Allocation Policy Analysis for Cache Coherence Protocols for STT-MRAM-based caches

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Pushkar Shridhar Nandkar

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Prof. David Lilja

Oct, 2014

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. David Lilja for the continuous support of my Master's study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in research as well as writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Ulya Karpuzku, and Prof. Pen Chung Yew, for their encouragement, insightful comments, and hard questions.

I thank my fellow lab mates in C-SPIN Group: William Tuohy, and Cong Ma, for the stimulating discussions. Also I thank my friends in Minneapolis, Manas Minglani, Ashwin Nagarajan, Nishant Borse and Shreyas Bhaban. I would also like to thank University of Minnesota Supercomputing Institute for letting me use the computing resources.

Last but not the least, I would like to thank my family: my parents Pramila Nandkar and Shridhar Nandkar, for supporting me throughout my life and my sister Aditi Nandkar for her encouragement.

Dedication

I dedicate this thesis to my parents Pramila Nandkar and Shridhar Nandkar

Abstract

Spintronic devices have demonstrated promising results to replace the traditional CMOS devices in Last Level Caches. Recent research have focussed on STT-CMOS hybrid caches and presented techniques to reduce leakage power and achieve performance benefit due to larger caches size that can be accomodated in the same footprint. Instead of using such hybrid caches, we use in-place STT-MRAM replacements for the complete cache hierarchy and show that we can achieve increased performance due to larger caches and significant power benefits due to decreased leakage. Further, we study different cache coherence protocols and with different allocation policies. Our preliminary results show that Non-inclusive protocols save write dynamic energy mostly due to reduced number of line fills compared to an inclusive protocol. We study the complete parsec benchmark suite and discuss the best allocation policy for each benchmark while considering the energy-delay trade off.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

CMOS based multi-core architectures have already gained base in embedded processors as well as general purpose computers. With the technology advancing every year, the CMOS based transistors kept getting smaller and smaller acknowledging what Moore predicted [2] in 1965. The Moore's law kept hold until now. However, CMOS technology is finding it difficult to sustain specially with the power wall [3] and the memory wall [4] dominating recent processor developments. Moreover, chip-multiprocessors (CMPs) which add multiple cores on a single chip aggravated the memory wall due to limited network resources and associated contention due to the shared memory space. As traditional CMOS based chips are facing these serious challenges, researchers for long have been looking out for alternatives devices. Many alternatives such as phase change random access memory(PRAM), resistive random access memory(RRAM) and magnetic random access memory(MRAM) are being explored to replace the existing CMOS based memory hierarchy. The non volatile alternatives offer low leakage and substantial area benefits. Perhaps, the most promising solution among them is the spin-transfer torque based magnetic random access memory as it shows compatibility with the existing cmos processes.

However, being non-volatile in nature, it suffers from high dynamic energy, primarily for write. It, being spin-based, changing the state of the device requires high currents to change the magnetic orientation of the electrons. It results in high write energy as well as the high write latency. To mitigate these performance hindrances, many architectural techniques are proposed in [5], [6], [7], [8], [9] for last level caches. These are mostly

based on hybrid caches or write buffers. Some of them [8], [10], [11] leverage the fact that non-volatility of the memory cell can be compromised since the cache entry will get replaced eventually. There are very few [12], [11], [13] which look at higher level caches. Moreover, these works do not evaluate the performance of a modern CMP. In any multithreaded CMP application, different caches will have multiple copies of the data. Due to sharing [14] and related coherence activities, data movement occur within these caches adding to network contention. Reference to a same cache line by multiple cores result in multiple updates in the various caches which ultimately leads to greater write energy. Factors like thread barriers, synchronizers which are critical components of multithreaded applications, also affect the IPC of a CMP.

In this work, we replace the CMOS based cache hierarchy with STT-MRAM based cache hierarchy. First, we analyze the effect of different allocation policies, based on inclusion property of coherence protocols, on different applications and understand its effect on IPC and Power. Next, we analyze the effect of larger caches on IPC, coherence traffic and on the power. Using larger capacity STT-MRAM L2s and smaller L1s, as compared to CMOS baseline, we achieve IPC improvement of 10.5% on an average for inclusive protocol and 4.6% for non-inclusive protocol along with power savings of 45%. Larger L2 decreases the off-chip traffic by 47%. Also, we observe that larger L1 decrease the coherence on-chip traffic by 45% resulting in savings in the network dynamic energy.

This works makes the following contribution:

- We propose the use of Non-Inclusive Protocols instead of Inclusive Protocol for Multi-core STT-MRAM based cache hierarchies to save L2 dynamic Power.
- We propose the use of STT-MRAM in L1 caches to reduce the leakage power.
- We evaluate the power, performance and network impact of replacing CMOS based caches with STT-MRAM based caches and show that it significantly reduces the power consumption and improves performance. We further evaluate the benefits of larger STT-MRAM caches on network contention. We conclude that smaller L1 STT-MRAM caches are better than larger counterparts because of the lower access latency

Chapter 2 briefly presents basics of STT-MRAM , sources of power dissipation and

the need for optimization. This chapter also discusses various cache coherence protocols with different allocation policies used in this work.

Chapter 3 explains the simulation methodology and various components used in these experiments for obtaining results. It briefly talks about the STT-MRAM modeling done in the simulators.

Chapter 4 analyses results for Parsec benchmark suite in detail for STT-MRAM based caches for its impact on Power, IPC and Traffic as compared to CMOS based caches.

Chapter 5 concludes the work by presenting incites of the analyses made in the thesis.

Chapter 2

Background and Motivation

We begin this chapter by describing the MTJ Device and explaining why we need the STT-MRAM in the cache hierarchy. We conclude it by detailed discussion of various Cache Coherence protocols used in this work.

2.1 STT MRAM basics

STT-MRAM is the second generation of Magnetic Random Access Memory(MRAM) which stores data in the form of magnetoresistance. The basic element of the STT-MRAM is the Magnetic Tunnel junction(MTJ). Fig. 2.1 depicts this device. It is a three layered stack, two ferromagnetic layers separated by a thin insulating layer. Out of the two ferromagnetic layers, one layer is fixed called as pinned layer and the other layer is free. The spin orientation of the free layer can be changed by passing current through the device. The current is such that the pinned layer spin orientation is not compromised. The fixed and the free layer can have the same orientation or opposite orientations. The resistance due to this magnetic orientation of the layers changes depending on that. When in parallel state, the resistance is less as compared to the anti-parallel state. These two states define the binary operation of the memory cell. The read operation is performed by sensing the resistance of the memory cell, whereas the write operation is performed by passing a current through the device which changes orientation of magnetic orientation of the free layer. The direction of the current determines whether the state will be parallel or anti-parallel.

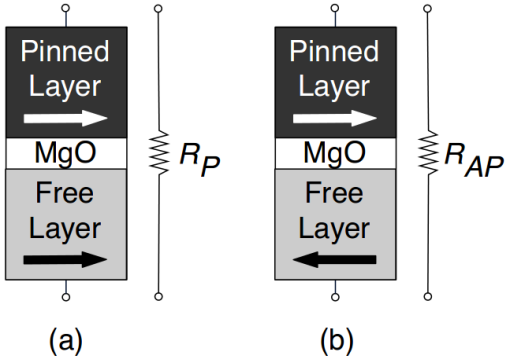

Figure 2.1: MTJ Device : (a) Parallel : low resistance, state '0' and (b) Anti-Parallel : high resistance, state '1'

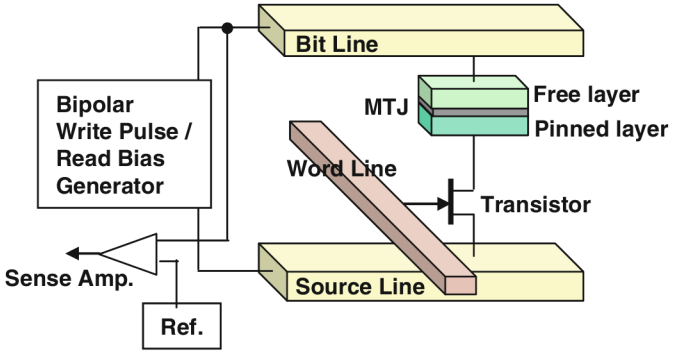

Figure 2.2: An illustration of STT-MRAM Cell

STT-MRAM memory cell is composed of one access NMOS transistor and one MTJ : 1T1J [15]. As Fig. 2.2 illustrates, the MTJ is connected in series with the access transistor. The operation of this access transistor is controlled by the wordline(WL).

2.1.1 Read Operation

A negative voltage is applied between the source-line(SL) and the bit-line(BL) with the access transistor turned on. The current passed through the series connection is enough to be sensed by the sense amplifier where is it compared with the reference current to determine state of the device.

2.1.2 Write Operation

A positive voltage is applied between the SL and BL for writing '0' and negative for writing a '1'. The current generated on the series connection is more than the threshold current which is enough to change the magnetic orientation of the electrons in the free layer.

2.2 Cache Coherence Protocols in Multi-threading environment

Coherence is the underlying hardware that programmers rely on for memory abstraction for communication between threads running on different cores in a shared memory space. In our experiments we use the MESI or the Illinois Protocol [16] which has four states **M**odified, **E**xclusive, **S**hared and **I**nvalid. In some variations that we describe and use later, the protocol looks like the MOESI protocol where O state makes it the owner of the cache line. Coherence protocols for multi-level hierarchy may further be classified based on the inclusion property the caches maintain with various other levels of caches. Namely, inclusive, exclusive and non-inclusive. This property is mainly for lower level caches with respect to higher level caches. Inclusive cache maintains the cache blocks present in the higher level caches. Exclusive caches do not cache the blocks present in the higher level caches. Non-inclusive caches neither force inclusion nor force exclusion. The non-inclusive protocols are generally bus based snooping protocols since no other cache maintains the information about where actually the data resides.

2.3 STT-MRAM Motivation

Inclusive protocols, since they need to maintain inclusion result in more line fills at L2 as compared to non-inclusive counter-parts. However, for non-inclusive caches, a bus-based snooping network will be needed since L2 may not have complete information of the data present in the cache hierarchy. Snoop-based protocol requires the node to be constantly active for snooping on bus for requests for the data blocks, it may have exclusive rights for. This will add to power dissipation [17] as well as require broadcast based networks to correctly model latency.

