Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing

Xiaochen Guo Engin İpek Tolga Soyata

University of Rochester
Rochester, NY 14627 USA
{xiguo, ipek, soyata}@ece.rochester.edu

ABSTRACT

As CMOS scales beyond the 45nm technology node, leakage concerns are starting to limit microprocessor performance growth. To keep dynamic power constant across process generations, traditional MOSFET scaling theory prescribes reducing supply and threshold voltages in proportion to device dimensions, a practice that induces an exponential increase in subthreshold leakage. As a result, leakage power has become comparable to dynamic power in current-generation processes, and will soon exceed it in magnitude if voltages are scaled down any further. Beyond this inflection point, multicore processors will not be able to afford keeping more than a small fraction of all cores active at any given moment. Multicore scaling will soon hit a power wall.

This paper presents *resistive computation*, a new technique that aims at avoiding the power wall by migrating most of the functionality of a modern microprocessor from CMOS to spin-torque transfer magnetoresistive RAM (STT-MRAM)—a CMOS-compatible, leakage-resistant, non-volatile resistive memory technology. By implementing much of the on-chip storage and combinational logic using leakage-resistant, scalable RAM blocks and lookup tables, and by carefully re-architecting the pipeline, an STT-MRAM based implementation of an eight-core Sun Niagara-like CMT processor reduces chip-wide power dissipation by 1.7× and leakage power by 2.1× at the 32nm technology node, while maintaining 93% of the system throughput of a CMOS-based design.

Categories and Subject Descriptors

B.3.1 [**Memory Structures**]: Semiconductor Memories; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms

Design, Performance

Keywords

Power-efficiency, STT-MRAM

1. INTRODUCTION

Over the past two decades, the CMOS microprocessor design process has been confronted by a number of seemingly insurmountable technological challenges (e.g., the memory wall [4] and the wire delay problem [1]). At each turn, new classes of systems have been architected to meet these challenges, and microprocessor performance has continued to scale with exponentially increasing transistor budgets. With more than two billion transistors integrated on a single die [27], power dissipation has become the current critical challenge facing modern chip design. On-chip power dissipation now exhausts the maximum capability of conventional cooling technologies; any further increases will require expensive and challenging solutions (e.g., liquid cooling), which would significantly increase overall system cost.

Multicore architectures emerged in the early 2000s as a means of avoiding the power wall, increasing parallelism under a constant clock frequency to avoid an increase in dynamic power consumption. Although multicore systems did manage to keep power dissipation at bay for the past decade, with the impending transition to 32nm CMOS, they are starting to experience scalability problems of their own. To maintain constant dynamic power at a given clock rate, supply and threshold voltages must scale with feature size, but this approach induces an exponential rise in leakage power, which is fast approaching dynamic power in magnitude. Under this poor scaling behavior, the number of active cores on a chip will have to grow much more slowly than the total transistor budget allows; indeed, at 11nm, over 80% of all cores may have to be dormant at all times to fit within the chip's thermal envelope [16].

This paper presents *resistive computation*, an architectural technique that aims at developing a new class of power-efficient, scalable microprocessors based on emerging resistive memory technologies. Power- and performance-critical hardware resources such as caches, memory controllers, and floating-point units are implemented using spin-torque transfer magnetoresistive RAM (STT-MRAM)—a CMOS- compatible, near-zero static-power, persistent memory that has been in development since the early 2000s [12], and is expected to replace commercially available magnetic RAMs by 2013 [13]. The key idea is to implement most of the on-chip storage and combinational logic using scalable, leakage-resistant RAM arrays and lookup tables (LUTs) constructed from STT-MRAM to lower leakage, thereby allowing many more active cores under a fixed power budget than a pure CMOS implementation could afford.

By adopting hardware structures amenable to fast and efficient LUT-based computing, and by carefully re-architecting the pipeline, an STT-MRAM based implementation of an eight-core, Sun Niagara-like CMT processor reduces leakage and total power at 32nm by 2.1× and 1.7×, respectively, while maintaining 93% of the system throughput of a pure CMOS implementation.

2. BACKGROUND AND MOTIVATION

Simultaneously to power-related problems in CMOS, DRAM is facing severe scalability problems due to precise charge placement and sensing hurdles in deep-submicron processes. In response, the industry is turning its attention to *resistive memory technologies* such as phase-change memory (PCM), memristors (RRAM), and spin-torque transfer magnetoresistive RAM (STT-MRAM)—memory technologies that rely on resistivity rather than charge as the information carrier, and thus hold the potential to scale to much smaller geometries than charge memories [13]. Unlike the case of SRAM or DRAM, resistive memories rely on non-volatile, resistive information storage in a cell, and thus exhibit near-zero leakage in the data array.

2.1 STT-MRAM

STT-MRAM [13,20,31–33] is a second generation MRAM technology that addresses many of the scaling problems of commercially available toggle-mode magnetic RAMs. Among all resistive memories, STT-MRAM is the closest to being a CMOS-compatible *universal memory* technology as it offers read speeds as fast as SRAM [39] (< 200ps in 90nm), density comparable to DRAM ($10F^2$), scalable energy characteristics [13], and infinite write endurance. Functional array prototypes [14, 20, 31], CAM circuits [37], and simulated FPGA chips [39] using STT-MRAM have already been demonstrated, and the technology is under rapid commercial development, with an expected industry-wide switch from toggle-mode MRAM to STT-MRAM by 2013 [13]. Although MRAM suffers from relatively high write power and write latency compared to SRAM, its near-zero leakage power dissipation, coupled with its fast read speed and scalability makes it a promising candidate to take over as the workhorse for on-chip storage in sub-45nm processes.

Memory Cells and Array Architecture. STT-MRAM relies on magnetoresistance to encode information. Figure 1 depicts the fundamental building block of an MRAM cell, the magnetic tunnel junction (MTJ). An MTJ consists of two ferromagnetic layers and a tunnel barrier layer, often implemented using a magnetic thin-film stack comprising $Co_{40}Fe_{40}B_{20}$ for the ferromagnetic layers, and MgO for the tunnel barrier. One of the ferromagnetic layers, the *pinned layer*, has a fixed magnetic spin, whereas the spin of the electrons in the *free layer* can be influenced by first applying a high-amplitude current pulse through the pinned layer to polarize the current, and then passing this spin-polarized current through the free layer. Depending on the direction of the current, the spin polarity of the free layer can be made either parallel or anti-parallel to that of the pinned layer.

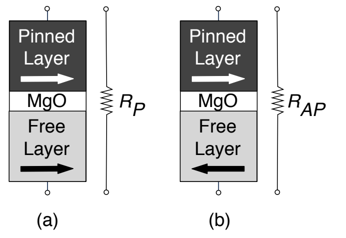

Figure 1: Illustrative example of a magneto-tunnel junction (MTJ) in (a) low-resistance parallel and (b) high-resistance anti-parallel states.

Applying a small bias voltage (typically 0.1V) across the MTJ causes a tunneling current to flow through the MgO tunnel barrier without perturbing the magnetic polarity of the free layer. The magnitude of the tunneling current—and thus, the resistance of the MTJ—is determined by the polarity of the two ferromagnetic layers: a lower, *parallel resistance* (R_P in Figure 1-a) state is experienced when the spin polarities agree, and a higher, *antiparallel resistance* state is observed when the polarities disagree (R_{AP} in Figure 1-b). When the polarities of the two layers are aligned, electrons with polarity anti-parallel to the two layers can travel through the MTJ easily, while electrons with the same spin as the two layers are scattered; in contrast, when the two layers have anti-parallel polarities, electrons of either polarity are largely scattered by one of the two layers, leading to much lower conductivity, and thus, higher resistance [6]. These low and high resistances are used to represent different logic values.

The most commonly used structure for an STT-MRAM memory cell is the 1T-1MTJ cell that comprises a single MTJ, and a single transistor that acts as an access device (Figure 2). Transistors are built in CMOS, and the MTJ magnetic material is grown over the source and drain regions of the transistors through a few (typically two or three) additional process steps. Similarly to SRAM and DRAM, 1T-1MTJ cells can be coupled through wordlines and bitlines to form memory arrays. Each cell is read by driving the appropriate wordline to connect the relevant MTJ to its bitline (BL) and source line (SL), applying a small bias voltage (typically $0.1V$) across the two, and by sensing the current passing through the MTJ using a current sense amplifier connected to the bitline. Read speed is determined by how fast the capacitive wordline can be charged to turn on the access transistor, and by how fast the bitline can be raised to the required read voltage to sample the read-out current. The write operation, on the other hand, requires activating the access transistor, and applying a much higher voltage (typically Vdd) that can generate enough current to modify the spin of the free layer.

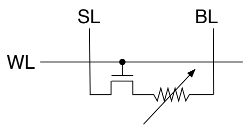

Figure 2: Illustrative example of a 1T-1MTJ cell.

An MTJ can be written in a thermal activation mode through the application of a long, low-amplitude current pulse (>10ns), under a dynamic reversal regime with intermediate current pulses (3-10ns), or in a precessional switching regime with a short (<3ns), high-amplitude current pulse [12]. In a 1T-1MTJ cell with a fixed-size MTJ, a tradeoff exists between switching time (i.e., current pulse width) and cell area. In precessional mode, the required current density $J_c(\tau)$ to switch the state of the MTJ is inversely proportional to switching time τ

$$J_c(\tau) \propto J_{c0} + \frac{C}{\tau}$$

where J_{c0} is a process-dependent intrinsic current density parameter, and C is a constant that depends on the angle of the magnetization vector of the free layer [12]. Hence, operating at a faster switching time increases energy-efficiency: a 2× shorter write pulse requires a less than 2× increase in write current, and thus, lower write energy [8, 20, 26]. Unfortunately, the highest switching speed possible with a fixed-size MTJ is restricted by two fundamental factors: (1) the maximum current that the cell can can support during an $R_{AP} \rightarrow R_P$ transition cannot exceed Vdd/R_{AP} since the cell has to deliver the necessary switching current over the MTJ in its high-resistance state, and (2) a higher switching current requires the access transistor to be sized larger so that it can source the required current, which increases cell area [1] and hurts read energy and delay due to higher gate capacitance.

