up using STT-RAM "Shadow" arrays of equal size. When a checkpoint is initiated, the SRAM entries that have been updated since the last checkpoint are transferred to the STT-RAM Shadow. To speed up the checkpointing process, all hybrid memory structures are banked, allowing all banks to be checkpointed in parallel. Since banking introduces additional area/power overheads, we experiment with various banking options to determine the optimal configuration. To reduce overhead, the shadow and master banks share row decoders. This is possible because during checkpointing and restore the same rows are being accessed in both the master and the shadow. The checkpoint is coordinated by control logic associated with each hybrid structure. The control logic generates addresses for the checkpoint and restore sequence and checks and updates "modified" bits used to identify blocks that have been updated since the last checkpoint.

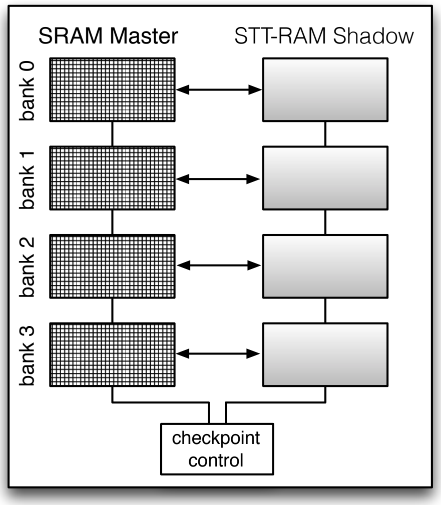

Fig. 1: SRAM memory structure with STT-RAM backup.

NVSleep Core Units	Technology
All Caches	STT-RAM
I-TLB and D-TLB	STT-RAM
Branch Prediction Table	STT-RAM
Register File	SRAM + STT-RAM Shadow
Instruction Window	SRAM + STT-RAM Shadow
Reorder Buffer	SRAM + STT-RAM Shadow
Load/Store Queue	SRAM + STT-RAM Shadow
Pipeline Registers	SRAM + STT-RAM Shadow
All Logic	CMOS

TABLE I: Technology choices for NVSleep structures.

All pipeline registers outside in the processor are implemented using CMOS flip-flops and backed-up with STT-RAM shadows. Their content is checkpointed in parallel in a single write cycle making banking unnecessary. For completeness, Table I enumerates the technologies used in the principal components of the NVSleep framework.

A. Checkpointing Control

NVSleep controls checkpointing, power-down, and wakeup of cores in a distributed fashion across the chip. It relies on the L1 cache controller of each core to help coordinate both the sleep and wakeup process. NVSleep uses two mechanisms for triggering the sleep sequence: one that is entirely hardware-initiated and managed, and one that uses a software API. The hardware-driven mechanism is appropriate for exploiting idle phases caused by events that can be easily identified by the hardware – such as misses in the last level cache.

The software API can be used by the system to request the shutting down of cores. The API relies on a dedicated `sleep()` instruction associated with a reserved memory address `0xADDR` that is tracked by the cache controller. The instruction can be used by the compiler or programmer when an idle execution phase is expected. For instance, a core can be shut down while it is blocked on a barrier, during long latency I/O transfers, or while it is waiting for a lock to be released.

B. Wakeup Mechanism

Wakeup mechanisms are based on the L1 cache controllers of each core to coordinate the wakeup processes. The cache controllers are not power gated and are therefore available to initiate and coordinate wakeup events. These events include returning misses or wakeup messages from other cores.

For the hardware-initiated sleep, if the sleep event has been triggered by an LLC miss, the cache controller wakes up the core once the missing data makes its way to the L1. The wakeup of a core that was explicitly shut down with the `sleep()` instruction has to be initiated by another core or service processor. The wakeup of a core is triggered by writing to the sleep `0xADDR` address associated with that core. An update or invalidate message for that address will direct the cache controller of the sleeping core to start waking up.

The wakeup overhead depends on how quickly the core can be brought back online and have its state restored. Bringing a system online after sleep can cause ringing in the power supply, which can lead to voltage droops. To prevent large droops we gradually ramp-up core wakeup. In the first phase of wakeup no computation is performed to allow the supply lines to settle. In the second phase, checkpointed data is restored from shadow STT-RAM structures. Finally, normal execution is resumed. More importantly, we do not allow multiple cores to wakeup simultaneously, limiting the current ramp-up to 1/N of the chip maximum, where N is the number of cores.