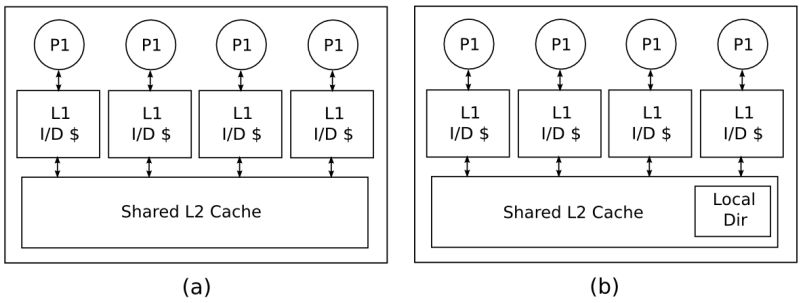

Figure 2.3: Cache Organization used (a) For Inclusive Protocol (b) For Non Inclusive Protocols

In inclusive protocols it is a given that L2 may have correct copy of the data or atleast have the information about the node which has the exclusive rights for that copy. L2 will take care of forwarding the request to the cache/node and the cache controllers do not need to have extra hardware mechanism for snooping. In order to avoid snooping for non-inclusive protocols and have L2 cache controller hold the information of the lines it does not have, we take a look at including small directories local to each L2 bank. In case the L2 bank does not have a valid entry, the request is forwarded to the local directory from where the request is forwarded to the concerned node or the main memory. The local directory is high associative cache.

CMOS based cache hierarchies may not include this directory since it may aggravate the already high L2 latency and increase the leakage power. We see that due to high density of STT-MRAM , we can fit a larger cache in the same CMOS cache footprint as well as include the local directory. In this work we evaluate how this local directory helps in reducing the number of writes to the L2 cache. Without loss of generality, we omit exploring ways to avoid L1 fills and stores.

In this work we explore the non-inclusive caches. In that we particularly look at the allocation policies mainly to reduce the number of writes to the L2 caches. We find that the writes to L2 being line fills are reduced in most of the cases for MESI NI protocol(See Sec. 2.6). This suggests that, for write sensitive technologies, we can make use of the non-inclusive nature of the coherence protocols. Instead of migrating the entire cache block from low retention region to high retention region [11] or from STT-MRAM to SRAM in hybrid cache [18], [10] we just save the state associated with the cache line to the local directory which is faster as well as leads to lower leakage.

2.4 Reasons for Cache line writes

There are various reasons which lead to writes in the cache. These reasons vary depending on whether the cache is a processor cache(L1) or a last level shared cache(L2/LLC) and depending on the inclusion policy that is applied.

Misses Both L1 and L2 misses will result in cache lines being brought in the cache. This leads to line fills. Writing of these lines in L2 will depend upon whether it maintains inclusion or not. If not, it will save the state instead of writing the lines to L2.

Stores Once the line is brought in the L1 Cache and used, a subsequent store will write new data to the line. In this case, only a word and not the entire line, is written to the cache. For a 64 bit system, a word write will be 8 bytes out the 64 bytes present at the line.

Writebacks Misses at L1 caches invariably result in eviction of the already present line. This leads to writebacks of these lines at L2 if the data has been modified at the concerned L1. Such lines can also be written at L2 if only the entry information is present at L2, in case of non-inclusive protocol, and the data is about to be deleted from the cache hierarchy due to this writeback.

Coherence For a shared last level cache, depending on what allocation policy is used, maintaining correct data at L2 cache becomes mandatory. This often results in multiple copies present in cache hierarchy. L2 may have a copy along with one or more L1s. A Read request on a line modified by other L1 will result in write at L2.

2.5 Coherence Protocols

Depending on weather to enforce inclusion or not, it can be decided whether to keep a copy at L2 or not. We come across such decisions on a number of occasions based on which we design a few allocation policies. The motive of these different allocation policies is to study effects of keeping the data on chip and the associated cost. In

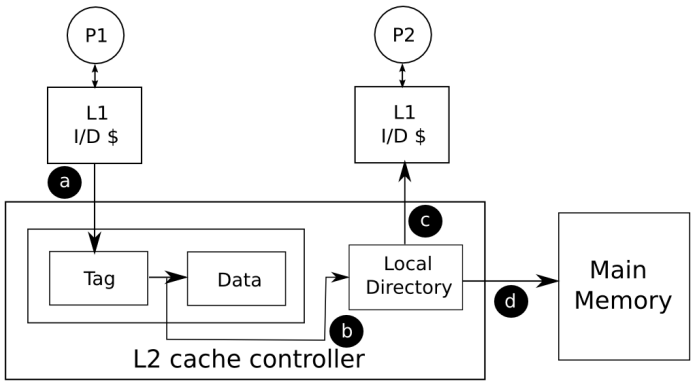

Figure 2.4: Request Flow

Chapter 4 we will see that non-inclusive protocols have similar performance compared the inclusive protocols but fair very well in terms of dynamic power consumption.

The exclusive and non-inclusive protocols that we will talk about maintain a Local Directory(LD) at L2 controller. As shown in Fig. 2.4 miss(a) at L2 will direct the request towards the LD(c). LD stores the state information about the data present in cache hierarchy. If the data is present at any of the L1s, the request is directed towards the correct L1(c) or directed towards the main memory(d). Sec. 2.6 details the different allocation policies.

Following are the typical **L1 Requests** that we will refer to interchangeably , while describing the protocols.

- **ReadS**:GETS:GET_INSTR : Ifetch as well as Load Requests which receive the data in **S**hared State. By default load request is served with a exclusive copy in the anticipation that loads may be followed by stores. Therefore, load requests for lines already present in other L1s will be served in shared state.

- **ReadX**:GETS : Load Requests which receive data in the **E**xclusive state since no other sharer requested the data at the same time.

- **WriteX**:GETX : Store Requests which receive data in the exclusive state and **M**odify it when received

- **Writeback**:WB : Writeback requests to L2 on a L1 replacements.

Following the various **L2 states**, the L2 controller will be in, take depending on whether the data is in L2 Cache or Local Directory(LD). Some protocols may not have MT or SLS states depending on when they decide to allocate data at L2.

- **I**: Neither in L2 nor in LD
- **SLS**: Correct copy at L2, L1 sharers exist
- **M**: Modified Copy (wrt main memory) at L2, No L1 Sharers
- **E**: Clean Copy at L2, No L1 Sharers
- **MT**: Stale Copy at L2, One of the L1 has modified it
- **ILS**: Entry in LD, L1 sharers exist
- **ILX**: Entry in LD, L1 exclusive holder exist

A L1 request on any of the L2 states will either result in serving the request by providing the data to the requesting L1 or forwarding the request to the concerned L1. It will result in either unblock messages to L2 or new data writes in L2 when the data is brought to the cache for maintaining inclusion.

2.6 Allocation policies

In this section, we describe the different protocols that we have used in this work. Each protocol has a different allocation policy. These allocation policies vary the number of writes to the L2 cache. Some workloads show similar number of write but the reasons for these writes vary. Table 2.6 shows the reasons for writes for various protocols. A '✓' shows that a write to L2 data array may occur for that particular event.

MESI Inclusive (MESI IN) L2 is inclusive of data present in the L1s, thus every line in L1 has a corresponding entry in L2 Cache. Every Request from L1 is allocated a cache line resulting in line fills (I GETS/GETX). L2 serves L1 requests when in states M, E, SLS. When in MT, L2 forwards the requests to the Exclusive L1 holder, which inturn serves the request and sends the data back to L2(MT GETS). This again results in L2 fill which helps L2 maintain correctness and the entry goes into SLS

Reasons For Writes										
MESI	MT GETS	I GETS	I GETX	ILS GETS	ILS GETX	ILX GETS	ILX GETX	MT WB	Dirty WB	Clean WB
IN	✓	✓	✓	-	-	-	-	✓	-	-
EX	-	-	-	-	-	-	-	-	✓	✓
NI	-	-	-	-	-	-	-	-	✓	✓
NE	-	✓	-	-	-	-	-	-	✓	✓
NIE	✓	✓	✓	-	-	✓	-	✓	✓	✓
NII	✓	✓	✓	-	-	✓	✓	✓	✓	✓
NEE	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
NEI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.1: Protocols along with various reasons for writes

state. Writebacks from L1 results in line fills for dirty data(MT WB). Shared Entry in L1 evicts silently and need not inform L2. L2 maintains a conservative list of sharers. Further, to maintain inclusion, replacements at the L2 cache will send invalidation requests to L1s which possess the same cache line. This results in the eviction of the cache line from the hierarchy.

MESI Exclusive (MESI EX) L2 is exclusive of the data present in L1. The data is either present in L1 or L2 but not at both. Therefore, the states MT and SLS don't exist in this protocol. L1 misses, do not allocate L2 cache. Instead, responses from the main memory are directly sent to the requesting L1 simultaneously allocating an entry in the local directory which finally goes to ILS or ILX states. Further requests for these entries are forwarded to the concerned L1 and the current requester is noted at L2. Replacements at L1 result in writebacks at L2. The writeback request, if entry not present in any other L1, results in a line fill at L2(Clean/Dirty WB). In this protocol, L2 maintains the correct list of sharer and each sharer informs the L2 before evicting. These L1 writebacks may result in replacements at L2 if the writeback data does not find any available space. Contrary to MESI IN, these replacements will not affect the data present at any of the L1s. They will simply evict the entry at L2. Local directory being a limited storage, the protocol has been modified to include replacements from the local directory which will result in invalidation to local owners or sharers. The data

from these L1s will be sent back to the main memory. However we do not observe such replacements since local directory is a high associative cache. Since this protocol maintains exclusion, the writes to L2 are bound reduce drastically than the MESI IN protocol.