Figure 3 shows 1T-1MTJ cell switching time as a function of cell area based on Cadence-Spectre analog circuit simulations of a single cell at the 32nm technology node, using ITRS 2009 projections on MTJ parameters (Table 1),

[1]The MTJ is grown above the source and drain regions of the access transistor and is typically much smaller than the transistor itself; consequently, the size of the access transistor determines cell area.

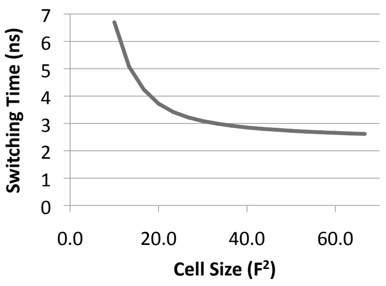

Figure 3: **1T-1MTJ cell switching time as a function of cell size based on Cadence-Spectre circuit simulations at 32nm.**

and the BSIM-4 predictive technology model (PTM) of an NMOS transistor [38]; results presented here are assumed in the rest of this paper whenever cell sizing needs to be optimized for write speed. As the precise value of intrinsic current density J_{c0} is not included in ITRS projections, J_{c0} is conservatively assumed to be zero, which requires a 2× increase in switching current for a 2× increase in switching speed. If feature size is given by F, then at a switching speed of 6.7ns, a 1T-1MTJ cell occupies $10F^2$ area—a 14.6× density advantage over SRAM, which is a $146F^2$ technology. As the access transistor's $\frac{W}{L}$ ratio is increased, its current sourcing capability improves, which reduces switching time to 3.1ns at a cell size of $30F^2$. Increasing the size of the transistor further causes a large voltage drop across the MTJ, which reduces the drain-source voltage of the access transistor and pushes the device into deep triode, and ultimately limits its current sourcing capability. As a result, switching time reaches an asymptote at 2.6ns, which is accomplished at a cell size of $65F^2$.

Parameter	Value
Cell Size	$10F^2$
Switching Current	$50\mu A$
Switching Time	$6.7ns$
Write Energy	$0.3pJ/bit$
MTJ Resistance (R_{LOW}/R_{HIGH})	$2.5k\Omega$ / $6.25k\Omega$

Table 1: **STT-MRAM parameters at 32nm based on ITRS'09 projections.**

2.2 Lookup-Table Based Computing

Field programmable gate arrays (FPGAs) adopt a versatile internal organization that leverages SRAM to store truth tables of logic functions [35]. This not only allows a wide variety of logic functions to be flexibly represented, but also allows FPGAs to be re-programmed almost indefinitely, making them suitable for rapid product prototyping. With technology scaling, FPGAs have gradually evolved from four-input SRAM-based truth tables to five- and six-input tables, named lookup tables (LUT) [7]. This evolution is due to increasing IC integration density—when LUTs are created with higher numbers of inputs, the area they occupy increases exponentially; however, place-and-route becomes significantly easier due to the increased functionality of each LUT. The selection of LUT size is technology dependent; for example, Xilinx Virtex-6 FPGAs use both five- and six-input LUTs, which represent the optimum sizing at the 40nm technology node [35].

This paper leverages an attractive feature of LUT-based computing other than reconfigurability: since LUTs are constructed from memory, it is possible to implement them using a leakage-resistant memory technology such as STT-MRAM, for dramatically reduced power consumption. Like other resistive memories, MRAM dissipates near-zero leakage power in the data array; consequently, power density can be kept under check by reducing the supply voltage with each new technology generation. (Typical MRAM read voltages of 0.1V are reported in the literature [20].) Due to its high write power, the technology is best suited to implementing hardware structures that are read-only or are seldom written. Previous work has explored the possibility of leveraging MRAM to design L2 caches [30,34], but this work is the first to consider the possibility of implementing much of the combinational logic on the chip, as well as microarchitectural structures such as register files and L1 caches, using STT-MRAM.

3. FUNDAMENTAL BUILDING BLOCKS

At a high-level, an STT-MRAM based resistive microprocessor consists of storage-oriented resources such as register files, caches, and queues; functional units and other combinational logic elements; and pipeline latches. Judicious partitioning of these hardware structures between CMOS and STT-MRAM is critical to designing a well-balanced system that exploits the unique area, speed, and power advantages of each technology. Making this selection correctly requires analyzing two broad categories of MRAM-based hardware units: those leveraging RAM arrays (queues, register files, caches), and those leveraging look-up tables (combinational logic, functional units).

3.1 RAM Arrays

Large SRAM arrays are commonly organized into hierarchical structures to optimize for area, speed, and power tradeoffs [3]. An array comprises multiple independent banks that can be simultaneously accessed through separate address and data busses to improve throughput. To minimize wordline and bitline delays and to simplify decoding complexity, each bank is further divided into subbanks sharing address and data busses; unlike the case of banks, only a single subbank can be accessed at a time (Figure 4). A subbank consists of multiple independent mats sharing an address line, each of which supplies a different portion of a requested data block on every access. Internally, each mat comprises multiple subarrays. Memory cells within each subarray are organized as $rows \times columns$; a decoder selects the cells connected to the relevant wordline, whose contents are driven onto a set of bitlines to be muxed and sensed by column sensing circuitry; the sensed value is routed back to the data bus of the requesting bank through a separate reply network. Different organizations of a fixed-size RAM array into different numbers of banks, subbanks, mats, and subarrays yield dramatically different area, speed, and power figures [22].

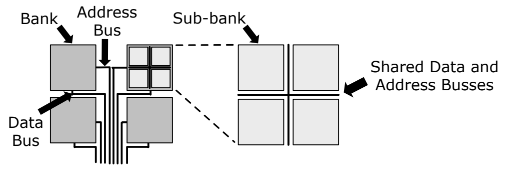

Figure 4: **Illustrative example of a RAM array organized into a hierarchy of banks and subbanks [22].**

MRAM and SRAM arrays share much of this high-level structure with some important differences arising from the size of a basic cell, from the loading on bitlines and wordlines, and from the underlying sensing mechanism. In turn, these differences result in different leakage power, access energy, delay, and area figures. Since STT-MRAM has a smaller cell size than SRAM ($10F^2$ vs $146F^2$), the length of the bitlines and wordlines within a subarray can be made shorter, which reduces bitline and wordline capacitance and resistance, and improves both delay and energy. In addition, unlike the case of 6T-SRAM where each cell has two access transistors, a 1T-1MTJ cell has a single access device whose size is typically smaller than the SRAM access transistor; this reduces the amount of gate capacitance on wordlines, as well as the drain capacitance attached to bitlines, which lowers energy and delay. The smaller cell size of STT-MRAM implies that subarrays can be made smaller, which shortens the global H-tree interconnect that is responsible for a large share of the overall power, area, and delay. Importantly, unlike the case of SRAM where each cell comprises a pair of cross-coupled inverters connected to the supply rail, STT-MRAM does not require constant connection to Vdd within a cell, which reduces leakage power within the data array to virtually zero.

Handling Long-Latency Writes. Despite these advantages, STT-MRAM suffers from relatively long write latencies compared to SRAM (Section 2.1). Leveraging STT-MRAM in designing frequently accessed hardware structures requires (1) ensuring that critical reads are not delayed by long-latency writes, and (2) long write latencies do not result in resource conflicts that hamper pipeline throughput.

One way of accomplishing both of these goals would be to choose a heavily multi-ported organization for frequently written hardware structures. Unfortunately, this results in an excessive number of ports, and as area and delay grow with port count, hurts performance significantly. For example, building an STT-MRAM based architectural register file that would support two reads and one write per cycle with fast, $30F^2$ cells at 32nm, 4GHz would require two read ports and 13 write ports, which would increase total port count from 3 to 15. An alternative option would be to go to a heavily multi-banked implementation without incurring the overhead of extreme multiporting. Unfortunately, as the number of banks are increased, so does the number of H-tree wiring resources, which quickly overrides the leakage and area benefits of using STT-MRAM.

Instead, this paper proposes an alternative strategy that allows high write throughput and read-write bypassing without incurring an increase in the wiring overhead. The key idea is to allow long-latency writes to complete locally within each sub-bank without unnecessarily occupying global H-tree wiring resources. To make this possible, each subbank is augmented with a *subbank buffer*—an array of flip-flops (physically distributed across all mats within a subbank) that latch in the data-in and address bits from the H-tree, and continue driving the subarray data and address wires throughout the duration of a write while bank-level wiring resources are released (Figure 5). In RAM arrays with separate read and write ports, subbank buffers drive the write port only; reads from other locations within the array can still complete unobstructed, and it also becomes possible to read the value being written to the array directly from the subbank buffer.

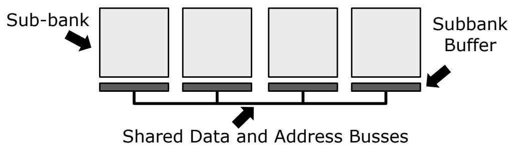

Figure 5: Illustrative example of subbank buffers.

