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

Xiang Pan, Anys Bacha, and Radu Teodorescu
Department of Computer Science and Engineering
The Ohio State University
{*panxi,bacha,teodores*}*@cse.ohio-state.edu*

***Abstract*—Near-threshold computing is emerging as a promising energy-efficient alternative for power-constrained environments. Unfortunately, aggressive reduction in supply voltage to the near-threshold range, albeit effective, faces a host of challenges. This includes higher relative leakage power and high error rates, particularly in dense SRAM structures such as on-chip caches.**

This paper presents an architecture that rethinks the cache hierarchy in near-threshold multiprocessors. Our design uses STT-RAM to implement all on-chip caches. STT-RAM has several advantages over SRAM at low voltages including low leakage, high density, and reliability. The design consolidates the private caches of near-threshold cores into shared L1 instruction/data caches organized in clusters. We find that our consolidated cache design can service more than 95% of incoming requests within a single cycle. We demonstrate that eliminating the coherence traffic associated with private caches results in a performance boost of 11%. In addition, we propose a hardware-based core management system that dynamically consolidates virtual cores into variable numbers of physical cores to increase resource efficiency. We demonstrate that this approach can save up to 33% in energy.

I. Introduction

Power consumption is now a primary constraint in microprocessor design spanning the entire spectrum of computing devices. Steady increase in the number of cores coupled with the growing inability to simultaneously activate most units of the chip prompted many to predict the end of traditional multicore scaling [1]. Such predictions emphasize the need to explore energy-efficient architectures that can continue to leverage advancements in process technology. Near-threshold (NT) computing [2], [3] has emerged as a potential solution for continuing to scale future processors to hundreds of cores. Near-threshold operation involves lowering the chip's supply voltage (V_{dd}) close to the transistor threshold voltage (V_{th}). Although this approach results in a 10× slowdown in chip speed, power consumption is lowered by 100×, potentially resulting in a full order of magnitude in energy savings. Unfortunately, near-threshold computing suffers from a number of drawbacks. These include decreased reliability, increased sensitivity to process variation, and higher relative leakage power [2], [3], [4].

Although near-threshold operations dramatically reduce power consumption, the contribution of dynamic and leakage components to the overall savings is not evenly distributed. While dynamic power reduction is cubic as a function of V_{dd} and frequency, leakage power only scales linearly. Caches are leakage dominated structures [5] that can account for 20% to 40% of the overall chip's power consumption depending on their size. Figure 1 shows the breakdown of leakage and dynamic power within a 64-core CMP at both nominal and NT V_{dd}. We observe that at a nominal V_{dd} of 1.0V, 14% of the total CMP power is attributed to cache leakage and another 14% to cache dynamic power. Overall, dynamic power represents 60% of the total CMP power consumption. However, when the same CMP operates in the near-threshold range, with a core V_{dd} of 400mV and a cache V_{dd} of 650mV, leakage power dominates, accounting for 75% of the total CMP power consumption. Close to half that leakage power is consumed by caches. While these numbers vary as a function of cache size, voltage and other factors, we find that reducing cache leakage will result in significant power savings at near-threshold voltages.

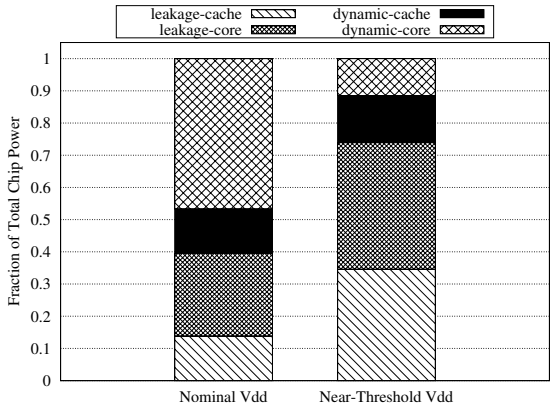

Figure 1: Dynamic and leakage power breakdown for a 64-core CMP at nominal and near-threshold voltages.

SRAM-based caches are generally the most vulnerable structure within the chip and are especially sensitive to low voltage. They are optimized for density and therefore rely on the smallest transistor design available for a given technology. While this approach enables larger cache capacities, it has the adverse effect of making such units

particularly vulnerable when operating at low voltages [6]. Process variation effects become increasingly pronounced as a function of V_{dd} reduction [7]. Consequently, this creates imbalances in the SRAM cells where a variety of failures can occur including timing and data retention errors. Such error rates are exacerbated in the near-threshold range, significantly compromising the ability of caches to reliably store data. Although a large body of error correction techniques have been proposed to deal with the high error rates in SRAM at low voltages [8], [4], the overhead associated with such approaches in the near-threshold range makes them inefficient.

This paper proposes a near-threshold chip multiprocessor design that uses Spin-Transfer Torque Random Access Memory (STT-RAM) to consolidate on-chip caches. We find STT-RAM to be an attractive candidate for implementing near-threshold systems for several reasons including low leakage, high density, and non-volatility [9], [10], [11], [12]. At one eighth the leakage of SRAM designs, STT-RAM based caches can operate at higher voltages and still save energy as a result of non-volatility. Raising the supply voltage has the advantage of alleviating the reliability concerns typically associated with low V_{dd}. Moreover, the inherently high write latencies of STT-RAM cells can be efficiently tolerated due to the low clock speeds at which near-threshold cores execute. This obviates the need for large SRAM buffers to mitigate performance bottlenecks caused by slow STT-RAM write speed [9], [13]. Moreover, unlike phase-change memory (PCM) and NAND flash memories [14], STT-RAM enjoys near-unlimited write endurance [10].

Based on these insights, we design a near-threshold chip multiprocessor (CMP) that utilizes dual voltage rails that can power processor cores and caches separately. We allocate a V_{dd} rail in the near-threshold range to the processing cores since they can operate at low frequencies. A second high V_{dd} supply rail is dedicated to the STT-RAM cache. This improves cache write latency relative to the cores. Furthermore, with this approach cache read latencies are substantially faster than the cycle time of the NT cores. This allows L1 caches to be shared by clusters of multiple cores, eliminating the need for cache coherence within the cluster. We show that this improves both latency and energy relative to traditional private cache designs. We redesign the shared cache controller to time-multiplex requests from different cores. The cluster size is chosen such that the vast majority of the read requests are serviced within a single core cycle to ensure no degradation in cache access latency.

