STT-RAM for Shared Memory in GPUs

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Master of Science (Engineering Physics)

by

Prateeksha Satyamoorthy

May 2011

Abstract

In the context of massively parallel processors such as Graphics Processing Units (GPUs), an emerging non-volatile memory – STT-RAM – provides substantial power, area savings, and increased capacity compared to the conventionally used SRAM. The use of highly dense, low static power STT-RAM in processors that run just few threads of execution does not seem attractive because of several times slower write latency, which can relatively impair the performance of the system. However, hundreds to thousands of threads executing in parallel in high-throughput GPUs hide the long write latency of STT-RAM through fine-grained multithreading.

In this thesis, evaluation of possibility of STT-RAM for the shared memory in GPUs was done. Performance, energy and area were evaluated across various configurations of shared memory capacity, banks and ports across a set of benchmarks displaying different characteristics. Results show performance degrades only up to 2% on average for an STT-RAM write latency, which is four times that of SRAM write latency. Performance is even increased by 20% on average when the denser STT-RAM is used to increase shared memory capacity, banks and ports. The energy savings are up to 17% and area savings up to 50%. The evaluation helps understand the trade-offs involved in the use of STT-RAM in GPUs. In the process, a few configurations were identified which encourage their use. To better understand the low impact of high shared memory latency on GPU's performance, a theoretical analysis was done.

Approval Sheet

This thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science (Engineering Physics)

Prateeksha Satyamoorthy

This thesis has been read and approved by the Examining Committee:

Kevin Skadron, Advisor

Sudhanva Gurumurthi, Committee Chair

Stuart Wolf, Curriculum Representative

Accepted for the School of Engineering and Applied Science:

Dean James H. Aylor, School of Engineering and Applied Science

May 2011

Dedicated to my parents

Acknowledgments

My years in graduate school at UVa have made significant impact on me. The learning has been very extensive, whether academics related or life lessons, and is always going to remain with me. Apart from enabling learning, academics and research has been exciting as well as challenging. And I would not have reached this point in graduate studies without the guidance, patience, kindness and just the presence of some wonderful people.

I would like to thank my advisor Prof. Kevin Skadron. He has shown immense patience and has advised me, as I worked my way through graduate school, for which I am extremely grateful. I have learnt a great deal working with him, and feel very enriched with the experience. I am especially very thankful to him for making the time in his busy schedule to provide comments during the writing of this thesis. I am extremely grateful to Prof. Sudhanva Gurumurthi for his valuable advice, specific and timely feedback, for being on my masters committee and making time for weekly meetings, all of which have been crucial for the progress I made. My program and department chairs, Prof. William Soffa, Prof. William Johnson and Prof. Petra Reinke have been very supportive and helpful, which I am very grateful for. I would also like to thank Prof. Stuart Wolf for making the time, and being on my masters committee.

The friends I made at UVa are a big part of my life here, and have been there during fun and rough times, being mentors, offering support, engaging in discussions, or just having fun together. I would like to thank Michael Boyer for always being interested in discussions, pointing the way to good practices, and proof-reading the thesis. I would like to thank Vidyabhushan Mohan and Taniya Siddiqua for the discussions, clarifications and support. I would like to thank Ray Buse for the many discussions, enthusiasm and support. I would like to thank Anurag Nigam for clarifications on STT-RAM. I would also like to take the opportunity to thank LavaLab members Mario Marino,

Brett Meyer, Runjie Zhang, Jiayuan Meng, Liang Wang, Shuai Che, Jeremy Sheaffer and Lukasz Szafaryn.

Outside of the departments and academics, I am deeply thankful to several friends at UVa, and friends from childhood and undergraduate days. I am also grateful to my wonderful family, for constantly believing, encouraging, and supporting me in all my endeavours.

Contents

List of Figures

Glossary

• "An **arithmetic logic unit (ALU)** is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of a processor, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern central processing units and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs." [27]

• Each **bank** in a multi-banked memory is a separate array that can be accessed independent of the others.

• **Benchmarks**, or workloads, are computer programs (applications) used to evaluate the performance of a processor.

• A **bottleneck** is "a phenomenon where the performance or capacity of an entire system is limited by a single or limited number of components or resources." [28]

• A **cache** "is a component that stores data so that future requests for that data can be served faster. The data that is stored within a cache might be values that have been computed earlier or duplicates of original values that are stored elsewhere. If requested data is contained in the cache (cache hit), this request can be served by simply reading the cache, which is comparatively faster. Otherwise (cache miss), the data has to be recomputed or fetched from its original storage location, which is comparatively slower. Hence, the more requests can be served from the cache the faster the overall system performance is." [29]

• A **clock cycle** (or clock period, cycle) is a discrete time intervals which determine when events take place the computer hardware. It is the inverse of clock rate, or frequency, of the processor. [25]

• Processor execution time for a program = Processor clock cycles for a program × Clock cycle time
Clock rate, or frequency = 1 / Clock cycle time

• "**CMOS** refers to both a particular style of digital circuitry design, and the family of processes used to implement that circuitry on integrated circuits (chips)." CMOS process is a method of semiconductor device fabrication which is used to create CMOS (Complementary metal-oxide-semiconductor) integrated circuits (silicon chips). [30]

• The **core** is "the part of the processor that actually performs the reading and executing of instructions. Single-core processors can process only one instruction at a time. A multi-core processor is composed of two or more independent cores." [31]

• **Data-level parallelism (DLP)** is the characteristic of a program where the same set of instructions can be concurrently executed on different data.

• An **execution unit** is usually implemented as a pipeline, which is the splitting "the processing of a computer instruction into a series of independent steps, with storage at the end of each step."

• "A basic **pipeline** is broken into five stages (and units) with a set of registers between each stage:
Instruction fetch Unit fetches the instruction from memory or cache, where it is stored
Instruction decode and register fetch interprets the instruction, and fetches the required data
Execute the operation specified by the instruction is performed
Memory access memory is accessed if needed
Write back either execution unit output, or a value loaded from memory is written into registers." [34]

• "**Ferromagnetism** is the spontaneous magnetization of small regions of a material that exists even in the absence of an external field of induction." [11]

• "**Magnetic anisotropy** is the direction dependence of a material's magnetic properties. A magnetically isotropic material has no preferential direction for its magnetic moment in zero field, while a magnetically anisotropic material will align its moment to an easy axis." [35]

• "The **magnetic moment** is the quantity that determines the torque (force about a pivot) that a magnetic field will exert on it." [36]

• **Magnetization** is the magnetic moment per unit volume.

• **Performance** of a processor depends on the amount of time it takes to execute a particular workload or set of workloads. If processor A executes a workload in lesser time than processor B, then processor A is said to provide better performance.

• **Runtime**, or execution time, of a workload is the time from the start to finish of its execution.

• "In atomic physics, the electron magnetic dipole moment is the magnetic moment of an electron caused by its intrinsic property of **spin**." [33] It is not associated with its orbital motion. [26]

• "In the solution to the Schrodinger equation for the hydrogen atom, three quantum numbers arise from the space geometry of the solution and a fourth arises from electron spin. No two electrons can have an identical set of quantum numbers according to the Pauli exclusion principle , so the quantum numbers set limits on the number of electrons which can occupy a given state. The different quantum numbers: (1) R(r) Principal quantum number, (2) P(θ) Orbital quantum number, (3) F(ϕ) Magnetic quantum number, (4) Spin quantum number" [42]. The two different orientations associated with spin quantum numbers, $+1/2$ and $-1/2$, are called **"spin up"** or **"spin down"**.

• A **thread** is "a separate process with its own instructions and data. A thread may represent a process that is part of a parallel program consisting of multiple processes, or it may represent an independent program on its own." [13]

• **Thread-level parallelism (TLP)** is the characteristic of a program where multiple threads can execute concurrently (i.e. at the same time).

• **Static random access memory (SRAM)** is "a type of semiconductor memory. SRAM does not need to be periodically refreshed. It is volatile in the conventional sense that data is eventually lost when the memory is not powered" [39]. It is a fast, but less dense memory than DRAM, Flash, STT-RAM, used to implement register files, caches, and other on-chip memory.

• **Dynamic random access memory (DRAM)** is "a type of random access memory that stores each bit of data in a separate capacitor within an integrated circuit. Since real capacitors leak charge, the information eventually fades unless the capacitor charge is refreshed periodically. Because of this refresh requirement, it is a dynamic memory as opposed to SRAM and other static memory." [32]

• A **lane** of a processor contains an execution unit and is able to perform one operation per cycle.

• In memory arrays with single **ports**, only one address can be read or written at a time. In multi-ported memory arrays, reads and writes equal to the number of ports on different addressed can be performed at a time.

• **Scratchpad** memory (SPM), "also known as scratchpad, scratchpad RAM or local store in computer terminology, is a high-speed internal memory used for temporary storage of calculations, data, and other work in progress." [38]

• **Scoreboarding** is a method for "dynamically scheduling a pipeline so that the instructions can execute out of order when there are no conflicts and the hardware is available. In a scoreboard, the data dependencies of every instruction are logged. Instructions are released only when the scoreboard determines that there are no conflicts with previously issued and incomplete instructions. If an instruction is stalled because it is unsafe to continue, the scoreboard monitors the flow of executing instructions until all dependencies have been resolved before the stalled instruction is issued." [37]

• A **vector processing unit** is an array of processing units, where one instruction operates on an array of data also called a vector. [40]

Chapter 1

Introduction

One technique of increasing processor performance is scaling CMOS semiconductor technology to smaller processes and increasing frequency, thereby decreasing time taken to process instructions. However, the performance increase, a fraction of original runtime, does not justify the complex, power hungry units used to perform out-of-order execution, super-scalar issue/execution, branch-prediction, speculation, etc. Another technique is to take advantage of the thread-level or data-level parallelism that applications exhibit, by providing the processor with more cores. These cores can be comparatively slower, but can speed up programs by executing many threads in parallel.

Multicore processors (composed of a few cores) and many-core processors (composed of a few hundred cores) like GPUs have demonstrated this point recently by providing more throughput per unit area for lesser power. While multicore processors are well suited for applications exhibiting thread-level parallelism, with several concurrent threads, many-core processors can tremendously boost performance for data-level parallel applications with thousands of threads. Many-core processors like the Graphics Processing Units (GPUs) usually provide up to hundreds of times increase in performance for data-parallel applications than single-core processors[22]. A few types of applications GPUs can provide this kind of performance for, are physics simulations, atmospheric simulations, bioinformatics applications, cryptography applications, image, audio, video processing; graphics, raytracing, computational finance, digital signal processing. These belong to an emerging set of data-parallel applications that are becoming increasing important.

Single-Instruction-Multiple-Data (SIMD) is a class of computers that provide high performance benefits for data-parallel applications, as the same operations are simultaneously applied to different

data at the same time. Based on this taxonomy, the term single-instruction-multiple-thread (SIMT) is used to describe the model of execution in which each data has a thread associated with it, maintaining its own state. Tens of thousands of these threads execute in parallel on the GPU.

Not all data-parallel applications obtain hundreds of times speedup on the GPU. The main hardware limitations causing this are the relatively small (1) on-chip and main memory capacity, compared to the number of cores on-chip, and (2) bandwidth to and from the CPU. For SIMD/SIMT applications with high levels of parallelism, tens of thousands of threads operate on tens of thousands of data elements in a single clock cycle. Here, memory is certainly a bottleneck, as it is limited to around 32 bits per thread every 2 cycles for fast on-chip memories: register and shared memory. For communication among all threads of an application global memory is used, whereas shared memory is used for threads belonging to a block.

An ideal memory system would be of unlimited capacity, and would be fast, providing data immediately when the processor requires it [13]. As applications and the data they operate upon get larger, it becomes impossible to fit them in a memory accessible within one clock cycle. Using a memory hierarchy provides faster access to programs with stronger locality of reference, and is an economical option. The first level of memory being the fastest with the smallest capacity (and expensive), the second slightly slower with a higher capacity, and so on, with the last level of memory being the slowest with the largest capacity (also the cheapest). It is not difficult to see that this trades away performance. To meet the demands of emerging applications one way of surely increasing performance is by increasing on-chip memory.

In increasing on-chip memory capacity one would be increasing die size, and thereby all the costs associated with it. Most on-chip memory is implemented in SRAM, whose memory cell area is $146F^2$ [16]. [1] Increasing the number of SRAM memory cells is expensive, for a relatively small increase in capacity, while dissipating a lot of static power. An alternative memory technology that seems viable is DRAM. While DRAM provides high density with cell size $6F^2$ [16], it is heavy on power consumption. Not only does it need to be refreshed frequently to retain data, it also dissipates a lot of static power.

[1] Where 'F' stands for size of the smallest feature on the die. 'F' is usually also the CMOS process used in manufacturing the die (e.g. 32nm).

Power is important

Another approach to increase memory is scaling to smaller CMOS process. Thus, increasing the number transistors available without increasing the die area. However, as SRAM and DRAM leakage power issues become even more pronounced at deep sub-micron feature sizes. Hence, they are not that suitable for scaling because as technology is scaled down, power per transistor scales slower than transistor size [15], meaning that power density increases.

A new class of memory technology called Non-Volatile Memory (NVM) seems appropriate. NVMs such as Phase-Change Memory (PCM), Magnetoresistive RAM (STT-RAM), NAND and NOR Flash offer higher density i.e. higher capacity per unit feature size than SRAM. In addition, the read latencies/access times are in the order of nanoseconds, comparable to SRAMs. One of the attractive features of NVM technology is its near-zero leakage power.

Moore's Law was initially an observed trend. Chip companies now target doubling the transistors on a chip approximately every two years. This is enabled by scaling down of CMOS processes, hence the reduction of transistor size. Chip manufacturers relied on increasing clock rate and single-thread performance, by making feature sizes smaller according to Moore's Law. Then power dissipation became significant enough that parallelism was used to keep scaling performance.

Power dissipated or consumed by a chip has two components Dynamic Power and Static Power, as shown in Equation (1.1)[18]. We can reduce dynamic power by reducing switching activity of the transistors (A). Reducing leakage power becomes challenging, as it is dissipated as long as power is on [24].

As device size is scaled down, operating voltages scale down (to reduce strength of electric field and dynamic power). Threshold voltages have to decrease too to keep performance, according to Equation (1.2) [18]

$$TotalPower = A \cdot C \cdot V^2 \cdot f + V \cdot I_{leak} \tag{1.1}$$

where A is the number of transistors actively switching, C is the capacitance of the transistors, V is the operating voltage, f is the frequency.

$$f \propto (V - V_{th})^{\alpha}/V \tag{1.2}$$

where V_{th} is the threshold voltage.

Reduction in the threshold voltage (V_{th}) (around even 65mV [24]) exponentially increases leakage current between the source and drain in transistors, even when they are OFF. Hence, leakage power is becoming a major issue in CMOS circuits as technology scales below 90nm [24]. This dissipation caused by leakage current takes place when transistors are active as well as inactive; whereas dynamic power is caused when transistors are active. Reducing power dissipation impacts performance, packaging, reliability, environmental impact, and heat removal costs [41]. One way to reduce on-chip power is to reduce power associated with memory. On-chip SRAM memories have large sections that are idle for relatively long periods of time. Thus, they dissipate considerable amounts of static power [8]. It is not only important to reduce power dissipation, but also power density. This influences cooling of the chip, cost of cooling, (which on insufficient cooling will lead to reliability problems, and ultimately failure of device), and packaging. With all these issues in sight, NVMs like STT-RAM are a great alternative. Not only do they eliminate majority of leakage power loss, they also aid in implementing very fine-grained complete local power shut-down (ultimate power-gating), and for logic-in-memory (where memory is distributed over a plane of logic) [23]. Along with leakage in memory, power density too is reduced, preventing hotspots.

In modern GPUs shared memory capacity is limited. Shared memory is a scratchpad, an addressable memory (like the global or main memory), as fast as registers (when there are no bank conflicts), which is a widely used for optimizing memory access in GPUs. STT-RAM offers the possibility of increasing shared memory capacity without increasing die size and potentially allowing the same block of threads to do more work with more data, or complete work faster by potentially having all (or most of) the required data on the scratchpad. This also allows for effective implementation of double buffering - so that the next thread block to start can be streaming in its data while the current one is computing. It also allows for having more processing elements for same area and shared memory capacity - providing more computing power per unit area.

The hypothesis is that using STT-RAM for shared memory in GPUs can save energy and area, and that GPUs can hide the long write latency of STT-RAM. The contributions of this thesis are (1) evaluation of using STT-RAM for the GPU shared memory in terms of performance, area and energy, and (2) a theoretical analysis of dependence of performance on shared memory access time. Various configurations for shared memory capacity per GPU multiprocessor, the number of banks per shared memory, and the number of ports per bank are explored, and configurations which provide similar or better performance, energy and area than SRAM are identified. It was found that using STT-RAM for shared memory in GPUs provides energy savings up to 17% and area savings up to 50% with performance at 98% to 120% on average, relative to the baseline. It was concluded that GPUs can hide long access latencies of STT-RAM with little performance loss. Performance can be improved by increasing the number of banks or increasing capacity while staying around energy and area budgets specified by the SRAM baseline system. The thesis itself is divided into the following sections and presented in that order: background and related work, methodology, results, and conclusion.

Chapter 2

Background and Related Work

2.1 Graphics Processing Units

Graphic Processing Units (GPUs) have traditionally been specialized co-processors for rendering graphics. At the heart of the architecture is the concept of many cores processing a very high number of threads at any given instance of time i.e., throughput. From being able to compute only graphics specific applications, GPUs have in recent years enabled computing general purpose applications exhibiting high amounts of parallelism. Although GPUs have been used in parallelizing graphic applications, writing and executing non-graphic applications on GPUs became considerably easier with the introduction of frameworks like CUDA and OpenCL. In this thesis we consider the NVIDIA GTX 285 of the GT200 architecture, for evaluation (Figure 2.1).

2.1.1 Architecture and Working of the GPU

CUDA stands for Compute Unified Device Architecture, and it defines the programming model and instruction set architecture for general purpose GPU platform. An application to be executed on the GPU is written using CUDA language extensions to C or Fortran [1]. The GPU is a co-processor, and the execution of an application using the GPU originates on the CPU. When an application running on the CPU arrives at a point requiring the GPU, a call is made to the GPU driver, invoking either data transfer to/from the GPU or a *kernel* launch on the GPU. Kernels are functions of the applications which are executed on the GPU. Each kernel specifies the data and number of threads

[1] The NVIDIA GPU also executes applications written in other languages or API like OpenCL and DirectCompute.

it will use for execution. The data which a kernel operates on has to be explicitly copied from the CPU to the GPU for computation, and copied back from the GPU to the CPU after computation. A kernel usually uses tens of thousands of threads , and the GPU executes the same kernel on all of the threads, and the data is usually different for each thread. This type of architecture is called Single-Instruction-Multiple-Thread (SIMT). SIMT is different from Single-Instruction-Multiple-Data (SIMD) in that SIMT maintains a thread context for each data element. All the threads of a kernel comprise a *grid*, which are further organized into equal sized *thread blocks*. These thread blocks can contain anywhere from 1 to 512 threads. [19] [21]

The GPU comprises a global work distribution unit, a memory hierarchy, and several multi-processing cores called Streaming Multiprocessors (SMs) [1]. The global work distribution unit, or the global block scheduler, is responsible for scheduling thread blocks in a load-balanced manner across all the SMs. The SMs contain eight lightweight, in-order execution units called Streaming Processors (SPs) each, a total of 240 SPs across the GPU. Groups of 32 threads called a *warp* are scheduled for concurrent execution on the eight SPs. A maximum of 32 such warps, or 1024 threads, can concurrently execute on an SM. 30 such SIMT SM cores execute a maximum of 32 x 30 warps concurrently across the entire GPU. The GPU usually has tens of thousands of threads executing on it at any given point of time.