Subbank buffers also make it possible to perform differential writes [18], where only bit positions that differ from their original contents are modified on a write. For this to work, the port attached to the subbank buffer must be designed as a read-write port; when a write is received, the subbank buffer (physically distributed across the mats) latches in the new data and initiates a read for the original contents. Once the data arrives, the original contents and the new contents are bitwise XOR'ed to generate a mask indicating those bit positions that need to be changed. This mask is sent to all relevant subarrays as the enable signals for the bitline drivers along with the actual data—in this way, it becomes possible to perform differential writes without incurring additional latency and energy on the global H-tree wiring. Differential writes can reduce the number of bit flips, and thus write energy, by significant margins, and can make STT-MRAM based implementation of heavily written arrays practical.

3.2 Lookup Tables

Although large STT-MRAM arrays dissipate near-zero leakage power in the subarrays, the leakage power of the peripheral circuitry can be appreciable and in fact dominant as the array size is reduced. With smaller arrays, opportunities to share sense amplifiers and decoding circuitry across multiple rows and multiple columns is significantly lower. One option to combat this problem would be to utilize very large arrays to implement lookup tables of logic functions; unfortunately, both access time and the area overhead deteriorate with larger arrays.

Rather than utilizing an STT-MRAM array to implement a logic function, we rely on a specialized STT-MRAM based lookup table employing differential current-mode logic (DCML). Recent work in this area has resulted in fabricated, two-input lookup tables [8] at 140nm, as well as a non-volatile full-adder prototype [26]. Figure 6 depicts an example three-input LUT. The circuit needs both complementary and pure forms of each of its inputs, and the LUT produces complementary outputs—when multiple LUTs are cascaded in a large circuit, there is no need to generate additional complementary outputs.

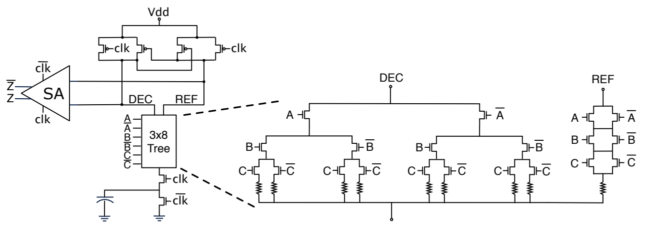

Figure 6: Illustrative example of a three-input lookup table.

This LUT circuit, an expanded version of what is proposed in [8], utilizes a dynamic current source by charging and discharging the capacitor shown in Figure 6. The capacitor is discharged during the $\overline{clk}$ phase, and sinks current through the 3×8 decode tree during the clk phase. Keeper PMOS transistors charge the two entry nodes of the sense amplifier (SA) during the $\overline{clk}$ phase and sensing is performed during the clk phase. These two entry nodes, named DEC and REF, reach different voltage values during the sensing phase (clk) since the sink paths from DEC to the capacitor vs. from REF to the capacitor exhibit different resistances. The reference MTJ needs to have a resistance between the low and high resistance values; since ITRS projects R_{LO} and R_{HIGH} values of $2.5k\Omega$ and $6.25k\Omega$ at 32nm, $4.375k\Omega$ is chosen for R_{REF}.

Although the MTJ decoding circuitry is connected to Vdd at the top and dynamically connected to GND at the bottom, the voltage swing on the capacitor is much smaller than Vdd, which dramatically reduces access energy. The output of this current mode logic operation is fed into a sense amplifier, which turns the low-swing operation into a full-swing complementary output.

In [8], it is observed that the circuit can be expanded to higher numbers of inputs by expanding the decode tree. However, it is important to note that expanding the tree beyond a certain height reduces noise margins and makes the LUT circuit vulnerable to process variations, since it becomes increasingly difficult to detect the difference between high and low MTJ states due to the additional resistance introduced by the transistors in series. As more and more transistors are added, their cumulative resistance can become comparable to MTJ resistance, and fluctuations among transistor resistances caused by process variations can make sensing challenging.

3.2.1 *Optimal LUT Sizing for Latency, Power, and Area*

Both the power and the performance of a resistive processor depend heavily on the LUT sizes chosen to implement combinational logic blocks. This makes it necessary to develop a detailed model to evaluate latency, area, and power tradeoffs as a function of STT-MRAM LUT size. Figure 7 depicts read energy, leakage power, read delay, and area as a function of the number of LUT inputs. LUTs with two to six inputs (4-64 MTJs) are studied, which represent realistic LUT sizes for real circuits. As a comparison, only five- and six-input LUTs are utilized in modern FPGAs (e.g., Xilinx Virtex 6) as higher sizes do not justify the increase in latency and area for the marginal gain in flexibility when implementing logic functions. As each LUT stores only one bit of output, multiple LUTs are accessed in parallel with the same inputs to produce multi-bit results (e.g., a three-bit adder that produces a four-bit output).

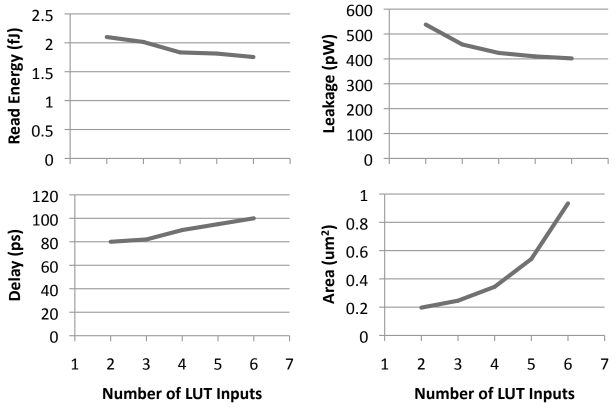

Figure 7: **Access energy, leakage power, read delay, and area of a single LUT as a function of the number of LUT inputs based on Cadence-Spectre circuit simulations at 32nm.**

Read Energy. Access energy decreases slightly as LUT sizes are increased. Although there are more internal nodes—and thus, higher gate and drain capacitances–to charge with each access on a larger LUT, the voltage swing on the footer capacitor is lower due to the increased series resistance charging the capacitor. As a design choice, it is possible to size up the transistors in the decode tree to trade off power against latency and area. The overall access energy goes down from 2fJ to 1.7fJ as LUT size is increased from two to six for the minimum-size transistors used in these simulations.

Leakage Power. Possible dominant leakage paths for the LUT circuit are: (1) from Vdd through the PMOS keeper transistors into the capacitor, (2) from Vdd through the footer charge/discharge NMOS to GND, and (3) the sense amplifier. Lower values of leakage power are observed at higher LUT sizes due to higher resistance along leakage paths (1) and (2), and due to the stack effect of the transistors in the 3×8 decode tree. However, similarly to the case of read energy, sizing the decoder transistors appropriately to trade-off speed against energy can change this balance. As LUT size is increased from two to six inputs, leakage power reduces from 550pW to 400pW.

Latency. Due to the increased series resistance of the decoder's pull-down network with larger LUTs, the RC time constant associated with charging the footer capacitor goes up, and latency increases from 80 to 100ps. However, LUT speed can be increased by sizing the decoder transistors higher, at the expense of larger area, and a higher load capacitance for the previous stage driving the LUT. For optimal results, the footer capacitor must also be sized appropriately. A higher capacitance allows the circuit to work with a lower voltage swing at the expense of increased area. Lower capacitance values cause higher voltage swings on the capacitor, thereby slowing down the reaction time of the sense amplifier due to the lower potential difference between the DEC and REF nodes. A 50fF capacitor was used in these simulations.

Area. Although larger LUTs amortize the leakage power of the peripheral circuitry better, and offer more functionality without incurring a large latency penalty, the area overhead of the lookup table increases exponentially with the number of inputs. Every new input doubles the number of transistors in the branches; as LUT size is increased from two to six inputs, the area of the LUT increases fivefold. Nevertheless, a single LUT can replace approximately 12 CMOS standard cells on average when implementing such complex combinational logic blocks as a floating-point unit (Section 4.5) or a memory controller's scheduling logic (Section 4.6.4); consequently, analyses shown later in the paper assume six-input LUTs unless otherwise stated.

3.2.2 *Case Study: Three-bit Adder using Static CMOS, ROM, and STT-MRAM LUT Circuits*

To study the power and performance advantages of STT-MRAM LUT-based computing on a realistic circuit, Table 2 compares access energy, leakage power, area, and delay figures obtained on three different implementations of a three-bit adder: (1) a conventional, static CMOS implementation, (2) a LUT-based implementation using the STT-MRAM (DCML) LUTs described in Section 3.2, and (3) a LUT-based implementation using conventional, CMOS-based static ROMs. Minimum size transistors are used in all three cases to keep the comparisons fair. Circuit simulations are performed using Cadence AMS (Spectre) with Verilog-based test vector generation; we use 32nm BSIM-4 predictive technology models (PTM) [38] of NMOS and PMOS transistors, and the MTJ parameters presented in Table 1 based on ITRS'09 projections. All results are obtained under identical input vectors, minimum transistor sizing, and a 370K temperature. Although simulations were also performed at $16nm$ and $22nm$ nodes, results showed similar tendencies to those presented here, and are not repeated.

Parameter	STT-MRAM LUT	Static CMOS	ROM-Based LUT
Delay	100ps	110ps	190ps
Access Energy	7.43fJ	11.1fJ	27.4fJ
Leakage Power	1.77nW	10.18nW	514nW
Area	$2.40\mu m^2$	$0.43\mu m^2$	$17.9\mu m^2$

Table 2: **Comparison of three-bit adder implementations using STT-MRAM LUTs, static CMOS, and a static CMOS ROM. Area estimates do not include wiring overhead.**

Static CMOS. A three-bit CMOS ripple-carry adder is built using one half-adder (HAX1) and two full-adder (FAX1) circuits based on circuit topologies used in the OSU standard cell library [29]. Static CMOS offers the smallest area among all three designs considered, since the layout is highly regular and only 70 transistors are required instead of 348, which is the case of the STT-MRAM LUT-based design. Leakage is 5.8× higher than MRAM since the CMOS implementation has a much higher number of leakage paths than an STT-MRAM LUT, whose subthreshold leakage is confined to its peripheral circuitry.