MESI Non Inclusive (MESI NI) Similar to MESI EX, L1 Misses do not allocate L2 cache and consequently LD attains either ILS or ILX state. Unlike MESI EX, evictions from L1 result in write-backs at L2 if the cache line is not present in the L2. Therefore, L2 can attain either SLS state if there are other sharers present or it may go to M state in case it is a single data holder. Any L1 ReadX or WriteX request on such L2 entries, removes the entry and allocates it to the local directory(ILX) after sending the data to requesting L1. As a result, any request from other L1 is forwarded towards the exclusive owner or sharer of the data which results in a 3-hop return to complete the request. Any ReadS request from L1 in states SLS and M will be served by the L2 cache and move to SLS state. Replacements at L2 on such lines move the entry from L2 to the local directory storing just the meta data. Writebacks on SLS will not result in data write at L2. This protocol is more inclined towards being a exclusive one. Intuitively it will result in less L2 writes than others described below but more than the MESI EX.

MESI Non Exclusive(MESI NE) Explain the differences later in analysis and results In this protocol, every ReadS request on L2 miss(I GETS) will result in L2 cache line fill. This will help any application with more sharing data to take advantage of 2-hop L2 return instead of 3-hop L1 return. Similar to MESI NI, further ReadX or WriteX request on such entries will move the entry from L2 cache to local directory. Apart from allocation of the L2 cache on ReadS miss, the protocol behaves same as MESI NI with regards to replacements at L2 and L1 writebacks. This new allocation is another cause for replacement at L2. The replaced entry looses the data and gets allotted to local directory.

MESI Non In-Exclusive(MESI NIE) Every L1 request which is a L2 miss(I GETS and I GETX) will fill the cache line. Moreover, a ReadS(GETS) request on ILX entry results in a line fill at the L2. For such a request, the L1 controller, for this

protocol, needs to change since the responsible L1 has to send the data to both, the requesting L1 and the L2. Since the L2 allocates cache on a GETX request, the state MT is again introduced in this protocol to keep such requests. Similar to MESI NI, a GETX on SLS moves the entry to local directory but on the contrary a GETX on MT keeps the entry in the cache and the state remains MT after changing the identity of the L1 owner. Any replacements on cache line in states SLS and MT will move the entry to the local directory in states ILS and ILX respectively. L1 writebacks work similar to MESI NI. In addition to that, writebacks on MT will result in data array writes if the cache line from the requesting L1 is dirty. Here again, replacements on L2 entries results in relocating the entry to the local directory.

MESI Non In-Inclusive(MESI NII) This protocol is identical to MESI NIE. The only change being, a ReadX or WriteX(GETX) request on ILX results in allocating the L2 and subsequent line fill makes to go MT.

MESI Non Ex-Exclusive(MESI NEE) In addition to being identical to MESI NII, L2 is allocated when it receives ReadS(GETS) request in the ILS state. The request is forwarded to the concerned L1, which sends the data to both, the requesting L1 and the L2. L1 controller is modified for this very purpose. The entry moves to the SLS state which serves further L1 requests.

MESI Non Ex-Inclusive(MESI NEI) The last protocol that we describe allocates L2 on every L1 request(GETS/GETX). The entry may be present in the local directory(ILX/ILS) or may not be present in the controller at all(I). The entry will move to either MT or SLS state depending on the request it satisfies.

Chapter 3

Experimental Setup

This chapter explains the experimental setup used in this work. It describes the simulator modelling, the simulation methodology used for running experiments and methods used gathering statistics and presenting results.

3.1 Simulator Modeling

We have modeled STT-MRAM in CACTI [19] and gem5 [20]. The power and access times numbers have been extracted from CACTI. These numbers are further pluged in gem5 to obtain IPC and event statistics for dynamic power and network traffic calculations.

3.1.1 CACTI

CACTI [19] is used widely in computer architecture research community for cache timing and power estimation. It is not a highly accurate tool like HSpice but gives us fair estimation about the power and time requirements for cache designs. It is used for CMOS based SRAM designs as well as DRAM designs. We have modified the tool to model the STT-MRAM based cache designs similar to the way described in [21].

Figs. 3.1 and 3.2 shows the difference between a SRAM and STT-MRAM basic cell. A SRAM cell has two access transistors whose gate capacitances form the load on the decoder driver. In case of STT-MRAM cell, this load is reduced to just one transistor. The wire-length almost reduces to half. Moreover, the SL and BL need

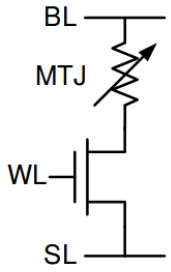

Figure 3.1: STT-MRAM Cell

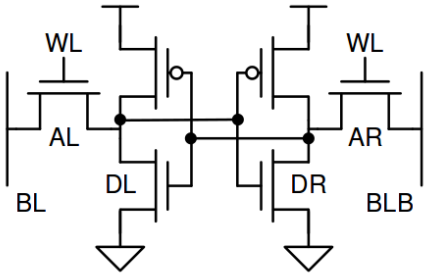

Figure 3.2: SRAM Cell

not be precharged every cycle reducing the precharging delay. Also, this precharges only to a point till we achieve the voltage differences as mentioned in Sec. 2.1.1. This decreases the read and write latency. This is true if we compare same cache sizes for both SRAM and STT-MRAM . However, if we compare a SRAM cache with a 4x large STT-MRAM cache, we find that it requires increased decoding and reading circuitry which will increase the access latency as well as the area.

Area & Latency

CACTI being mainly used for SRAM and DRAM, the current sensing schemes are not available. The voltage-divider sensing scheme used in NVSim [22] has been used here. The way CACTI models and generates sub-arrays,mats and banks for a cache is different from NVSim. Plus, NVSim does not include the Energy-Delay-Product (EDP) optimization for overall cache operation. Hence, to have better comparison between the two caches, STT-MRAM and SRAM, we added the voltage-divider sensing scheme used in NVSim to CACTI. The write circuits remain the same. We have used the cell describe in [1]. The number of access transistors is changed to one. We use a more conservative approach in our work than Table 3.1 and Fig. 3.3 from [1]. For e.g, in L1 we have used a cell with a size of $30F^2$ and a 5ns write pulse. Whereas, for L2 we have used a cell with a size of $20F^2$ and 7ns write pulse. The cell size is plugged in the CACTI technology description files.

The reduced cell size of STT-MRAM cell is certainly a major advantage with it comes to calculation of the area. 32nm Technology with $30F^2$ area [1] for STT-MRAM cell certainly shows almost 4x improvement over same sized SRAM cache, shown in

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

Table 3.1: STT-MRAM Parameters [1]

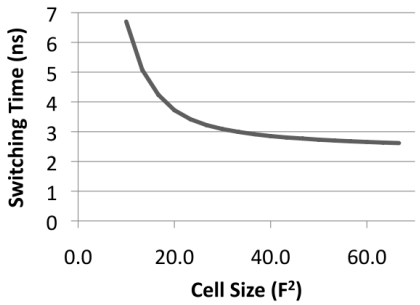

Figure 3.3: Switching time as a function of cell size [1]

Table 3.3. The latency also increases as the cache size increases for STT-MRAM based caches. This is attributed to the fact that the decoding and sensing circuits required for larger caches make it slower even though the interconnect latency may not be dominated as is the general case of larger SRAM caches. The latency of 128kB STT-MRAM and 64kB SRAM are around the same and have considered the same way in our experiments.

Energy/Power

The read energy of the STT-MRAM cell is comparable to CMOS SRAM cell design since it includes the energy consumed in data array peripherals and tag array comparators and dominated by H-tree routing in larger sized caches. The bit cell write energy is 0.3pJ [1]. This value is added to the write energy for dynamic power calculation along with the peripheral circuitry needed to write the cell. A CPU store will write 64 bits while a line fill will write 64 bytes of data. We separate both the events and calculate energy accordingly. A zero leakage power is considered for the STT-MRAM cell array, be it a tag array or a data array. All the peripheral circuits will leak throughout the

duration of the program.

3.1.2 GEM5

gem5 [20] is a full system, event driven simulator. It supports multiple ISAs on modular simulation framework which is widely used by computer architecture researchers. It models two different memory models, *Classic* and *Ruby*. The *Classic* model provides a fast memory system, while the Ruby model provides a flexible system which is simulates variety of cache coherent memory systems accurately. In this work, we use this *Ruby* model in gem5 for simulating the cache hierarchy and various coherence protocols using SLICC[23].

L1 access latency

The access time for each of the cache (See Table 3.3) is obtained from CACTI. One cycle of routing from CPU core to the cache is added to it and together considered as access latency for L1. The cycle time from CACTI is used as response latency for requests forwarded from L2. L1 has a faster response time hence accesses data and tag in parallel. This consumes extra energy on a miss when it access the invalid data before the tag array match. In contrast, L2 is considered to have a sequential access. Once the tag access confirms that the entry is invalid, the access request gets forwarded to local directory. At local directory, this request is either served by forwarding it to the correct L1 or considered a miss and allocated MSHR for the same. The dynamic power required and the time spent in such a response and similar other responses is correctly modeled in using gem5.

Write Latency

The write pulse latency of each STT-MRAM cache is modeled as a part of Cache Controller(L1/L2) in the SLICC compiler. This is done by adding a few extra states and transitions in the cache controller. A pseudo timer is used to determine when the writing finishes. Whenever the cache needs to write to the Data Array on reception of data block, the timer is started to model the write pulse. The write port is blocked for further write requests. As soon as the timer finishes(5ns for L1 and 7ns for L2) the

cache line moves to a stable state and can accept further requests.

Allocation Policies

The Ruby memory system use SLICC compiler to generate Cache controller C++ files which are further compiled into gem5. Each controller has states, actions, events and transitions. A state can be stable or transient state. A cache line in stable state is able to serve any request directed to it. A cache line in transient state is blocked from serving any requests until it receives any message it is waiting for. It goes to stable state when it receives either unblock message or acknowledgement message. Whenever a controller receives any message it generates an event. For e.g *GETS*. This triggers few actions that are taken by the controller. These action may be writing the data to the cache or send the data to the requestor or forwarding the request to the L1 cache or requesting the main memory. After any of the actions, depending on whether the controller will wait for any ack or not it will go to a transient or stable state.