III. NVSleep Framework Implementation

We developed two applications of the NVSleep framework. The first, which we call *NVSleepMiss* is hardware-controlled and shuts down cores that block on misses in the last level cache. The second, which we call *NVSleepBarrier*, is software-controlled and turns off cores that are blocked on barrier synchronization.

A. NVSleepMiss

Last level cache misses can lead to hundreds of cycles of stalled execution due to long latency memory accesses. Even though out-of-order processors can hide some of that latency, pipelines will eventually stall when independent instructions are no longer available. Even though stalled cores don't consume as much power as active cores, their idle power is still significant. For instance, modern Intel processors idle at anywhere between 10W and 50W [4]. NVSleepMiss addresses this inefficiency.

NVSleepMiss requires identification and tracking of misses in the last level cache. For this purpose we augment the miss handling status register (MSHR) of the L1 cache controller. We add two new fields to the standard MSHR design, as shown in Table II. The first field, labeled "LLC Miss", is a one-bit tag used to indicate whether this L1 cache miss ends up missing in the last level cache as well. This tag is used to decide when a core should be asked to sleep. The second field is "Pending LD" which is used to keep track of other L1 loads that might still be pending when the core shuts down.

Figure 2 illustrates the steps involved in the NVSleepMiss shutdown and wakeup. In this example a load request misses in the L1 cache in step ①. The controller of the last level cache informs the L1 cache controller of an LLC miss event in step ②. The L1 cache controller will find the related entry in the MSHR table and update its "LLC Miss" field to 1. If there are no other pending LLC misses, a shutdown signal will be sent to the core (step ③).

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

TABLE II: NVSleep MSHR with additional fields.

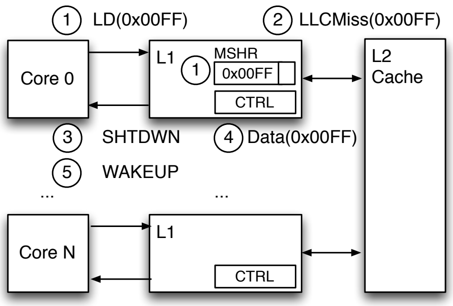

Fig. 2: NVSleepMiss hardware-initiated sleep and wakeup.

A core initiate the shutdown sequence upon receipt of a sleep signal from the cache controller. To ensure the core wakes up in a consistent state, all instructions that follow the LLC miss in program order are allowed to complete and are retired from the reorder buffer. Any instructions that are still "in-execution" (meaning they occupy the execution cluster) are allowed to complete and no new instructions are dispatched. The checkpointing sequence begins as soon as the core receives the sleep signal and takes place in parallel with the draining of the execution pipeline. Only SRAM/STT-RAM hybrid structures require explicit checkpointing. The checkpointing control unit copies all modified entries in the SRAM section to STT-RAM. Once the pipeline is drained, a second checkpointing phase is initiated to save any modified entries still remaining (such as those modified while draining the execution cluster).

While draining the execution pipeline after receiving the sleep signal the core could still send load requests to the cache. The core does not need to wait for these loads to be serviced in case they miss in the L1. The cache controller will keep track of what requests have completed while the core is sleeping using the "Pending LD" field of the MSHR. All load requests in the MSHR that are marked as "Pending LD"'s will be re-issued to the cache by the cache controller after the core wakes up. This ensures the latest copy of the data is supplied to the core after wakeup. Only loads need to be tracked because stores can complete while the processor is sleeping and do not require any data to be sent to the core.

If data arrives at the L1 from a lower-level cache while the core is sleeping, that data will be written into the cache. If the data corresponds to a load request, the associated MSHR "Pending LD" field will be set to 1. If this is the last pending LLC miss, a wakeup signal will be sent to wake up the sleeping core. This is illustrated by steps ④ and ⑤ in Figure 2. Otherwise, the "LLC Miss" tag will be set to 0 but no wakeup

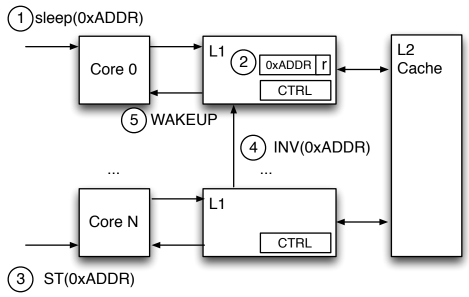

Fig. 3: NVSleepBarrier software-initiated sleep and wakeup.

signal will be generated. This is because other LLC misses are still pending so there is no reason to wake up the core just yet. After the core is woken up, all pending loads in the MSHR table will be re-issued to the L1 cache. To avoid a deadlock, these re-issued requests will not ask the processor to sleep, even if they result in an LLC miss.