STT-MRAM LUTs. A three-input half-adder requires four STT-MRAM LUTs, one for each output of the adder (three sum bits plus a carry-out bit). Since the least significant bit of the sum depends only on two bits, it can be calculate using a two-input LUT. Similarly, the second bit of the sum depends on a total of four bits, and can be implemented using a four-input LUT. The most significant bit and the carry-out bit each depend on six bits, and each of them requires a six-input LUT. Although results presented here are based on unoptimized, minimum-size STT-MRAM LUTs, it is possible to slow down the two- and four-input LUTs to save access energy by sizing their transistors. The results presented here are conservative compared to this best-case optimization scenario.

An STT-MRAM based three-bit adder has 1.5× lower access energy than its static CMOS counterpart due to its energy-efficient, low-swing, differential current-mode logic implementation; however, these energy savings are achieved at the expense of a 5.6× increase in area. In a three-bit adder, a six-input STT-MRAM LUT replaces three CMOS standard cells. Area overhead can be expected to be lower when implementing more complex logic functions that require the realization of many minterms, which is when LUT-based computation is most beneficial; for instance, a single six-input LUT is expected to replace 12 CMOS standard cells on average when implementing the FPU (Section 4.5) and the memory controller scheduling logic (Section 4.6.4).

The most notable advantage of the STT-MRAM LUT over static CMOS is the 5.8× reduction in leakage. This is due to the significantly smaller number of leakage paths that are possible with an STT-MRAM LUT, which exhibits subthreshold leakage only through its peripheral circuitry. The speed of the STT-MRAM LUT is similar to static CMOS: although CMOS uses higher-speed standard cells, an STT-MRAM LUT calculates all four bits in parallel using independent LUTs.

CMOS ROM-Based LUTs. To perform a head-on comparison against a LUT-based CMOS adder, we build a 64×4 static ROM circuit that can read all three bits of the sum and the carry-out bit with a single lookup. Compared to a 6T-SRAM based, reconfigurable LUT used in an FPGA, a ROM-based, fixed-function LUT is more energy efficient, since each table entry requires either a single transistor (in the case of a logic 1) or no transistors at all (in the case of a logic 0), rather than the six transistors required by an SRAM cell. A 6-to-64 decoder drives one of 64 wordlines, which activates the transistors on cells representing a logic 1. A minimum sized PMOS pull-up transistor and a skewed inverter are employed to sense the stored logic value. Four parallel bitlines are used for the four outputs of the adder, amortizing dynamic energy and leakage power of the decoder over four output bits.

The ROM-based LUT dissipates 290× higher leakage than its STT-MRAM based counterpart. This is due to two factors: (1) transistors in the decoder circuit of the ROM represent a significant source of subthreshold leakage, whereas the STT-MRAM LUT uses differential current-mode logic, which connects a number of access devices in series with each MTJ on a decode tree, without any direct connections between the access devices and Vdd, and (2) the ROM-based readout mechanism suffers from significant leakage paths within the data array itself, since all unselected devices represent sneak paths for active leakage during each access. The access energy of the ROM-based LUT is 3.7× higher than the STT-MRAM LUT, since (1) the decoder has to be activated with every access, and (2) the bitlines are charged to Vdd and discharged to GND using full-swing voltages, whereas the differential current-sensing mechanism of the STT-MRAM LUT operates with low-swing voltages.

The ROM-based LUT also runs 1.9× slower than its STT-MRAM based counterpart due to the serialization of the decoder access and cell readout: the input signal has to traverse through the decoder to activate one of the wordlines, which then selects the transistors along that wordline. Two thirds of the delay is incurred in the decoder. Overall, the ROM-based LUT delivers the worst results on all metrics considered due to its inherently more complex and leakage-prone design.

3.2.3 *Deciding When to Use LUTs*

Consider a three-bit adder which has two three-bit inputs and four one-bit outputs. This function can be implemented using four six-input LUTs, whereas the VLSI implementation requires only three standard cells, resulting in a $\frac{stdcell}{LUT}$ ratio of less than one. On the other hand, an unsigned multiplier with two three-bit inputs and a six-bit output requires six six-input LUTs or 36 standard cells, raising the same ratio to six. As the size and complexity of a Boolean function increases, thereby requiring more minterms after logic minimization, this ratio can be as high as 12 [5]. This is due not only to the increased complexity of the function better utilizing the fixed size of the LUTs, but also to the sheer size of the circuit allowing the boolean minimizer to amortize complex functions over multiple LUTs. As this ratio gets higher, power consumption and leakage advantage of LUT based circuits improve dramatically. This observation that LUT-based implementations work significantly better for large and complex circuits is one of our guidelines for choosing which parts of a microprocessor should be implemented using LUTs vs. conventional CMOS.

4. STRUCTURE AND OPERATION OF AN STT-MRAM BASED CMT PIPELINE

Figure 8 shows how hardware resources are partitioned between CMOS and STT-MRAM in an example CMT system with eight single-issue in-order cores, and eight hardware thread contexts per core. Whether a resource can be effectively implemented in STT-MRAM depends on both its size and on the expected number of writes it incurs per cycle. STT-MRAM offers dramatically lower leakage and much higher density than SRAM, but suffers from long write latency and high write energy. Large, wire-delay dominated RAM arrays—L1 and L2 caches, TLBs, memory controller queues, and register files—are implemented in STT-MRAM to reduce leakage and interconnect power, and to improve interconnect delay. Instruction and store buffers, PC registers, and pipeline latches are kept in CMOS due to their small size and relatively high write activity. Since LUTs are never written at runtime, they are used to implement such complex combinational logic blocks as the front-end thread selection, decode, and next-PC generation logic, the floating-point unit, and the memory controller's scheduling logic.

An important issue that affects both power and performance for caches, TLBs, and register files is the size of a basic STT-MRAM cell used to implement the subarrays. With $30F^2$ cells, write latency can be reduced by 2.2× over $10F^2$ cells (Section 2.1) at the expense of lower density, higher read energy, and longer read latency. Lookup tables are constructed from dense, $10F^2$ cells as they are never written at runtime. The register file and the L1 d-cache use $30F^2$ cells with 3.1ns switching time as the 6.7ns write occupancy of a $10F^2$ cell has a prohibitive impact on throughput. The L2 cache and the memory controller queues are implemented with $10F^2$ cells and are optimized for density and power rather than write speed; similarly, TLBs and the L1 i-cache are implemented using $10F^2$ cells due to their relatively low miss rate, and thus, low write probability.

4.1 Instruction Fetch

Each core's front-end is quite typical, with a separate PC register and an eight-deep instruction buffer per thread. The i-TLB, i-cache, next-PC generation logic, and front-end thread selection logic are shared among all eight threads. The i-TLB and the i-cache are built using STT-MRAM arrays; thread selection and next-PC generation logic are implemented with STT-MRAM LUTs. Due to their small size and high write activity, instruction buffers and PC registers are left in CMOS.

4.1.1 *Program Counter Generation*

Each thread has a dedicated, CMOS-based PC register. To compute the next sequential PC with minimum power and area overhead, a special 6×7 "add one" LUT is used rather than a general-purpose adder LUT. A 6×7 LUT accepts six bits of the current PC plus a carry-in bit to calculate the corresponding six bits of the next PC and a carry-out bit; internally, the circuit consists of two-, three-, four-, five-, and six-input LUTs (one of each), each of which computes a different bit of the seven bit output in parallel.

The overall next sequential PC computation unit comprises five such 6×7 LUTs arranged in a carry-select configuration (Figure 9). Carry out bits are used as the select signals for a chain of CMOS-based multiplexers that choose either the new or the original six bits of the PC. Hence, the delay of the PC generation logic is four multiplexer delays, plus a single six-input LUT delay, which comfortably fits

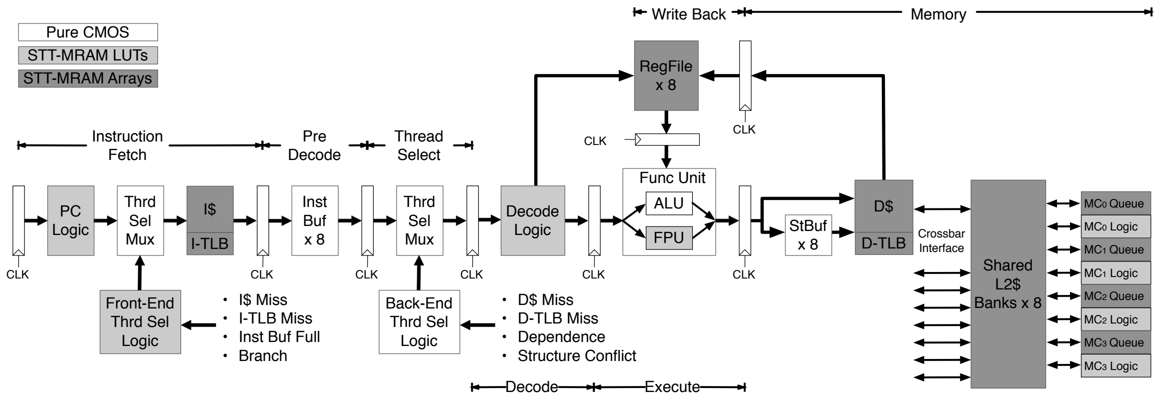

Figure 8: Illustrative example of a resistive CMT pipeline.

within a 250ps clock period in PTM-based circuit simulations (Section 6).