This flexibility provided in Ruby to model the coherent systems helps us in modeling the various protocols with different allocation policies. These policies are already explained in Sec. 2.6. The main difference is the events that result in data writes at L2. The reasons for these writes are summarized Table 2.6.

Processor	2 GHz processor Out-of-Order, ISA-ALPHA
L1 Cache	64 KB per-core(SRAM) or 64/128/256 KB per-core(STT-MRAM) (private) I/D cache, 2-way 64B block size, write-back, 4MSHRs/WBs, 5ns write-pulse
L2 Cache	1MB (SRAM) or 4MB (STT-RAM) bank, shared, 8-way, 64B block size, 8MSHRs/WBs, 7ns write-pulse
Main Memory	1GB, 120 cycle access

Table 3.2: System Parameters

Tech	**Cache**	**Size**	**Associa-tivity**	**Latency** (ns)	**Read Energy** (nJ)	**Write Energy** (nJ)	**Leakage** (mW)	**Area** (mm^2)
CMOS	L1	64kB	2	0.71	0.024	0.030	25.06	0.241
	MSHR/WB	256B	FA	0.33	0.013	0.013	1.55	
	L2	1MB	8	4.47	0.133	0.165	89.6	5.23
	MSHR/WB	512B	FA	0.34	0.0133	0.014	1.56	
	LD	8kB	16	2.14	0.01	0.023	3.56	0.038
STT	STT256 -L1	256kB	2	1.02	0.037	0.210	6.52	0.311
	STT128 -L1	128kB	2	0.80	0.019	0.202	6.05	0.171
	STT64 -L1	64kB	2	0.40	0.011	0.176	2.94	0.046
	MSHR/WB	512B	FA	0.34	0.0133	0.014	1.56	
	L2	4MB	8	5.35	0.213	0.369	14.29	5.06
	MSHR/WB	1024B	FA	0.35	0.0134	0.0152	1.6	
	LD	8kB	16	2.14	0.01	0.023	3.56	0.038

Table 3.3: System Configurations

In Ruby, we have modeled a cache controller comprising of a Tag Array, Data Array, Writeback Buffer and MSHR. The local directory used for Exclusive and Non Inclusive protocols is modeled as well. WB and MSHR are modeled as fully associative caches. On the other hand, local directory is a high associative cache which stores all the meta data for an entry. The line size is 2-bytes.

Table 3.2 shows the parameters that we have used for SRAM and STT-MRAM based systems. Table 3.3 gives precise energy, latency, leakage and area numbers obtained from CACTI for each cache component used in the our experiments. We use the 32nm ITRS technology for this calculations. L1 cache use the HP(High Performance) cells for the data array whereas we use the LSTP(Low Stand-by Power) cells are used for data array in the L2 cache.

3.2 Simulation Methodology

This work is evaluated using the GEM5 Simulator [20] running the PARSEC benchmark suite [24]. A 2GHz, four-core out-of-order processor is modeled. A sampling technique similar to SMARTs [25] is used for reducing the runtime of the simulation without compromising its accuracy. We create checkpoints using GEM5 by running simple

atomic CPU model at every 50 billion cycles of the atomic CPU. Then, for a complete run of the checkpoint, we switch the CPU model from simple timing to detailed out-of-order and back to simple timing, periodically. Timing simple CPU periods between the detailed ones help in fast-forwarding the simulation. The timing and the detailed CPU model use the same Ruby memory model. Therefore, the caches and related dynamic structure remain active and maintain their state throughout the checkpoint run. Apart from this, to increase throughout of the simulation runs, we runs these checkpoints in parallel, on hundreds of cores using GNU Parallel [26]. These checkpoints are taken from the parallel Region-of-Interest (ROI), compiled from the source of each benchmark using [27].

We collect statistics for the period in which the detailed out-of-order CPU runs, interleaved between the simple timing CPU runs. Such detailed CPU period is a single sample similar to the SMARTs strategy. This methodology helps us cover the complete benchmark in a short amount of time as well as consider the various phases of distinct behavior of the benchmark. The data collected for such samples is used to estimate performance and rates of cache events needed for estimating power dissipation and network traffic. To verify the accuracy, we simulated complete run for one of the configurations in Table 3.3 and find that the sampled runs are quite accurate. Confidence intervals for 95% confidence were computed using the techniques presented in [28] and we find that for most of the data presented in Chapter 4 have a very tight confidence interval, around 0% to 3%. This technique is statistically verified in one of our works [29].

3.2.1 Workloads

Princeton Application Repository for Shared-Memory Computers[24](PARSEC), is a set of parallel programs used for studying Chip-Multiprocessors (CMPs). These benchmarks include state-of-the-art algorithms used in scientific applications and provide diversity in locality, data sharing, synchronization and off-chip/on-chip traffic which is needed for our analysis of these protocols. Out of various datasets like simsmall, simmedium, simlarge and native, we present the results for simmedium dataset. The cpu execution time for simmedium is 3 to 5 seconds whereas it is 10 to 20 seconds for simlarge. We find that our protocol policies changes do not affect the performance across datasets. We did not simulate native dataset since it is very large, only meant

Benchmark	Description	Problem Size
blackscholes	calculates portfolio price using Black-Scholes PDE	16,384 options
bodytrack	computer vision, tracks 3D pose of human body	4 frames, 2,000 particles
canneal	synthetic chip design, routing	200,000 netlist elements
dedup	pipelined compression kernel	31 MB
facesim	physics simulation, models a human face	372,126 tetrahedra, 1 frame
ferret	pipelined audio, image and video searches	64 image queries, 13,787 images
fluidanimate	physics simulation, animation of fluids	100,000 particles, 5 frames
freqmine	data mining application	500,000 transactions
streamcluster	kernel to solve the online clustering problem	8,192 input points
swaptions	computes portfolio prices using Monte-Carlo simulation	32 swaptions
vips	image processing, image transformations	2,336 2,336 pixels
x264	H.264 video encoder	32 frames

Table 3.4: Parsec Benchmark Suite

for real machines and not simulators. Table 3.4 lists these program that were simulated from the PARSEC benchmark suite.

3.2.2 Calculation Methodology

Each sample collected is a 0.25ms of detailed out-of-order CPU run. The total number of samples in each benchmark vary depending on the length of benchmark. Shorter runtime benchmarks like canneal have 200 samples, whereas the longest benchmark freqmine has 4350 samples.

Performance

The performance impact of the system is measured using the instructions-per-cycle (IPC) for different protocols with various configurations. The IPC is the weighted harmonic mean of IPCs over all the samples in the benchmark.

Power

The power consumption results are presented by calculating the dynamic and leakage power dissipated per sample. The dynamic power is calculated by counting the number of accesses to each component and summing up the energy spent in each of these accesses. The energy per access is given in the Table 3.3. This energy spend per sample is divided by the total simulated seconds(0.25ms) to get the dynamic power. Leakage

power is obtained directly from CACTI. We also present number in terms of dynamic energy per instruction.

Traffic

The network traffic is presented in two ways. First, the number of messages transferred per instruction and second, the number of bytes transferred per instructions. The number of messages shows how each of this protocol allocation policies change the way and to which destination, the messages are transferred. The number of bytes per instruction give an idea on how the total traffic changes on an average for each protocol or for each configuration. We differentiate the traffic messages into different type depending on what kind of message it carries. Most of the traffic is due to misses and some of it is contributed by coherence. It changes depending on the cache size.

Chapter 4

Results and Analysis

This chapter presents the results for the various coherence protocols and analyze them for every benchmark program. We analyze number of writes for each protocol, the distribution of the dynamic energy, ways to get better performance. We find the best protocol to work with for each workload and finally some discuss about network traffic due to larger size caches.

4.1 Power Analysis

4.1.1 Leakage Power

	L1 leakage	L2 leakage		Total leakage	
Configs	MESI	MESI	MESI NI	MESI	MESI NI
CMOS	112.6	102.1	116.2	214.6	228.8
STT64	24.24	27.1	41.3	51.34	65.54
STT128	36.7	27.1	41.3	63.8	78
STT256	38.56	27.1	41.3	65.66	79.86

Table 4.1: Leakage Power (mW)

This section studies the dynamic and the leakage power of the cache hierarchy. In any Chip-Multiprocessor, the cache hierarchy is the largest source of leakage power. STT-MRAM is used in our the cache hierarchy to decrease this leakage power. Table 4.1 shows

the leakage power for various configurations that we have used for our experiments. Addition of the two will give us the total leakage power dissipated in the system. As shown in Table 4.1, L2 itself is not the largest source of leakage power, in fact, L1 and L2 have quite comparable leakage. This is because, there are four L1s and each use high leakage HP cells needed for faster operation. Whereas, L2 uses LSTP cells which have very low leakage power dissipation. The peripheral circuitry used in these caches also contributes to the leakage. Moreover, the exclusive and non-inclusive protocols have cmos-based local directory added as another component which adds to the leakage power.

4.1.2 Dynamic Power

Using STT-MRAM based caches decrease leakage, but as discussed in Sec. 2.1 writes to STT-MRAM caches result in higher dynamic energy. The protocols discussed in this work differ in the way they write to L2 cache. As we show later, the L2 dynamic energy is mostly dominated by the L2 writes for STT-MRAM based caches. However, this dynamic energy is lower than that of the L1 cache since number of accesses and overall activity in L2 are far less than any of the L1 cache.