2.1.2 The Memory Hierarchy

GPUs have on-chip memories such as the register file, shared memory and constant cache in an SM. Texture caches shared amongst three SMs and a global L2 (Figure 2.2), and an off-chip DRAM. What is called the global memory that is visible to all threads of a kernel resides in the off-chip DRAM. A thread's local memory is visible to only itself, and this too resides in the off-chip DRAM. Constant memory and texture memory resides in the off-chip DRAM, and is cached by the constant and texture caches respectively. Shared memory is visible to all the threads of a block and resides on the on-chip scratchpad present on each SM. The shared memory has a separate address space which is disjoint from the main memory, as opposed to the single main memory used by global memory, texture and constant caches [19] [21]. Global, local, and shared memory are the read/write memories, while the constant and texture memories are read-only. Global and local memory accesses

[1] There are 30 SMs in GT200, and 15 SMs in Fermi

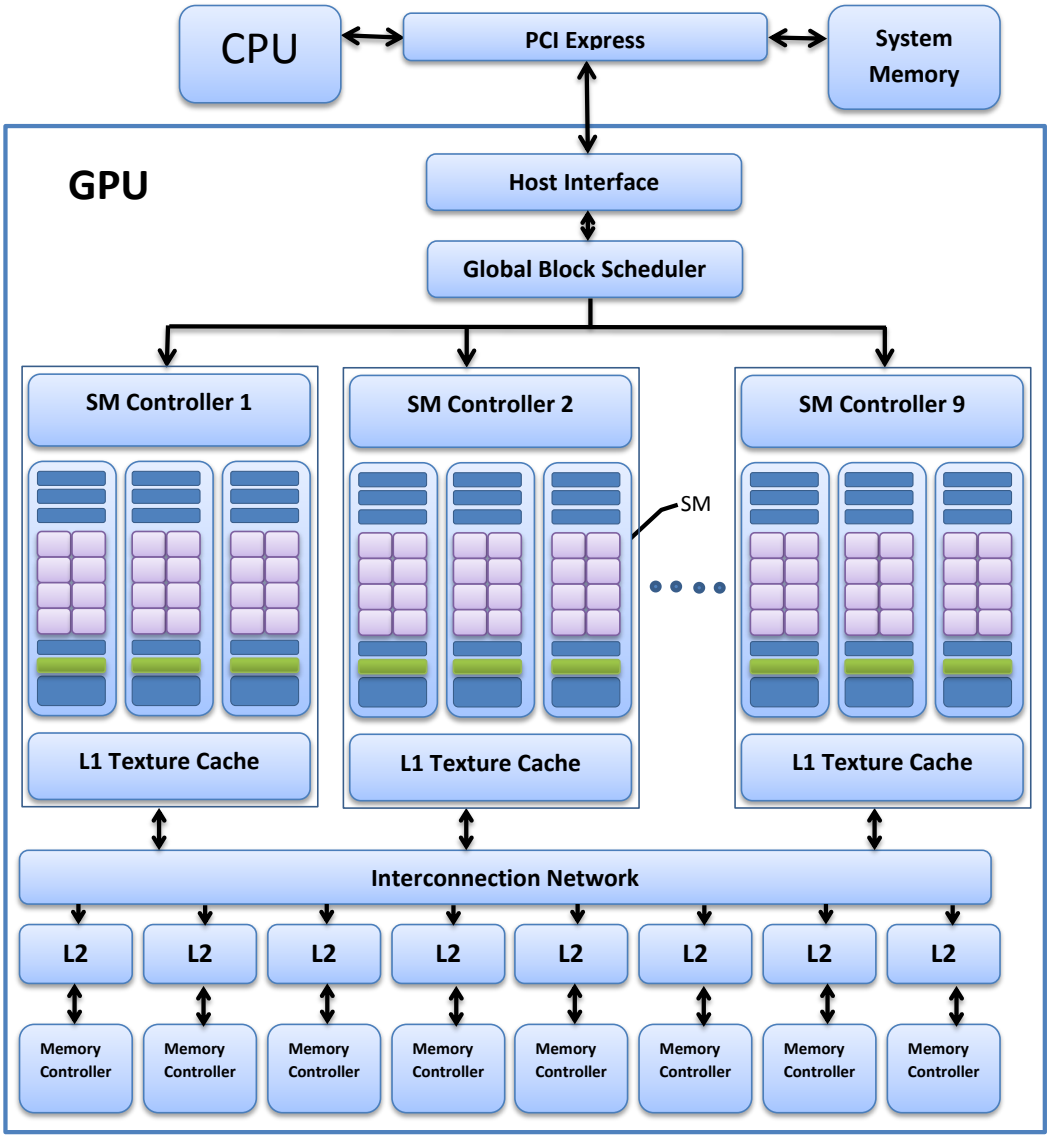


Figure 2.1: The GT200 Architecture. Adapted from Lindholm et al. [19] and Kanter [17]

.

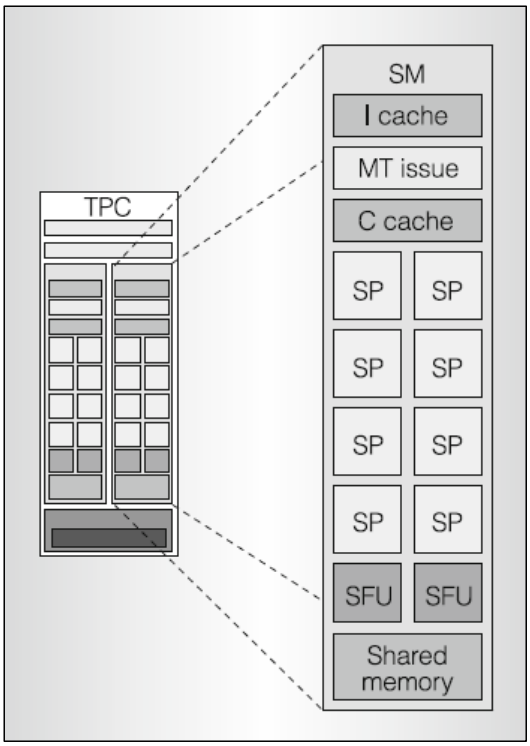


Figure 2.2: Streaming Multiprocessor (SM). Source:Lindholm et al. [19]

.

are 400-600 clock cycles, and are not cached in the GT200 architecture, while being cached in the Fermi architecture[2]. To use global memory bandwidth efficiently, if memory accesses by threads belonging to a half-warp take place, the separate memory accesses are 'coalesced' into a single memory request. [19] [21]

The register file is 64kB in size, and is divided into 16 banks for high concurrent access. The shared memory is 16kB and 48kB in size, in the GT200 and Fermi architectures respectively. It too is divided into 16 banks, and when a warp accesses shared memory, the request is sent in two groups of 16 accesses each [3]. The constant and texture caches are 8kB and 24kB in size, respectively. The L2 cache is 256kB and 768kB in GT200 and Fermi architectures, respectively.

[2]The Fermi architecture was recently introduced. Though its basic architecture is similar to GT200, Fermi has more SPs per SM, providing L1 caches for SMs, and supports concurrent execution of multiple kernels among other differences.

2.1.3 Streaming Multiprocessor

Three SMs comprise a Texture/Processor Cluster [19], also called Thread Processing Cluster, (TPC) [17], and share an SM controller and common texture cache. A single SM consists of eight arithmetic logic units (ALUs) and multiply-add (MAD) units which operate on single precision floating point, one fused multiply-add unit for double-precision floating point, two special-function units which perform transcendental functions, interpolation, reciprocal and other uncommon functions, one multithreaded instruction fetch and issue unit, one instruction cache, one read-only constant cache, and one read/write shared memory. See Figure 2.2.

The SM schedules the warps in a loose round-robin style, i.e. fine-grained multithreading [3]. In fine-grained multithreading every cycle execution switches to a new warp, skipping stalled threads [13]. Hence, whenever there is a long latency memory operation, that warp is taken out of the SM's scheduling pool until the memory operation completes.

The execution units (ALU, MAD, SFU) run at twice the frequency of fetch, issue, registers and shared memory, which run on the slower core clock. To execute each instruction the ALU and MAD units take 4 fast clock cycles, the FPU takes 32+ fast clock cycles, the SFU takes 16-32 fast clock cycles. Together, one ALU and one MAD unit represent a streaming processor (SP). These functional units can be executed in parallel. Although, single and double-precision units share logic, only one can be executing at any particular time.

The SM schedules and executes threads in groups of 32 threads called a warp, on the 8 SPs. A maximum of 8 blocks of threads, or 1024 threads gets scheduled on an SM for execution. All the threads of a warp execute in lock-step executing the same instruction. So, a warp waits until all its threads have finished execution and the next instruction is fetched. [17] [19]

As stated previously, a warp waits till all its threads finish execution. So, a warp waits on branches a particular thread might take. Hence, the divergent paths at branches are serialized. More diverging paths means corresponding decrease in performance.

2.1.4 Shared Memory

The shared memory in each SM is a scratchpad memory. Scratchpads are local memories that are fast and directly addressable, with no tag array. It has the same access latency as the register file

when there are no bank conflicts. The shared memory is divided into 16 banks. Each successive 32-bit word is assigned to a successive bank. Each bank can be accessed simultaneously providing higher memory bandwidth. A memory request consisting of n addresses are mapped to n distinct banks which are accessible simultaneously. Suppose two addresses of a memory request map to the same bank, a bank conflict occurs and the two addresses are now accessed serially, one after the other. Therefore, a memory request is split into as many conflict-free requests as needed. This decreases the bandwidth as many times as the separate requests. As the warp size is 32 and number of banks are 16, shared memory requests of a warp is split into groups two corresponding to the first and second halves of the warp. Hence, bank conflicts do not occur between threads belonging to the different halves of a warp [21].

Having a shared memory enables quicker access to data, which reduces the need for global memory accesses. Intuitively, one would want to keep all of the data required by the threads of an SM on the shared memory and obtain performance benefits. However, scratchpads are only 16-48kB in size. This is one of the main factors limiting the amount of data that can be processed either at any given instance, or by a single thread block [19]. A shared memory variable is only visible to threads belonging to a block. The number of threads per block is therefore restricted by the limited memory resources of a processor core. The number of blocks an SM has in its scheduling pool at any given time is referred to as the *number of active blocks*, [19] depends on how much of resources like registers per thread and shared memory per block is available on the SM.[19].

To use the shared memory, data needs to be copied from global memory to shared memory via explicit instructions. So, data is loaded to registers and then to shared memory. If data computed on shared memory is needed by threads from other thread blocks or by the CPU, it is first copied explicitly to the global memory.

It is desirable to increase the number of active blocks on an SM as it would increase performance by masking latency. Though, it might cause contention in some memory bound workloads. For example, consider a non-memory-bound application using only double precision values: the 16KB shared memory can accommodate at most 2K double precision values. If this could be increased, fewer global memory accesses would be required to obtain data for processing. This implies that the same thread block can now spend more time dedicated to processing. This shows that there exists a need for a higher capacity memory within the fixed area and power constraints.

2.2 STT-RAM

The previous sections discussed how the next generation of GPUs with more number of cores (hence concurrency) need higher memory capacity. The next generations of GPUs will move into deep sub-micron processes in a few years. Therefore, in order to avoid power wall, they need to dissipate and consume lesser power too. This work looks at leakage power in memory, as SRAM's leakage power is a serious issue beyond 32nm.

This need for a dense and lower power memory technology is addressed by a new class of memory technology called Non-Volatile Memory (NVM). NVMs are memories which can retain the data stored in them even when the power source is removed. Many NVMs such as Phase-Change Memory (PCM), Magnetoresistive RAM (STT-RAM), NAND and NOR Flash offer higher density i.e. higher capacity per unit area than SRAM. In addition, the read latencies of these NVMs are in the order of nanoseconds comparable to SRAMs (flash is in microseconds). One of the attractive features of NVMs are its low leakage power. This thesis proposes replacing the SRAM shared memory in GPUs with a magnetoresistive STT-RAM to tackle the issue of providing higher capacity for a given area budget. STT-RAM, or spin transfer torque random access memory, stores the data in form of magnetic moments, and is written and read by passing a spin-polarized current through it. STT-RAM provides increased density, which can be taken advantage of in many ways. The extra capacity provided by STT-RAM, of same area as SRAM, could be used for double buffering where the next thread block to be executed can stream in its data while the current one is computing. Or, instead of increasing the scratchpad size, more processing elements could be accommodated in the same area without increasing the shared memory capacity.

STT-RAM is a magnetoresistive memory that employs Magnetic Tunnel Junctions (MTJs) as memory elements exhibiting two distinct resistances corresponding to 0 and 1 states. MTJs consist of two layers ferromagnetic (FM) material sandwiching a layer of insulating material. One of the two FM layers has its magnetization, or magnetic moment fixed. The other FM layer, called the free layer, can change its magnetization direction. Hence, stores the data bit in form of the orientation of magnetization of relative to a fixed magnetization. The orientation of the changeable magnetization relative to the fixed one can be in two stable states: parallel and anti-parallel, giving two corresponding different resistances. A bit is stored in an MTJ through current-induced

magnetization switching known as the spin transfer torque (STT) effect. The stored data is read by passing a sensing current through the MTJ, indicating its resistance state.

2.2.1 Tunneling Magnetoresistance and the Magnetic Tunnel Junction

The tunneling magnetoresistance (TMR) effect is observed in magnetic tunnel junctions (MTJ). MTJ structure is very similar to the GMR structure. The difference is that MTJ uses an insulating spacer layer (Eg. Al_20_3, MgO), called the tunnel barrier, instead of the metallic one used by GMR structures.

In parallel (P) magnetizations electrons in FM layer will find more empty states to tunnel through the barrier than anti-parallel (AP) magnetizations. Effectively making the resistance of the MTJ low in parallel magnetizations and high in anti-parallel magnetizations [20]. The conducting capability or resistance of the MTJ is defined by the magnetoresistance ratio (MR),

$$MR = \frac{(R_{AP} - R_P)}{R_P} = \frac{\Delta R}{R_P} \tag{2.1}$$

where R_{AP} is the resistance of the MTJ in antiparallel state, R_P is the resitance of the MTJ in parallel state, and ΔR is the difference in resistances of parallel and anti-parallel magnetizations of the FM layers.

The output voltage of the MTJ structure is given by,

$$\Delta V = k.J.\Delta R.A \tag{2.2}$$

where k is the efficiency, J is the current density, A is the cross-sectional area through which current flows. From Equation (2.2) we can see that in order to have a readable output voltage it is necessary to have a high ΔRA, which corresponds to high MR [20]. MTJ structures can have MR as high as 1010% [4].

2.2.2 Current-driven Magnetization Switching

This section explains how the magnetization of the free layer is changed with a spin-polarized current. Consider the case where the magnetizations of layers between two ferromagnetic layers

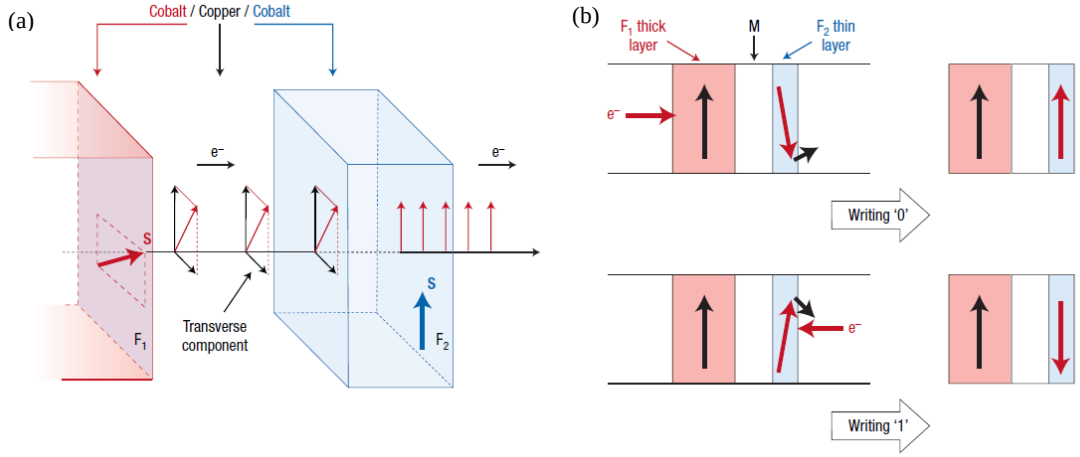


Figure 2.3: (a) Depiction of the spin transfer torque (STT) effect, where the spin polarized current passing from one ferromagnetic layer to another through a spacer layer, switches the magnetization of a second layer through transfer of angular momentum. (b) STT effect applied to the writing an STT-RAM cell. While one direction of current induces parallel state of magnetization in second layer relative to the first, the opposite direction induces a anti-parallel magnetization. The parallel and anti-parallel states corresponding to '0' and '1' data respectively. Source: Chappert et al. [4]

.

(FM1 and FM2) are opposite. When a current of s-orbital electrons are passed from FM1 through to FM2, the FM1 layer acts as a spin polarizer and polarizing the average spin moment of the electrons along its magnetization. Upon reaching layer FM2 the s-d exchange interaction occurs and the average spin moment of the electrons are now directed along the magnetization of layer FM2. This results in the loss of the transverse components of the spin angular momentum of the s-orbital electrons. Since, total angular momentum is conserved in the system (as no external forces are acting), the spin is transferred to the magnetization of layer FM2, causing a torque to act on the FM2 layer aligning its magnetization along the average spin of the s-orbital electrons which is along the magnetization of layer FM1. Thus 'switching' the magnetization of FM2, due to spin transfer torque. The amount of torque per unit area is directly proportional to the density of a current of polarized s-orbital electrons (which cause the spin transfer). This means that the writing current is reduced as cross-sectional area of the structure is reduced [4]. This phenomenon is illustrated in Figure 2.3.

The critical current density needed for magnetization switching at zero temperature [20] is given by,

$$J_{c0} = \frac{2e}{\hbar}\frac{\alpha}{\eta}M_S t_F \left(\pm H_{ext} + H_K + 2\pi M_S - \frac{H_{K\perp}}{2}\right) \tag{2.3}$$

where e is the electron charge, $\hbar$ is the reduced Plank's constant, α is the damping constant, η is the spin transfer efficiency, M_S is the saturation magnetization of the free layer, t_F is the thickness of the free layer, H_{ext} is the externally applied magnetic field, H_K is the in-plane uniaxial anisotropy field and $H_{K\perp}$ is the out-of-plane perpendicular anisotropy field of the free layer. Hence, the corresponding current is given by,

$$I_{c0} = J_{c0}.A \tag{2.4}$$

where J_{c0} is current density and A is the cross-sectional area of the free layer. From Equation (2.3) and (2.4) we can see how the critical current density J_{c0} and current I_{c0} can be reduced. Apart from varying material properties like M_S and η, having a smaller free layer size (A t_F) gives smaller I_{c0}. As thickness of free layer is very small, varying area of free layer has a greater impact on I_{c0}.