In the NVSleep framework the cache controller of a sleeping core is never shut down. This allows the cache controller to initiate the wakeup process. It also ensures that all coherence requests continue to be served even when the core is sleeping. As a result, no changes to the coherence protocol are needed.

B. NVSleepBarrier

Parallel applications often use software barriers to coordinate execution of multiple threads. These threads are sometimes unbalanced for algorithmic or runtime-related reasons. As a result many cores may end up spending a significant portion of time idle while waiting for slow threads to reach barriers. NVSleepBarrier addresses this inefficiency by shutting down cores blocked on barrier synchronization.

The NVSleepBarrier implementation uses the NVSleep API through the `sleep(0xADDR)` instruction. The instruction is treated like a special load instruction that reads from the `0xADDR` address. Figure 3 illustrates this process. When the `sleep(0xADDR)` instruction is executed, a load from address `0xADDR` is sent to the cache (1). The cache allocates a line for the data at `0xADDR` and marks it as reserved by setting a dedicated bit in the tag (2). This reserved address will be used to trigger the wakeup process.

The `sleep()` instruction also acts as a memory fence instruction, not allowing any memory access reordering with respect to the load to address `0xADDR`. This will ensure that when the `sleep()` instruction retires, there will be no pending loads or stores in the cache that follow the `sleep()` instruction in program order. This will allow the core to go to sleep as soon as the `sleep()` instruction retires and no loads will have to be re-issued when the core wakes up.

With this instruction in place, NVSleepBarrier requires minimal changes to the standard software barrier implementation. Figure 4 shows code for the implementation of NVSleepBarrier in a generic sense-reversing barrier. The `sleep()` instruction is executed by threads that block on the barrier. The global "sense" variable address is passed as a parameter to the `sleep()` instruction. As a result, the "sense" variable becomes the *wakeup trigger* for all the sleeping cores. Since threads that block on the barrier will only read the "sense"

```
void barrier(int count, int sense, int
    num_threads)
{
 int local_sense;
 // Each core toggles its own sense.
 local_sense = !sense;
 if(count != num_threads-1) {
  // Not the last thread, block.
  while(local_sense != sense) {
    // NVSleep instruction, with sense as
    // wakeup trigger.
    sleep(&sense);
  }
 } else {
  // Last thread in the barrier.
  count = 0;
  // By writing to the sense variable
  // blocked cores are woken up and
  // subsequently released from barrier.
  sense = local_sense;
 }
}
```

Fig. 4: NVSleepBarrier implementation.

variable, they will not trigger a wakeup. The last thread to reach the barrier will follow the *else* path through the code and will write to the "sense" variable, inverting its direction. The write will also trigger an "invalidate" message to the caches of all blocked cores. This is illustrated in Figure 3 by steps (3) and (4). These trigger messages will wake up the sleeping cores, allowing them to resume execution (5).

IV. Evaluation Methodology

We modeled a 64-core CMP in 32nm technology. Each core is a dual-issue out-of-order architecture. We used SESC [9] to simulate the baseline SRAM-based CMP as well as the NVSleep framework. Table III summarizes the architectural parameters used in our simulations. We used CACTI [7] to extract energy per access for all SRAM memory structures including register file, reorder buffer, instruction window, etc. CACTI was also used to model the energy and area overhead for the banked SRAM memory structures (hybrid register file and reorder buffer, etc.).

For modeling STT-RAM structures we used data from [3] for the access latency and energy, and NVSim [2] to estimate chip area overhead. We also modeled leakage power based on estimated unit area and technology (CMOS vs. STT). We plugged these energy numbers into the activity model of the SESC simulator to obtain power consumption and energy.

We ran benchmarks from the SPEC CPU2000, SPLASH2, and PARSEC suites. The benchmark sets include single-threaded and multi-threaded benchmarks. Some parallel benchmarks have heavy barrier activity while others use barriers sporadically or not at all.

V. Evaluation

We evaluate the energy and performance implications of the two implementations of NVSleep: NVSleepMiss and NVSleepBarrier. We also evaluate the time and energy cost of checkpointing and show some sensitivity analysis results. We compare NVSleep with a baseline system (*SRAM Baseline*) in which all memory structures are built with SRAM. The baseline system employs clock-gating of idle functional units to reduce power.

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
SRAM Read/Write Time	1 cycle
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

TABLE III: Summary of the experimental parameters.