The shared L1 cache enables another key feature of our CMP design. Since the L1 is shared by all cores within a cluster, migrating threads from one core to another has very low overhead compared to private cache designs because cached data is not lost in the migration. We take advantage of this feature to further reduce energy consumption with a dynamic core consolidation mechanism. The technique dynamically co-locates threads on the most energy efficient cores shutting down the less efficient ones depending on the characteristics of the workload. A runtime mechanism uses a greedy optimization that dynamically chooses the active core count which minimizes energy consumption. The motivation behind core consolidation is two-fold: (1) NTV cores have high leakage and powering some of them off can sometimes lead to net energy gains and (2) applications have low-IPC phases during which multiple threads can be consolidated on a single core with small impact on performance.

Evaluation using SPLASH2 and PARSEC benchmarks shows 11% performance improvement with the shared cache design and 33% combined energy savings with the dynamic core consolidation optimization enabled.

Overall, this paper makes the following contributions:

- Proposes STT-RAM as a great candidate for saving leakage and improving performance in near-threshold chips. To the best of our knowledge, this is the first work to use non-volatile caches in near-threshold chip multiprocessors.
- Introduces a novel process variation aware shared cache controller design that efficiently accommodates requests from cores running at different frequencies.
- Presents a low overhead dynamic core consolidation system that transparently virtualizes hardware resources to save energy.

The rest of this paper is organized as follows: Sections II and III present the design and implementation for the proposed near-threshold CMP with STT-RAM caches. Sections IV and V describe experimental methodology and evaluation. Section VI details some related work and Section VII concludes.

II. NT CMP with STT-RAM Caches

We design an NT CMP that uses STT-RAM for all on-chip caches. Figure 2 illustrates the chip's floorplan. The CMP is organized in clusters within which all cores share single L1 and L2 caches. The clusters themselves share the last-level cache (L3). The CMP makes use of two externally regulated voltage domains. One domain, which contains the core logic is set to low NT V_{dd}. The second, which encompasses the entire STT-RAM cache hierarchy and a few logic units, runs at high nominal V_{dd}. Note that two voltage domains are generally needed for SRAM-based NTV systems also because SRAM requires a higher voltage to operate reliably.

Running the STT-RAM caches at nominal V_{dd} dramatically improves write speed relative to the NT cores, reducing write latency from 10 cycles to about 3 cycles for a core running at 500MHz. Level-shifters [15] are needed for all cross voltage domain up-shift transitions (from low to high voltage domain). The delay overheads introduced by these circuits are compensated by the speed gain in the units running at higher voltages. We account for the level-shifting delay and power overhead in our evaluation.

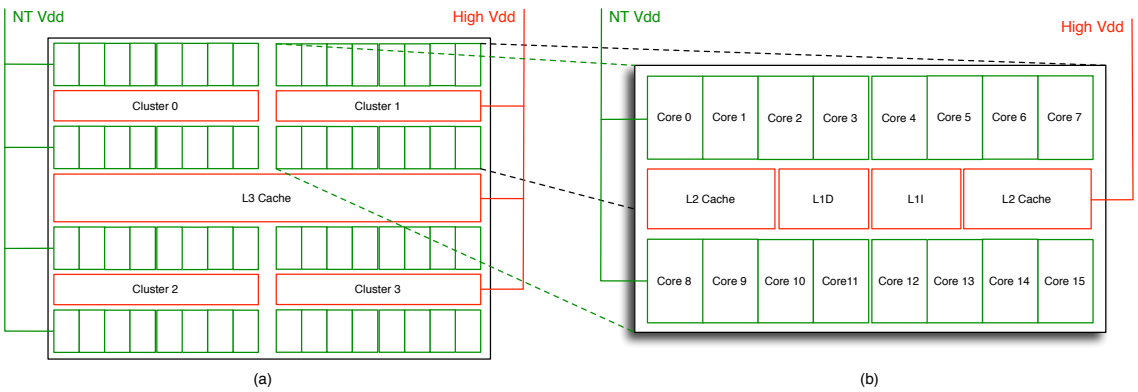

Figure 2: Floorplan of the clustered 64-core CMP design (a) with details of the cluster layout (b).

An additional benefit of the high voltage cache is that read accesses are very fast relative to the core speeds. For example, a 256KB STT-RAM L1 cache has a read speed around 0.4ns (in line with data reported by recent work [9], [10], [12]). The level shifters needed to access the high-V_{dd} shared cache add some delay overhead (0.75ns according to [15]). This overhead is incurred only when the voltage is upshifted from the NT V_{dd} of the cores to the high V_{dd} of the cache. Even with the level-shifting overhead (which can be pipelined at the cache side), the cache response time is significantly faster than the cycle time of the NT cores (ranging between 1.6ns and 2.4ns).

We exploit the fast read speeds and share a single L1 instruction, L1 data, and L2 cache among all the cores for each cluster. This is accomplished by running the shared L1 cache at a high frequency (2.5GHz in our experiments to match the 0.4ns access time) and time-multiplexing requests from the cores in each cluster. The main advantage of the shared cache design is that coherence is no longer necessary within each cluster. This greatly reduces the latency cost of sharing data between threads that are executing on cores in the same cluster. It also reduces coherence traffic, design complexity, and energy cost.

The large core-to-core variation associated with NT operation [7], [16] makes the approach of limiting the entire CMP to match the frequency of the slowest core very inefficient. Since fast cores are almost twice as fast as slow ones, we allow the respective cores across the CMP to run at the highest frequencies they can achieve. To keep the design cost effective, each cluster uses a single PLL for generating its base clock. The reference clock that feeds this PLL is based on the maximum frequency of the cache (e.g. 2.5GHz corresponding to 0.4ns). The cores run at integer multiples of the reference clock (e.g. 1.6ns, 2.0ns, 2.4ns) generated through clock multipliers. As a result, all cache access requests will align at cycle boundaries with the cache's reference clock, enabling the cache controller to efficiently arbitrate between requests from different cores.

A. Time-Multiplexing Cache Accesses

The shared cache controller handles multiple parallel requests from different cores using a form of time multiplexing. The primary goal of the cache controller is to return read hit requests to individual cores within a single core cycle. Since cores have cycle times, slower cores have more time slacks to have their requests serviced compared to faster cores. As a result, requests arriving at the same time are ordered based on the frequency of the requesting cores. Higher frequency cores are serviced first, while requests from slower cores receive lower priority. If the cache bandwidth is exceeded and a hit request cannot be serviced in time, a "half-miss" response is sent to the core and the request is serviced in the following cycle. In our evaluation, only about 4% of cache acesses result in half-misses.