2.2.3 Switching Regimes

Based on analytical and numerical estimations three switching modes dependent on current pulse widths have been identified for the free layer in an MTJ structure. *Thermal Activation* - where a long current pulse ($>$ 10ns) is applied cause magnetization switching which is a thermally activated process. This switching process is determined by thermal agitations, and is independent of initial conditions. *Precessional Switching* - where a short current pulse ($<$ 3ns) is applied to cause magnetization switching which is nearly independent of the thermal agitation during the switching process; though, it is dependent on the initial thermal distribution. *Dynamic Reversal* - where current pulses intermediate to that of Thermal Activation and Precessional Switching modes cause dynamic reversal magnetization switching. These current pulses are the speed of operation of practical STT-RAMs. This mode is both precessional and thermally activated process which depends on initial thermal distribution and thermal agitation during the switching process [6].

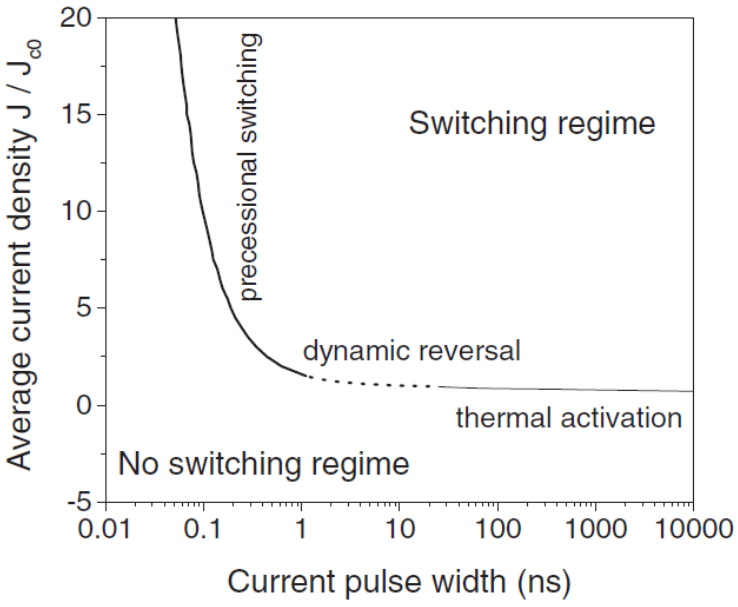


Figure 2.4: Switching modes corresponding to current pulse width: thermal activation, dynamic reversal, and precessional switching. Source: Diao et al. [6]

.

Consulting Figure 2.4 we assume a 5ns current pulse width under dynamic reversal regime for the magnetization switching of the MTJ free layer, as it neither has the very high switching current density of precessional switching, or very long switching speed of thermally activated regime. This is the write speed of solely the MTJ. The extra STT-RAM array access times, address decoding etc, is assumed to be 0.3ns as explained in the Methodology section.

2.2.4 STT-RAM Cell

The process of the spin-transfer-torque based writing of a STT-RAM cell is shown in Figure 2.3. Electrons flowing from the thick 'polarizing' layer to the thin free layer favor a parallel orientation of the magnetizations: if the initial state is antiparallel, then beyond a threshold current density J_C the free layer will switch. When the electrons flow from the free to the polarizing layer, the effective spin moment injected in the free layer is opposed to the magnetization of the polarizing layer, writing an antiparallel configuration beyond the threshold current density.

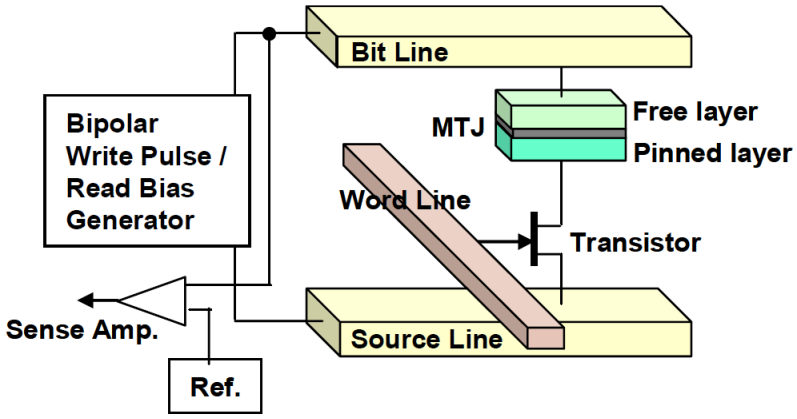


Figure 2.5: The 1T/1MTJ STT-RAM cell, consisting of an MTJ, a transistor to access the MTJ device, and bit line, word line and source line used for reading and writing the device by controlling the access transistor. Source: Hosomi [14]

.

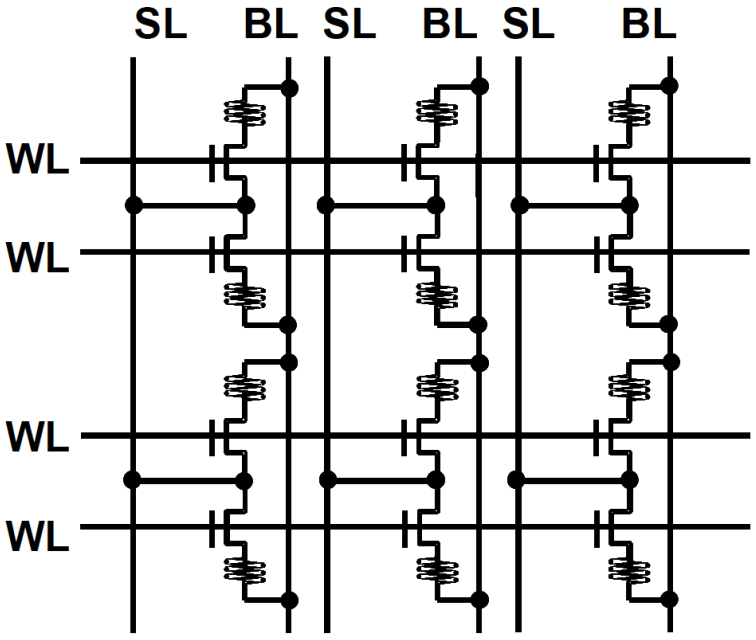


Figure 2.6: Schematic of the STT-RAM cell array. Source: Hosomi [14]

.

2.2.5 Related Work

Guo et al. [12] developed an architectural technique that targets power-efficient, scalable microprocessor design. They use STT-RAM memory technology for this purpose. Their key idea is implementing power and performance critical hardware resources like register files, caches, memory controllers, FPUs using scalable, leakage-resistant RAM arrays and lookup tables (LUTs) designed with STT-RAM. To deal with long write latency of STT-RAM they incorporate write-buffers, which allow the long-latency write to complete locally within each subbank, without going through the H-trees, or the interconnect between elements of the memory array. For STT-RAM used in register files, caches, TLB, and memory controller queues, they modify a standard SRAM array to include buffers for subbanks.

Combinational logic and functional units like front-end thread scheduler, decode, next-PC generation logic, FPU, and memory controller scheduling logic use specialized STT-RAM array based LUTs employing 2-bit differential current-mode logic (DCML) and extend it to 3-input LUT. They develop a detailed model to evaluate latency, area and power trade-offs depending on LUT size. They find that increasing size of the transistors leads to savings in power trading off performance and area.

Their main argument is that a single LUT can replace around 12 CMOS cells, especially while implementing complex combinational logic such as FPUs. Area is really traded off when STT-RAM LUTs are used, by a factor of 5.6x. Though the leakage power savings are 5.8x. They show that power consumption and leakage savings of LUT based circuits improve dramatically as complexity of logic increases. Their evaluation is in the context of FPGAs, which do use LUTs heavily. Hence, traditional processors might not benefit much from STT-RAM LUT implementation.

Their baseline is SRAM for register files, L1 and L2 caches, and incorporates STT-RAM LUTs. For performance evaluation they used different configurations: 1) register file implemented in STT-RAM, L1 and L2 are also in STT-RAM with same capacity as CMOS - this gave 89% baseline performance 2) register file is moved to CMOS - this gave 93% baseline performance 3) L1 and L2 were enlarged keeping the same area budget as CMOS - this gave 1.02% baseline performance 4) for the same area budget as CMOS L2 capacity is increased, and L1 is moved to STT-RAM - this gave 1.02% baseline performance. Their microprocessors are barrel-processors, which employ fine-grain

multithreading. They do not evaluate the contribution of fine-grained multithreading in hiding STT-RAM latency separately. Hence, it is difficult to ascertain how much latency hiding is due to the subbank buffers, and how much due to the fine-grain multithreading nature of the processor they use. In their power evaluation they find that STT-RAM configurations that maintain same cache sizes as CMOS reduce total power by 1.7x (while degrading performance by 7%), and increase in cache capacity under same area budget increase power by 1.2x (while improving performance by 2%).

Dong et al. [9] developed an STT-RAM cache model, and used that to evaluate implementation of different levels of the memory hierarchy in STT-RAM. Based on that model, STT-RAM, SRAM and DRAM are compared for performance, area and energy. Also, 3D stacking of STT-RAM is evaluated. For the design of the STT-RAM cell, they first bound the area of the cell by bounding the current required for the write operation. As critical current will increase exponentially for writing pulses shorter than 10ns, they assume that as MTJ write latency value.They use HSPICE model to determine W/L ratio of the STT-RAM cell driving a current of 216uA, which they obtained by scaling all MTJ-parameters from 180nm technology to 90nm technology. To simplify the model they approximate the MTJ to be a static resistor. They also determine dynamic read/write energies for the STT-RAM cell. The STT-RAM cell values are then used in CACTI to obtain values for STT-RAM arrays of 4MB-16MB.

SRAM, DRAM array values for area, performance and delay are compared to that of STT-RAM. STT-RAM cell area is found to be 25% of SRAM cell area, and 70% more than DRAM cell area. The read latencies for SRAM, DRAM and STT-RAM are found to be very close. SRAM leakage power per unit area is 9.33x of STT-RAM, and DRAM leakage power per unit area is 3.14x of STT-RAM. Then the cache hierarchy design of replacing the SRAM/DRAM L2 with 3D stacked STT-RAM L2 is evaluated.They find that STT-RAM L2 performs 13% better than same size SRAM L2. Though increasing STT-RAM size further leads to dramatic increase in delay as interconnect delay starts to dominate. DRAM was found to perform 10% worse than STT-RAM of same size. According to their simulations, STT-RAM L2 can save 89% more power than SRAM L2 and 70% more than DRAM L2.

Desikan et al. [5] explore STT-RAM as an on chip main memory by 3D stacking it on the processor. This study proposes having STT-RAM on chip as a fast accessible alternative to having

an off-chip main memory whose access latency is hundreds of cycles. They propose STT-RAM as a high-bandwidth low latency technology and do not touch on the latency issue any further. This limits their findings to 3D on-die integration of STT-RAM

STT-RAM in GPUs Al Maashri et al. [2] focus on 3D stacking of texture and Z caches. Hence, they evaluate GPUs in the context of graphics applications and do not really address the use of GPUs for general purpose computing. They address the latency problem by 3D stacking STT-RAM over the GPU. The reasoning behind this being that 3D stacking allows shorter wires and hence reduced latencies. Their primary evaluation identifies texture and Z caches as highest impacted by hit rate when they change the organization of streamer, texture, Z, color write caches in GPUs. They replace all SRAM with STT-RAM to save on leakage power when cache size increase beyond 512kB. Read and write speeds for STT-RAM are not specified. From results there is up to 20% degradation in performance. They note that this is not beneficial to applications which require performance, and is more suitable to power conserving ones. However, they do not explore replacing the SRAM shared memory with an STT-RAM based one.

Chapter 3

Methodology

3.1 Architectural Simulation with GPGPU-Sim

To evaluate implementing the GPU shared memory in STT-RAM, the GPGPU-Sim microarchitectural simulator [3] was chosen. GPGPU-Sim provides a functional and timing model for microarchitecture for GPUs with general compute capability. GPGPU-Sim models timing for the SMs, SPs, on-chip memory, interconnection network, memory controllers, and graphics DRAM. It does not model timing for CPU execution or data transfer to and from the CPU. Hence, GPGPU-Sim just reports the number of cycles applications take to execute on the GPU [1].

GPGPU-Sim models the CUDA parallel thread execution (PTX) instruction set. It can run unmodified CUDA applications, as it uses the CUDA compiler *nvcc* to convert the CUDA application code into host CPU C code and GPU PTX instructions. However, instead of using CUDA API library that interfaces applications with the GPU, the simulator employs a customized library that emulates the CUDA API. The customized version contains stub functions which makes calls to GPGPU-Sim instead of the GPU. [1]

GPGPU-Sim provides many configurable parameters such as number of shader cores, warp size, number of threads, number of registers, shared memory size, constant cache size, texture cache size, DRAM bandwidth and number of access ports [1]. By changing the shared memory capacity, the number of shared memory banks, or access ports, modeling performance for various configurations of shared memory is possible. Hence, identification of favorable STT-RAM shared memory configurations can be done.

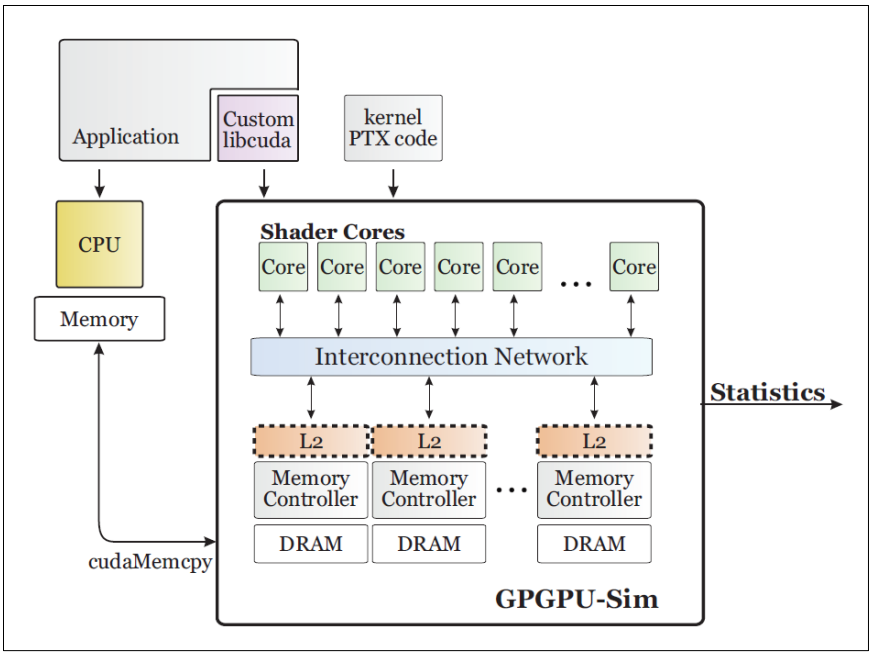


Figure 3.1: GPGPU-Sim Compilation Flow. (GPGPU-Sim uses the term shader cores for SMs.) Source: Bakhoda et al. [3]

3.2 STT-RAM Shared Memory

In this study, the STT-RAM Shared Memory element is taken to be MTJ characterized by Diao et al. [7]. The MTJ implements dual MgO barriers, with a resulting critical switching current density J_{c0} of $1.0MA/cm^2$. This MTJ structure has a TMR of 70%, and dimensions of 120 nm x 240 nm. The STT-RAM Shared Memory is in the 32nm CMOS process.

The three operating regimes identified by Diao et al. [6] are considered for the write operation, to determine switching current J_c, and switching time, or write latency. In the *thermal activation* regime, the switching current density is $J_c = 0.75 \times J_{c0}$, while having a comparatively long switching time t_{sw}, i.e. $t_{sw} > 10$ns. In the *dynamic reversal* regime, the switching current density is $J_c = 1.5 \times J_{c0}$, while having a relatively shorter switching time t_{sw}, i.e. $3\text{ns} < t_{sw} < 10\text{ns}$. In the *precessional switching* regime, the switching current density is $J_c = 5 \times J_{c0}$, while having a very short switching time t_{sw}, i.e. $t_{sw} < 3$ns . In order to keep the switching current as low as possible without having very long write latencies of tens of clock cycles, the MTJ is selected to operate in the *dynamic reversal* switching regime, with $J_c = 1.5MA/cm^2$ and switching time $t_{sw} = 5$ ns. [6] The MTJ's access transistor should have width corresponding to the amount of switching current.

To drive a current of 432μA, the access transistor's size is determined as $6.1F \times 3F = 18.3F^2$, from transistor current specifications in CACTI. The write energy per bit, calculated as writing current times write time times supply voltage (Focus C Tables,ITRS [16]), is 2.06 pJ/bit.

For the read operation, the latency is taken to be the speed of SRAM read access. This is consistent with the STT-RAM cell parameters assumed and with other studies performed [9] [12]. The sensing voltage for the read operation is chosen as 0.46V, to keep the probability of read disturb around 10^{-20} with read current $\mathrm{I}_{RD} = 0.2 \times I_{c0}$. [10]

3.3 Modifications to GPGPU-Sim

The conventional shared memory in the GPU is implemented in SRAM. SRAM access times are smaller than the clock cycle time (0.3ns [16]). Nevertheless, the clock cycle time, which is usually more than SRAM access time, determines the completion of the access. In GPGPU-Sim shared memory reads and writes are modeled to take one clock cycle to complete [21]. According to NVIDIA, shared memory operations are the same latency as register file access if there are no bank conflicts.

As the core clock cycle is 1.54ns, the delay for STT-RAM array access is taken to be 0.3ns for read operations, and 5.3ns for write operations. As 5ns is MTJ switching time, and the array access times are assumed to be 0.3ns. Giving a total of 5.3ns for write accesses. Therefore an STT-RAM shared memory access time in terms of clock cycles is 1 clock cycle for read, and four clock cycles for write operations. Implementing the shared memory in STT-RAM required write latencies of more than one clock cycle to be incorporated in GPGPU-Sim. As the core clock cycle is 1.54ns, the delay for STT-RAM array access is taken to be 0.3ns for read operations, and 5.3ns. As 5ns is MTJ switching time, and the array access times are assumed to be 0.3ns. Giving a total of 5.3ns for write operations. Therefore an STT-RAM shared memory access time in terms of clock cycles is 1 clock cycle for read, and four clock cycles for write operations. Implementing the shared memory in STT-RAM required write latencies of more than one clock cycle to be incorporated in GPGPU-Sim.

To model the STT-RAM shared memory and obtain performance data for the chosen benchmarks, we incorporated the latency of shared memory writes depending on relative difference between

SRAM and STT-RAM latencies. A write to shared memory is not marked as completed until the number of clock cycles corresponding to the STT-RAM write latency have passed.

3.3.1 Baseline

The baseline configuration chosen for the evaluation was a GPU with 30 SMs, 16kB SRAM shared memory, 16 banks, 1 read-write port per bank, 16kB register file, 1024 active threads / SM, 1.3Ghz shader (fast) clock. The execution units run on the fast clock, at 1.3GHz. The core clock which drives the shared memory and register files runs at half the speed of the fast clock, at 650MHz. This is a representation of the GT200 GPU architecture (e.g., GTX 285).