A. *Application Idle Time Analysis*

For the purpose of this analysis we define idle time as cycles in which no instruction is retired and no instruction is in execution in a functional unit. During those cycles the processor is virtually stalled. Figure 5 shows the percentage of idle time in the total execution time for single-threaded benchmarks. The large number of misses in memory-bound applications like *mcf*, *equake*, *mgrid*, and *swim* leads to idle cycle counts that exceed 50%. Compute-bound applications such as *bzip2* on the other hand have very little idle time. We expect NVSleepMiss to benefit memory-bound applications, with little or no benefit to compute-intensive applications that experience relatively few misses.

For multi-threaded applications, in addition to looking at pipeline stalls due to LLC misses, we also examine idle time spent by cores blocked on barriers. The idle time relative to the total execution time broken down into pipeline stalls and barrier blocked time is shown in Figure 6. We find that for the multi-threaded applications we examine, stalls due to LLC misses account for very small fraction of execution time (less than 2% on average). The amount of idle time spent in barriers depends on two factors: the number of barriers and the level of imbalance between work done by individual threads. Applications with large numbers of barriers such as *streamcluster* or with significant imbalance such as *lu* and *fluidanimate* spend up to 89% of their execution time idling inside barriers. *ocean*, on the other hand, is stalled only about 4.7% of its time, even though it runs through about 900 dynamic barrier instances. This is because *ocean* is very balanced, with threads reaching barriers almost simultaneously and leaving them rapidly. We expect unbalanced applications to benefit most from NVSleepBarrier. Naturally, benchmarks that use no barriers such as *dedup* will see no benefit from NVSleepBarrier.

B. *NVSleep Energy Savings*

Figure 7 shows the energy consumption for NVSleepMiss relative to the SRAM baseline for single-threaded benchmarks. Applications with high number of misses and frequent stalls benefit greatly from NVSleepMiss. For example, *mcf* with the longest idle time achieves 54% energy reduction using NVSleepMiss compared to the baseline. Similar energy reduction is achieved by *equake*, *mgrid*, and *swim*. On average, the energy savings of NVSleepMiss are 17.2% for SPEC Int benchmarks and 23.3% for SPEC FP benchmarks.

We compare NVSleepMiss with two reference designs. One is the NVSleep framework with the sleep option disabled – we call this *NVNoSleep*. In NVNoSleep all energy savings come from leakage reduction from STT-RAM structures. On average, energy is 8-15% higher for NVNoSleep compared to NVSleepMiss. We also compare to an ideal version of NVSleep that has no checkpointing overhead (*NVSleepIdeal*). NVSleepIdeal is 3.3% and 9.1% more energy efficient than NVSleepMiss for SPEC Int and SPEC FP respectively.

For the multi-threaded benchmarks we examine the energy benefits of NVSleepMiss, NVSleepBarrier, and the combined application of the two techniques (*NVSleepCombined*). Figure 8 shows the energy savings achieved by the three NVSleep techniques compared to the baseline. Applications with fewer than 10 barriers get virtually no benefit from NVSleepBarrier. For applications with more than 10 barriers the energy savings depend on the level of workload imbalance and the number of barriers. For *lu*, which is extremely unbalanced, the energy reduction exceeds 80%. *streamcluster* has over 4000 barriers but is fairly balanced. Its energy reduction is 22.1%. On average, applications with more than 10 barriers achieve a very significant energy reduction of 33.8% with NVSleepBarrier. This represents a 22.4% improvement over NVNoSleep.

NVSleepMiss does not help much in the case of multi-threaded benchmarks since miss-related idle time is small. As a result NVSleepCombined is only marginally more energy efficient than NVSleepBarrier.

C. *NVSleep Overheads*

1) Performance: Figures 9 and 10 show the runtimes of NVSleep implementations for single-threaded and multi-threaded benchmarks respectively. There are mainly two sources of performance overhead in NVSleep. The first one is caused by the pipeline drain required to shut down cores in consistent states. This drain means that cores resume execution after a shutdown with fewer instructions in their instruction windows than they would otherwise have available. This might slow down execution in some cases after the core is woken up.

The other source of performance overhead is related to the way NVSleep handles multiple LLC misses that occur in close temporal proximity. In NVSleep, a core is only woken up after all LLC misses have returned. As a result, the core will miss the opportunity to work on some instructions that are dependent on data that has become available since the core has been shut down. This explains why we observe more than 5% performance overhead in benchmarks like *mcf*, *art*, and *swim*, in which multiple misses per sleep event are common. On average, the performance overhead of NVSleepMiss is less than 3% and that of NVSleepBarrier and NVSleepCombined is less than 1%.