L2 Data Write Analysis

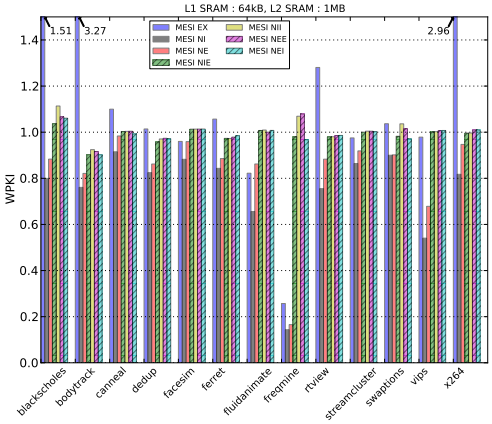

Figure 4.1: CMOS WPKI

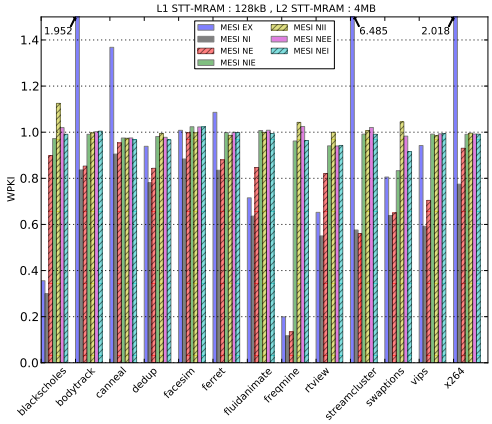

Figure 4.2: STT128 WPKI

Fig. 4.1 and Fig. 4.2 show the number of writes per kilo instruction for each PARSEC

benchmark program for CMOS and STT128 (see Table 3.3) respectively. The writes are normalized to the MESI IN protocol for respective configurations. In the figures, each bar represents each protocol. Depending on the allocation policy that we apply, the writes vary. As we increase the events when the L2 cache gets written/allocated the number of writes increase.The number of writes is least for MESI NI protocol for every benchmark followed by the MESI EX protocol for most of the workloads. For *bodytrack, canneal, streamcluster* and *x264*, MESI EX shows a very large write count. Maintaining exclusivity often leads to increased number of evictions and line fills if the cache line is referred again. Therefore, exclusive protocol is not beneficial enough when it comes to the number of writes to the cache.

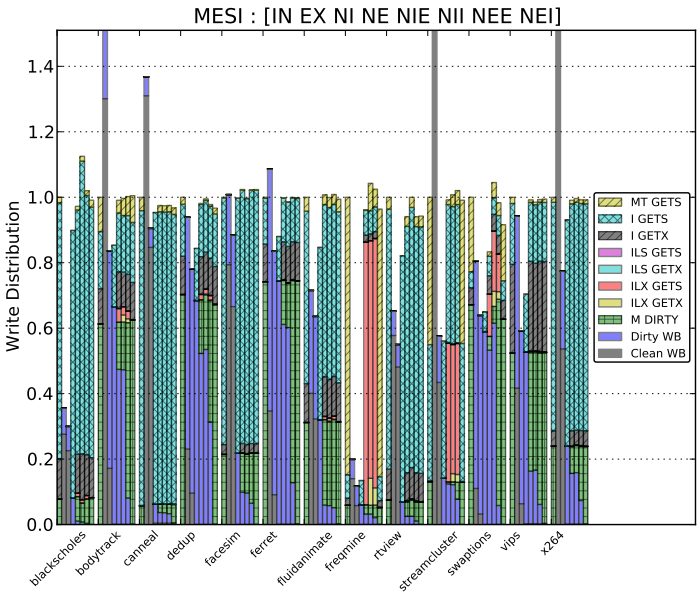

Figure 4.3: Normalized Write Distribution for STT128

Fig. 4.3 demonstrates in detail how the increase in the types of write allocate events increases the the number of writes to the cache. The MESI NI and MESI EX are only writeback events that result in writes to the cache. While MESI EX maintains exclusivity with L1 cache, MESI NI allocates the cache for every writeback even when the data is present in other caches. As a result, further L1 requests can be served by the L2 cache. This strategy makes the number of writes the least. As we move on to add more events for further protocols, the number of writes increase. The MESI NIE

protocol shows that it has almost the same number of writes as MESI IN protocol for all the workloads barring *bodytrack, freqmine* and *swaptions*. In fact, the MESI NIE, MESI NII, MESI NEE and MESI NEI have more or less similar number of writes for all the benchmarks. Even the reasons for these writes are similar. The number of writes due to I.GETS and I.GETX are almost the same for all the benchmarks. As one looks from protocol MESI NIE to MESI NEI, the writes due to writebacks on ILS/ILX decrease and writebacks on MT increases, however, the total remains the same. Only *freqmine, streamcluster* and *swaptions* show different behavior. *Freqmine* and *streamcluster* have a lot of producer-consumer data which is evicted from L2 due to replacements. It is then brought back on a L1 request result in a L2 write(ILX.GETS/ILX.GETX).

Overall, the L2 writes reduce for non-inclusive protocols, particularly more for the MESI NI by 36% on average for STT-MRAM based caches. It has the least number of writes for all the benchmarks. Six of the benchmarks show around 20% decrease in the number of writes, five of them show around 50%, *blackscholes* show 70% decrease while *freqmine* shows the maximum decrease i.e 87%.

L2 Dynamic Energy/Power

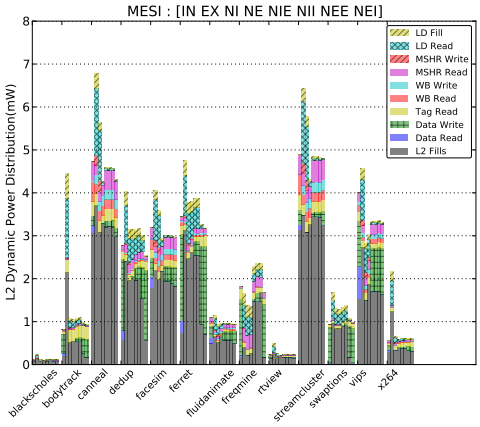

Figure 4.4: L2 Dynamic Power Distribution For CMOS

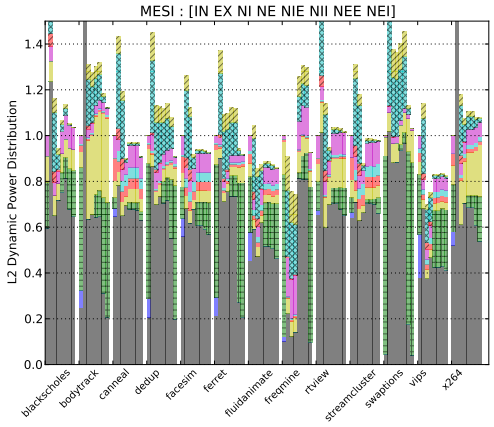

Figure 4.5: Normalized L2 Dynamic Power Distribution For CMOS

Fig. 4.5 and Fig. 4.7 show the distribution of the dynamic power dissipated in the L2 cache for all the protocols normalized to MESI IN for CMOS and STT128 configurations

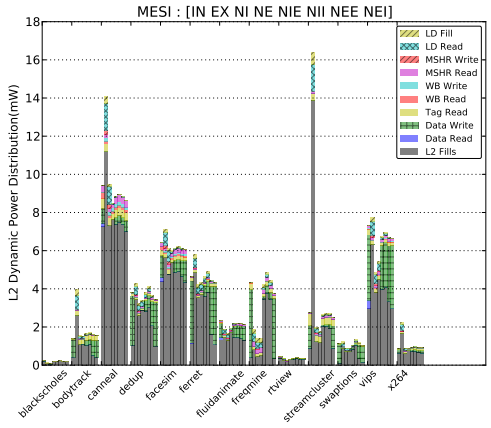

Figure 4.6: L2 Dynamic Power Distribution For STT128

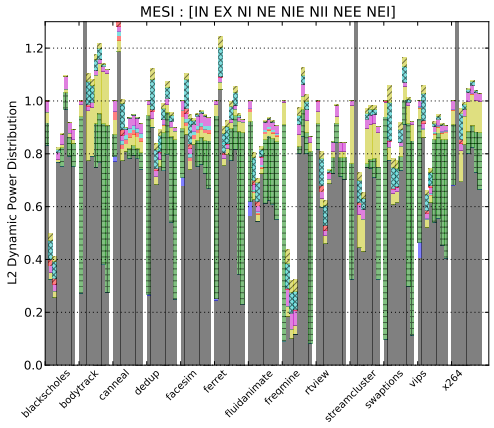

Figure 4.7: Normalized L2 Dynamic Power Distribution For STT128

respectively. The dynamic power include energy spent not only in the data and tag arrays but also in the different cache components. These components include MSHR, Write Buffer and Local Directory. The read and write energy requirements are given the Table 3.3. As Fig. 4.5 demonstrates, Exclusive Protocols are perform worst for almost all the benchmarks when it comes to Dynamic Energy. As the protocol goes from Exclusive to Non-Inclusive and further upto inclusive, the energy spent in the local directory decreases. This is the case for most of the benchmarks. For benchmarks *blackscholes, canneal, facesim, fluidanimate, rtview, streamcluster* and *vips* this trend stops at MESI NE. For the protocols thereafter the L1 request for these benchmarks is served by L2 cache without accessing the local directory. Except for *freqmine* and *vips*, no other benchmark performs well for non-inclusive protocol for CMOS baseline configuration. On an average the dynamic power at L2 for NI protocol increases by 7% as compared to IN protocol.

The trend is however different when it comes to STT-MRAM based caches. The dynamic energy for L2 caches is dominated by L2 data writes and line fills, evident in Fig. 4.7. A substantial amount of energy is spent in the local directory for CMOS. However, the domination of write energy in STT128 makes the local directory energy look negligible. The trend for dynamic energy in local directory is the same even for larger STT-MRAM based cache. Here again the exclusive protocol is the worst performer. The MESI NI performs well for most of the workloads while it is at par

with the MESI IN for canneal, facesim and x264. This further shows that non-inclusive protocols will help reduce power consumption at the L2 cache. On an average, for STT128 the dynamic power at L2 for NI protocol decreases by 25% as compared to IN protocol.

CMOS system showed that the complexity and hardware included in the non-inclusive L2 does not bode well when it comes to power dissipation. However, for STT-MRAM based system, a protocol which reduces writes definitely achieves the highest power gains since the write energy dominates the power consumed in the added hardware complexity.