3.4 Benchmarks and their Characteristics

Performance of an application depends on the amount of parallelism it has, the processor speed and architecture, and the amount and availability of resources. For highly parallel applications running on a processor with hundreds of execution units, amount and availability of resources becomes an important factor in determining performance. In a GPU, whether a block of threads gets scheduled on an SM for execution is determined by its shared memory usage per block, register usage per thread, number of threads per block, texture memory usage, constant memory usage, and how much of these resources is available on a particular SM. For example, consider a block of 128 threads using 8kB of shared memory. Though a maximum of 1024 threads can concurrently execute on an SM, the number of blocks - each consisting of 128 threads - that can be executed in this case is only two, as only 16kB shared memory capacity [1] is available on an SM. Hence, limitation of that resource poses a bottleneck for performance of that application.

The following are the characteristics of which show a benchmark's usage of resources relevant to the study:

1. Shared memory usage - limits number of threads that can concurrently execute on an SM
2. Percentage of shared memory writes - as shared memory write latency is high, the greater the shared memory writes, the shared memory bank is unavailable for a larger amount of time

[1] 16kB shared memory per SM on GT200 architecture. The Fermi architecture has up to 48kB shared memory capacity per SM.

3. Percentage of shared memory reads - when shared memory write latencies are several clock cycles, having more shared memory accesses (reads or writes) results in loss of performance as shared memory requests have to wait before a bank in use is freed

4. Register usage per thread - as the number of physical registers available is finite, high register usage results stalls which affects performance

5. Number of threads - defines the throughput of the application

6. Percentage of memory instructions - global memory accesses are 400-600 clock cycles. Since there are no caches, the more there are global memory accesses, the less the access latencies can be overlapped with execution, posing a bottleneck for system performance

7. Runtime - time taken for the execution of a particular benchmark is presented for comparison with other benchmarks' runtimes, and ultimately leakage energy

As it can be seen from Figure 3.2 that different benchmarks exhibit different characteristics, which shows the diversity of the benchmark set. Henceforth, all shared memory reads and writes will be referred to as reads and writes, unless explicitly stated otherwise.

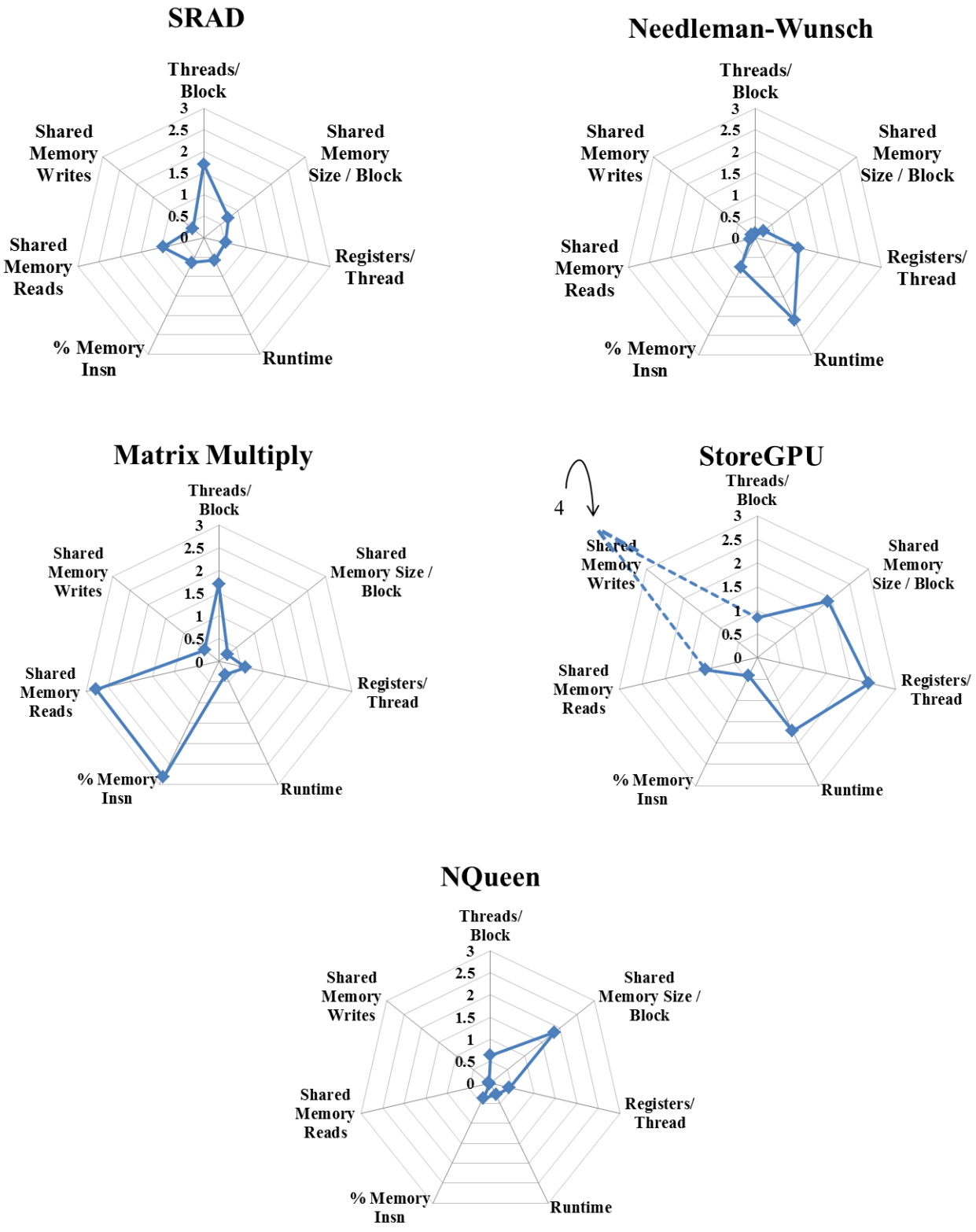


Figure 3.2: Each benchmark is characterized by threads per block, shared memory size utilized per block, registers used per thread, percentage of memory instructions, percentage of shared memory reads and writes and runtime. Each characteristic is normalized against its average across all benchmarks. The plots show that the set of benchmarks display diversity.

SRAD

Structural grid applications involve the decomposition of the computation into highly spatial sub-blocks such that any change to one element depends on its neighbors. SRAD (Speckle Reducing Anisotropic Diffusion) is an example of a structural grid application. It is essentially a diffusion algorithm for ultrasonic and radar imaging applications that uses partial differential equations. It processes the images by identifying and removing locally correlated noise known as speckles without destroying important image features. Since SRAD is a structural grid application with computations over sets of neighboring pixels, it takes advantage of the on-chip shared memory.

Needleman-Wunsch

Needleman-Wunsch is a dynamic programming application which solves an optimization problem by storing and reusing the results of its sub problem solutions. Needleman-Wunsch is a global optimization method for the alignment of sequences typically used for protein and DNA sequences. The pairs of sequences are organized in a 2D matrix and the algorithm is carried out in two steps. In the first step the sequence values are populated in the matrix from top left to bottom right. To get the optimal alignment, the pathway with the maximum score is obtained where a score is the value of the maximum path ending at that cell. This benchmark exploits the advantages of shared memory as each data element is used four times to calculate the values of four different elements. Data elements on the same diagonal in a thread block are concurrently executed, while thread blocks on the same diagonal within the entire matrix are executed in parallel.

MatrixMultiply

The implementation of matrix multiply uses shared memory blocking, where the size of the block is 256 elements.

NQueen

The N-Queen solver tackles a classic puzzle of placing N queens on a NxN chess board such that no queen can capture another. It uses a simple backtracking algorithm to try to determine all possible solutions. The search space implies that the execution time grows exponentially with N. Our analysis shows that most of the computation is performed by a single thread, which explains the low IPC.

StoreGPU

StoreGPU is a library that accelerates hashing-based primitives designed for middle-ware. The

benchmark consists of a sliding-window implementation of the MD5 algorithm for an input file of size 192KB. The off-chip memory traffic is minimized by using the fast shared memory.

3.5 Power and Area Evaluation with CACTI

To evaluate power and area of the STT-RAM shared memory array, we use the CACTI model with parameters like access transistor current, cell area and sense voltage set to STT-RAM specific values. CACTI is a tool which provides a model to determine cache and memory access time, area, leakage, and dynamic power. It models delay, power and area of these major components: decoder, wordline, bitline, sense amplifier, comparator, multiplexor, output driver, inter-bank wires for memory and caches made of SRAM and DRAM arrays. The peripheral circuitry STT-RAM arrays use are similar to those used in SRAM arrays (as shown in Figure 2.6. Hence, CACTI can be used to estimate area, and approximate energy for STT-RAM arrays.

In CACTI the structure of the memory consists of banks, consisting of data and tag arrays, which can be accessed simultaneously. The data or tag array is made up of sub-arrays which support a single access. Four sub-arrays make a mat. CACTI performs an exhaustive search on different number of subarrays i.e., different number of vertical and horizontal partitions of the data or tag array.

The STT-RAM memory array read access starts when the address is inputed to decoder, then wordline in data array is activated, the source line is kept low and the bitline is precharged to voltage V_{RD} not high enough to cause a read disturb, the bitline then discharges through the cell, a reference bitline is simultaneously discharged using a reference cell, these are input to a sense amplifier and sensing is performed. The same sequence of operations are performed for the write access, except that after wordline is activated, to write a '1' the bitline is kept low and source line is driven high, and to write a '0' the bitline is driven high while the source line is kept low.

In CACTI, the scratch-ram type memory which is direct-mapped and with no tag-array is used to model shared memory. Power and area for various configurations of capacity, banks and ports is obtained. For static or leakage power calculations, power consumed by peripheral circuitry and access transistors of the STT-RAM cell that in the 'off' state while reading or idling are taken

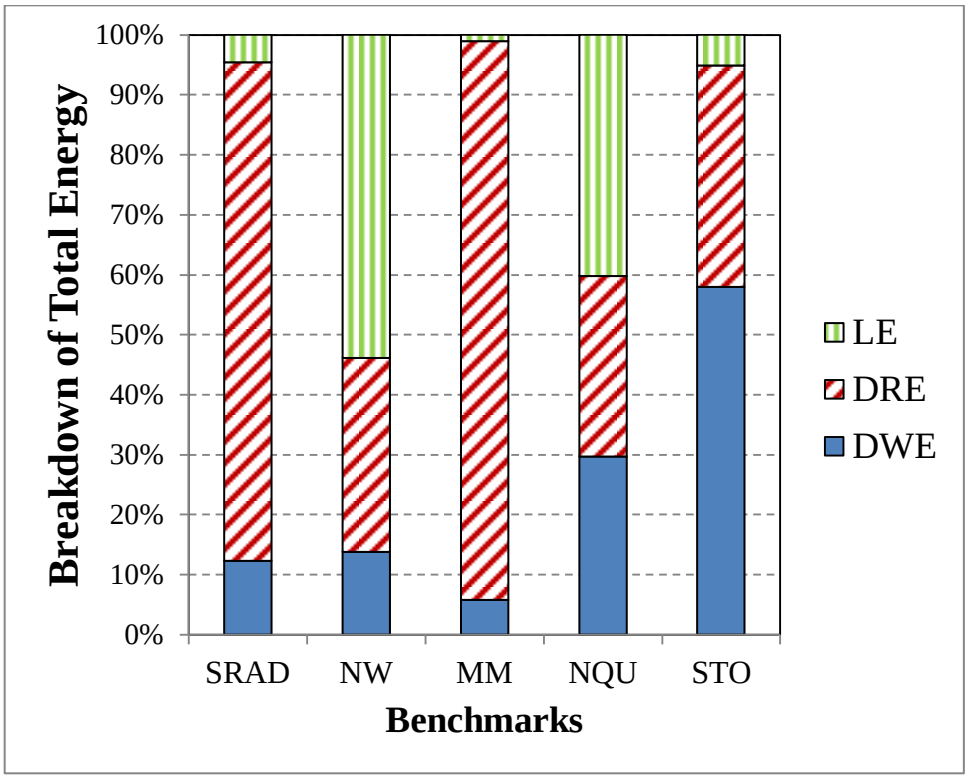


Figure 3.3: Percentage of leakage energy, dynamic read and write energy of the benchmarks for an SRAM shared memory.

into consideration. The MTJ does not leak any power. For dynamic power calculations, peripheral circuity and access transistor power consumption is taken in consideration.

The baseline SRAM cell area is assumed to be $146F^2$, while the STT-RAM cell area is assumed to be $18.3F^2$. STT-RAM access transistors with operating current 432μA is assumed. For the interconnect, semi-global wires which are smaller and consume lesser power with more delay than global wires are considered.

Chapter 4

Results

First we present performance results for all the benchmarks, then area and energy evaluations. Finally, to visualize trade-offs, performance, area and energy results will be put together in the context of various configurations of shared memory capacity, number of banks per shared memory, number of ports per bank, frequency of shared memory access, shared memory area and static and dynamic energy.

The list of configurations: abbreviations and full-forms used is presented in Table 4.1. All SMs contains one shared memory array, and each array consists of several banks, and each of those banks can have 1 or more ports. For the sake of simplicity, henceforth, reference to shared memory capacity pertains to one SM, number of banks pertains to one shared memory array, and number of ports pertains to one bank.

Abbreviation	**Full-form**
16kB 16b 1p	16kB shared memory capacity per SM, 16 banks per shared memory, 1 port per bank
16kB 16b 2p	16kB shared memory capacity per SM, 16 banks per shared memory, 2 port per bank
16kB 32b 1p	16kB shared memory capacity per SM, 32 banks per shared memory, 1 port per bank
64kB 16b 1p	64kB shared memory capacity per SM, 16 banks per shared memory, 1 port per bank
64kB 16b 2p	64kB shared memory capacity per SM, 16 banks per shared memory, 2 port per bank

Table 4.1: List of abbreviations and their full forms.

4.1 Performance

STT-RAM access latency varies with the design parameters of the MTJ. Design parameters of the MTJ include structural properties such as the thickness and area of the free layer, and material properties such as the saturation magnetization and spin transfer efficiency, offering a variety of design possibilities. These designs usually offer substantial power savings, at the cost of performance. These issues are studied in the context of GPUs, by implementing the fast shared memory in STT-RAM. To see the trend of the latency's effect on performance, shared memory access latency is varied from 2-40 times relative to SRAM as speedup was measured across all the benchmarks via simulation the results are shown in Figure 4.1 through Figure 4.5. SRAD's performance is degraded by less than 1% up to 59%.NW's performance is degraded by less than 1%-7%. MatMul's performance is degraded by less than 1% up to 57%. NQU's performance is degraded by less than 1% up to 48%. STO's performance is degraded by less than 1% up to 59%. For STT-RAM write latency 4 times SRAM write latency (4 clock cycles). SRAD experienced a degradation of 2%, NW 1%, MatMul 16%, NQU 1%, and STO 2%.These results related to each benchmark's number of shared memory instructions per set of active warps. The greater the number of shared memory instructions in a benchmark's set of active warps, the more vulnerable it is to shared memory access latency. The reason the correspondence is not exact is due to the fine-grained multithreading of all the active warps on an SM. Every issue cycle, the warp scheduler selects ready warps and issues them to the SPs in 'loose' round robin policy [3]. Warps which are non-ready, like those with threads waiting on long memory accesses, are taken out of the scheduling pool till the access is completed, enabling the SM to hide long latency operations and provide throughput.

From this point onwards all evaluations assume STT-RAM write latency equal to 4 clock cycles, which is 4 times SRAM write latency. STT-RAM read latency is assumed to be 1 clock cycle, which is the same as SRAM read latency.

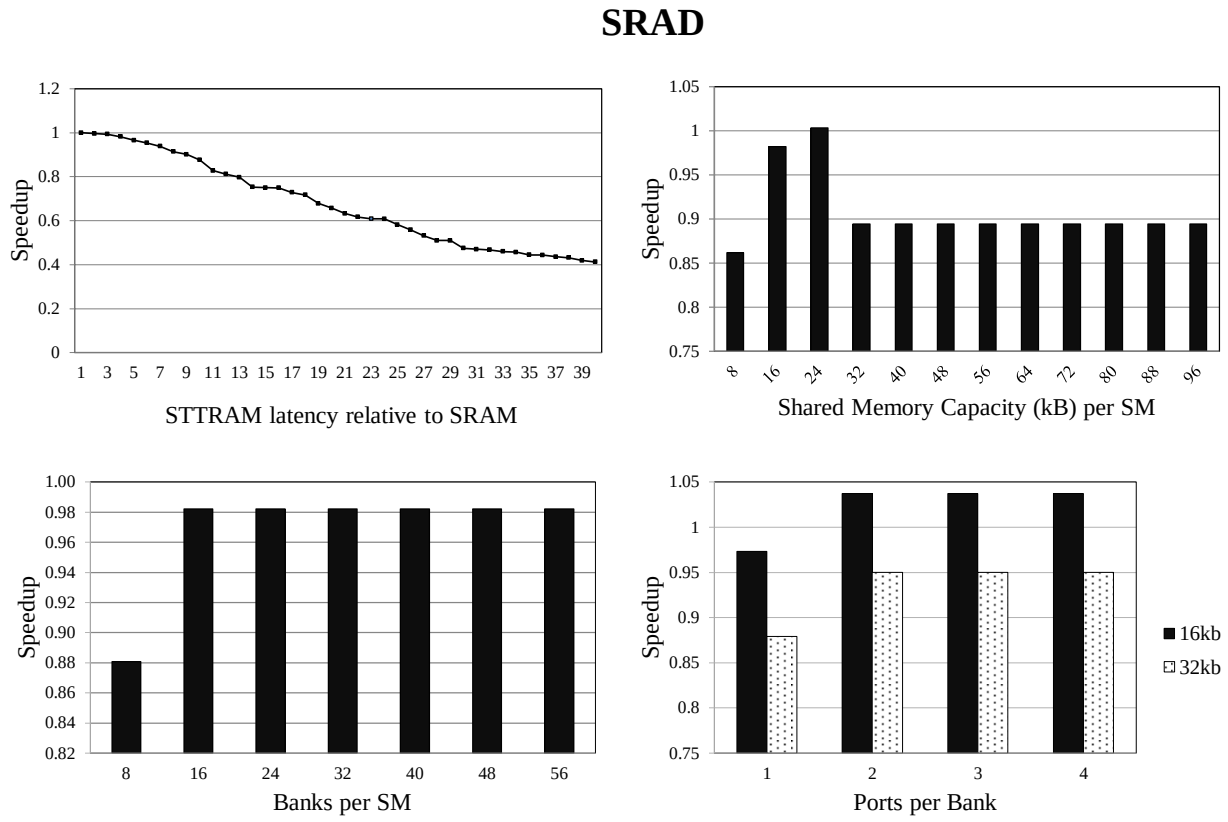


Figure 4.1: Speedup of benchmark SRAD relative to baseline when (a) shared memory latency is varied; then keeping latency constant at 4 clock cycles (b) shared memory capacity is varied (c) number of banks per shared memory (or SM) is varied keeping capacity constant at 16kB (d) number of ports per bank is varied while keeping banks at 16.