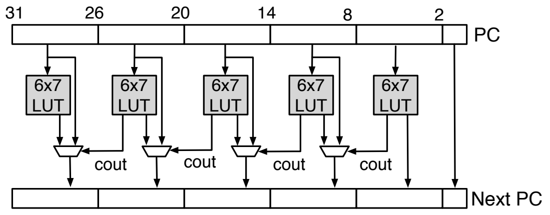

Figure 9: Next PC generation using five add-one LUTS in a carry-select configuration.

4.1.2 *Front-End Thread Selection*

Every cycle, the front-end selects one of the available threads to fetch in round-robin order, which promotes fairness and facilitates a simple implementation. The following conditions make a thread unselectable in the front-end: (1) an i-cache or an i-TLB miss, (2) a full instruction buffer, or (3) a branch or jump instruction. On an i-cache or an i-TLB miss, the thread is marked unselectable for fetch, and is reset to a selectable state when the refill of the i-cache or the i-TLB is complete. To facilitate front-end thread selection, the ID of the last selected thread is kept in a three-bit CMOS register, and the next thread to fetch from is determined as the next available, ublocked thread in round-robin order. The complete thread selection mechanism thus requires an 11-to-3 LUT, which is built from 96 six-input LUTs sharing a data bus with tri-state buffers—six bits of the input are sent to all LUTs, and the remaining five bits are used to generate the enable signals for all LUTs in parallel with the LUT access. (It is also possible to optimize for power by serializing the decoding of the five bits with the LUT access, and by using the enable signal to control the LUT clk input.)

4.1.3 *L1 Instruction Cache and TLB*

The i-cache and and the i-TLB are both implemented in STT-MRAM due to their large size and relatively low write activity. Since writes are infrequent, these resources are each organized into a single subbank to minimize the overhead of the peripheral circuitry, and are built using $10F^2$ cells that reduce area, read energy, and read latency at the expense of longer writes. The i-cache is designed with a dedicated read port and a dedicated write port to ensure that the front-end does not come to a complete stall during refills; this ensures that threads can still fetch from the read port in the shadow of an ongoing write. To accommodate multiple outstanding misses from different threads, the i-cache is augmented with an eight-entry refill queue. When a block returns from the L2 on an i-cache miss, it starts writing to the cache immediately if the write port is available; otherwise, it is placed in the refill queue while it waits for the write port to free up.

Parameter	SRAM (32KB)	STT-MRAM (32KB)	STT-MRAM (128KB)
Read Delay	397ps	238ps	474ps
Write Delay	397ps	6932ps	7036ps
Read Energy	35pJ	13pJ	50pJ
Write Energy	35pJ	90pJ	127pJ
Leakage Power	75.7mW	6.6mW	41.4mW
Area	$0.31mm^2$	$0.06mm^2$	$0.26mm^2$

Table 3: Instruction cache parameters.

It is possible to leverage the 14.6× density advantage of STT-MRAM over SRAM by either designing a similar-capacity L1 i-cache with shorter wire delays, lower read energy, and lower area and leakage, or by designing a higher-capacity cache with similar read latency and read energy under a similar area budget. Table 3 presents latency, power, and area comparisons between a 32KB, SRAM-based i-cache; its 32KB, STT-MRAM counterpart; and a larger, 128KB STT-MRAM configuration that fits under the same area budget [2]. Simply migrating the 32KB i-cache from SRAM to STT-MRAM reduces area by 5.2×, leakage by 11.5×, read energy by 2.7×, and read delay by one cycle at 4GHz. Leveraging the density advantage to build a larger, 128KB cache results in more modest savings in leakage (45%) due to the higher overhead of the CMOS-based peripheral circuitry. Write energy increases by $2.6 - 3.6\times$ over CMOS with 32KB and 128KB STT-MRAM caches, respectively.

4.2 Predecode

After fetch, instructions go through a predecode stage where a set of predecode bits for back-end thread selection are extracted and written into the CMOS-based instruction buffer. Predecode bits indicate if the instruction is a member of the following equivalence classes: (1) a load or a store, (2) a floating-point or integer divide, (3) a floating-point add/sub, compare, multiply, or an integer multiply, (4) a brach or a jump, or (5) any other ALU operations. Each flag is generated by inspecting the six-bit opcode, which requires a total of five six-input LUTs. The subbank ID of the destination register is also extracted and recorded in the instruction buffer during the predecode stage to faciliate back-end thread selection.

4.3 Thread Select

Every cycle, the back-end thread selection unit issues an instruction from one of the available, unblocked threads. The goal is to derive a correct and balanced issue schedule that prevents out-of-order completion; avoids structural hazards and conflicts on L1 d-cache and register file subbanks; maintains fairness; and delivers high throughput.

[2]The experimental setup is described in Section 5.

4.3.1 *Instruction Buffer*

Each thread has a private, eight-deep instruction buffer organized as a FIFO queue. Since buffers are small and are written every few cycles with up to four new instructions, they are implemented in CMOS as opposed to STT-MRAM.

4.3.2 *Back-End Thread Selection Logic*

Every cycle, back-end thread selection logic issues the instruction at the head of one of the instruction buffers to be decoded and executed. The following events make a thread unschedulable: (1) an L1 d-cache or d-TLB miss, (2) a structural hazard on a register file subbank, (3) a store buffer overflow, (4) a data dependency on an ongoing long-latency floating-point, integer multiply, or integer divide instruction, (5) a structural hazard on the (unpipelined) floating-point divider, and (6) the possibility of out-of-order completion.

A load's buffer entry is not recycled at the time the load issues; instead, the entry is retained until the load is known to hit in the L1 d-cache or in the store buffer. In the case of a miss, the thread is marked as unschedulable; when the L1 d-cache refill process starts, the thread transitions to a schedulable state, and the load is replayed from the instruction buffer. On a hit, the load's instruction buffer entry is recycled as soon as the load enters the writeback stage.

Long-latency floating-point instructions and integer multiplies from a single thread can be scheduled back-to-back so long as there are no dependencies between them. In the case of an out-of-order completion possibility—a floating-point divide followed by any other instruction, or any floating-point instruction other than a divide followed by an integer instruction—, the offending thread is made unschedulable for as many cycles as needed for the danger to disappear.

Threads can also become unschedulable due to structural hazards on the unpipelined floating-point divider, on register file subbank write ports, or on store buffers. As the register file is built using $30F^2$ STT-MRAM cells with 3.1ns switching time, the register file subbank write occupancy is 13 cycles at 4GHz. Throughout the duration of an on-going write, the subbank is unavailable for a new write (unless it is the same register that is being overwritten), but the read ports remain available; hence, register file reads are not stalled by long-latency writes. If the destination subbank of an instruction conflicts with an ongoing write to the same bank, the thread becomes unschedulable until the target subbank is available. If the head of the instruction buffer is a store and the store buffer of the thread is full, the thread becomes unschedulable until there is an opening in the store buffer.

In order to avoid starvation, a least recently selected (LRS) policy is used to pick among all schedulable threads. The LRS policy is implemented using CMOS gates.

4.4 Decode

In the decode stage, the six-bit opcode of the instruction is inspected to generate internal control signals for the following stages of the pipeline, and the architectural register file is accessed to read the input operands. Every decoded signal propagated to the execution stage thus requires a six-input LUT. For a typical, five-stage MIPS pipeline [15] with 16 output control signals, 16 six-input LUTs suffice to accomplish this.

4.4.1 *Register File*

Every thread has 32 integer registers and 32 floating-point registers, for a total of 512 registers (2kB of storage) per core. To enable a high-performance, low-leakage, STT-MRAM based register file that can deliver the necessary write throughput and single-thread latency, integer and floating-point register from all threads are aggregated in a subbanked STT-MRAM array as shown in Figure 10. The overall register file consists of 32 subbanks of 16 registers each, sharing a common address bus and a 64-bit data bus. The register file has two read ports and a write port, and the write ports are augmented with subbank buffers to allow multiple writes to proceed in parallel on different subbanks without adding too much area, leakage, or latency overhead (Section 3.1). Mapping each thread's integer and floating-point registers to a common subbank would significantly degrade throughput when a single thread is running in the system, or during periods where only a few threads are schedulable due to L1 d-cache misses. To avert this problem, each thread's registers are are striped across consecutive subbanks to improve throughput and to minimize the chance of a subbank write port conflict. Double-precision floating-point operations require reading two consecutive floating-point registers starting with an even-numbered register, which is accomplished by accessing two consecutive subbanks and driving the 64-bit data bus in parallel.

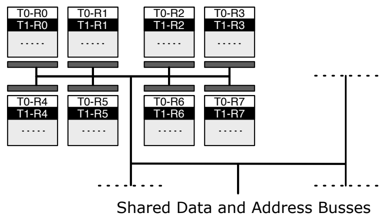

Figure 10: **Illustrative example of a subbanked register file.**

Table 4 lists area, read energy, and leakage power advantages that are possible by implementing the register file in STT-MRAM. The STT-MRAM implementation reduces leakage by 2.4× and read energy by 1.4× over CMOS; however, energy for a full 32-bit write is increased by 22.2×. Whether the end result turns out to be a net power savings depends on how frequently the register file is updated, and on how effective differential writes are on a given workload.