2) Area: The NVSleep framework has some area overhead due to the banked SRAM blocks and STT-RAM shadow

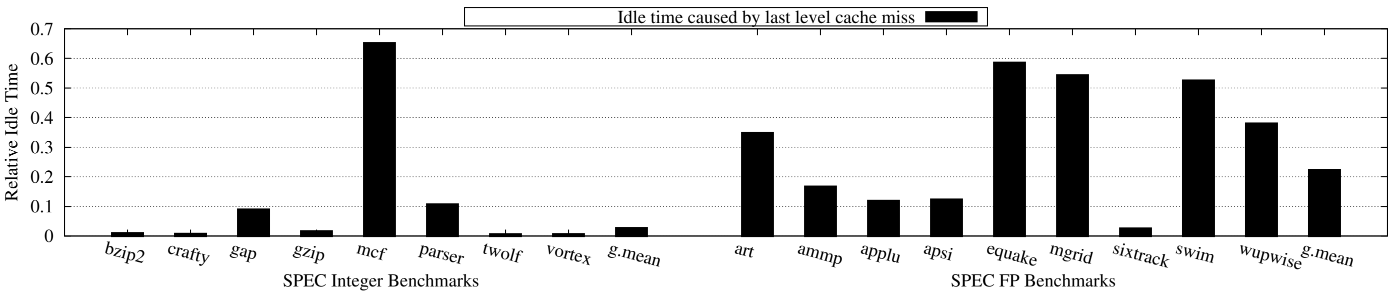

Fig. 5: Idle fraction of the execution time for single-threaded benchmarks.

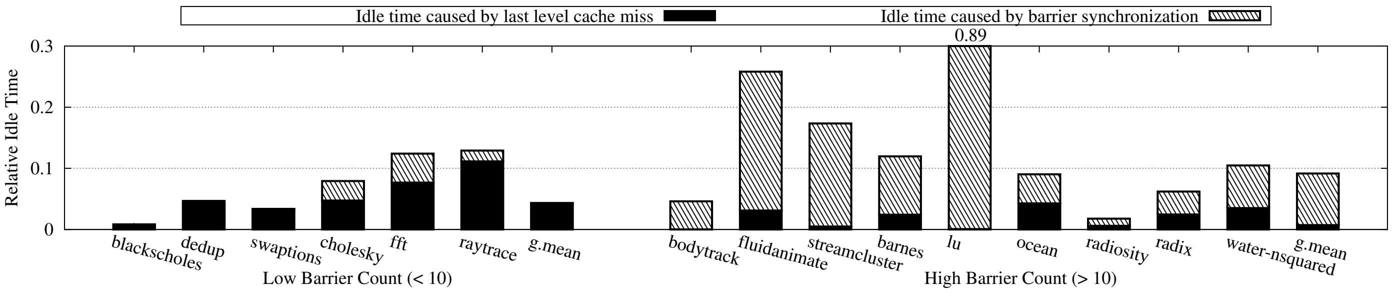

Fig. 6: Idle fraction of the execution time for multi-threaded benchmarks.

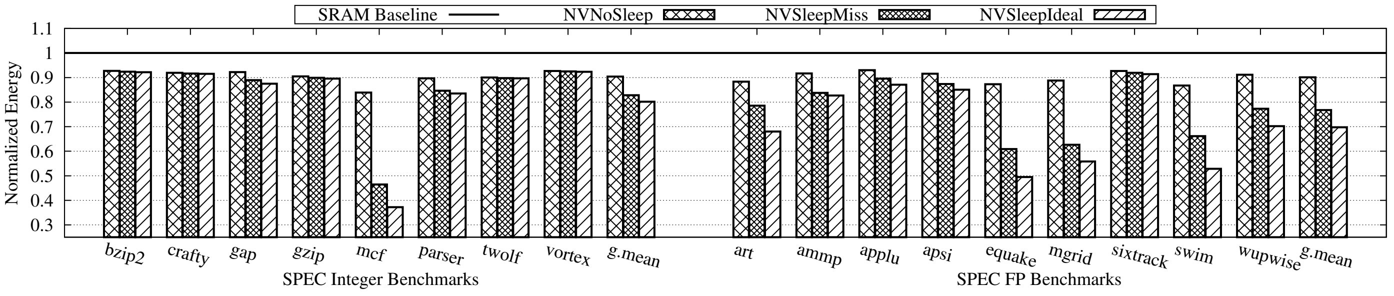

Fig. 7: Energy consumption of NVSleepMiss for single-threaded benchmarks relative to SRAM baseline.