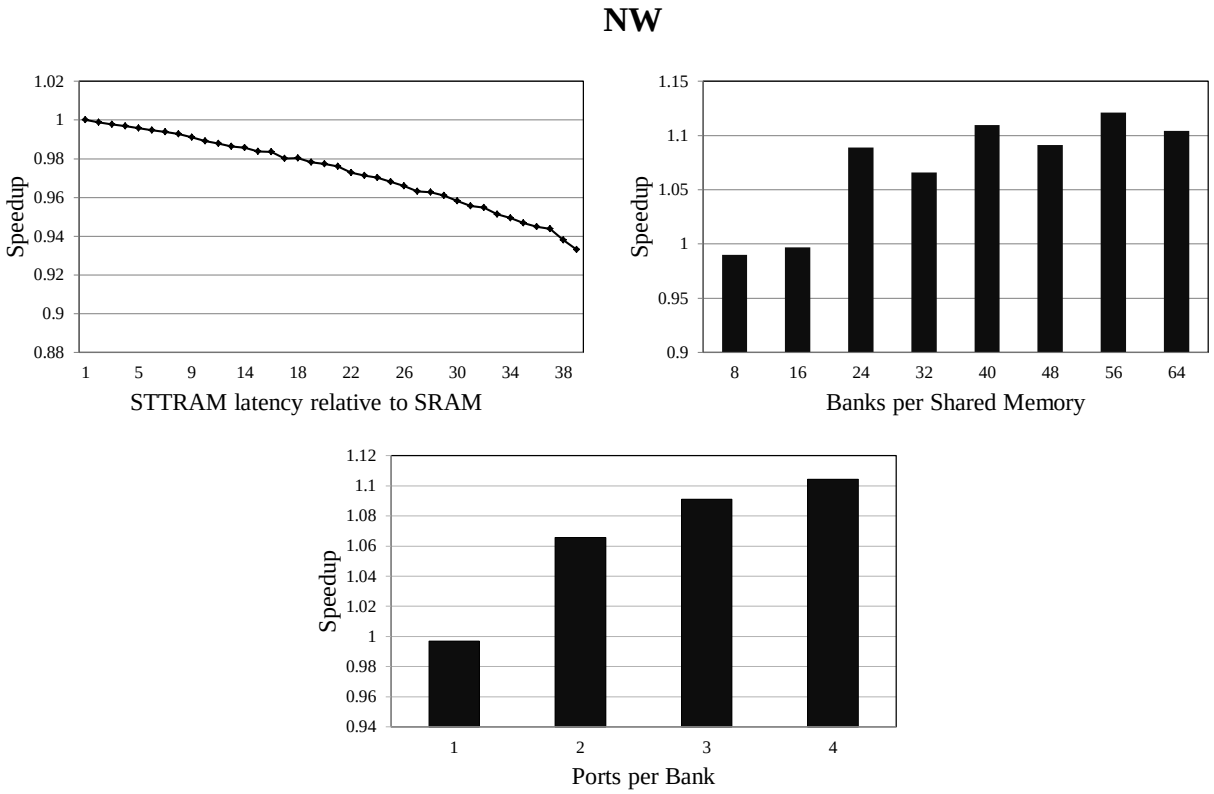


Figure 4.2: Speedup of benchmark NW relative to baseline when (a) shared memory latency is varied; and keeping latency constant at 4x (b) number of banks per shared memory (or SM) is varied keeping capacity constant at 16kB (d) number of ports per bank is varied while keeping banks at 16 and capacity at 16kB. The number of threads in NW are very low (96 per SM). Implying a low frequency

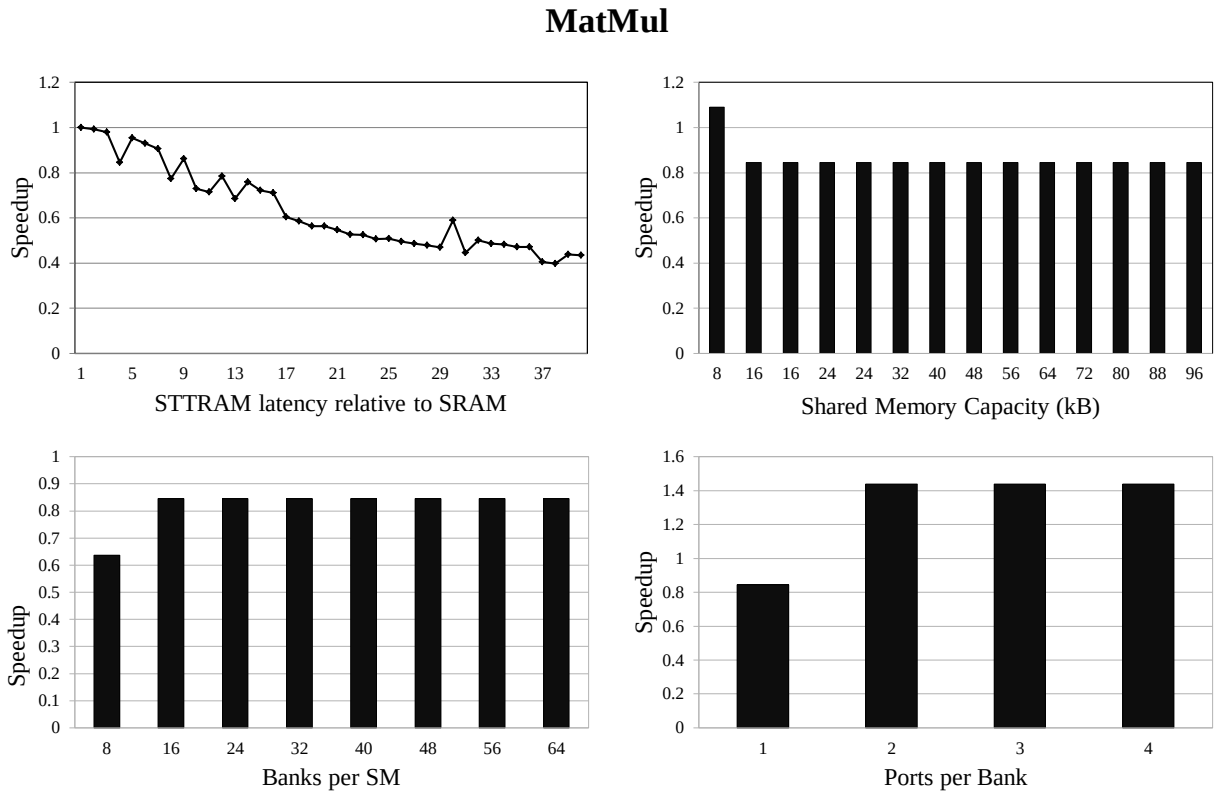


Figure 4.3: Speedup of benchmark MatMul relative to baseline when (a) shared memory latency is varied; and keeping latency constant at 4x (b) shared memory capacity is varied (c) number of banks per shared memory (or SM) is varied keeping capacity constant at 16kB (d) number of ports per bank is varied while keeping banks at 16 and capacity at 16kB.

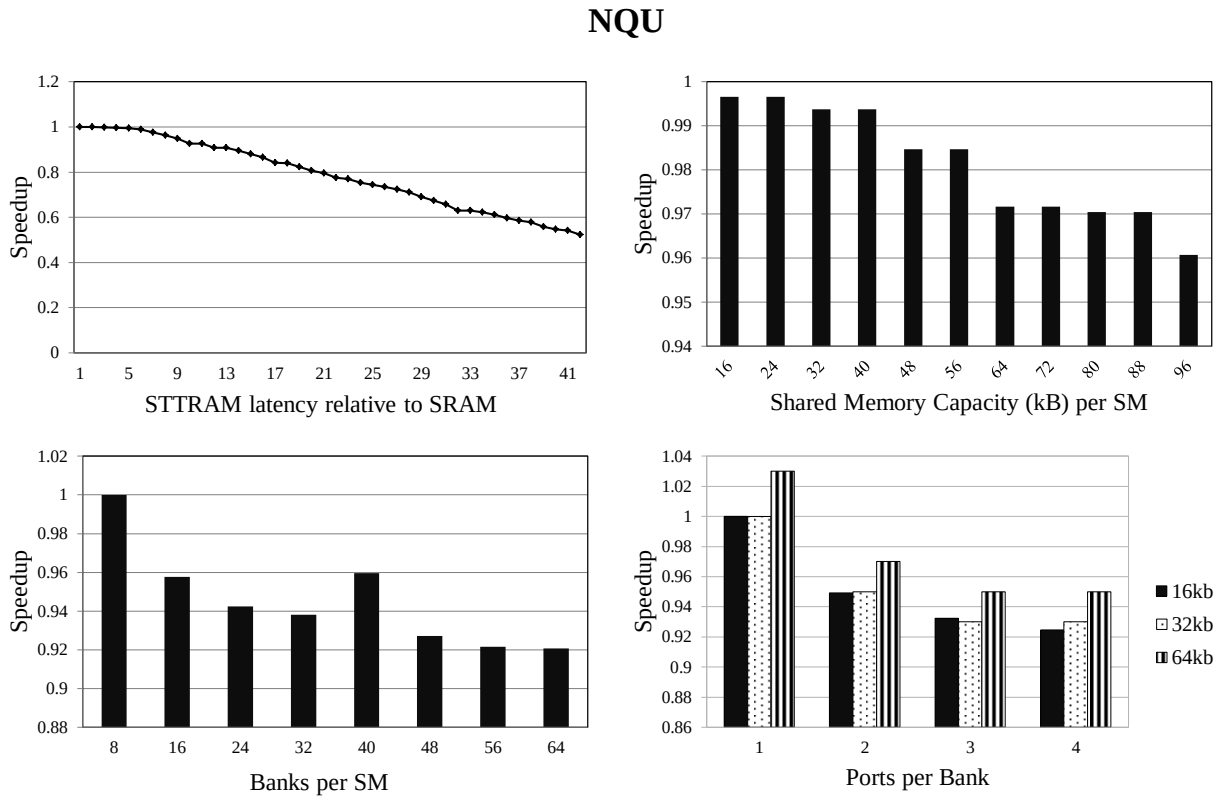


Figure 4.4: Speedup of benchmark NQU relative to baseline when (a) shared memory latency is varied; and keeping latency constant at 4x (b) shared memory capacity is varied (c) number of banks per shared memory (or SM) is varied keeping capacity constant at 16kB (d) number of ports per bank is varied while keeping banks at 16.

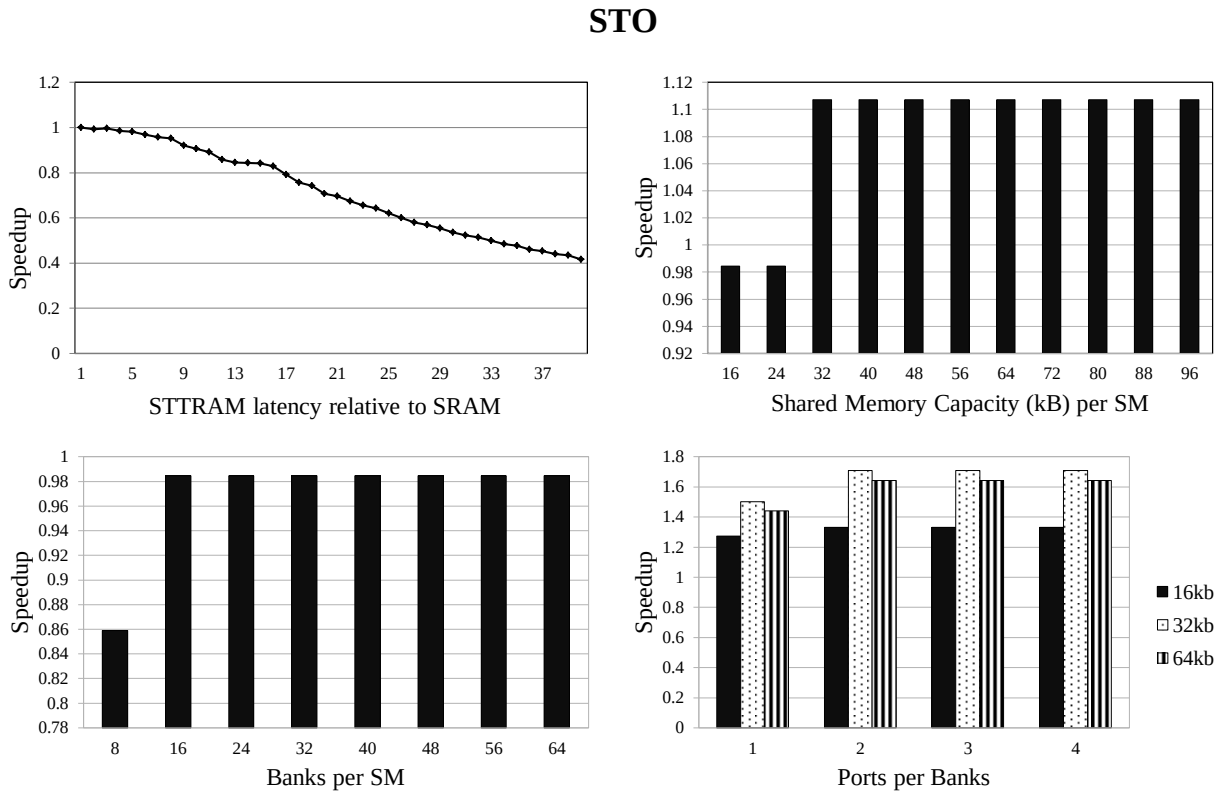


Figure 4.5: Speedup of benchmark STO relative to baseline when (a) shared memory latency is varied; and keeping latency constant at 4× (b) shared memory capacity is varied (c) number of banks per shared memory (or SM) is varied keeping capacity constant at 16kB (d) number of ports per bank is varied while keeping banks at 16.

4.2 Theoretical analysis with performance equation

To theoretically analyze performance of the benchmarks, we developed the equations shown below. Equation (4.1) and Equation (4.2) identify execution time as the sum of both the runtime of the benchmarks and the clock cycle time multiplied by memory stall cycles, and are taken from Hennessey and Patterson [13].

Memory stalls are caused by different levels of the memory hierarchy. As shown below in Equation (4.3), the number of memory stall cycles for each memory of the hierarchy is a product of the number of memory accesses per instruction times access latency.

$$\begin{aligned} Execution\,Time_{SM} = & \\ & = (SM\,Clock\,Cycles + Memory\,Stall\,Cycles) \\ & \times Clock\,Cycle\,Time \end{aligned} \tag{4.1}$$

$$\begin{aligned} Execution\,Time_{SM} = & \\ & = Instruction\,Count \times (Clock\,Per\,Instruction_{runtime} \\ & + \frac{Memory\,Stall\,Clock\,Cycles}{Instruction}) \\ & \times Clock\,Cycle\,Time \end{aligned} \tag{4.2}$$

$$\begin{aligned} Execution\,Time_{SM} = & \\ & = Instruction\,Count \times (CPI_{runtime} \\ & + \frac{Global\,Memory\,Accesses}{Instruction} \\ & \times Global\,Memory\,AccessLatency \\ & + \frac{Shared\,Memory\,Accesses}{Instruction} \\ & \times Shared\,Memory\,Access\,Latency) \\ & \times Clock\,Cycle\,Time \end{aligned} \tag{4.3}$$

The SM performs loose-round-robin scheduling, somewhat like an out-of-order processor. Hence, there will be overlapping of execution and memory access. Therefore, only *Non-overlapped Access Latency* contributes to the total execution time, as shown in Equation (4.4).

$$
\begin{aligned}
Execution\,Time_{SM} = & \\
& = Instruction\,Count \times (CPI_{runtime} \\
& + \frac{Global\,Memory\,Accesses}{Instruction} \\
& \times Global\,Memory\,Non-overlapped\,Access\,Latency \\
& + \frac{Shared\,Memory\,Accesses}{Instruction} \\
& \times Shared\,Memory\,Non-overlapped\,Access\,Latency) \\
& \times Clock\,Cycle\,Time
\end{aligned} \tag{4.4}
$$

The *Non-overlapped Access Latency* can be said to be the access penalty times the non-overlapped fraction of the memory access. The greater the non-overlap of accesses to a memory, the greater the corresponding penalty incurred, as shown in Equation (4.5).

$$
\begin{aligned}
ExecutionTime_{SM} = & \\
& = Instruction\,Count \times (CPI_{runtime} \\
& + \frac{Global\,Memory\,Accesses}{Instruction} \\
& \times Global\,Memory\,Access\,Latency \\
& \times Global\,Memory\,Access\,Non-overlapFraction \\
& + \frac{Shared\,Memory\,Accesses}{Instruction} \\
& \times Shared\,Memory\,Access\,Latency \\
& \times Shared\,Memory\,Access\,Non-overlapFraction) \\
& \times Clock\,Cycle\,Time
\end{aligned} \tag{4.5}
$$

The *non-overlapped fraction* of shared memory accesses increases as the latency of shared memory write increases. This is because shared memory is occupied for a longer time for writes, and

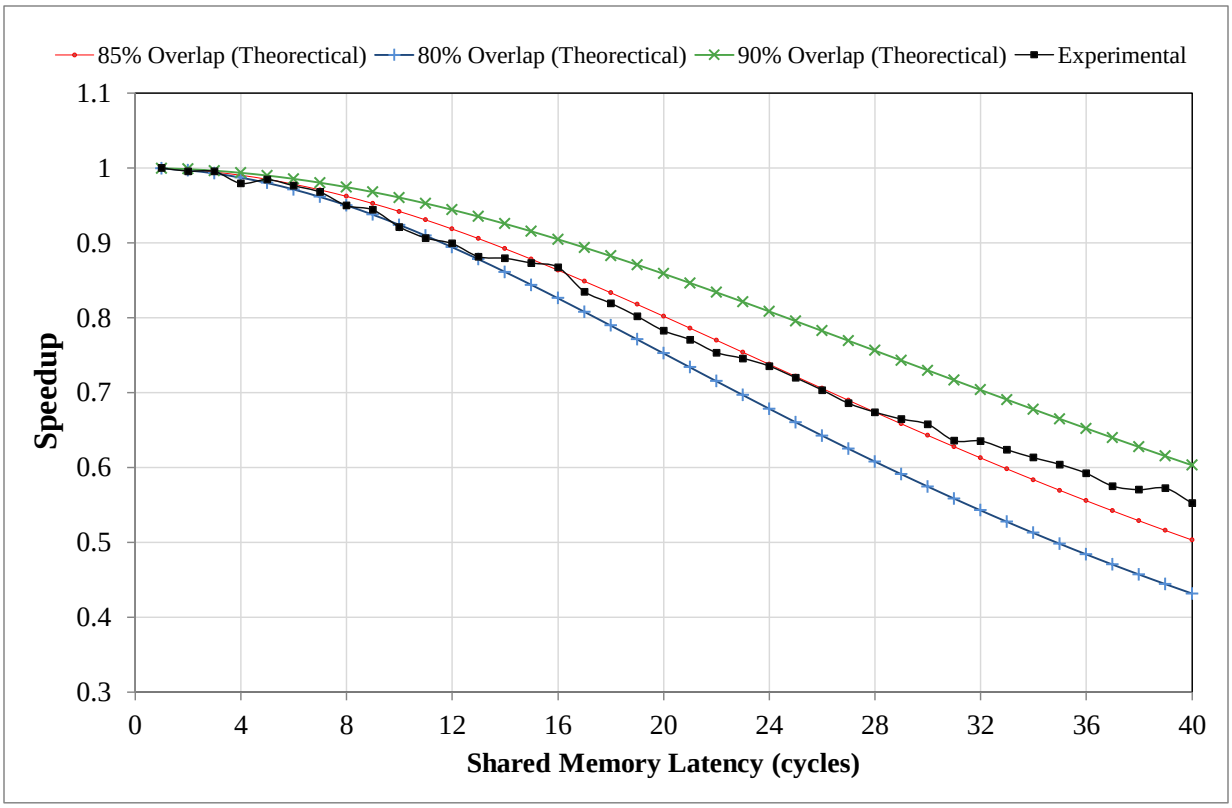


Figure 4.6: Dependence of Theoretical and Experimental Speedup on Shared Memory Write Latency

other shared memory access cannot take place. Although non-shared-memory instructions can be executed in parallel with shared memory accesses, the data-dependency between instructions can start limiting this parallelism. Hence, increasing the write latency increases the degree of non-overlap; or, $non-overlapped\,fraction \propto shared\,memory\,write\,latency$.