Figure 3 shows an example of how multiple access requests from cores that are using different clock periods (1.6ns-2.4ns) are handled by a shared cache operating at 2.5GHz (0.4ns clock period). A cycle-by-cycle timeline of such requests is outlined in Figure 3 (a). In this figure, each request is associated with a line segment that represents the cycle time of the original core that issued it. This is a multiple of the reference clock that is used by the cache. For instance, Core 0 is running at 625MHz, which means its cycle time of 1.6ns is equal to 4 cache cycles. Since Core 0's request is received in cycle 0, to ensure that the cache responds within a single core clock cycle, the cache must send the data (or miss signal) by the end of cycle 3. Each core's request takes 2 fast cache cycles (0.8ns) to arrive at the cache due to wire and level-shifting overhead.

To keep track of all requests, the cache controller maintains a request register and a priority register for each core in the cluster. The request register stores the requested address, type, and the data for the read requests. The priority registers are shift registers preloaded with a representation of the number of fast cache cycles available for each request. Figure 3 (b) shows a view of the priority registers for the same example. For instance, for Core 0, which needs to be

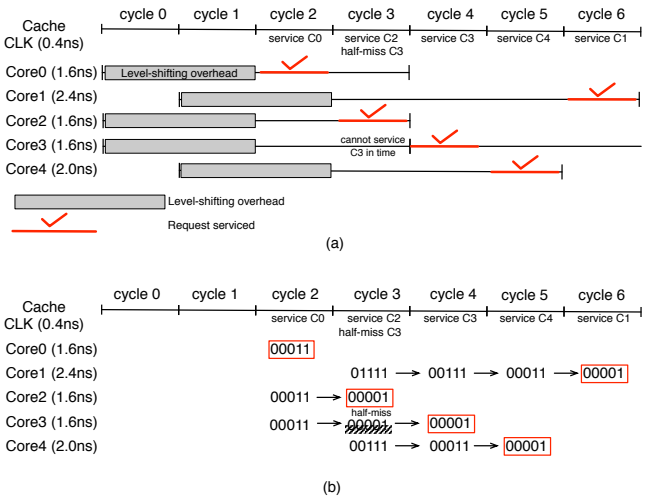

Figure 3: Example timeline of access requests from cores running at different frequencies to a shared cache running at high frequency.

serviced in two cache cycles, the request register is preloaded with "00011". Note that the cycles required to service each request account for the level shifting overhead. In other words, even though Core 0's request needs to be serviced in 1.6ns or four cache cycles, two of those are spent in the level shifters and wires. The remaining two are recorded in the priority register. For Core 1's request, which needs to be serviced in four cache cycles, the register is preloaded with "01111". All priority registers are right-shifted by one position each cache cycle to indicate a reduction in the available time for all unserviced requests.

At the end of cycle 2, the cache has three requests from Core 0, Core 2, and Core 3, out of which the cache can only service one. The cache controller picks the request that expires the soonest (i.e. the one with the fewer "1" bits) using simple selection logic. In this example all three requests have equal priority so the cache randomly chooses to service Core 0. This is indicated by the red "checkmarks" in Figure 3 (a) and the red rectangles in Figure 3 (b). The priority register corresponding to Core 0 is cleared and becomes available for a new request in the following cycle.

In cycle 3, the requests from Core 2 and Core 3 are both critical, meaning they have to be serviced in the current cycle (priority register is "00001"). Since the cache can only service one request it will choose Core 2's. A "half-miss" event will be sent to Core 3 to indicate that the request could not be fulfilled in a single cycle, but this is not necessarily an L1 miss. Core 3's request will be rescheduled through a reinitialization of the priority register. To increase its priority the register will be initialized to a lower value (in this example "00001"). Core 3's request will be serviced in cycle 4, which corresponds to a 2-cycle total hit latency. Requests from Core 4 and Core 1, issued in cycle 1 will be serviced in their priority order in cycles 5 and 6 respectively.

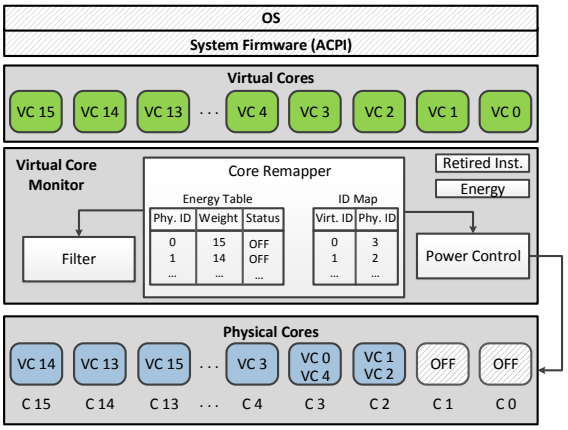

Figure 4: Overview of the virtual core management system integrated in one cluster.

III. Dynamic Core Management

The shared L1 cache design significantly reduces the performance overhead of migrating threads within the same cluster. This is because no cache data is lost after the migration. We take advantage of this feature to further reduce energy consumption with a dynamic core consolidation mechanism. The motivation behind core consolidation stems from the fact that NT cores exhibit high variability in maximum operating frequency. They also have a high ratio of leakage to dynamic power. As a result, cores that achieve a higher frequency at the same voltage are more energy efficient that then the low-frequency ones. In some situations it is therefore more energy efficient to power off the least efficient cores and consolidate their threads to the more efficient ones. This is generally true in low-IPC phases.

A. Core Virtualization

We find that low-IPC execution phases are relatively short and therefore taking advantage of them requires a low overhead, fast reacting mechanism for migrating threads and shutting down cores. We present a new hardware management mechanism that dynamically consolidates cores through a virtualization extension. The proposed system takes advantage of shared resources to transparently remap running applications across a set of heterogeneous cores.

Implementing this management system in hardware as opposed to the OS enables faster response times and lower performance overhead. In addition, the hardware-based core consolidation system is transparent to the OS and does not require OS intervention or support. This makes the solution easily deployable and backward compatible irrespective of the underlying hardware differences. Figure 4 depicts an overview of how our core management system would be integrated into a chip multiprocessor.

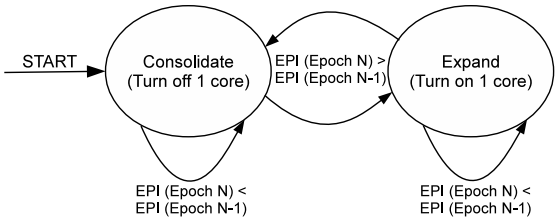

Figure 5: Greedy selection for dynamic core consolidation.

A key feature of our design is the ability to autonomously and transparently migrate threads to different physical cores without OS intervention. To that end our system makes use of virtual cores that provide a homogeneous view of processor resources to the OS. The virtual resources are made visible to the OS via the Advanced Configuration and Power Interface (ACPI) available within system firmware.