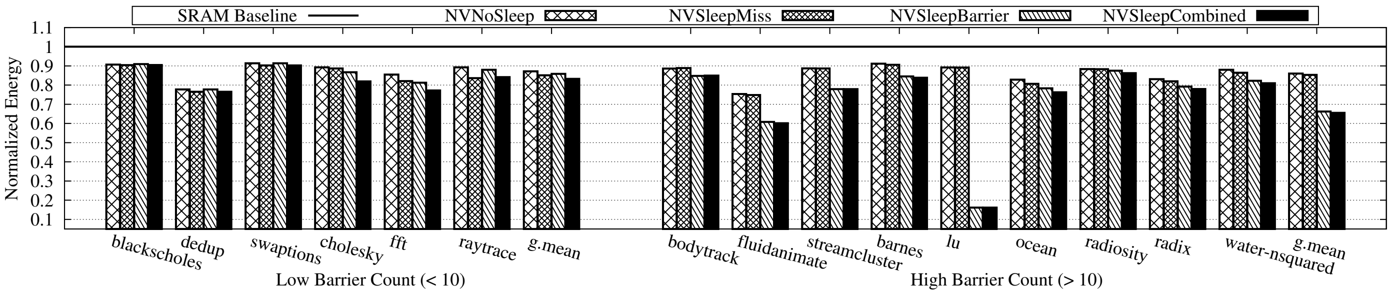

Fig. 8: NVSleepMiss, NVSleepBarrier, and NVSleepCombined energy for multi-threaded benchmarks.

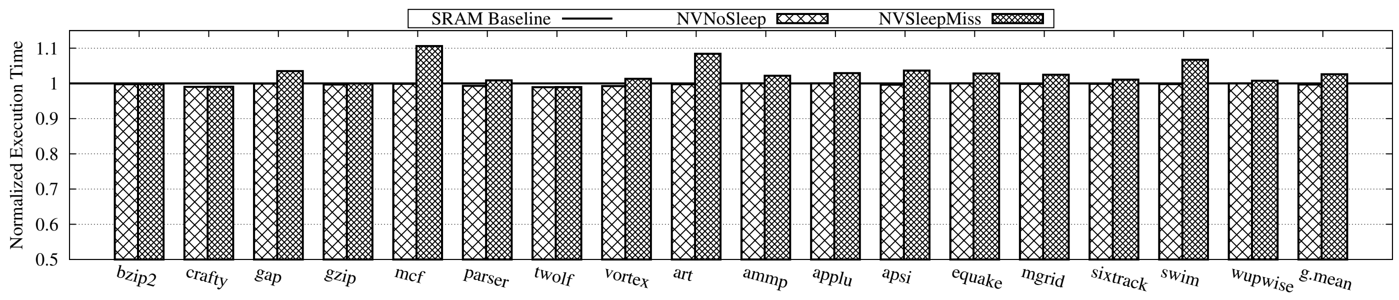

Fig. 9: Runtime of NVSleepMiss design for single-threaded benchmarks.

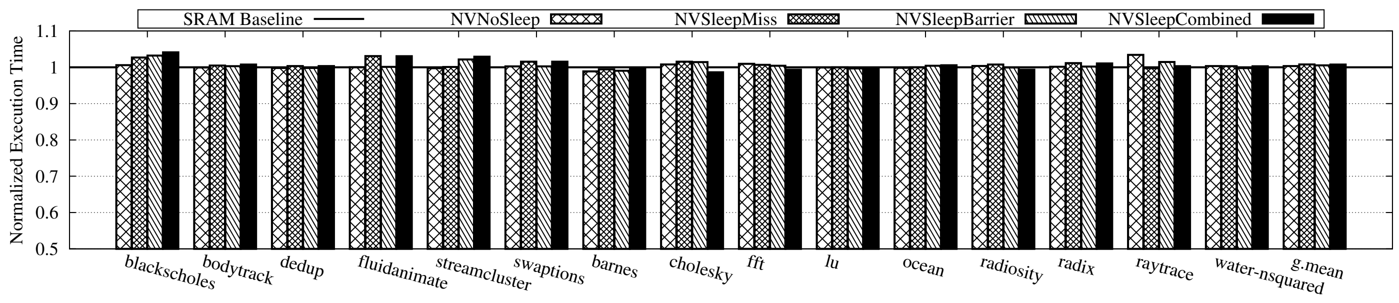

Fig. 10: Runtime of different NVSleep designs for multi-threaded benchmarks.