L1 Dynamic Energy/Power

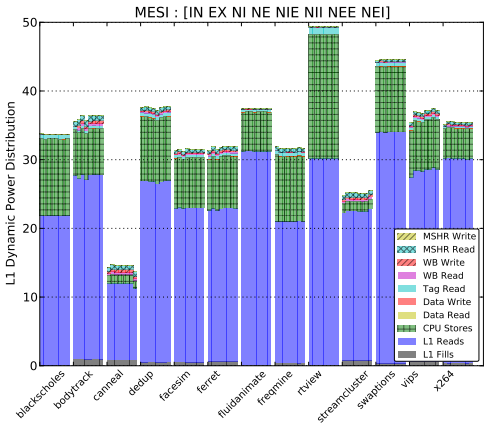

Figure 4.8: L1 Dynamic Power Distribution(mW) For CMOS

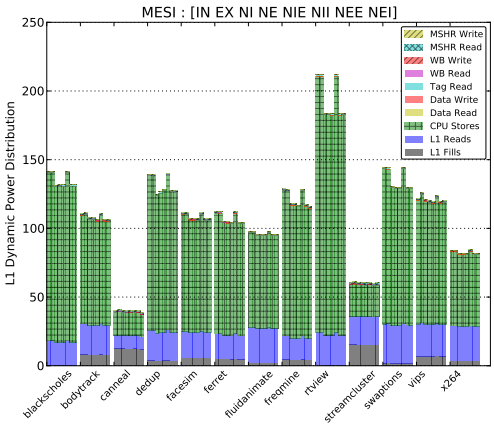

Figure 4.9: L1 Dynamic Power Distribution(mW) For STT128

Next, we take a look at dynamic power for L1 caches for all the protocols. Fig. 4.8 and Fig. 4.9 demonstrate that the dynamic energy does not change across protocols for both the systems. The L1 cache controller is changed for two protocols where instead of sending acknowledge message to L2 controller, it sends the data when it receives a forward request. This does not affect L1 performance and it remains the same over all the protocols. In CMOS, reads dominate the dynamic power, whereas CPU stores dominate the dynamic power dissipation in STT128 . We can observe almost 2 to 4 times increase in the dynamic power dissipation. The read energy of STT128 L1 is

comparable to CMOS L1. However, the power dissipated due to reads which is around 70% to 80% of the total dynamic power for CMOS L1 becomes around 10% of the total dynamic power for STT128 L1.

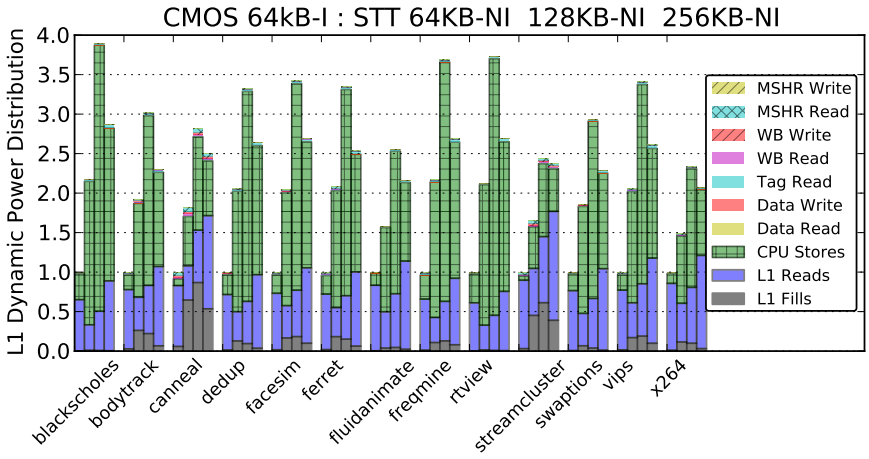

Figure 4.10: Comparison of L1 Dynamic Energy. Normalized to MESI Inclusive for CMOS baseline

Fig. 4.10 shows the dynamic energy for different L1 caches sizes, normalized to CMOS baseline. Here we compare the CMOS MESI IN with MESI NI for STT-MRAM since in Sec. 4.1.2 we saw MESI IN performed best for CMOS while MESI NI performed best for STT64 . STT64 performs better in comparison with other STT-MRAM based systems. Although the energy dissipated is still more than the CMOS system, primarily due to the high write energy. The dynamic energy is dominated by CPU stores. Store dominated benchmarks like *rtview*, *freqmine* and *blackscholes* show more than 3.5times increase in the L1 dynamic power consumption for STT128 . The energy increase for STT64 is around 2 times. There is a decrease in the power consumption in case of STT256 as compared to STT128 primarily because of high access latency leading to low IPC, hence less number of stores executed per sample. It can be concluded that STT64 configuration consumes the least amount of dynamic power for all the benchmarks.

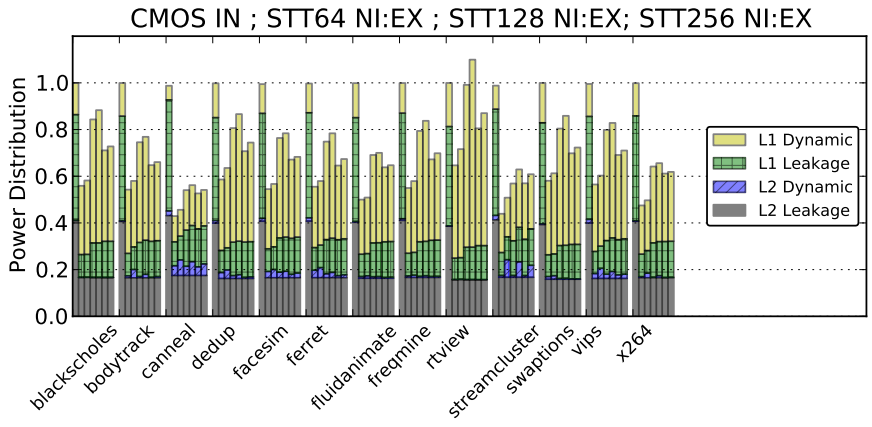

Figure 4.11: Comparison of Total Power. Normalized to CMOS MESI IN

Finally, we look at the total power dissipation in the system in Fig. 4.11. Normalized to CMOS MESI IN, it compares the total power dissipation over all the STT-MRAM based systems for MESI NI and MESI EX protocols. Except *rtview*, every other benchmark performs well for this metric for configurations. Even rtview performs better than CMOS for STT64 configuration. For every benchmark, the MESI EX shows more dynamic energy consumption than MESI NI both in L1 and L2. However, it is always less than the total energy consumed in a CMOS based system. We definitely achieve over 40% decrease in the total power dissipation for all the benchmarks for STT64 configuration. Canneal, streamcluster and x264 show more than 50% savings. Canneal shows the highest savings with 57% and 55% for MESI NI and MESI EX respectively. On an average it is 45% power savings for STT64 MESI NI as compared to CMOS baseline MESI IN over all the benchmarks. Thus, it can be concluded that even though the write energy dominates the power consumption for both L1 and L2, the benefits due to zero leakage energy(STT-MRAM) supersedes the system having high leakage power and lower dynamic power(CMOS).

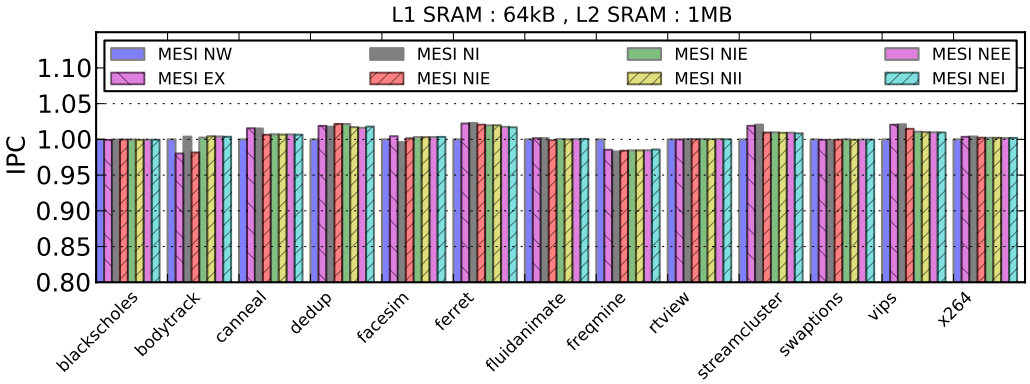

Figure 4.12: IPC for CMOS. Normalized to MESI Inclusive

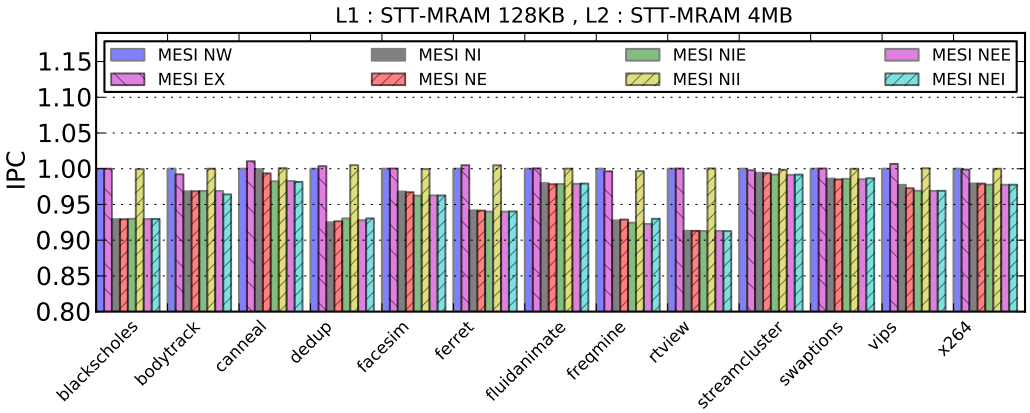

Figure 4.13: IPC for STT128 . Normalized to MESI Inclusive

4.2 IPC Analysis

Fig. 4.12 and Fig. 4.13 show the IPC trend over different protocols, normalized to MESI Inclusive protocol for CMOS and STT128 respectively. The trend is the same for STT-MRAM based STT64 and STT256 as well. For CMOS, the IPC does not vary much over these protocols. There is hardly a one or two percent deviation.This means, the changing the allocation policies for the same protocol does not affect performance much since we do not change any processor core related parameters. However, for STT-MRAM based configurations it does change. It is evident from Fig. 4.13 and Fig. 4.7 that MESI NII behaves similar to MESI IN both in terms of power and now IPC.