The core consolidation mechanism dynamically shuts down physical cores following an energy optimization algorithm. However, from the OS point of view, all virtual cores are always available. If some physical cores are off, a core mapping mechanism assigns multiple virtual cores to a single physical core.

B. Energy Optimization Algorithm

An energy monitoring and optimization system is implemented in firmware running on a dedicated on-chip microcontroller that is deployed in many of today's processors [17] for energy management.

A virtual core monitor (VCM) block, shown in Figure 4 is responsible for monitoring the energy per instruction (EPI) for each virtual core using hardware performance counters. The VCM also runs the energy optimization algorithm designed to dynamically search for the optimal number of active cores.

A simple greedy search algorithm (illustrated in Figure 5) guides the energy optimization. Execution is broken down into multiple epochs. At the end of each epoch the algorithm decides whether a physical core should be shut down, turned on, or if nothing needs to change. The EPI of the current epoch is compared to that of the previous one. If the difference exceeds a predefined threshold, then physical cores are either turned off or on.

The system starts with all physical cores on for an entire epoch. At the end of the first epoch, one physical core is shut down and its virtual core migrated to another core. The new EPI is measured at the end of the epoch. If energy is lower, the greedy search continues by progressively shutting down additional cores. If energy is higher, the search reverses direction. If EPI difference between the current and previous epoch is lower than the threshold, the current state is maintained for the next epoch. This is done to avoid excessive state changes for minor energy benefits.

In addition, the algorithm applies an exponential back-off to eliminate unnecessary oscillations between neighboring states. The history of recent state changes within each cluster is recorded. If the system detects an oscillating pattern, it exponentially increases the number of epochs during which it will hold the current state before attempting a state change (e.g. 2, 4, 8, 16, and 32 epochs).

C. Virtual Core Consolidation

Core consolidation within a cluster is handled by the core remapper module depicted in Figure 4. Whenever a power down/up event is required, the remapper examines the pool of active physical cores. An energy efficiency score is precomputed and recorded in a table. The score is determined based on the frequency of the core. Faster cores are more energy efficient because they can achieve a lower energy per instruction at the same voltage than lower frequency cores. The primary reason is that the high leakage power that dominates NT cores is a fixed cost independent of frequency. Using the energy profile, the system will turn off the least efficient active core, or turn on the most efficient inactive core as dictated by the greedy search.

Once a core is marked for deconfiguration, the remapper assigns one of the remaining active physical cores as a host for the unassigned virtual core. To keep the design simple, allocations to active physical cores are performed in a round robin fashion. We start allocations with the most efficient core (fastest) and move down to the least efficient one (slowest). This means that multiple virtual cores are more likely to be consolidated on the faster physical cores, thus alleviating the performance impact of consolidation.

The migration of the virtual core follows two main phases. In the first phase, the deconfigured core stops fetching new instructions and saves the next PC into a consolidation register. The core continues to execute instructions until all in-flight instructions are committed. The register file content is then saved. In the second phase, the target physical core is interrupted and the register file image and the PC are transferred. Execution resumes on the new core. Once the remapping is complete, the virtual-to-physical ID map is updated accordingly to reflect the new association. A request is then issued to the power control module to power gate the deconfigured core. A similar migration process is followed when a new physical core is activated and a virtual core is migrated to it.

If multiple virtual cores are mapped to a single physical resource, hardware-based context switches are performed at regular intervals that are much smaller than the typical OS context-switch interval. This ensures fairness and uniform progress of the virtual cores such that they all appear to be running simultaneously.

D. Mitigating Core Consolidation Overhead

There are a few sources of potential performance overhead associated with our core consolidation mechanism. The biggest potential cost for our remapping scheme is the loss of data stored in local caches. If remapping is frequent, "cold-cache" effects can severely degrade performance. In our CMP design, we restrict the remapping of virtual cores to occur only within clusters. This means that application level threads that are associated with virtual cores don't lose any data locality since the entire cache hierarchy is shared at the cluster level.

Another source of overhead is the loss of architectural state associated with each individual thread including branch prediction history and on-chip data stored in register files or reorder buffers. After every consolidation the architectural information of each newly remapped thread is lost. It takes tens of cycles to rebuild those states before the thread can perform any useful work. Therefore if remapping occurs too frequently, the overall performance can suffer. We address this issue by carefully choosing a reasonable consolidation interval. With experiments we find that remapping performed every 160K instructions carries only a small performance penalty and returns optimal energy savings.

Finally, another potential source of overhead is related to the action of powering on cores. After a core is turned on from a power-gated state, voltage noise can cause timing errors [18]. To prevent that, the core is stalled for a brief period of time. However, because the cores run at NT voltage and their power is relatively low compared to their available capacitance the noise is small. As a result the penalty for voltage stabilization is only about 10-30ns [7] or 5-15 cycles for a core running at 500MHz.

All types of overheads discussed above are properly reflected in our design and included in the evaluation results shown in Section V.

IV. Evaluation Methodology

We modeled a 64-core CMP with a range of cluster sizes from 4 to 32 cores. Most experiments were conducted with a cluster size of 16 cores, which we found to be optimal. We also experimented with three cache configurations: small, medium, and large. The size of the caches were chosen to provide between 1MB (small) and 4MB (large) of cache for each core, in line with existing commercial designs [17], [19]. Also in line with existing designs, our medium cache configuration accounts for approximately 25% of the total chip area. In the large configuration the total cache area represents 50% of the chip area. Most of our results are reported for the medium cache configuration. The small and large are included for reference and trend analysis. Table I summarizes our cache configurations at different levels.

In our experiments each core has a dual-issue out-of-order architecture. We used SESC [20] to perform all of our simulations. We collected runtime, power, and energy information. Table II summarizes the baseline architecture configuration parameters. NVSim [21] combined with CACTI [22] was used to obtain STT-RAM latency, energy, and area. Similarly, per access energy for all SRAM memory structures including register file, reorder buffer, load/store queue, and instruction window were extracted through CACTI. McPAT [5] was used to model energy per access for all CMOS logic units such as ALUs and FPUs. We included a model for leakage power based on estimated unit area and technology (CMOS vs. MTJ). This information was inserted into SESC's activity model in order to obtain total power and energy consumption. Table III lists the technology parameters we obtained from NVSim and CACTI for various types of L1 data caches. The cache areas reported take into account the higher density of STT-RAM compared to SRAM. We rounded STT-RAM cache read latency up to 0.4ns to align clock edges between the shared cache and cores. Parameters of other cache hierarchies are similarly simulated and properly fed into our architecture simulations.