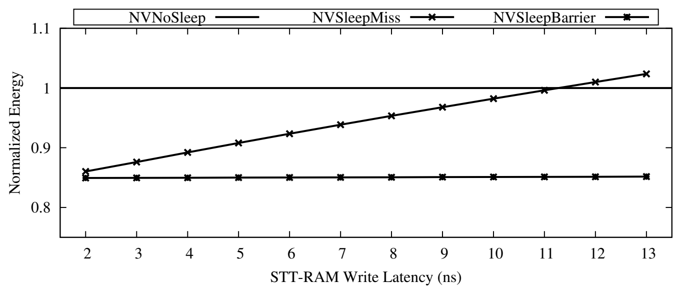

Fig. 11: NVSleepMiss and NVSleepBarrier energy for different STT-RAM write latencies, relative to NVNoSleep.

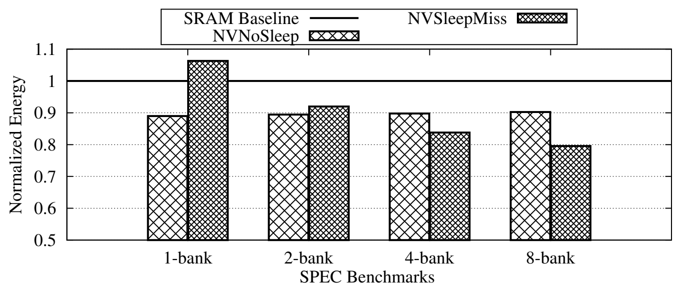

Fig. 12: NVSleepMiss energy for different numbers of banks.

Num of Banks	Energy/Access (nJ)	Area (mm^2)
1	0.000448	0.007543
2	0.000552	0.012091
4	0.000628	0.018883
8	0.000741	0.029032

TABLE IV: Energy and area for banked 1KB 32-bit SRAM.

structures. According to CACTI and NVSim simulations with our area model, the 8-bank NVSleep framework design will increase the processor core area by 60%. However, since STT-RAM is much denser than SRAM, the cache area is only 16% of the baseline design. Overall, the total chip area overhead of NVSleep adds up to less than 3% of the SRAM baseline.

D. Sensitivity Studies

The main overhead of STT-RAM is high write latency. To better understand its impact on overall energy savings, we experimented with various STT-RAM write pulses ranging from aggressive 2ns to conservative 13ns. Figure 11 shows that average energy savings for NVSleepMiss with single-threaded benchmarks gradually become smaller as the STT-RAM write latency increases. When the write latency reaches 11ns NVSleepMiss saves almost no energy. On the other hand, since checkpointing is rare in NVSleepBarrier, the increasing STT-RAM write latency has almost no impact on the overall energy savings of NVSleepBarrier. All our previous experiments have assumed an STT-RAM write latency of 3.3ns, also used in prior work [3].

The number of banks in the hybrid SRAM/STT-RAM structures has an impact on the overall energy savings because it directly affects the performance and energy overhead of checkpointing. Figure 12 shows the energy consumption of NVSleepMiss with the hybrid memory structures configured with 1, 2, 4, and 8 banks. We show average energy across the SPEC benchmarks. Using higher number of banks reduces performance overhead parallelism because all banks can be checkpointed in parallel. However, increasing the number of banks also has an energy and area cost. The energy and area sensitivity with the number of banks is shown in Table IV. The optimal configuration for our system is the 8-bank design.

To examine the scalability of NVSleepBarrier we run the same experiments on simulated 16 and 32-core systems in addition to the 64-core system. In general, we observe that

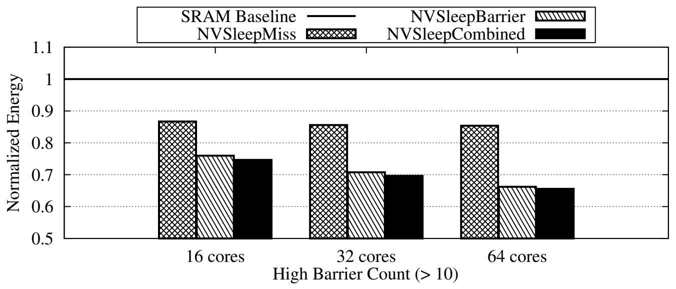

Fig. 13: NVSleep energy savings in CMPs with 16, 32, and 64 cores.