Parameter	SRAM	STT-MRAM
Read Delay	137ps	122ps
Write Delay	137ps	3231ps
Read Energy	0.45pJ	0.33pJ
Write Energy	0.45pJ	10.0pJ
Leakage Power	3.71mW	1.53mW
Area	$0.038mm^2$	$0.042mm^2$

Table 4: **Register file parameters.**

4.5 Execute

After decode, instructions are sent to functional units to complete their execution. Bitwise logical operations, integer addition and subtraction, and logical shifts are handled by the integer ALU, whereas floating-point addition, multiplication, and division are handled by the floating-point unit. Similar to Sun's Niagara-1 processor [17], integer multiply and divide operations are also sent to the FPU rather than a dedicated integer multiplier to save area and leakage power. Although the integer ALU is responsible for 5% of the baseline leakage power consumption, many of the operations it supports (e.g., bitwise logical operations) do not have enough circuit complexity (i.e., minterms) to amortize the peripheral circuitry in a LUT-based implementation. Moreover, operating an STT-MRAM based integer adder (the power- and area-limiting unit in a typical integer ALU [28]) at single-cycle throughput requires the adder to be pipelined in two stages, but the additional power overhead of the pipeline flip-flops largely offsets the benefits of transitioning to STT-MRAM. Consequently, the integer ALU is left in CMOS. The FPU, on the other hand, is responsible for a large fraction of the per-core leakage power and dynamic access energy, and is thus implemented with STT-MRAM LUTs.

Floating-Point Unit. To compare ASIC- and LUT-based implementations of the floating-point unit, an industrial FPU design from Gaisler Research, the GRFPU [5], is taken as a baseline. A VHDL implementation of the GRFPU synthesizes to 100,000 gates on an ASIC design flow, and runs at 250MHz at 130nm; on a Xilinx Virtex-2 FPGA, the unit synthesizes to 8,500 LUTs, and runs at 65MHz. Floating-point addition, subtraction, and multiplication are fully pipelined and execute with a three-cycle latency; floating-point division is unpipelined and takes 16 cycles.

To estimate the required pipeline depth for an STT-MRAM LUT-based implementation of the GRFPU to operate at 4GHz at 32nm, we use published numbers on configurable logic block (CLB) delays on a Virtex-2 FPGA [2]. A CLB has a LUT+MUX delay of 630ps and an interconnect delay of 1 to 2ns based on its placement, which corresponds to a critical path of six to ten CLB delays. For STT-MRAM, we assume a critical path delay of eight LUTs, which represents the average of these two extremes. Assuming a buffered six-input STT-MRAM LUT delay of 130ps and a flip-flop sequencing overhead ($t_{setup} + t_{C\rightarrow Q}$) of 50ps, and conservatively assuming a perfectly-balanced pipeline for the baseline GRFPU, we estimate that the STT-MRAM implementation would need to be pipelined eight times deeper than the original to operate at 4GHz, with floating-point addition, subtraction, and multiplication latencies of 24 cycles, and an unpipelined, 64-cycle floating-point divide latency. When calculating leakage power, area, and access energy, we account for the overhead of the increased number of flip-flops due to this deeper pipeline (flip-flop power, area, and speed are extracted from 32nm circuit simulations of the topology used in the OSU standard cell library [29]). We characterize and account for the impact of loading on an STT-MRAM LUT when driving another LUT stage or a flip-flop via Cadence-Spectre circuit simulations.

To estimate pipeline depth for the CMOS implementation of the GRFPU running at 4GHz, we first scale the baseline 250MHz frequency linearly from 130nm to 32nm, which corresponds to an intrinsic frequency of 1GHz at 32nm. Thus, conservatively ignoring the sequencing overhead, to operate at 4GHz, the circuit needs to be pipelined 4× deeper, with 12-cycle floating-point addition, subtraction, and multiplication latencies, and a 64-cycle, unpipelined floating-point divide. Estimating power for CMOS (100,000 gates) requires estimating dynamic and leakage power for an average gate in a standard-cell library. We characterize the following OSU standard cells using circuit simulations at 32nm, and use their average to estimate power for the CMOS-based GRFPU design: INVX2, NAND2X1, NAND3X1, BUFX2, BUFX4, AOI22X1, MUX2X1, DFFPOSX1, and XNORX1.

Table 5 shows the estimated leakage, dynamic energy, and area of the GRFPU in both pure CMOS and STT-MRAM. The CMOS implementation uses $100,000$ gates whereas the STT-MRAM implementation uses 8,500 LUTs. Although each CMOS gate has lower dynamic energy than a six-input LUT, each LUT can replace 12 logic gates on average. This 12× reduction in unit count results in an overall reduction of the total dynamic energy. Similarly, although each LUT has higher leakage than a CMOS gate, the cumulative leakage of 8,500 LUTs reduces leakage by 4× over the combined leakage of $100,000$ gates. Area, on the other hand, is comparable due to the reduced unit count compensating for the 5× higher area of each LUT and the additional buffering required to cascade the LUTs. (Note that these area estimates do not account for wiring overheads in either the CMOS or the STT-MRAM implementations.) In summary, the FPU is a good candidate to place in STT-MRAM since its high circuit complexity produces logic functions with many minterms that require many CMOS gates to implement, which is exactly when a LUT-based implementation is advantageous.

Parameter	CMOS FPU	STT-MRAM FPU
Dynamic Energy	36pJ	26.7pJ
Leakage Power	259mW	61mW
Area	$0.22mm^2$	$0.20mm^2$

Table 5: **FPU parameters. Area estimates do not include wiring overhead.**

4.6 Memory

In the memory stage, load and store instructions access the STT-MRAM based L1 d-cache and d-TLB. To simplify the scheduling of stores and to minimize the performance impact of contention on subbank write ports, each thread is allocated a CMOS-based, eight-deep store buffer holding in-flight store instructions.

4.6.1 Store Buffers

One problem that comes up when scheduling stores is the possibility of a d-cache subbank conflict at the time the store reaches the memory stage. Since stores require address computation before their target d-cache subbank is known, thread selection logic cannot determine if a store will experience a port conflict in advance. To address this problem, the memory stage of the pipeline includes a CMOS-based, private, eight-deep store buffer per thread. So long as a thread's store buffer is not full, the thread selection logic can schedule the store without knowing the destination subbank. Stores are dispatched into and issued from store buffers in FIFO order; store buffers also provide an associative search port to support store-to-load forwarding, similar to Sun's Niagara-1 processor. We assume relaxed consistency models where special synchronization primitives (e.g., memory fences in weak consistency, or acquire/release operations in release consistency) are inserted into store buffers, and the store buffer enforces the semantics of the primitives when retiring stores and when forwarding to loads. Since the L1 d-cache supports a single write port (but multiple subbank buffers), only a single store can issue per cycle. Store buffers, and the L1 refill queue contend for access to this shared resource, and priority is determined based on a round-robin policy.

4.6.2 L1 Data Cache and TLB

Both the L1 d-cache and the d-TLB are implemented using STT-MRAM arrays. The d-cache is equipped with two read ports (one for snooping, and one for the core) and a write port shared among all subbanks. At the time a load issues, the corresponding thread is marked unschedulable and recycling of the instruction buffer entry holding the load is postponed until it is ascertained that the load will not experience a d-cache miss. Loads search the store buffer of the corresponding thread and access the L1 d-cache in parallel, and forward from the store buffer in the case of a hit. On a d-cache miss, the thread is marked unschedulable, and is transitioned back to a schedulable state once the data arrives. To accommodate refills returning from the L2, the L1 has a 16-deep, CMOS-based refill queue holding incoming data blocks. Store buffers and the refill queue contend for access to the two subbanks of the L1, and are given access using a round-robin policy. Since the L1 is written frequently, it is optimized for write throughput using $30F^2$ cells. L1 subbank buffers perform internal differential writes to reduce write energy.

Parameter	SRAM (32KB)	STT-MRAM (32KB, $30F^2$)	STT-MRAM (64KB, $30F^2$)
Read Delay	344ps	236ps	369ps
Write Delay	344ps	3331ps	3399ps
Read Energy	60pJ	31pJ	53pJ
Write Energy	60pJ	109pJ	131pJ
Leakage Power	78.4mW	11.0mW	31.3mW
Area	$0.54mm^2$	$0.19mm^2$	$0.39mm^2$

Table 6: **L1 d-cache parameters.**

Table 6 compares the power, area, and latency characteristics of two different STT-MRAM based L1 configurations to a baseline, 32KB CMOS implementation. A capacity-equivalent, 32KB d-cache reduces access latency from two clock cycles to one, and cuts down the read energy by 1.9× due to shorter interconnect lengths possible with the density advantage of STT-MRAM. Leakage power is reduced by 7.1×, and area is reduced by 2.8×. An alternative, 64kB configuration requires 72% of the area of the CMOS baseline, but increases capacity by 2×; this configuration takes two cycles to read, and delivers a 2.5× leakage reduction over CMOS.

4.6.3 *L2 Cache*

The L2 cache is designed using $10F^2$ STT-MRAM cells to optimize for density and access energy rather than write speed. To ensure adequate throughput, the cache is equipped with eight banks, each of which supports four subbanks, for a total of 32. Each L2 bank has a single read/write port shared among all subbanks; unlike the L1 d-cache and the register file, L2 subbanks are not equipped with differential writing circuitry to minimize leakage due to the CMOS-based periphery.

Table 7 compares two different STT-MRAM L2 organizations to a baseline, 4MB CMOS L2. To optimize for leakage, the baseline CMOS L2 cache uses high-Vt transistors in the data array, whereas the peripheral circuitry needs to be implemented using low-Vt, high-performance transistors to maintain a 4GHz cycle time. A capacity-equivalent, 4MB STT-MRAM based L2 reduces leakage by 2.0× and read access energy by 63% compared to a CMOS baseline. Alternatively, it is possible to increase capacity to 32MB while maintaining lower area, but the leakage overhead of the peripheral circuitry increases with capacity, and results in twice as much leakage as the baseline.