Two benchmark suites were adopted in the evaluation: SPLASH2 and PARSEC. SPLASH2 (*barnes, cholesky, fft, lu, ocean, radiosity, radix, raytrace,* and *water-nsquared*) was configured to run with reference input sets. PARSEC (*blackscholes, bodytrack, streamcluster,* and *swaptions*), on

Hierarchy	Size	Block Size	Assoc.	Rd/Wr Ports
L1I (Private/ Shared w/i Cluster)	16KB (Private)/ 256KB (Shared w/i Cluster)	32B	2-way	1/1
L1D (Private/ Shared w/i Cluster)	16KB (Private)/ 256KB (Shared w/i Cluster)	32B	4-way	1/1
L2 (Shared w/i Cluster)	8MB (Small)/ 16MB (Medium)/ 32MB (Large)	64B	8-way	1/1
L3 (Shared w/i Chip)	24MB (Small)/ 48MB (Medium)/ 96MB (Large)	128B	16-way	1/1

Table I: Summary of cache configurations.

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-V_{dd}	0.4V (Core), 0.65V (Cache)
Nominal-V_{dd}	1.0V
Core Frequency Range	375MHz-725MHz
Median Core Frequency	500MHz
Variation Parameters	
V_{th} std. dev./mean (σ/μ)	12% (chip), 10% (cluster)

Table II: CMP architecture parameters.

	V_{dd} Rail	Area (mm^2)	Rd/Wr Lat. (ps)	Rd/Wr Eng. (pJ)	Leakage (mW)
SRAM (16KB×16)	Low (0.65V)	0.9176	1337	2.578	573
SRAM (16KB×16)	High (1.0V)	0.9176	211.90	6.102	881
SRAM (256KB)	High (1.0V)	0.9176	533.60	42.41	881
STT-RAM (256KB)	High (1.0V)	0.2451	388.20/ 5208	29.32/ 209.30	114

Table III: L1 data cache technology parameters.

Configuration & Description	
PR-SRAM-NT (baseline)	NT chip with SRAM private L1(I/D) cache and shared L2/L3 cache
HP-SRAM-CMP (alt. baseline)	Traditional high-performance CMP with cores and caches at nominal V_{dd}
SH-SRAM-Nom	NT core with nominal V_{dd} SRAM shared L1(I/D) cache and shared L2/L3 cache
SH-STT	SH-SRAM-Nom with all caches built in STT-RAM
SH-STT-CC	SH-STT that performs hardware-managed dynamic core consolidation
SH-STT-CC-Oracle	SH-STT-CC with oracle knowledge for dynamic core consolidation
PR-STT-CC	SH-STT-CC with private L1(I/D) cache
SH-STT-CC-OS	SH-STT-CC with OS-managed dynamic core consolidation

Table IV: Architecture configurations used in the evaluation.

the other hand, was launched with sim-small input sets. We used VARIUS [23] to model variation effects on threshold voltages (V_{th}) across the CMP. We generated distributions of core frequencies that were used in the simulations.

V. Evaluation

In this section we show performance and energy benefits of the proposed architecture. We also include sensitivity studies on optimal cluster size, shared cache behavior, and dynamic core consolidation mechanism. For easy reference, Table IV summarizes all the architecture configurations used in our evaluation.

A. Power Analysis

Figure 6 shows the reduction in power consumption from the proposed STT-RAM-based CMP architecture without dynamic core consolidation (SH-STT). Since the power savings we obtain are dependent on the size of the cache, we show results for three cache configurations (Table I): small, medium, and large. The medium size cache is the most typical one, with about 2MB/core of total cache capacity. In this configuration, the cache accounts for approximately 25% of the chip area.

We compare to a baseline that uses SRAM caches running at a low voltage rail (0.65V) in a traditional private cache hierarchy (PR-SRAM-NT). This is the most typical near-threshold CMP design. The reason why SRAM caches run at a higher voltage rail is to ensure acceptable reliability

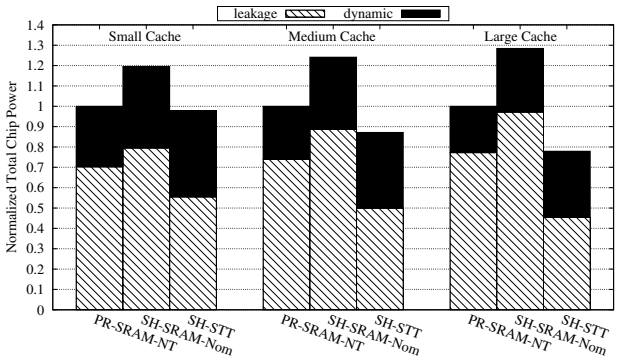

Figure 6: Power reduction of proposed design for three L2/L3 cache sizes: small, medium, and large.

since SRAM caches running at NT V_{dd} would be unusable without cell resizing or strong error correction [24], [4] – both of which carry significant overheads.

We can see that power is lower for SH-STT compared to the baseline in all configurations. The reduction in total power comes from lower leakage power at the cost of slightly increased dynamic power (due to nominal voltage STT-RAM reads and the high cost of STT-RAM writes). For the small cache configuration the power is only about 2.1% lower. For the medium and large configurations the power savings are significant, at 12.9% and 22.1% respectively.

Figure 6 also shows a breakdown of leakage and dynamic power for each configuration. We can see that for STT-RAM, even though dynamic power is higher due to the nominal voltage cache operations, the reduction in leakage power compensates for it in all three cache size configurations.

For reference, we also compare to an SRAM design in which the cache is shared and also running at nominal voltage (SH-SRAM-Nom), the same configuration used by our proposed design but with SRAM caches. This ensures reliable operation but is costly in terms of power. SH-SRAM-Nom uses between 22% and 65% more power than SH-STT for the three cache sizes. This is due to the much higher leakage power consumed by SRAM running at nominal voltage.

B. Performance Analysis

The shared cache design brings significant performance improvements compared to the baseline system. Figure 7 shows the execution time of the proposed STT-RAM design (SH-STT) with medium-sized cache. The results are normalized to the PR-SRAM-NT baseline. Process variation effects (core frequency distributions) are modeled in all configurations. We can see that the SH-STT configuration reduces execution time by an average of 11%. This performance improvement is due to the benefits of within-cluster cache sharing. Applications that benefit the most are those in which there is significant data sharing and reuse such as *raytrace*.