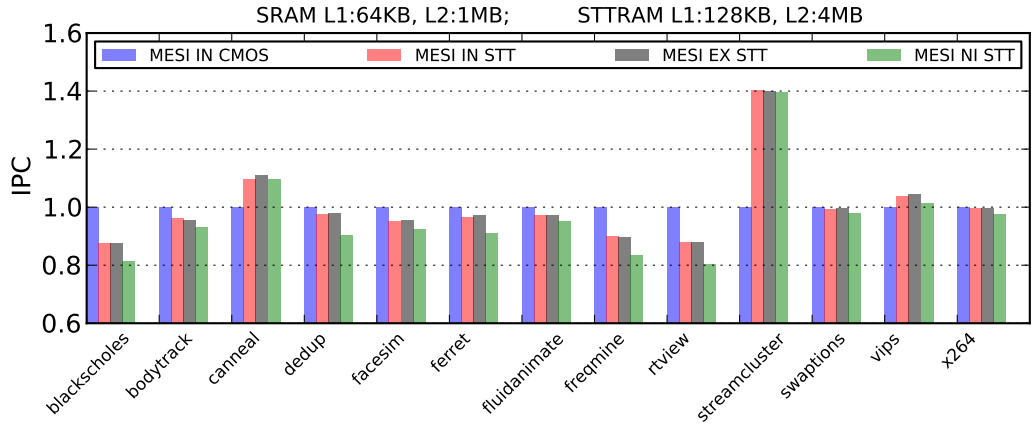

Figure 4.14: IPC Comparision between SRAM and STT-MRAM based systems. Normalized to SRAM MESI

Fig. 4.14 compares the IPC of systems that we have seen so far i.e CMOS and STT128 for MESI IN, MESI EX and MESI NI protocols. Most of the benchmarks show decrease in the performance baring *canneal* and *streamcluster* which show 10% and 40% increase in the IPC and *swaptions, vips* and *x264* whose performance is similar to CMOS MESI IN. Apart from blackscholes, freqmine and rtview, every benchmark has atleast 90% same performance as that of CMOS MESI IN. However, these gains are substantial considering that we get around 40% benefit in the power consumption but not sufficient.

A significant difference can been seen when we change the cache size. Since the access latency is a function of cache size, decreasing the cache size for STT-MRAM

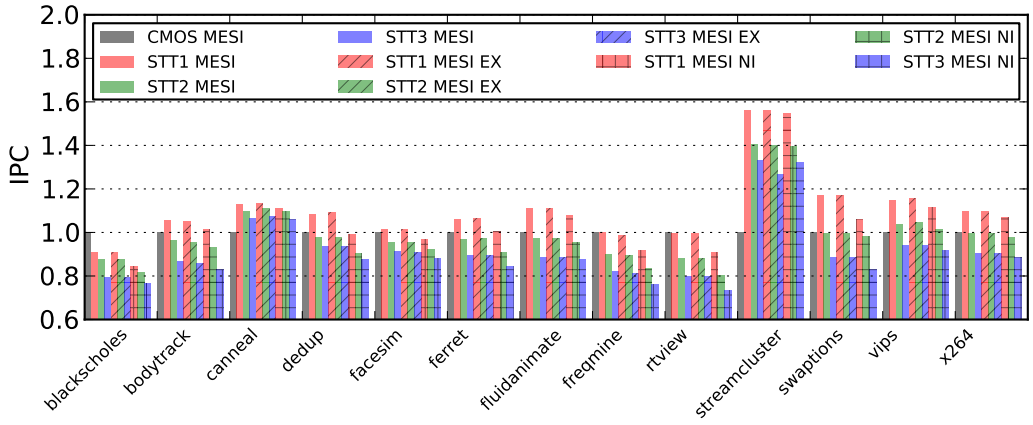

Figure 4.15: IPC Comparision between for various configurations presented in Table 3.3. Normalized to CMOS MESI Inclusive

will have positive effect on the performance. Fig. 4.15 confirms this. It compares the performance of STT64 , STT128 and STT256 for MESI IN, MESI EX and MESI NI with CMOS MESI IN. We see that apart from *blackscholes*, every other benchmark fairs well for STT-MRAM based MESI IN and MESI EX. *Streamcluster* shows the maximum benefit with 57% performance improvement followed by swaptions which achieves 17%. Apart from *freqmine, rtview* and *blackscholes* even MESI NI does a good job. On an average STT64 shows IPC improvement over CMOS baseline by 10.5% for MESI IN and MESI EX protocols and by 4.6% for MESI NI protocol.

The access latency of 64kB STT-MRAM cache is better than the 64kB CMOS cache which makes it a attractive solution for replacing CMOS based L1 caches. Sec. 4.1 and Sec. 4.2 show that instead of using larger STT-MRAM L1 cache, smaller STT-MRAM L1 with larger STT-MRAM L2 is a better solution. However, due to the large write latency, write intensive benchmarks like *blackscholes, facesim, freqmine* and textitrtview do not perform that well.

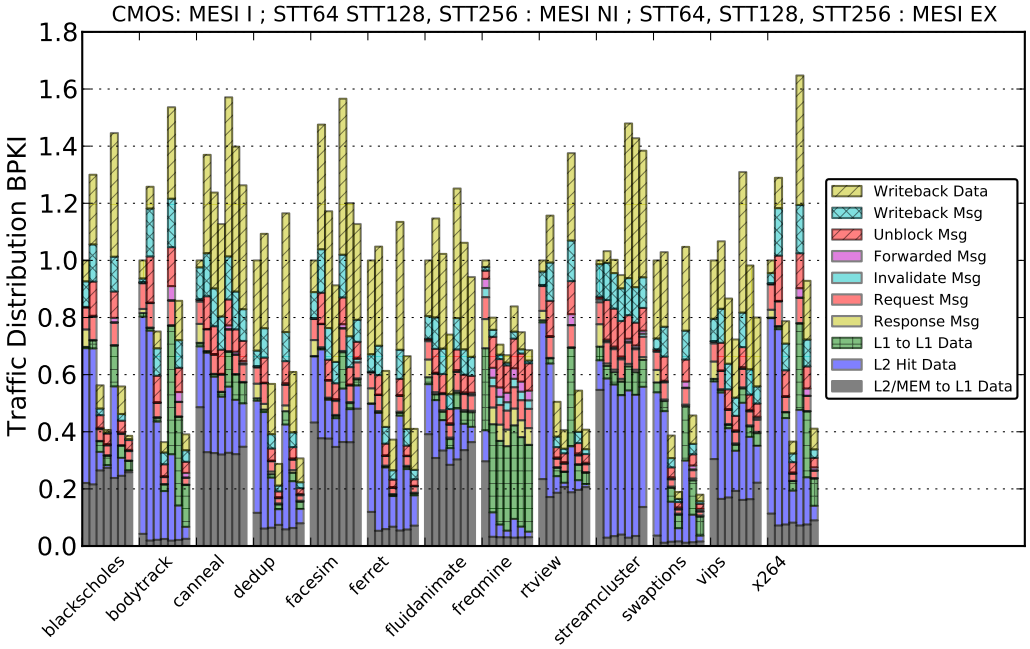

Figure 4.16: On chip traffic Distribution. Normalized to CMOS MESI configuration

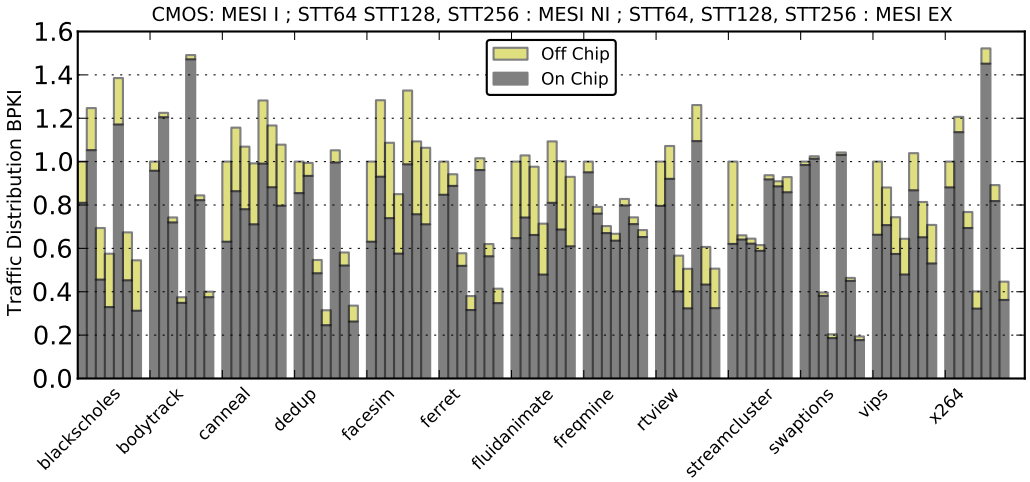

Figure 4.17: Traffic Over the network. Bytes per Kilo Instruction. Normalized to MESI Inclusive CMOS

4.3 Network Traffic

One of the advantages of increased cache size is less number of replacements from the higher level caches. We calculate the traffic by counting the number of bytes transferred over the network per kilo instruction. There is drastic decrease in the on-chip network traffic when the size of the cache increases for most of the benchmarks as shown in figure Fig. 4.16. The write-back data traffic for MESI NI and MESI EX is increased as compared to the MESI IN protocol. This write-back traffic decreases as we increase the cache size for MESI NI protocol barring *canneal* and *vips* and it decreases for every benchmark in case of MESI EX protocol. We also see that the byte transfers increase in case of MESI EX as compared to MESI NI for the same size cache. On an average the on-chip traffic decreases by 22% for STT128 MESI NI and by 43% for STT256 MESI NI as compared to CMOS baseline MESI IN.

The off-chip traffic largely remains the same across protocols since the size of L2 does not change. See Fig. 4.17. The off-chip traffic decrease when we change 1MB CMOS to 4MB STT-MRAM L2 by 47% on an average. The network, as expected is dominated by - however, as the size of the L1 increase, due to increased latency the IPC does not scale. For smaller sttram caches, the onchip traffic remains the same. However we see increase ipc due to faster access of the smaller size. The offchip traffic has reduced due to larger L2 size.