Parameter	SRAM (4MB)	STT-MRAM (4MB)	STT-MRAM (32MB)
Read Delay	2364ps	1956ps	2760ps
Write Delay	2364ps	7752ps	8387ps
Read Energy	1268pJ	798pJ	1322pJ
Write Energy	1268pJ	952pJ	1477pJ
Leakage Power	6578mW	3343mW	12489mW
Area	$82.33mm^2$	$32.00mm^2$	$70.45mm^2$

Table 7: L2 cache parameters.

4.6.4 *Memory Controllers*

To provide adequate memory bandwidth to eight cores, the system is equipped with four DDR2-800 memory controllers. Memory controller read and write queues are implemented in STT-MRAM using $10F^2$ cells. Since the controller needs to make decisions only every DRAM clock cycle (10 processor cycles in our baseline), the impact of write latency on scheduling efficiency and performance is negligible.

The controller's scheduling logic is implemented using STT-MRAM LUTs. To estimate power, performance, and area under CMOS- and MRAM-based implementations, we use a methodology similar to that employed for the floating-point unit. We use a DDR2-800 memory controller IP core developed by HiTech [11] as our baseline; on an ASIC design flow, the controller synthesizes to $13,700$ gates and runs at 400MHz; on a Xilinx Virtex-5 FPGA, the same controller synthesizes to 920 CLBs and runs at 333MHz. Replacing CLB delays with STT-MRAM LUT delays, we find that an STT-MRAM based implementation of the controller would meet the 400MHz cycle time without further modifications.

Table 8 compares the parameters of the CMOS and STT-MRAM based implementations. Similarly to the case of the FPU, the controller logic benefits significantly from a LUT based design. Leakage power is reduced by 7.2×, while the energy of writing to the scheduling queue increases by 24.4×.

Parameter	CMOS	STT-MRAM
Read Delay	185ps	154ps
Write Delay	185ps	6830ps
Read Energy	7.1pJ	5.6pJ
Write Energy	7.1pJ	173pJ
MC Logic Energy	30.0pJ	1.6pJ
Leakage Power	41.4mW	5.72mW
Area	$0.097mm^2$	$0.051mm^2$

Table 8: Memory controller parameters. Area estimates do not include wiring overhead.

4.7 Write Back

In the write-back stage, an instruction writes its result back into the architectural register file through the write port. No conflicts are possible during this stage since the thread selection logic schedules instructions by taking register file subbank conflicts into account. Differential writes within the register file reduce write power during write backs.

5. EXPERIMENTAL SETUP

We use a heavily modified version of the SESC simulator [25] to model a Niagara-like in-order CMT system with eight cores, and eight hardware thread contexts per core. Table 9 lists the microarchitectural configuration of the baseline cores and the shared memory subsystem.

Processor Parameters	
Frequency	4 GHz
Number of cores	8
Number of SMT contexts	8 per core
Front-end thread select	Round Robin
Back-end thread select	Least Recently Selected
Pipeline organization	Single-issue, in-order
Store buffer entries	8 per thread
L1 Caches	
iL1/dL1 size	32kB/32kB
iL1/dL1 block size	32B/32B
iL1/dL1 round-trip latency	2/2 cycles(uncontended)
iL1/dL1 ports	1 / 2
iL1/dL1 banks	1 / 2
iL1/dL1 MSHR entries	16/16
iL1/dL1 associativity	direct mapped/2-way
Coherence protocol	MESI
Consistency model	Release consistency
Shared L2 Cache and Main Memory	
Shared L2 cache	4MB, 64B block, 8-way
L2 MSHR entries	64
L2 round-trip latency	10 cycles (uncontended)
Write buffer	64 entries
DRAM subsystem	DDR2-800 SDRAM [21]
Memory controllers	4

Table 9: Parameters of baseline.

For STT-MRAM, we experiment with two different design points for L1 and L2 caches: (1) configurations with capacity equivalent to the CMOS baseline, where STT-MRAM enjoys the benefits of lower interconnect delays (Table 10-Small), and (2) configurations with larger capacity that still fit in under same area budget as the CMOS baseline, where STT-MRAM benefits from fewer misses (Table 10-Large). STT-MRAM memory controller queue write delay is set to 27 processor cycles. We experiment with an MRAM-based register file with 32 subbanks and a write delay of 13 cycles each, and we also evaluate the possibility of leaving the register file in CMOS.

To derive latency, power, and area figures for STT-MRAM arrays, we use a modified version of CACTI 6.5 [23] augmented with $10F^2$ and $30F^2$ STT-MRAM cell models. We use BSIM-4 predictive technology models (PTM) of NMOS and PMOS transistors at 32nm, and perform circuit simulations using Cadence AMS (Spectre) mixed signal analyses with Verilog-based input test vectors. Only high performance transistors were used in all circuit simulations. Temperature is set to 370K in all cases, which is a meaningful thermal design point for the proposed processor operating at 4GHz [24].

For structures that reside in CMOS in both the baseline and the proposed architecture (e.g., pipeline latches, store buffers), McPAT [19] is used to estimate power, area, and latency.

6. EVALUATION

6.1 Performance

Figure 11 compares the performance of four different MRAM-based CMT configurations to the CMOS baseline. When the register file is placed in STT-MRAM, and the L1 and L2 cache capacities are made equivalent to CMOS, performance degrades by 11%. Moving the register file to CMOS improves performance, at which point the system achieves 93% of the baseline performance. Enlarging both L1 and L2 cache capacities under the same area budget reduces miss rates but loses the latency advantage of the smaller caches; this configuration outperforms CMOS by 2% on average.

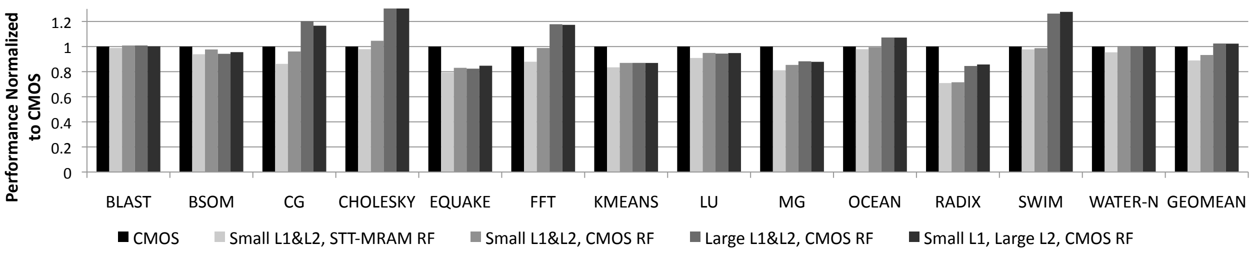

Figure 11: Performance

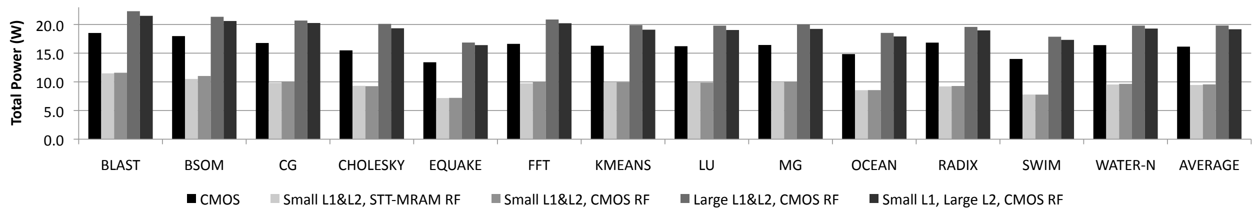

Figure 12: Total Power

	Small	Large
iL1/dL1 size	32kB/32kB	128kB/64kB
iL1/dL1 latency	1/1 cycles	2/2 cycles
L1s write occupancy	13 cycles	13 cycles
L2 size	4MB	32MB
L2 latency	8 cycles	12 cycles
L2 write occupancy	24 cycles	23 cycles

Table 10: STT-MRAM caches parameters

Benchmark	Description	Problem size
Data Mining		
BLAST	Protein matching	12.3k sequences
BSOM	Self-organizing map	2,048 rec., 100 epochs
KMEANS	K-means clustering	18k pts., 18 attributes
NAS OpenMP		
MG	Multigrid Solver	Class A
CG	Conjugate Gradient	Class A
SPEC OpenMP		
SWIM	Shallow water model	MinneSpec-Large
EQUAKE	Earthquake model	MinneSpec-Large
Splash-2 Kernels		
CHOLESKY	Cholesky factorization	tk29.O
FFT	Fast Fourier transform	1M points
LU	Dense matrix division	512 × 512to16 × 16
RADIX	Integer radix sort	2M integers
Splash-2 Applications		
OCEAN	Ocean movements	514×514 ocean
WATER-N	Water-Nsquared	512 molecules

Table 11: Simulated applications and their input sizes.

Optimizing the L2 for fewer misses (by increasing capacity under a the same area budget) while optimizing the L1s for fast hits (by migrating to a denser, STT-MRAM cache with same capacity) delivers similar results.

In general, performance bottlenecks are application dependent. For applications like CG, FFT and WATER, the MRAM-based register file represents the biggest performance hurdle. These applications encounter a higher number of subbank conflicts than others, and when the register file is moved to CMOS, their performance improves significantly. EQUAKE, KMEANS, MG, and RADIX are found sensitive to floating-point instruction latencies as they encounter many stalls due to dependents of long-latency floating-point instructions in the 24-cycle, STT-MRAM based floating-point pipeline. CG, CHOLESKY, FFT, RADIX, and SWIM benefit most from increasing cache capacities under the same area budget as CMOS, by leveraging the density advantage of STT-MRAM.