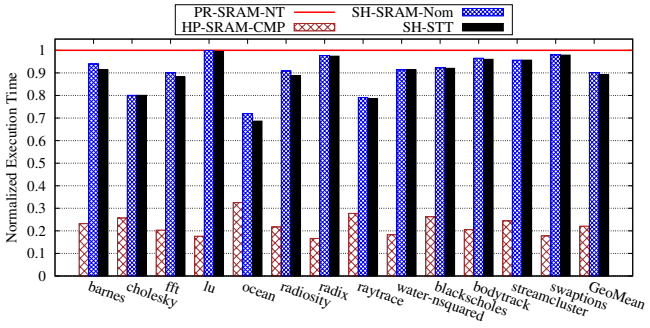

Figure 7: Relative runtime of SPLASH2 and PARSEC benchmarks for various designs with medium-sized cache.

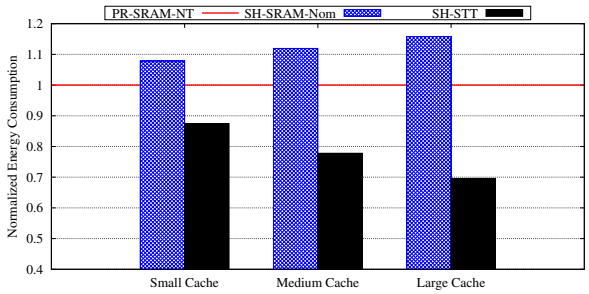

Figure 8: Energy consumption for small, medium, and large L2 and L3 cache configurations.

Applications such as *ocean* also benefit significantly because they make heavy use of synchronization (*ocean* has hundreds of barriers). Synchronization is much faster in the shared cache design because it involves much less coherence traffic.

We also compare SH-STT to SH-SRAM-Nom (as before) and we add another baseline, HP-SRAM-CMP. HP-SRAM-CMP represents a conventional high-performance design in which the entire CMP (cores plus caches) run at nominal voltage. Figure 7 shows that compared to SH-SRAM-Nom our proposed SH-STT design achieves marginally better performance (1.2% on average) because of slightly faster read speed of STT-RAM compared to SRAM. The high-performance HP-SRAM-CMP achieves the lowest execution time because it runs at high voltage and high frequency. This performance, however, comes at a much higher energy cost.

C. Energy Analysis

Our design reduces both power consumption and execution time resulting in important energy savings. Figure 8 shows that SH-STT has between 13% and 31% lower energy than PR-SRAM-NT baseline depending on cache sizes. As expected we see larger energy savings for larger cache sizes. We also show that the SH-SRAM-Nom configuration which uses shared SRAM caches at nominal V_{dd} uses 8-16% more energy than the NT SRAM baseline (PR-SRAM-NT).

Figure 9 shows the energy breakdown by benchmark for our designs with the medium-sized cache relative to the PR-SRAM-NT baseline. The shared STT-RAM cache design (SH-STT) reduces energy by an average of 23%. This is in stark contrast with a similar shared cache configuration that uses SRAM at nominal V_{dd} (SH-SRAM-Nom), which increases energy by 12%. The high-performance baseline HP-SRAM-CMP consumes 40% more energy on average than the PR-SRAM-NT baseline. Relative to HP-SRAM-CMP, our SH-STT design has an average of 45% lower energy consumption. When we add dynamic core consolidation (SH-STT-CC), we reduce energy by an additional 10% for a combined 33% reduction relative to PR-SRAM-NT (51% reduction relative to HP-SRAM-CMP).

We also include an oracle version of the dynamic core consolidation solution (SH-STT-CC-Oracle) to show the limits of our greedy-search-based energy optimization. We obtained SH-STT-CC-Oracle by choosing the optimal number of cores to consolidate at each evaluation interval. SH-STT-CC-Oracle reduces energy consumption by 36%. The small 3% difference between the Oracle and our implementation is due to the slight sub-optimality of the greedy search we perform. Overall it is a small penalty to pay for a fast optimization that can be deployed in production systems.

Figure 9 also compares the energy reduction of SH-STT-CC relative to other possible alternatives for implementing core consolidation. PR-STT-CC shows the energy of a solution that attempts core consolidation with private STT-RAM caches. Because of the overhead of consolidating cores with private caches (which results in loss of cache locality after consolidation), PR-STT-CC reduces energy by only 24% compared to 33% for SH-STT-CC.

We also compare with an approach in which core consolidation is handled by the OS at coarser time intervals (1ms). SH-STT-CC-OS does not require any hardware support since consolidation is controlled by the OS. However, because consolidated threads are context-switched at coarser intervals, critical threads can easily bottleneck the entire application when they are not running. This hurts performance significantly to the point where energy actually increases by 27% compared to SH-STT.

D. Optimal Cluster Size

A key parameter for our design is the cluster size. We run simulations with cluster sizes of 4, 8, 16, and 32 cores. Table V summarizes the results. Note that, as we increase the cluster size we also proportionally increase the shared L1 cache size. For the entire CMP, the total core count and the sum of all L1 cache capacities remain constant.

Performance improves in SH-STT when going from 4 to 16 cores per cluster by 5% to 11% compared to PR-SRAM-NT baseline. This is due to the increased opportunity for data sharing and reduced coherence traffic. The downside is increased bandwidth pressure on the shared cache. When the

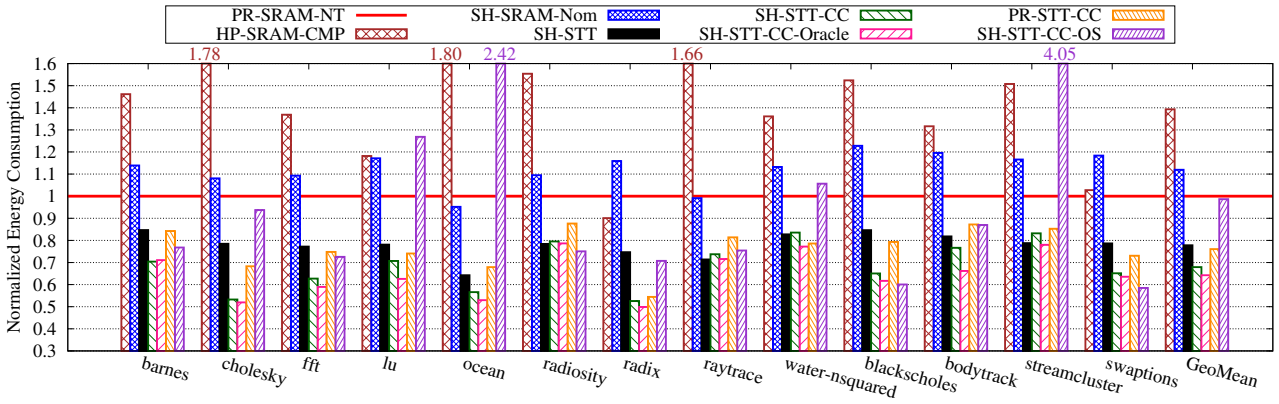

Figure 9: Energy consumption for SPLASH2 and PARSEC benchmarks with a core consolidation interval of 160K instructions and a medium-sized L2 and L3 cache.