In *Execution Time with SRAM Shared Memory*, as shown in Equation (4.6), stalls due to shared memory were not accounted for, as access to shared memory is as fast as register access, when there are no bank conflicts. Accounting for bank conflicts had less than 1% impact on execution time.

Figure 4.6 compares the average simulated speedup across all benchmarks with the theoretical speedup calculated as shown in Equation (4.6).

$$Speedup = \frac{Execution\,Time\,with\,SRAM\,Shared\,Memory}{Execution\,Time\,with\,STT-RAM\,Shared\,Memory} \tag{4.6}$$

In results shown in Figure 4.6, the average fraction of global and shared memory accesses across all benchmarks is 23%, and 18%, respectively. The latency of global memory access is taken to be

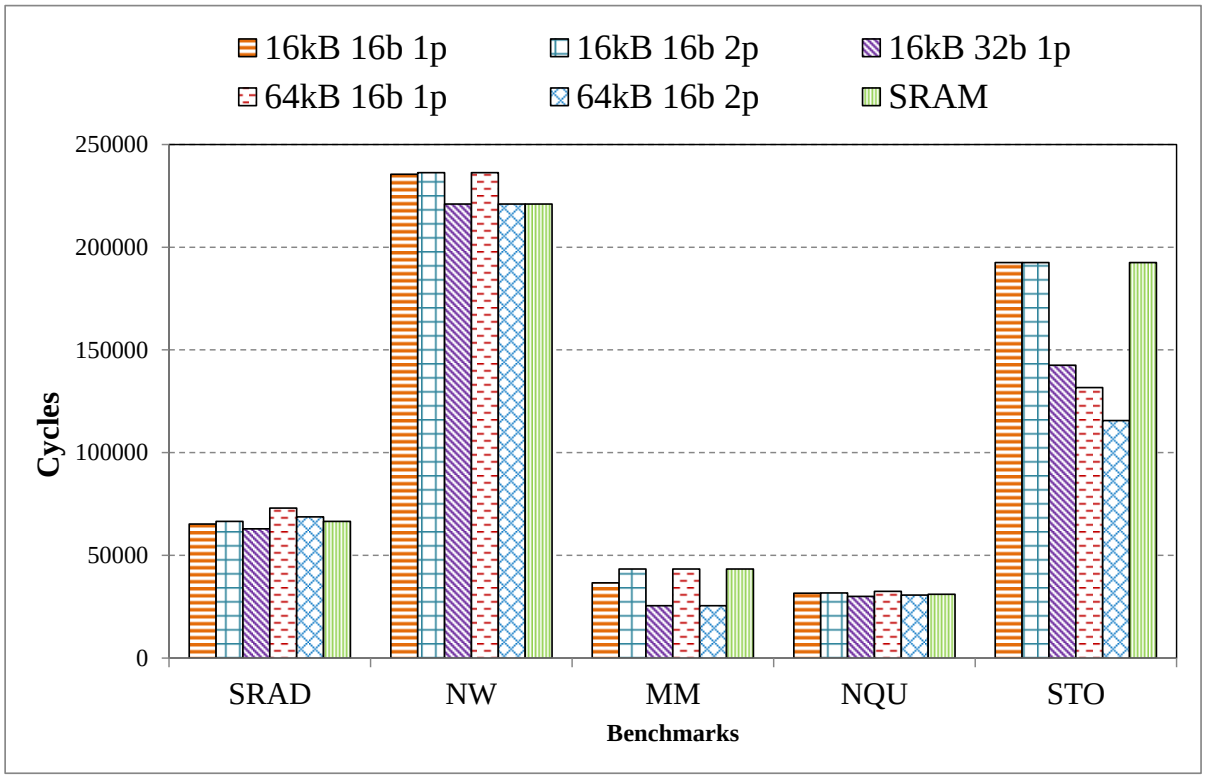


Figure 4.7: Runtimes for various configurations.

600 clock cycles [21], while latency of shared memory access is varied from 1 to 40 clock cycles . The percentage of overlap of global memory accesses is taken to be 70% for both SRAM and STT-RAM shared memory. Whereas, the percentage of overlap of shared memory access in SRAM shared memory is taken to be 99%, as shared memory access is as fast as register access except when bank conflicts occur. Three values of overlap of STT-RAM shared memory access are considered 80%, 85%, 90%. It can be seen in Figure 4.6, that the degree of overlap is 80% from 1× to around 13× shared memory write latency, 85% from 14× to 29×, and 90% 30× .

4.2.1 Exploring Various Configurations

To evaluate performance, energy and area trade-offs various configurations of STT-RAM shared memory are evaluated. The runtimes of all benchmarks for the chosen configurations is presented in Figure 4.7.

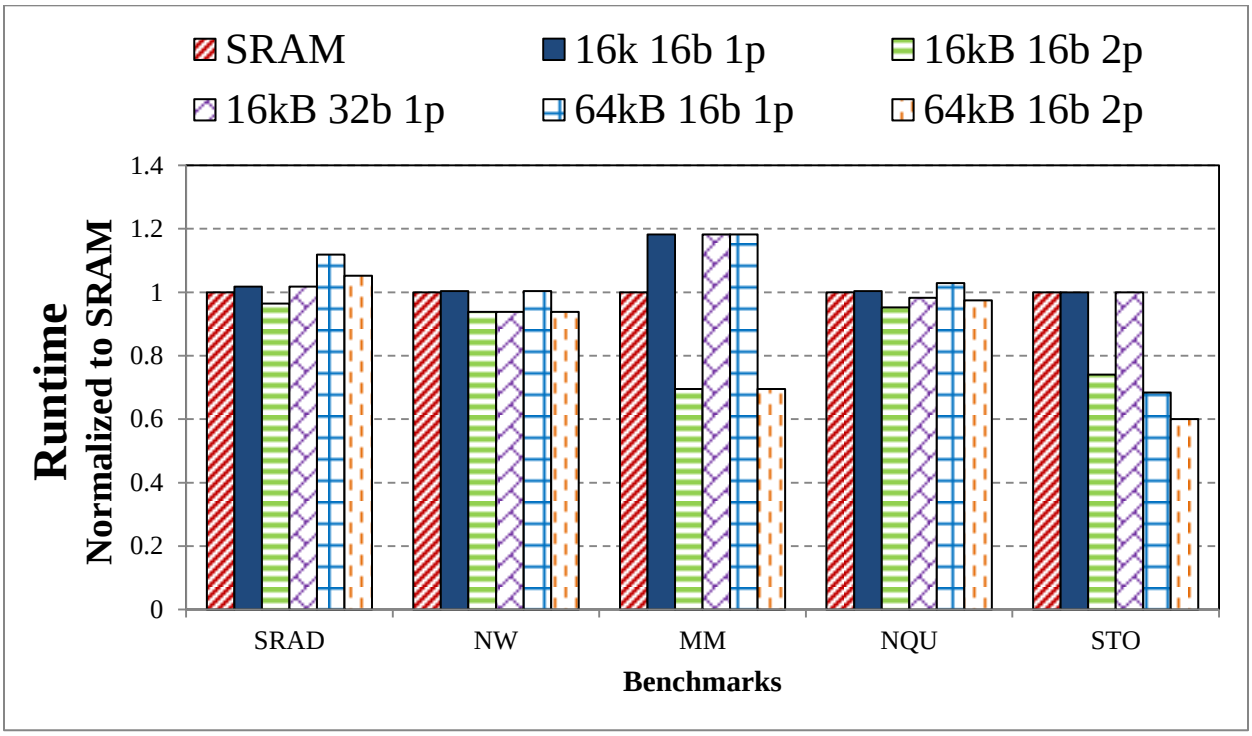


Figure 4.8: Runtimes for various configurations normalized to SRAM.

4.3 Area

The area of shared memory, implemented in STT-RAM, was determined using CACTI, as described in the methodology section. The area of STT-RAM memory with 16kB 16b 1p configuration is 0.57x the area of the baseline SRAM. Adding one extra port per bank resulted in 2.12x the area of the baseline, while adding 16 extra banks per shared memory resulted in only 0.86x area of baseline. Increasing the capacity by four times increased the area to 0.98x of the baseline, making the STT-RAM shared memory four times denser than the baseline. Adding an extra port per bank to the above configuration resulted in 3.22x baseline area.

A steeper rise in area is encountered when the number of ports is increased than when capacity is increased for the same number of banks, because adding ports involves laying out new bitlines and associated peripheral circuitry, the area increases substantially as ports are increased. Increasing the number of banks for the same capacity involves adding more block decoders and interconnect; the resulting area increase is around 27% going from the 16kB 16b 1p to the 16kB 32b 1p configuration.

Out of the five configurations explored, three of them — 16kB 16b 1p, 16kB 32b 1p, and 64kB 16b 1p — have areas within the budget set by the baseline SRAM 16kB 16b 1p configuration.

4.4 Energy

The energy consumption for STT-RAM shared memory was obtained using CACTI, as mentioned in the methodology section. Figure 4.9 and 4.10 show the energy, area and capacity results for STT-RAM shared memory. The values obtained are normalized to the SRAM baseline, which is 16 kB of shared memory, 16 banks and 1 port. SRAM values are represented by a horizontal black line at y=1 in Figure 4.9 and 4.10.

Figure 4.9(a) shows results for 16kB 16b and 1p of STT-RAM shared memory. Using STT-RAM for this configuration shared results in only 35% leakage power, 45% read energy, 60% area and 425% write energy, relative to SRAM. Introducing an extra port leads to substantial increase in leakage power, dynamic read and write energies, and area, as seen in Figure 4.9(b). This is because adding a port involves adding peripheral circuitry for the extra bitlines, and an access transistor per port of a memory cell. The overhead, in terms of energy or area, for adding 16 extra banks is just a few percent more compared to configuration shown in Figure 4.9(a). Adding a bank involves adding peripheral circuitry like decoders for each bank, whereas adding ports involves adding peripheral circuitry per bitline.

Figure 4.10(a) shows results for 64kB 16b 1p of STT-RAM shared memory. For this configuration, leakage power is 80%, read energy is 50%, write energy is 460%, while area is the same as baseline SRAM, for a factor of four increase in capacity. From Figure 4.10(b) we can see that adding ports substantially increases area and energy due to the extra peripheral circuitry.

The total energy consumed by each benchmark for various configurations of shared memory is shown in Figure 4.11. If we assume a total energy and area budget set by the baseline SRAM, we can see that configurations 16kB 16b 1p, 16kB 32b 1p, and 64kB 16b 1p fall within or just around it, for benchmarks SRAD, NW and MM. These benchmarks have a high ratio of read to write accesses, and the write energy percentage is 10% or less of the total energy consumed by each application. Hence, the high write energy due to STT-RAM is compensated by lowered leakage and read energies. From SRAM data we know that STO and NQU have as many reads as writes, with

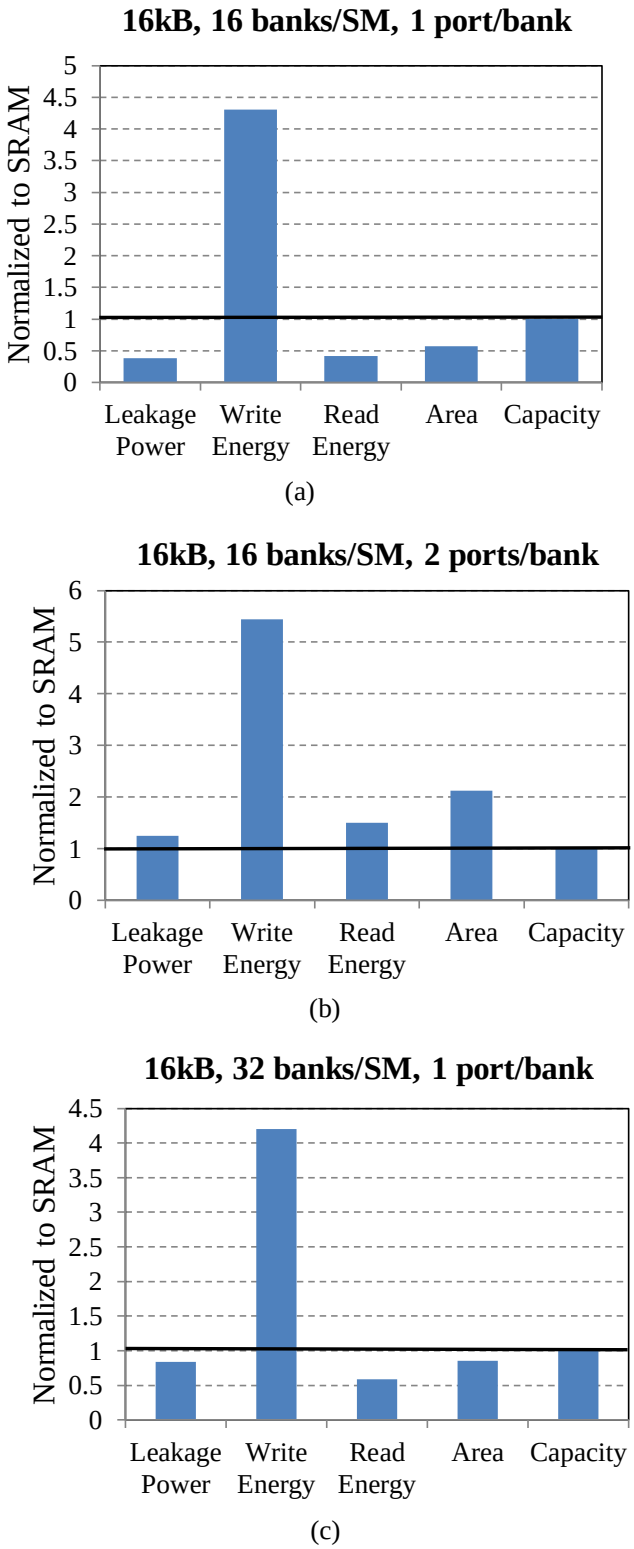


Figure 4.9: Configurations of 16kB shared memory capacity per SM: (a) 16 banks per SM and 1 port per bank, (b) 16 banks per SM and 2 ports per bank, (c) 32 banks per SM and 1 port per bank.

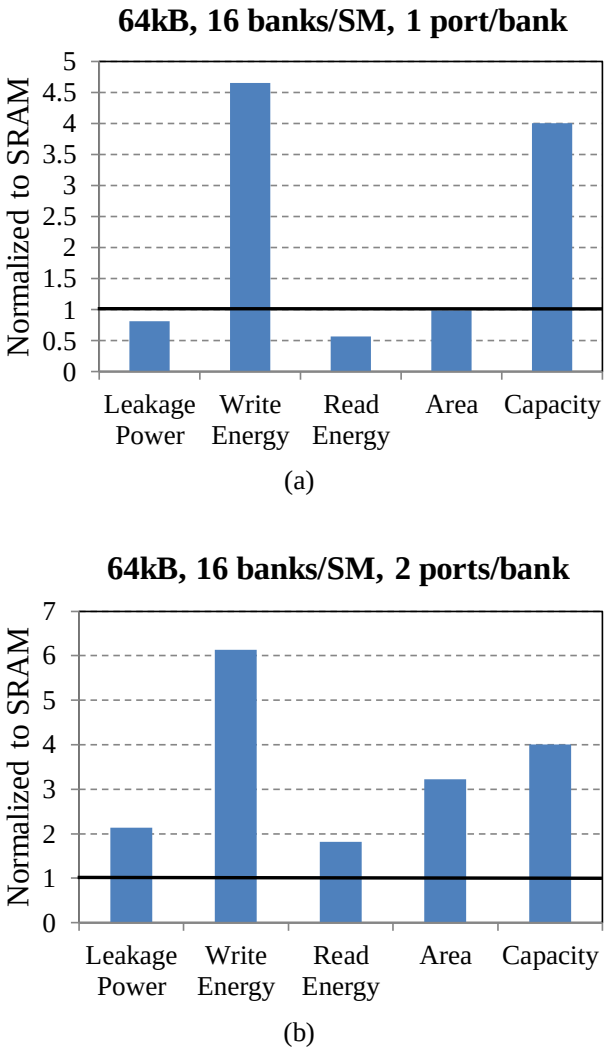


Figure 4.10: Configurations of 64kB shared memory capacity per SM: (a) 16 banks per SM and 1 port per bank, (b) 16 banks per SM and 2 ports per bank.

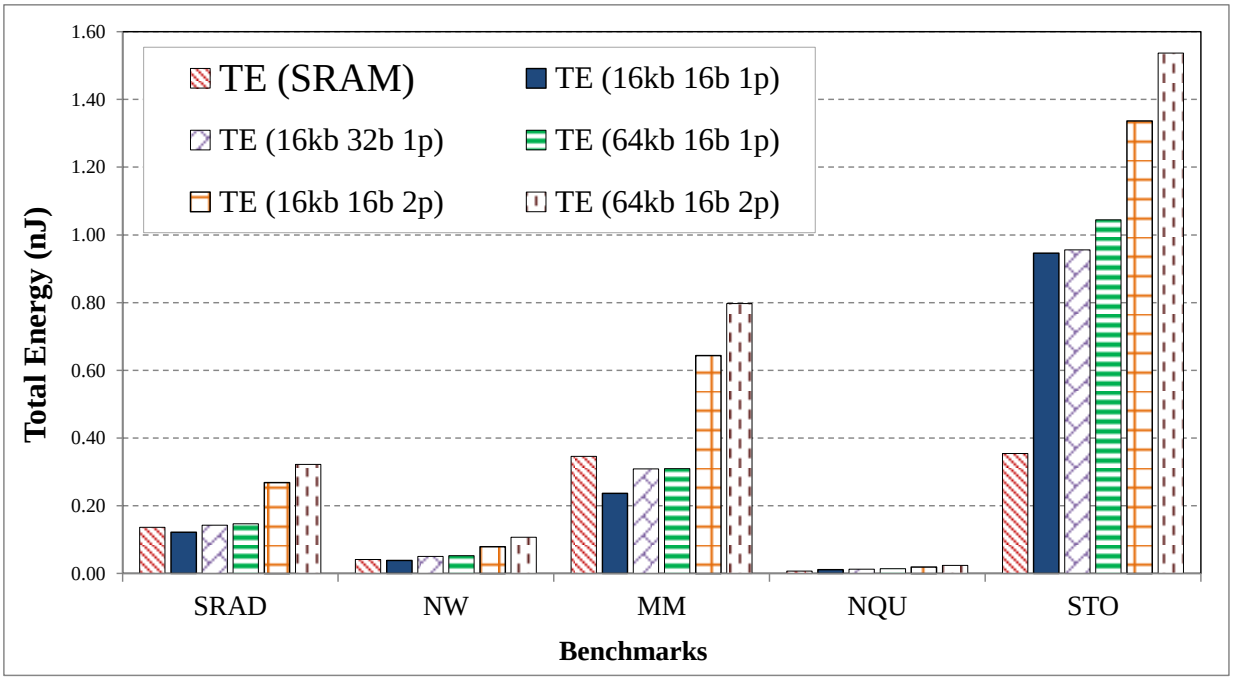


Figure 4.11: Total energy consumption of various configurations for all the benchmarks.

low leakage energy. Hence, for these benchmarks, the high write energy of STT-RAM results in four to six times higher total energy consumption for any configuration.