6.2 Power

Figure 12 compares total power dissipation across all five systems. STT-MRAM configurations that maintain the same cache sizes as CMOS reduce total power by 1.7× over CMOS. Despite their higher performance potential, configurations which increase cache capacity under the same area budget increase power by 1.2× over CMOS, due to the significant amount of leakage power dissipated in the CMOS-based decoding and sensing circuitry in the 32MB L2 cache. Although a larger L2 can reduce write power by allowing for fewer L2 refills and writes to memory controllers' scheduling queues, the increased leakage power consumed by the peripheral circuitry outweighs the savings on dynamic power.

Figure 13 shows the breakdown of leakage power across different components for all evaluated systems. Total leakage power is reduced by 2.1× over CMOS when cache capacities are kept the same. Systems with a large L2 cache increase leakage power by 1.3× due to the CMOS-based periphery. The floating-point units, which consume 18% of the total leakage power in the CMOS baseline, benefit significantly from an STT-MRAM based implementation. STT-MRAM based L1 caches and TLBs together reduce leakage power by another 10%. The leakage power of the memory controllers in STT-MRAM is negligible, whereas in CMOS it is 1.5%.

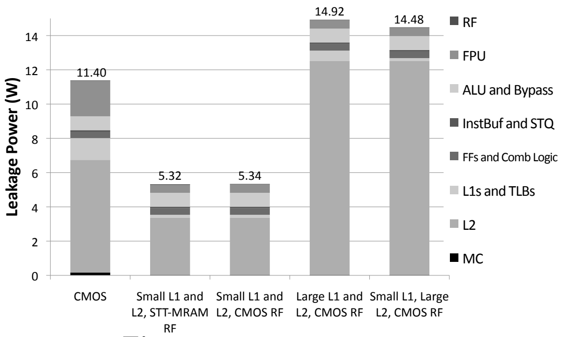

Figure 13: Leakage Power.

7. RELATED WORK

STT-MRAM has received increasing attention in recent years at the device and circuit levels [8,12,26,32,33,37,39]. At the architecture level, Desikan et al. [9] explore using MRAM as a DRAM replacement to improve memory bandwidth and latency. Dong et al. [10] explore 3D-stacked MRAM, and propose a model to estimate the power and area of MRAM arrays. Sun et al. [30] present a read-preemptive write technique which allows an SRAM-MRAM hybrid L2 cache to get performance improvements and power reductions. Zhou et al. [40] apply an early write termination technique at the circuit level to reduce STT-MRAM write energy. Wu et al. [34] propose a data migration scheme to a hybrid cache architecture to reduce the number of writes to resistive memories. Xu et al. [36] propose a circuit technique, which sizes transistors smaller than the worst case size required to generate the switching current to improve density. Most of this earlier work on MRAM considers it as a DRAM or SRAM cache replacement in the system and none of them discusses how to use resistive memories to build combinational logic.

8. CONCLUSIONS

In this paper, we have presented a new technique that reduces leakage and dynamic power in a deep-submicron microprocessor by migrating power- and performance-critical hardware resources from CMOS to STT-MRAM. We have evaluated the power and performance impact of implementing on-chip caches, register files, memory controllers, floating-point units, and various combinational logic blocks using magnetoresistive circuits, and we have explored the critical issues that affect whether a RAM array or a combinational logic block can be effectively implemented in MRAM. We have observed significant gains in power-efficiency by partitioning on-chip hardware resources among STT-MRAM and CMOS judiciously to exploit the unique power, area, and speed benefits of each technology, and by carefully re-architecting the pipeline to mitigate the performance impact of long write latencies and high write power.

We believe this paper is part of an exciting new trend toward leveraging resistive memories in effecting a significant leap in the performance and efficiency of computer systems.

9. ACKNOWLEDGMENTS

The authors would like to thank Yanwei Song, Ravi Patel, Sheng Li, and Todd Austin for useful feedback.

10. REFERENCES

[1] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger. Clock rate vs. IPC: End of the road for conventional microprocessors. In *International Symposium on Computer Architecture*, Vancouver, Canada, June 2000.

[2] ALTERA. Stratix vs. Virtex-2 Pro FPGA performance analysis, 2004.

[3] B. Amrutur and M. Horowitz. Speed and power scaling of SRAMs. 2000.

[4] D. Burger, J. R. Goodman, and A. Kagi. Memory bandwidth limitations of future microprocessors. In *International Symposium on Computer Architecture*, Philedelphia, PA, May 1996.

[5] E. Catovic. GRFPU-high performance IEEE-754 floating-point unit. `http://www.gaisler.com/doc/grfpu_dasia.pdf`.

[6] C. Chappert, A. Fert, and F. N. V. Dau. The emergence of spin electronics in data storage. *Nature Materials*, 6:813–823, November 2007.

[7] M. D. Ciletti. *Advanced Digital Design with the Verilog HDL*. 2004.

[8] D. Suzuki *et al.* Fabrication of a nonvolatile lookup table circuit chip using magneto/semiconductor hybrid structure for an immediate power up field programmable gate array. In *Symposium on VLSI Circuits*, 2009.

[9] R. Desikan, C. R. Lefurgy, S. W. Keckler, and D. C. Burger. On-chip MRAM as a high-bandwidth, low-latency replacement for DRAM physical memories. In *IBM Austin Center for Advanced Studies Conference*, 2003.

[10] X. Dong, X. Wu, G. Sun, H. Li, Y. Chen, and Y. Xie. Circuit and mircoarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Design Automation Conference*, 2008.

[11] HiTech. DDR2 memory controller IP core for FPGA and ASIC. `http://www.hitechglobal.com/IPCores/DDR2Controller.htm`.

[12] Y. Huai. Spin-transfer torque MRAM (STT-MRAM) challenges and prospects. *AAPPS Bulletin*, 18(6):33–40, December 2008.

[13] ITRS. *International Technology Roadmap for Semiconductors: 2009 Executive Summary*. `http://www.itrs.net/Links/2009ITRS/Home2009.htm`.

[14] K. Tsuchida *et al.* A 64Mb MRAM with clamped-reference and adequate-reference schemes. In *Proceedings of the IEEE International Solid-State Circuits Conference*, 2010.

[15] G. Kane. *MIPS RISC Architecture*. 1988.

[16] U. R. Karpuzcu, B. Greskamp, and J. Torellas. The bubblewrap many-core: Popping cores for sequential acceleration. In *International Symposium on Microarchitecutre*, 2009.

[17] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[18] B. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase-change memory as a scalable dram alternative. In *International Symposium on Computer Architecture*, Austin, TX, June 2009.

[19] 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 Computer Architecture*, 2009.

[20] M. Hosomi and H. Yamagishi and T. Yamamoto and K. Bessha *et al.* A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. In *IEDM Technical Digest*, pages 459–462, 2005.

[21] Micron. *512Mb DDR2 SDRAM Component Data Sheet: MT47H128M4B6-25*, March 2006. `http://download.micron.com/pdf/datasheets/dram/ddr2/512MbDDR2.pdf`.

[22] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. Chicago, IL, Dec. 2007.

[23] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. NAS parallel benchmarks. Technical report, NASA Ames Research Center, March 1994. Tech. Rep. RNR-94-007.

[24] U. G. Nawathe, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *IEEE Journal of Solid-State Circuits*, 43(1):6–20, January 2008.

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

[26] S. Matsunaga *et al.* Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions. *Applied Physics Express*, 1(9), 2008.

[27] S. Rusu *et al.* A 45nm 8-Core Enterprise Xeon Processor. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 56–57, Feb. 2009.

[28] Sanu K. Mathew and Mark A. Anders and Brad Bloechel *et al.* A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS. *IEEE Journal of Solid-State Circuits*, 40(1):44–51, January 2005.

[29] J. E. Stine, I. Castellanos, M. Wood, J. Henson, and F. Love. Freepdk: An open-source variation-aware design kit. In *International Conference on Microelectronic Systems Education*, 2007. `http://vcag.ecen.okstate.edu/projects/scells/`.

[30] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel 3D stacked MRAM cache architecture for CMPs. In *High-Performance Computer Architecture*, 2009.

[31] T. Kawahara *et al.* 2 Mb SPRAM (spin-transfer torque RAM) with bit-by-bit bi-directional current write and parallelizing-direction current read. *IEEE Journal of Solid-State Circuits*, 43(1):109–120, January 2008.

[32] T. Kishi and H. Yoda and T. Kai *et al.* Lower-current and fast switching of a perpendicular TMR for high speed and high density spin-transfer-torque MRAM. In *IEEE International Electron Devices Meeting*, 2008.

[33] U. K. Klostermann *et al.* A perpendicular spin torque switching based MRAM for the 28 nm technology node. In *IEEE International Electron Devices Meeting*, 2007.

[34] 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*, 2009.

[35] Xilinx. *Virtex-6 FPGA Family Overview*, November 2009. `http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf`.

[36] W. Xu, Y. Chen, X. Wang, and T. Zhang. Improving STT MRAM storage density through smaller-than-worst-case transistor sizing. In *Design Automation Conference*, 2009.

[37] W. Xu, T. Zhang, and Y. Chen. Spin-transfer torque magnetoresistive content addressable memory (CAM) cell structure design with enhanced search noise margin. In *International Symposium on Circuits and Systems*, 2008.

[38] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45nm design exploration. In *International Symposium on Quality Electronic Design*, 2006. `http://ptm.asu.edu/`.

[39] W. Zhao, C. Chappert, and P. Mazoyer. Spin transfer torque (STT) MRAM-based runtime reconfiguration FPGA circuit. In *ACM Transactions on Embedded Computing Systems*, 2009.

[40] 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*, 2009.