Cluster Size (#cores)	Shared Cache Size (KB)	Performance Gain (%)
4	64	4.82
8	128	6.29
16	256	10.81
32	512	2.50

Table V: Cluster size impact on performance.

cluster size is increased to 32 cores, performance improvement drops to only 2.5%. The larger cache size (512KB for 32 cores vs. 256KB for 16 cores) has higher access latency and lower bandwidth. At the same time the number of cores goes from 16 to 32, generating a lot more requests and overwhelming the reduced bandwidth. The optimal cluster size for this design is therefore 16 cores.

E. Shared Cache Impact on Access Latency

The shared cache design cannot guarantee single cycle access to all cache read hits. If requests cannot be serviced in the equivalent of a core cycle, a "half-miss" response is returned to the core. In order to better understand the impact of the shared cache contention on access latency, we conducted two sets of experiments.

The first experiment measures cache utilization by looking at the number of requests arriving at the shared cache each cycle. Figure 10 shows percentage of the total cache cycles in which a given number of requests arrive at the shared cache. We count all requests handled by the cache including reads, writes, line fills, etc. We show numbers for five different benchmarks and the arithmetic mean *of all* our benchmarks.

We can see that, on average, almost half of the cache cycles (49%) have no incoming requests, 21% with one request, 15% with two requests, 9% with three requests, and 6% with more than four requests. This shows that requests exceeding the number of available ports (1 read/1 write) occur in about 30% of the cache cycles. However, these are

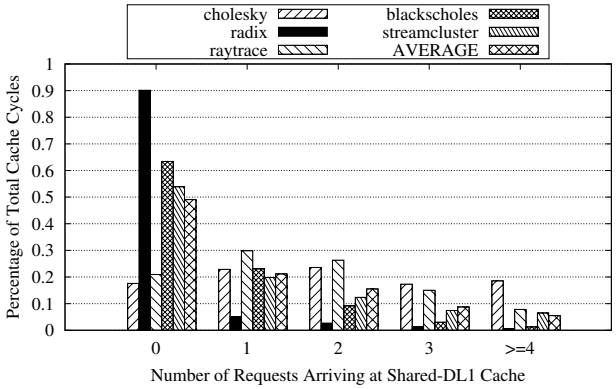

Figure 10: Shared DL1 cache utilization rate in one cluster.

fast cache cycles and each requesting core has considerable time slacks in which to receive a response. As a result, most of these requests will not receive a delayed response.

Figure 11 shows a histogram of the percentages of read hit requests serviced in 1, 2, or more core cycles. We can see that the vast majority of requests are handled in 1 cycle (95.8%). About 4% of requests result in half-misses and over 99% of those are handled in 2 cycles. As a result, the performance impact of the cache contention is small, and more than compensated by the benefits of the shared cache.

F. Dynamic Core Consolidation

Figure 12 shows a detailed runtime trace of *radix* when performing dynamic core consolidation. We show traces for both SH-STT-CC and SH-STT-CC-Oracle to compare the effectiveness of our consolidation mechanism. We can see that except for a few data points, our consolidation trace matches very well with the oracle trace. This leads to very close energy savings for SH-STT-CC (48%) and SH-STT-CC-Oracle (50%) compared to PR-SRAM-NT baseline.

Occasionally the greedy search does not respond sufficiently fast to keep up with workload changes, whereas the

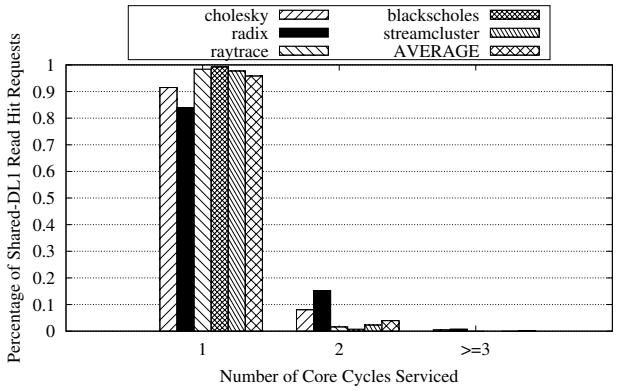

Figure 11: Fraction of read hit requests serviced by the shared DL1 cache in 1, 2, or more core cycles.

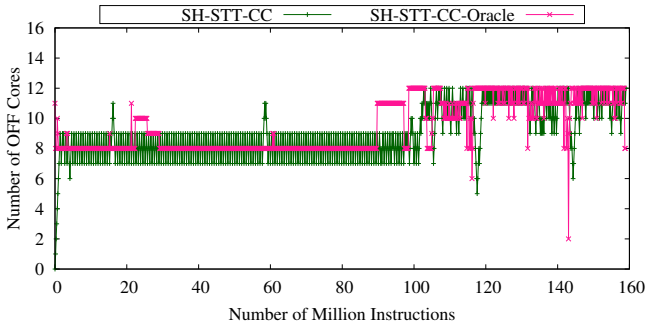

Figure 12: Core consolidation trace of *radix*.

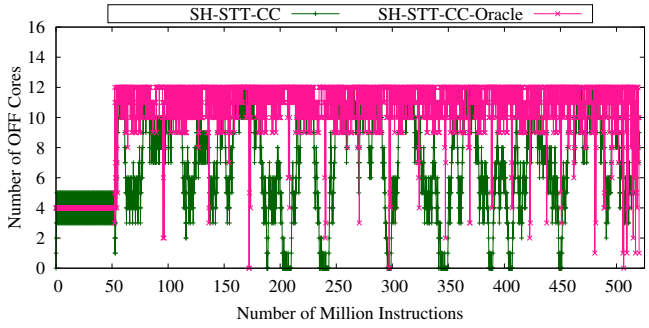

Figure 13: Core consolidation trace of *lu*.