Overall, the traffic does decrease when we increase the size of the L1 cache. The data set becomes available at L1 and the L1 cache does not have to request L2 for further data. This leads to less coherence and request traffic. With just four CPUs, the traffic contention with 64kB cache is not a limiting factor for performance. It is when, the number of CPU is increased beyond 8 or 16 that the network contention will start affecting the performance of system.

4.4 Related Work

Most of recent researches have focused on hybrid caches mostly at L2 and a few of at L1 for mitigating write latency and energy overhead. [11] [8] [10] propose relaxing the non-volatility by reducing the retention time, in turn, reducing the energy and latency requirements. [30], [5], [6], [7], [8], [9], [18] and [31] propose architectural techniques

which reduce the number of writes in the high energy structure either using hybrid caches or multi-level retention caches. Decreasing the retention time affects the reliability of the device while the architectural techniques increase the reduce the energy consumption. While these techniques achieve significant energy reduction there is minimal performance improvement with added cost of extra hardware for data migration or prediction. In contrast, we have used STT-MRAM for L1 caches as well and studied its performance and energy effects. And used non-inclusive protocols to reduce the number of writes at L2 instead of any complicated architectural technique.

Chapter 5

Conclusion and Discussion

Over the years, researchers are finding it difficult to maintain the technology advancement in accordance with the Moore's Law. When CMOS started showing signs that it cannot be scaled as expected in the early 2000s, thread parallelism and multi-core processors eased the burden from CMOS for maintaining Moore's Law. However, as the number of cores increase, the request contention at the CMOS based Last Level caches increase. Moreover, the CMOS based higher level caches cannot increase in size because of higher onchip leakage than the chip can itself accomodate. Researchers, for past few years, have focused on STT-MRAM based LLCs. STT-MRAM caches being denser provide the flexibility of having capacity about 4 times than that of CMOS in the same footprint. Moreover, the extra write pulse latency does not affect the overall performance. The excess write latency is buffered in the savings that a larger cache provides by reducing main memory accesses and thereby reducing related latency. STT-MRAM provide another advantage of zero leakage which makes its case stronger for CMOS replacement. However, this write pulse latency have kept researchers away from using STT-MRAM for L1. As more and more cores are getting added to the CMPs, keeping the required data on smaller size L1 is getting to maintain for new state-of-the-art algorithms and applications. This leads to contention of requests at L2. The primary reason that takes researchers away from looking in STT-MRAM based L1 are the large number of CPU stores. It will consumer higher energy and impact the processor throughput.

The focus of this work is to replace the CMOS based cache hierarchy by STT-MRAM based one. In this regard, we first discussed the various allocation policies that

for different protocols. The allocation policies were meant to reduce the number of writes in the L2. This led us to non-inclusive protocols. These protocols require the usage of a small high associative cache, local directory. Whenever the data is not present in the cache, the local directory is referred from where the request gets forwarded to the concerned L1. We found that power dissipated in these extra transactions resulted in larger power dissipation for CMOS based caches which increased the L2 dynamic power by 7%. The dynamic power dissipation in STT-MRAM based caches is dominated by data writes and line fills. In case of non-inclusive protocols, the power dissipated by the extra hardware amounts to almost 10% out of the total power dissipated. We achieve a maximum 70% benefit in reducing the L2 dynamic power with an average of 36% power reduction for the MESI NI protocol in comparison with MESI IN protocol. We concluded that, in order to reduce the L2 dynamic energy, intelligent allocation of the L2 cache line helps reducing the dynamic energy. We then looked at the L1 dynamic power. It remains the same over all the protocols. As mentioned earlier, it is dominated by CPU stores and line fills. The total power dissipation, dynamic and leakage, reduces by 45% on average for all the workloads for MESI NI protocol when we use smaller and faster 64kB STT-MRAM L1 cache. Moreover, the dynamic energy of L2 becomes insignificant when we look at the complete picture of power dissipation. The decrease in the leakage power for both the caches, L1 and L2, plays a significant role in bringing down the overall power dissipation.

We also found that the protocols MESI IN and MESI EX have better IPC for all the workloads. Since the L2 dynamic power affects the total energy consumption by a very fractional amount, using MESI IN or MESI EX over other non-inclusive protocols would not be a bad choice. Moreover, when we use smaller L1 caches, a 64kB STT-MRAM L1 cache gives us on an average 10.5% performance benefit for MESI IN and MESI EX. In this work we have use two types of STT-MRAM cells. These cells have same write energy per bit. The trade off is between the write pulse latency and the area per cell. We use $30F^2$ cell area for L1 cache and consider that it has 5ns write pulse latency. Whereas for L2 we consider a $20F^2$ cell are with a 7ns write pulse latency. This is a conservative approach. However, the results that we have obtained are quite promising. The denser L2 cache helps us to accommodate the extra space required by the local directory and the necessary circuitry. We can conclude that a STT-MRAM based cache

hierarchy with smaller L1 cache along with a larger L2 helps in reducing the power by 45% and increase the performance by 10.5% on average.

At the end of this work, we looked at the network traffic contention. The general idea is, as the L1 cache size increases, the concerned processor will find most of the data in its respective L1. As a result it will reduce the load on the network. For our work, the traffic reduces by around 45% on an average for a 256kB L1 STT-MRAM when compared to a CMOS 64kB L1. However, since we have modeled a four core processor, though the reduction in traffic is evident, its effect on performance is not yet evident. For a 64 core processor, a larger L1 will benefit its performance when it reduces the network traffic contention.

References

[1] Xiaochen Guo, Engin Ipek, and Tolga Soyata. Resistive computation: avoiding the power wall with low-leakage, stt-mram based computing. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 371–382. ACM, 2010.

[2] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[3] Mark Horowitz, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. Scaling, power, and the future of cmos. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 7–pp. IEEE, 2005.

[4] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[5] Wei Xu, Hongbin Sun, Xiaobin Wang, Yiran Chen, and Tong Zhang. Design of last-level on-chip cache using spin-torque transfer ram (stt ram). *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(3):483–493, 2011.

[6] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid cache architecture with disparate memory technologies. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 34–45. ACM, 2009.

[7] Mitchelle Rasquinha, Dhruv Choudhary, Subho Chatterjee, Saibal Mukhopadhyay, and Sudhakar Yalamanchili. An energy efficient cache design using spin torque transfer (stt) ram. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 389–394. ACM, 2010.

[8] Adwait Jog, Asit K Mishra, Cong Xu, Yuan Xie, Vijaykrishnan Narayanan, Ravishankar Iyer, and Chita R Das. Cache revive: architecting volatile stt-ram caches for enhanced performance in cmps. In *Proceedings of the 49th Annual Design Automation Conference*, pages 243–252. ACM, 2012.

[9] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 239–249. IEEE, 2009.

[10] Clinton W Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R Stan. Relaxing non-volatility for fast and energy-efficient STT-RAM caches. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 50–61. IEEE, 2011.

[11] Zhenyu Sun, Xiuyuan Bi, Hai Helen Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. Multi retention level stt-ram cache designs with a dynamic refresh scheme. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 329–338. ACM, 2011.

[12] Jianxing Wang, Yenni Tim, Weng-Fai Wong, Zhong-Liang Ong, Zhenyu Sun, and Hai Helen Li. A coherent hybrid sram and stt-ram l1 cache architecture for shared memory multicores. In *ASP-DAC*, pages 610–615, 2014.

[13] Yong Li, Yaojun Zhang, Hai Li, Yiran Chen, and Alex K Jones. C1c: A configurable, compiler-guided stt-ram l1 cache. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):52, 2013.

[14] John K Bennett, John B Carter, and Willy Zwaenepoel. *Munin: Distributed shared memory based on type-specific memory coherence*, volume 25. ACM, 1990.

[15] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 459–462. IEEE, 2005.

[16] Mark S Papamarcos and Janak H Patel. A low-overhead coherence solution for multiprocessors with private cache memories. *ACM SIGARCH Computer Architecture News*, 12(3):348–354, 1984.

[17] Samaher Al-Hothali, Safeeullah Soomro, Khurram Tanvir, and Ruchi Tuli. Snoopy and directory based cache coherence protocols: A critical analysis.

[18] Zhe Wang, Daniel A Jiménez, Cong Xu, Guangyu Sun, and Yuan Xie. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 13–24. IEEE, 2014.

[19] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, 2009.

[20] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[21] Mu-Tien Chang, Paul Rosenfeld, Shih-Lien Lu, and Bruce Jacob. Technology comparison for large last-level caches (L^3Cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 143–154. IEEE, 2013.

[22] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P Jouppi. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(7):994–1007, 2012.

[23] SLICC, domain specific language for specifying cache coherence protocols. `http://www.m5sim.org/SLICC`.

[24] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: characterization and architectural implications. *PACT '08:*

Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pages 72–81, 2008.

[25] Roland E Wunderlich, Thomas F Wenisch, Babak Falsafi, and James C Hoe. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*. ACM, June 2003.

[26] Ole Tange. Gnu parallelthe command-line power tool. *The USENIX Magazine*, 36(1):42–47, 2011.

[27] Mark Gebhart, Joel Hestness, Ehsan Fatehi, Paul Gratz, and Stephen W Keckler. Running parsec 2.1 on m5. *University of Texas at Austin, Department of Computer Science, Technical Report# TR-09-32*, 2009.

[28] Shruti Patil and David J Lilja. Using Resampling Techniques to Compute Confidence Intervals for the Harmonic Mean of Rate-Based Performance Metrics. *IEEE Computer Architecture Letters*, 9(1):1–4.

[29] William Tuohy, Cong Ma, Pushkar Nandkar, and David Lilja. Statistical validation of parallel sampling simulation for performance and energy comparisons of multicore cache hierarchies. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015. submitted to.

[30] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, and Yuan Xie. Power and performance of read-write aware hybrid caches with non-volatile memories. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 737–742. IEEE, 2009.

[31] Junwhan Ahn, Sungjoo Yoo, and Kiyoung Choi. Dasca: Dead write prediction assisted stt-ram cache architecture. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 25–36. IEEE, 2014.