4.4.1 Breakdown of Total Energy for the Configurations

The following section discusses leakage power, read and write energy for each of the benchmarks. The benchmarks' energy consumptions reflect their characteristics. Leakage energy depends on the runtime of the benchmark, while read and write energies depend on the number of reads and writes. Figure 4.13 shows the energy consumption of SRAD. Leakage energy consumption of SRAD for 16kB 16b 1p configuration is 0.38x of the baseline, while read and write energies are 0.58x and 4.2x of baseline. Adding an extra port results in 1.27x leakage energy, 1.49x read energy and 5.43x write energy compared to the baseline. With 16 extra banks, for 16kB 32b 1p configuration, leakage is 0.86x of baseline. For the 64kB 16b 1p configuration, leakage energy is 0.82x with read and write energies of 0.56x and 4.65x of baseline. Adding an extra port to the above configuration resulted in 2.18x leakage, 1.81x read and 6.13x write energies relative to the baseline. The number of reads in SRAD is nearly 7 times the number of writes, hence read energy is a biggest component of the total energy consumption of SRAD. However, the write energy is nearly equal to the read energy for 16kB

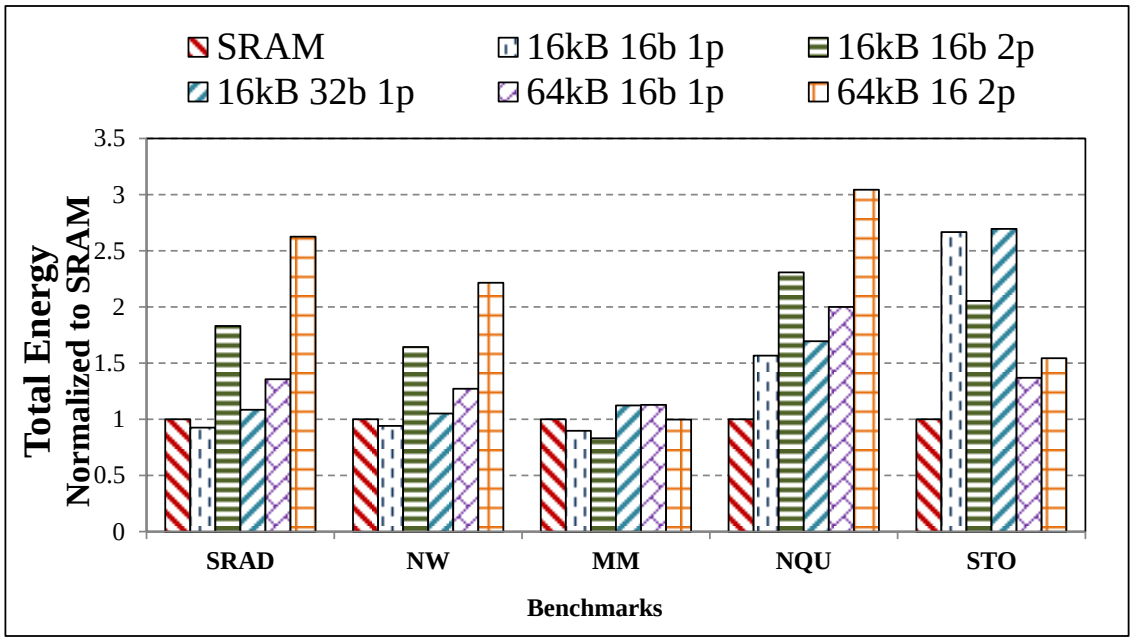


Figure 4.12: Total energy consumption of various configurations for all the benchmarks, normalized to SRAM.

16b 1p, 16kB 32b 1p, and 64kB 16b 1p, yet resulting in total energy consumption of 0.89x, 1.04x, and 1.08x of the baseline, which can be said to be around the energy budget set by the baseline SRAM. The other two configurations, 16kB 16b 2p and 64kB 16b 2p, consume around 2x the energy of the baseline, mainly due to the extra port.

The energy breakdown of benchmark NW for different configurations is shown in Figure 4.14. NW has the longest runtime among the benchmarks and thus consumes the most leakage energy. The number of reads and writes is fairly low (Figure 3.2), so the respective energies are not increased as substantially as in other benchmarks. By reducing the leakage energy and read energy, the total energy of the 16kB 16b 1p configuration is 0.93x relative to the baseline SRAM. Adding 16 extra banks results in 1.22x of the total energy of the baseline. Increasing the capacity by a factor of four in configuration 64kB 16b 1p increases the total energy consumption by 1.26x. While increasing the ports to 2 per bank, increases the total energy by 1.9x and 2.59x for 16kB 16b 2p and 64kB 16b 2p, respectively, relative to the baseline.

Figure 4.15 shows the energy breakdown for the MM benchmark. As MM is a benchmark that runs for a shorter time compared to the average benchmark runtime 3.2, the leakage energy is also relatively less. MM's number of reads are 16 times the number of writes. Therefore, its dynamic

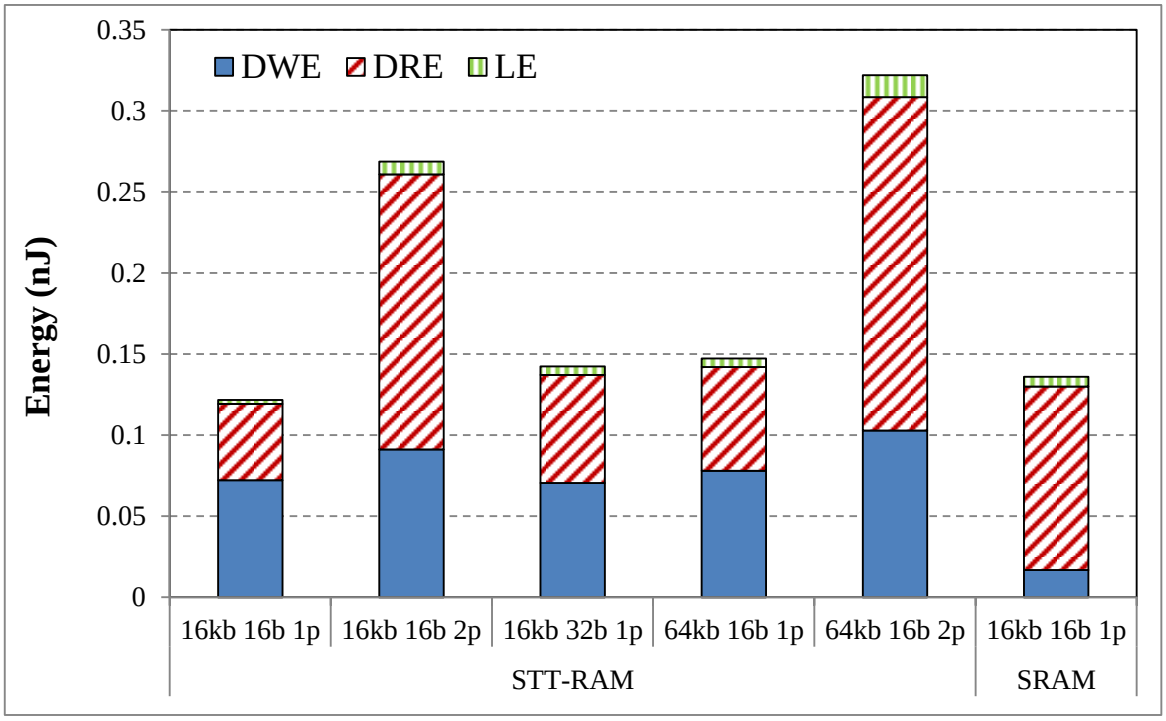


Figure 4.13: Leakage energy (LE), dynamic read energy (DRE) and dynamic write energy (DWE) of various configurations for the SRAD benchmark.

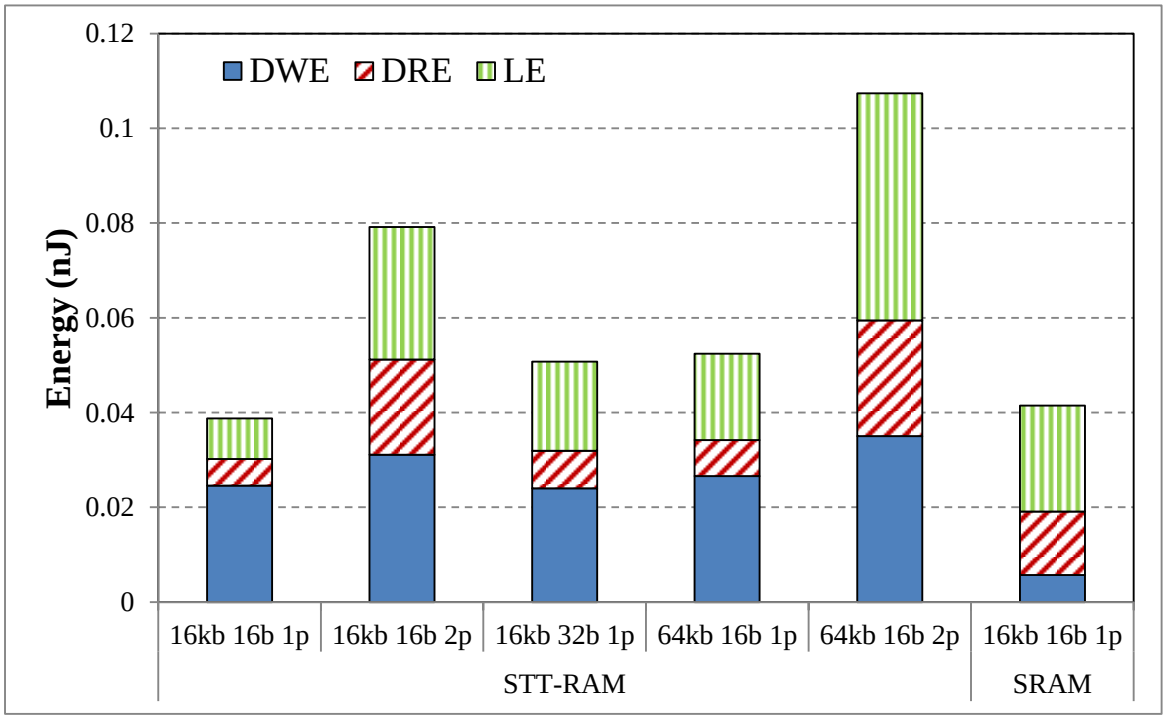


Figure 4.14: Leakage energy (LE), dynamic read energy (DRE) and dynamic write energy (DWE) of various configurations for the NW benchmark.

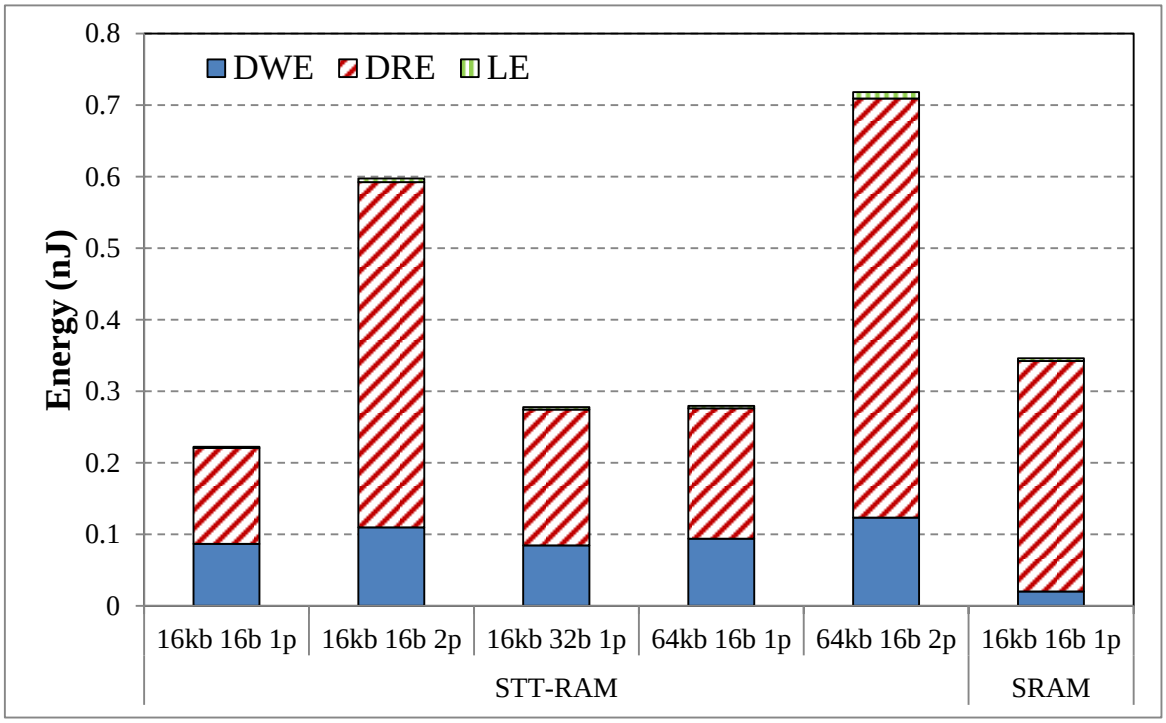


Figure 4.15: Leakage energy (LE), dynamic read energy (DRE) and dynamic write energy (DWE) of various configurations for the MatMul(MM) benchmark.

energy consumption is dominated by read energy. In the 16kB 16b 1p, 16kB 32b 1p and 64kB 16b 1p configurations, although write energy is ∼4x higher for STT-RAM, leakage energy and ready energy decrease to consume less total energy relative to the baseline. For 16kB 16b 2p and 64kB 16b 2p, increasing ports to two per bank increases write energy by 30%, while read energy is increased three times compared to the single port configurations. This results in high read energy, and therefore high total energy for these configurations.

Figure 4.16 shows the energy breakdown for the NQU benchmark. While having a comparatively short runtime, NQU also has a fairly low number of reads and writes. In the total energy breakdown of NQU for the baseline SRAM (3.3), it can be seen that leakage, read and write energies are almost the same. Though leakage and read energy are decreased in the single port configurations 16kB 16b 1p, 16kB 32b 1p and 64kB 16b 1p, the 4-5x increase in write energy increases the total energy consumption of NQU by 55% - 87% compared to the baseline.

The STO benchmark's total energy consumption breakdown is presented in Figure 4.17. STO has the highest number of writes (four times the average) compared to other benchmarks (Figure

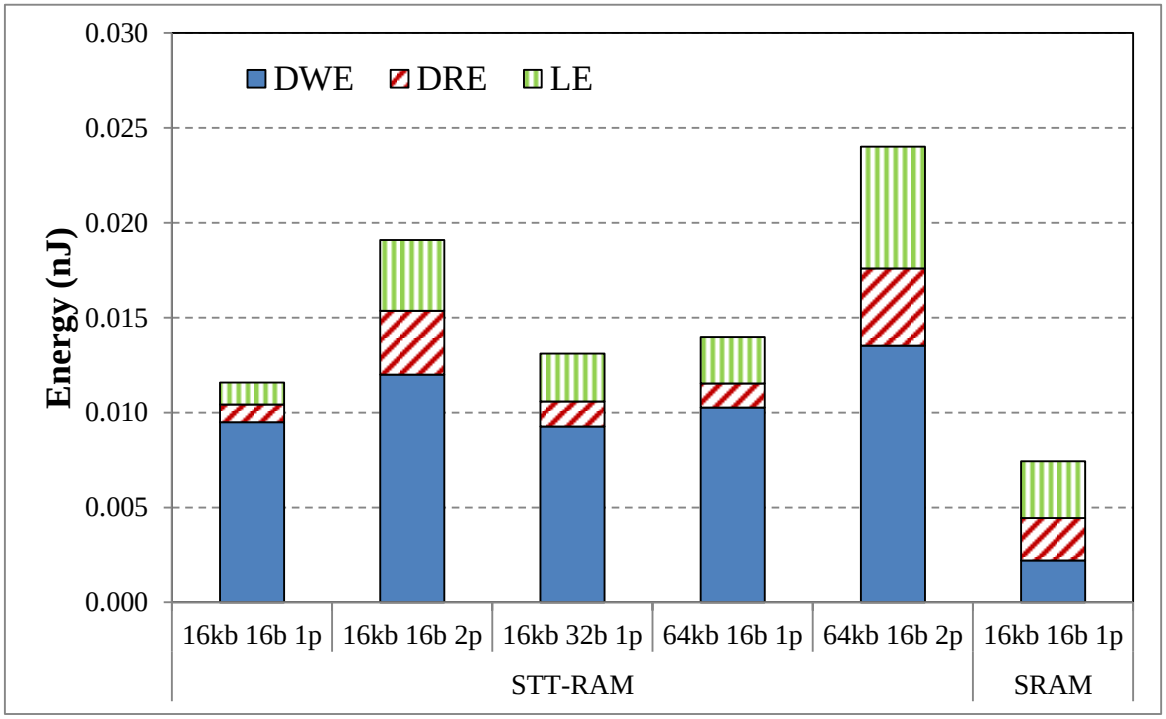


Figure 4.16: Leakage energy (LE), dynamic read energy (DRE) and dynamic write energy (DWE) of various configurations for the NQU benchmark.

3.2). STO has comparatively long runtimes too, although the energy consumed by writes dominates the total energy consumption even for the SRAM shared memory. Although leakage and read energy were decreased for the single port configurations, total energy consumption of STO for 16kB 16b 1p, 16kB 32b 1p and 64kB 16b 1p is 2.66x, 2.69x and 2.94x of the baseline, respectively. Adding an extra port per bank further increases the energy to 3.7x and 4.33x of the baseline.