NVSleepBarrier saves more energy in higher core count systems. This is because barrier idle time and workload imbalance tends to increase with the number of cores. NVSleepBarrier lowers energy by 24% on the 16-core system, by 29.3% on the 32-core system, and by 33.8% on the 64-core system, as shown in Figure 13. Energy savings from NVSleepMiss are only marginally higher with increasing number of cores.

VI. Related Work

As STT-RAM has gained more and more attention recently, many researchers have focused on solving the long-latency and high-energy write issues associated with the technology in order to make it a feasible SRAM replacement. For instance, Zhou et al. [14] proposed Early Write Termination to terminate redundant bit writes at their early stages to reduce write energy. Other work [11], [13] has explored factors which could affect data retention time of STT-RAM cells and found that there is a trade-off between non-volatility and write performance and energy of those cells. By relaxing the non-volatility requirement they observe that they can improve energy by using shorter write times.

Guo et al. [3] explored replacing large, wire-delay dominated SRAM arrays, such as caches, TLBs, and register files with STT-RAM. Some of the latency-critical units as well as pipeline registers are implemented using SRAM. They have built an in-order 8-core processor with this hybrid design. Their goal is to save energy by replacing high-leakage CMOS with low-leakage STT-RAM. Their design is similar to our NVNoSleep system except that we use out-of-order cores and hybrid SRAM/STT-RAM structures for memory units that are updated frequently. We show that significant energy reductions can be achieved by aggressively turning cores off when idle, in addition to simply replacing SRAM with STT-RAM.

Previous work [10] has proposed building the last level cache or main memory with non-volatile memory like STT-RAM or PCRAM (Phase Change RAM) to provide checkpointing capabilities for reliability and power reduction. Their solution is targeted at coarse-grained server/system level checkpoints that can tolerate much higher checkpointing/restore overheads. Our checkpointing has much lower performance overhead.

Kvatinsky et al. [6] proposed a memristor-based multistate pipeline register design to provide low penalty switch-on-event multithreading capabilities. To the best of our knowledge this is the first paper that exploits the non-volatility properties of STT-RAM to enable microarchitecture-level rapid checkpoint/restoration of cores with the goal of saving energy.

VII. Conclusion and Future Work

Non-volatile memory can be used effectively to implement rapid checkpoint/wakeup of idle cores. This paper has explored a framework for implementing rapid checkpoint using STT-RAM and two applications of that framework: NVSleepMiss and NVSleepBarrier. Evaluation showed average energy savings of 21% for NVSleepMiss in single-threaded applications and 34% for NVSleepBarrier in high barrier count multi-threaded workloads, both with very small performance overhead. In future work we will explore other opportunities for shutting down cores when idle such as spinning due to lock contention or other synchronization events.

References

[1] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement," in *Design Automation Conference (DAC)*, June 2008, pp. 554–559.

[2] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, July 2012.

[3] X. Guo, E. Ipek, and T. Soyata, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *International Symposium on Computer Architecture (ISCA)*, June 2010, pp. 371–382.

[4] "Intel Core i7-800 Processor Series and the Intel Core i5-700 Processor Series Based on Intel Microarchitecture (Nehalem)," Intel White Paper, 2009, http://download.intel.com/products/processor/corei7/319724.pdf.

[5] "International Technology Roadmap for Semiconductors (2009)," http://www.itrs.net.

[6] S. Kvatinsky, Y. H. Etsion, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-Based Multithreading," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 41–44, March 2013.

[7] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," HP Labs, Tech. Rep. HPL-2009-85, 2009.

[8] X. Pan and R. Teodorescu, "Using STT-RAM to Enable Energy-Efficient Near-Threshold Chip Multiprocessors," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, August 2014, pp. 485–486.

[9] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos, "SESC Simulator," January 2005, http://sesc.sourceforge.net.

[10] S. Sardashti and D. A. Wood, "UniFI: Leveraging Non-Volatile Memories for a Unified Fault Tolerence and Idle Power Management Technique," in *International Conference on Supercomputing (ICS)*, June 2012, pp. 59–68.

[11] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2011, pp. 50–61.

[12] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2009, pp. 239–249.

[13] Z. Sun, X. Bi, H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme," in *International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 329–338.

[14] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy Reduction for STT-RAM Using Early Write Termination," in *International Conference on Computer-aided Design (ICCAD)*, November 2009, pp. 264–268.