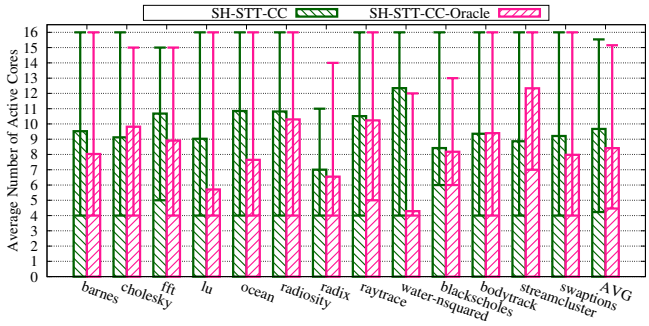

Figure 14: Average number of active cores (and min and max values) using core consolidation for SPLASH2 and PARSEC benchmarks.

oracle adapts immediately. This can be observed in benchmarks such as *lu*, shown in Figure 13. The greedy search gradually searches for the optimal energy point, resulting in some temporary sub-optimal behavior. As a result, for the *lu* benchmark, our proposed SH-STT-CC design saves 29% energy while SH-STT-CC-Oracle saves 38%.

Dynamic core consolidation takes advantage of the large variability in application behavior both within and across workloads. To illustrate this, Figure 14 shows the average number of active cores in a cluster for each benchmark. We can see that on average only 10 out of 16 cores in a cluster are used. Note that, however, there is high variability in the number of active cores both across and within benchmarks. The markers on each bar indicate the range of active cores throughout the execution. The startup phase of each benchmark is excluded. We can see that for most benchmarks, core consolidation takes advantage of the full dynamic range from 16 to 4 active cores per cluster. Some exceptions include *radix* which only activates 11 cores per cluster at the most and *blackscholes* which never uses fewer than 6 physical cores.

VI. Related Work

The idea of STT-RAM caches in NT multiprocessors was first proposed by the authors in an extended abstract [25]. This paper greatly expands that initial idea with design details, evaluation, and the core consolidation mechanism. Previous work by Zhai et al. [26] has proposed grouping several slower near-threshold cores into a cluster that shares a faster L1 cache in order to eliminate cache coherence traffic. This can speed up system performance and also reduce coherence energy. In their design they applied a relatively higher voltage to the shared SRAM L1 cache and found the optimal energy efficiency configuration is 2 cores per cluster with 2 clusters. They did not explicitly consider variation effects or heterogeneous core frequencies in their design. We use nominal voltage STT-RAM to build the shared L1 cache. In our design the STT-RAM shared cache is much faster than the cores, making much larger clusters (16 cores) become optimal. In addition, our work takes advantage of this shared cache design to perform dynamic core consolidation to further optimize energy consumption.

Prior work [13], [14] has proposed implementing caches with STT-RAM. They take advantage of its high-density characteristic to build large capacity on-chip STT-RAM or SRAM with STT-RAM hybrid caches. To the best of our knowledge this is the first work that examines STT-RAM in the context of near-threshold CMPs.

VII. Conclusion

This is the first paper to explore the use of STT-RAM in near-threshold processors. We find STT-RAM to be an ideal SRAM replacement at near-threshold for two reasons: first, it has very low leakage, which dominates near-threshold designs; second, it can efficiently run at nominal voltages, avoiding the reliability problems of low-V_{dd} SRAM.

Acknowledgment

The authors would like to thank the anonymous reviewers for their feedback. This work was supported in part by the National Science Foundation under grants CCF-1253933 and CCF-1629392, and the Defense Advanced Research Projects Agency under the PERFECT (DARPA-BAA-12-24) program.

References

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *International Symposium on Computer Architecture (ISCA)*, pp. 365–376, 2011.

[2] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, pp. 253–266, February 2010.

[3] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey, "Ultralow-Power Design in Near-Threshold Region," *Proceedings of the IEEE*, vol. 98, pp. 237–252, February 2010.

[4] T. N. Miller, R. Thomas, J. Dinan, B. Adcock, and R. Teodorescu, "Parichute: Generalized Turbocode-Based Error Correction for Near-Threshold Caches," in *International Symposium on Microarchitecture (MICRO)*, pp. 351–362, 2010.

[5] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *International Symposium on Microarchitecture (MICRO)*, pp. 469–480, 2009.

[6] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson, "A Sub-200mV 6T SRAM in 0.13μm CMOS," in *International Solid-State Circuits Conference (ISSCC)*, pp. 332–606, 2007.

[7] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 27–38, 2012.

[8] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving Cache Lifetime Reliability at Ultra-Low Voltages," in *International Symposium on Microarchitecture (MICRO)*, pp. 89–99, 2009.

[9] 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)*, pp. 371–382, 2010.

[10] 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)*, pp. 50–61, 2011.

[11] X. Pan and R. Teodorescu, "NVSleep: Using Non-Volatile Memory to Enable Fast Sleep/Wakeup of Idle Cores," in *International Conference on Computer Design (ICCD)*, pp. 400–407, 2014.

[12] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in *Design Automation Conference (DAC)*, pp. 243–252, 2012.

[13] 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)*, pp. 239–249, 2009.

[14] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," in *International Symposium on Computer Architecture (ISCA)*, pp. 34–45, 2009.

[15] J. Garcia, J. Montiel-Nelson, J. Sosa, and S. Nooshabadi, "High Performance Single Supply CMOS Inverter Level Up Shifter for Multi-Supply Voltages Domains," in *Design Automation and Test in Europe (DATE)*, pp. 1273–1276, 2015.

[16] T. N. Miller, R. Thomas, and R. Teodorescu, "Mitigating the Effects of Process Variation in Ultra-low Voltage Chip Multiprocessors using Dual Supply Voltages and Half-Speed Stages," *IEEE Computer Architecture Letters*, vol. 11, July 2012.

[17] "Intel Core™ i7 Processor." http://www.intel.com.

[18] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and Eliminating Synchronization-Induced Voltage Emergencies in Many-core Processors," in *International Symposium on Computer Architecture (ISCA)*, pp. 249–260, 2012.

[19] R. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski, "A 32nm 3.1 Billion Transistor 12-Wide-Issue Itanium Processor for Mission-Critical Servers," in *International Solid-State Circuits Conference (ISSCC)*, pp. 84–86, 2011.

[20] "SESC Simulator." http://sesc.sourceforge.net.

[21] 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, pp. 994–1007, July 2012.

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

[23] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Parameter Variation and Resulting Timing Errors for Microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, pp. 3–13, February 2008.

[24] H. R. Ghasemi, S. Draper, and N. S. Kim, "Low-Voltage On-Chip Cache Architecture Using Heterogeneous Cell Sizes for High-Performance Processors," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 38–49, 2011.

[25] 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)*, pp. 485–486, 2014.

[26] B. Zhai, R. G. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester, "Energy Efficient Near-Threshold Chip Multi-Processing," in *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 32–37, 2007.