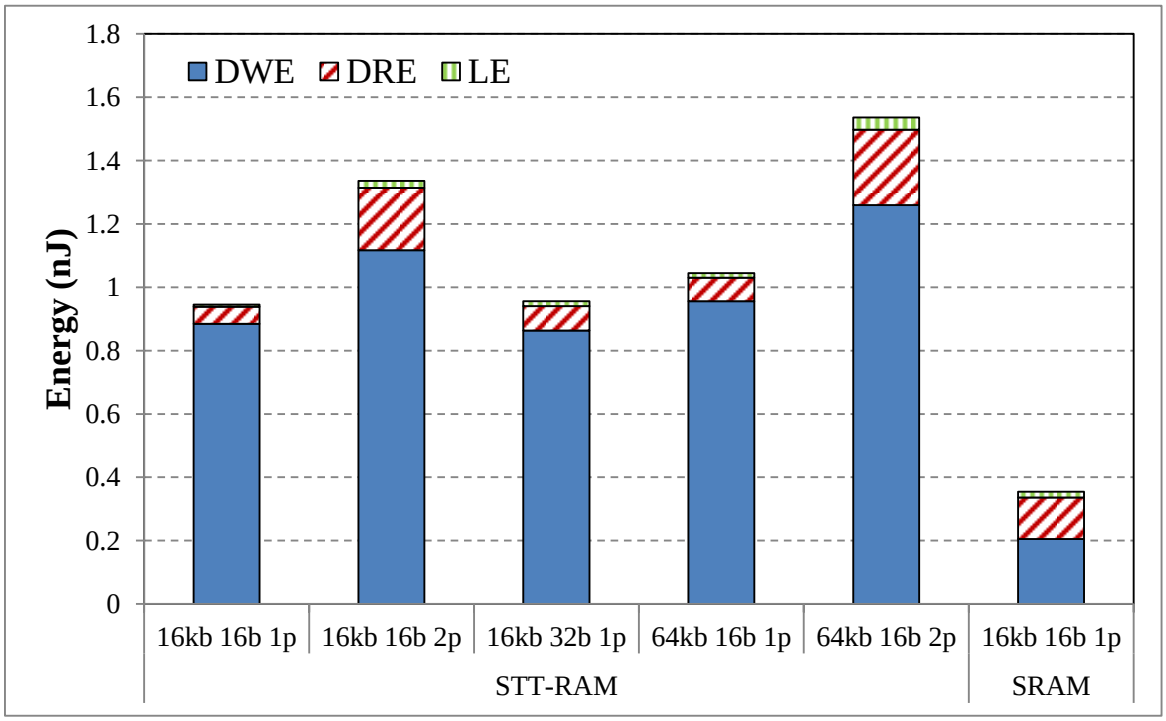
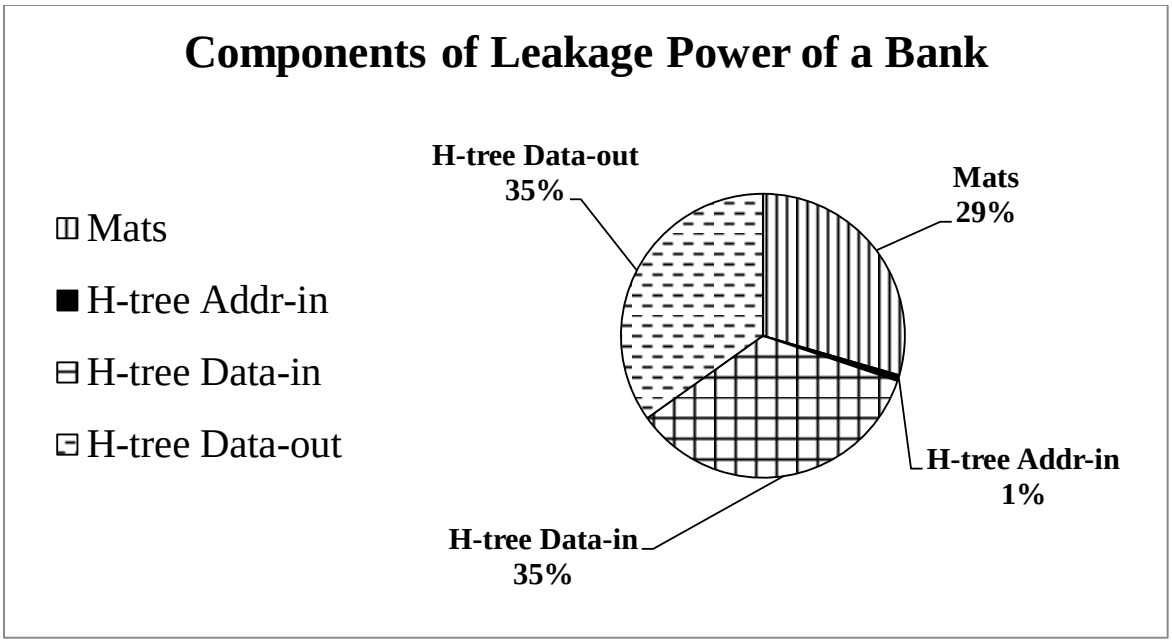
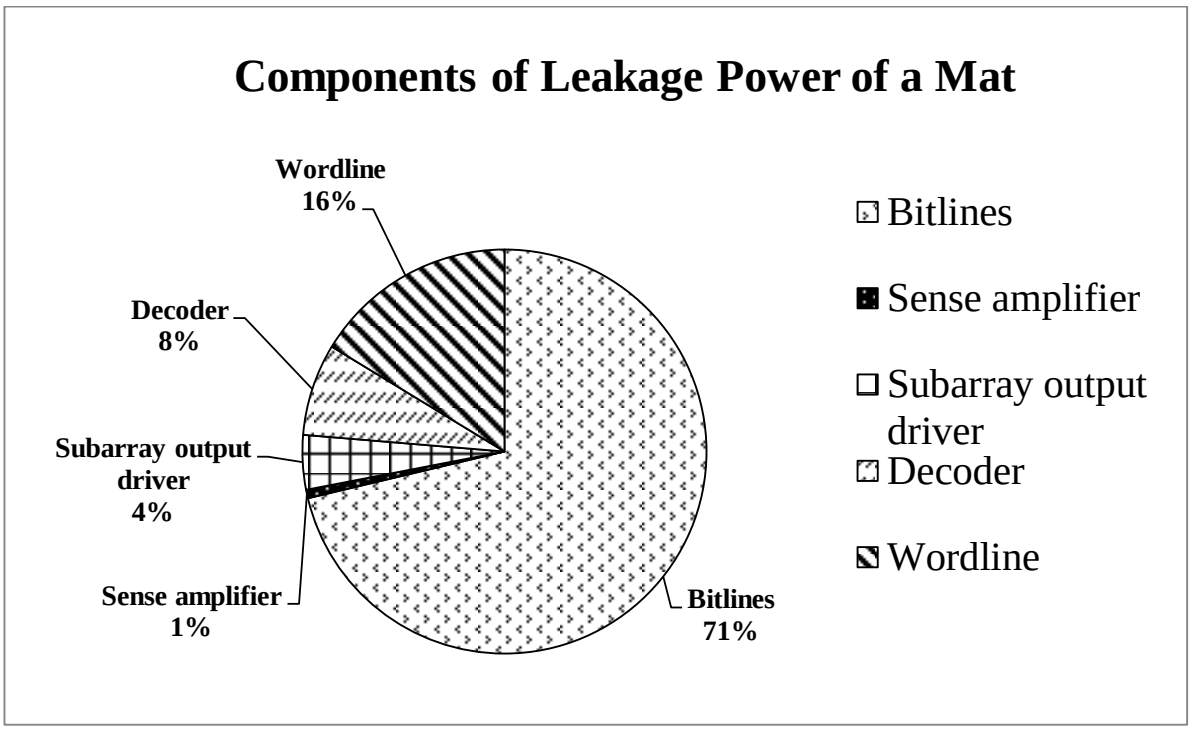


Figure 4.17: Leakage energy (LE), dynamic read energy (DRE) and dynamic write energy (DWE) of various configurations for the STO benchmark.



(a)



(b)

Figure 4.18: Components of Leakage Power

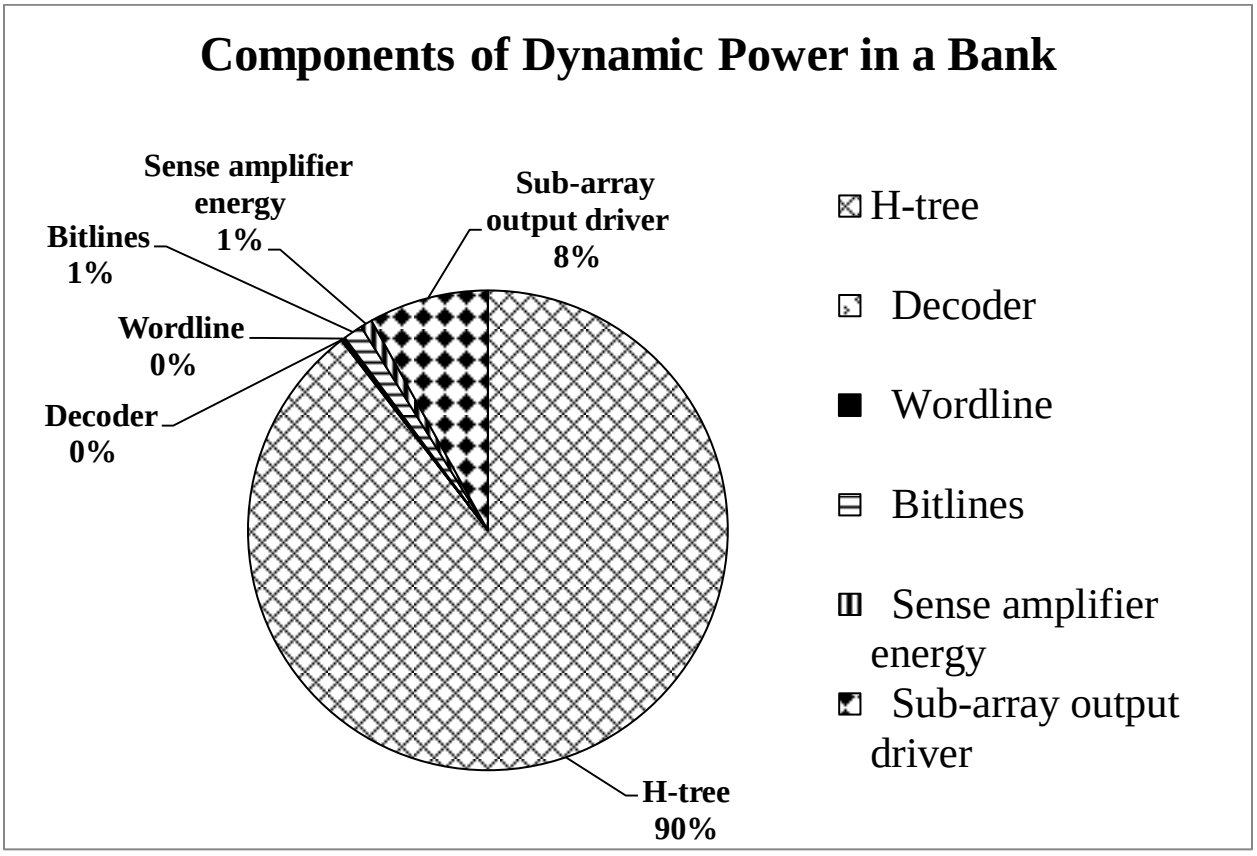


Figure 4.19: Components of Dynamic Energy

Chapter 5

Conclusion

In this thesis we have explored the use of STT-RAM for shared memory in GPUs, to study the impact of its latency, area and energy consumption. An equation was developed to estimate the performance of GPUs as a function of shared memory latency. Since the performance depends both on how much memory access and instruction execution overlap,and on the memory access latency, it was found that shared memory latencies which are up to 4 times slower for write access, resulted in only 2% percent loss of performance on average.

Five configurations of STT-RAM shared memory differing in capacity, number of banks per shared memory, and number of ports per bank were studied in detail. Except for the configuration with 16 kB capacity, 16 banks, and 1 port, which takes a hit in performance of 2%-16%, all other configurations resulted in higher performance than the baseline. More specifically, 1% to 21% more than the performance of the baseline SRAM system. All the STT-RAM configurations with a single port per bank occupy a smaller area than the baseline SRAM. The configurations with two ports per bank have two to three times the area of baseline.

The total energy consumed by benchmarks SRAD, NW and MM is less than the baseline, for one or more configurations. STO is write energy dominated, and STT-RAM's high write energy increases the total energy consumption of the benchmark. NQU also suffers from a large number of writes, hence the increase in write energy results in an increase in total energy consumed by the benchmark.

Because of the higher density of STT-RAM, for three out of five benchmarks it is possible to increase the capacity or number of banks to obtain a higher performance in spite of the higher access

latency, within the area and energy budgets set by the baseline SRAM shared memory system.For the more write-bound benchmarks, the substantial increase in write energy results in higher total energy consumption. Having two ports per bank increases the area and energy of shared memory significantly, whereas the same performance can be obtained by increasing the number of banks and keeping energy consumption and area relatively low.

Compared to earlier works on non-volatile memory in GPUs, more specifically STT-RAM, this work evaluates using STT-RAM based shared memory for performance, energy and area, and identifies favorable configurations. New workloads can be written, or existing ones can be modified, to make use of the increase in shared memory capacity with STT-RAM. The benchmarks were evaluated without any modifications to them, as it was beyond the scope of this work. For configurations with 16kB capacity 16 banks and 1 port per bank, and 16kB capacity 32 banks and 1 port per bank, the decrease in area was not large enough to study the effects of adding more SMs to the GPU.

For the reasons stated above, STT-RAM is a viable option for implementing the GPU shared memory. The area savings with STT-RAM shared memory area up to 50%. The high write energy of STT-RAM needs to be addressed by development of MTJs with lower switching current densities, even if it means higher switching time. Incurring a performance hit is acceptable, as the performance lost can be recovered by increasing the number of banks or increasing capacity while staying around energy and area budgets specified by the SRAM baseline system. Therefore, we were able to verify that GPUs are effective for hiding long access latencies. And that it possible to save up to 17% total energy by taking a performance hit, or stay around a few percent of energy budget and increase performance.

Bibliography

[1] Tor M. Aamodt, Ali Bakhoda, and Wilson W.L. Fung. Tutorial on gpgpu-sim: A performance simulator for massively multithreaded processor research.

[2] A. Al Maashri, Guangyu Sun, Xiangyu Dong, V. Narayanan, and Yuan Xie. 3d gpu architecture using cache stacking: Performance, cost, power and thermal analysis. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, pages 254 –259, 2009. doi: 10.1109/ICCD.2009.5413147.

[3] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163 –174, 2009. doi: 10.1109/ISPASS.2009.4919648.

[4] C. Chappert, A. Fert, and F. N. van Dau. The emergence of spin electronics in data storage. *Nature Materials*, 6:813–823, November 2007. doi: 10.1038/nmat2024.

[5] Rajagopalan Desikan, Charles R. Lefurgy, Stephen W. Keckler, and Doug Burger. On-chip stt-ram as a high-bandwidth, low-latency replacement for dram physical memories, 2002.

[6] Zhitao Diao, Zhanjie Li, Shengyuang Wang, Yunfei Ding, Alex Panchula, Eugene Chen, Lien-Chang Wang, and Yiming Huai. Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *Journal of Physics: Condensed Matter*, 19(16):165209, 2007.

[7] Zhitao Diao, Alex Panchula, Yunfei Ding, Mahendra Pakala, Shengyuan Wang, Zhanjie Li, Dmytro Apalkov, Hideyasu Nagai, Alexander Driskill-Smith, Lien-Chang Wang, Eugene Chen, and Yiming Huai. Spin transfer switching in dual mgo magnetic tunnel junctions. *Applied Physics Letters*, 90(13):132508, 2007. doi: 10.1063/1.2717556. URL `http://link.aip.org/link/?APL/90/132508/1`.

[8] Minh Q. Do, Mindaugas Drazdziulis, Per Larsson-Edefors, and Lars Bengtsson. Leakage-conscious architecture-level power estimation for partitioned and power-gated sram arrays. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 185 –191, 2007. doi: 10.1109/ISQED.2007.97.

[9] Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, Helen Li, and Yiran Chen. Circuit and microarchitecture evaluation of 3d stacking magnetic ram (stt-ram) as a universal memory replacement. In *DAC '08: Proceedings of the 45th annual Design Automation Conference*, pages 554–559, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-115-6. doi: http://doi.acm.org/10.1145/1391469.1391610.

[10] A. Driskill-Smith, S. Watts, D. Apalkov, D. Druist, X. Tang, Z. Diao, X. Luo, A. Ong, V. Nikitin, and E. Chen. Non-volatile spin-transfer torque ram (stt-ram): An analysis of chip data, thermal stability and scalability. In *Memory Workshop (IMW), 2010 IEEE International*, pages 1 –3, May 2010. doi: 10.1109/IMW.2010.5488325.

[11] Robert Eisberg and Robert Resnick. Quantum physics of atoms, molecules, solids, nuclei, and particles.

[12] Xiaochen Guo, Engin Ipek, and Tolga Soyata. Resistive computation: avoiding the power wall with low-leakage, stt-stt-ram based computing. In *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, pages 371–382, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0053-7. doi: http://doi.acm.org/10.1145/1815961.1816012.

[13] John Hennessey and David Patterson. A quantitative approach, fourth edition. *Computer Architecture*.

[14] Nagao Hosomi. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. 2005. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1609379`.

[15] Wei Huang, K. Skadron, S. Gurumurthi, R.J. Ribando, and M.R. Stan. Exploring the thermal impact on manycore processor performance. In *Semiconductor Thermal Measurement and Management Symposium, 2010. SEMI-THERM 2010. 26th Annual IEEE*, pages 191 –197, 2010. doi: 10.1109/STHERM.2010.5444293.

[16] ITRS. Itrs 2009. *International Technology Roadmap for Semiconductors*.

[17] David Kanter. Nvidia's gt200: Inside a parallel processor. *http://www.realworldtech.com/*.

[18] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68 – 75, 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1250885.

[19] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008. ISSN 0272-1732. doi: http://dx.doi.org/10.1109/MM.2008.31.

[20] Iong Ying Loh. Mechanism and assessment of spin transfer torque (stt) based memory. *Thesis (M. Eng.), Massachusetts Institute of Technology, Dept. of Materials Science and Engineering)*, 2009.

[21] NVIDIA. Cuda programming guide 3.1, 2010.

[22] NVIDIA. Cudazone. *CUDAZone*.

[23] H. Ohno. A hybrid cmos/magnetic tunnel junction approach for nonvolatile integrated circuits. In *VLSI Technology, 2009 Symposium on*, pages 122 –123, 2009.

[24] Barry Pangrle and Shekhar Kapoor. Leakage power at 90nm and below.

[25] David A. Patterson and John L. Hennessy. Computer organization and design: the hardware/software interface.

[26] R. Shankar. Principles of quantum mechanics.

[27] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Arithmetic_logic_unit.

[28] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Bottleneck.

[29] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Cache.

[30] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/CMOS.

[31] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Multi-core_processor.

[32] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/DRAM.

[33] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Electron_magnetic_dipole_moment.

[34] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Instruction_pipeline.

[35] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Magnetic_anisotropy.

[36] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Magnetic_moment.

[37] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Scoreboarding.

[38] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Scratchpad_RAM.

[39] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Static_random_access_memory.

[40] Source:Wikipedia. . URL http://en.wikipedia.org/wiki/Vector_processor.

[41] M.R. Stan and K. Skadron. Power-aware computing. *Computer*, 36(12):35 – 38, 2003. ISSN 0018-9162. doi: 10.1109/MC.2003.1250876.

[42] Source:Hyperphysics Website. URL http://hyperphysics.phy-astr.gsu.edu/hbase/spin.html.