DRAM-based Coherent Caches and How to Take Advantage of the Coherence Protocol to Reduce the Refresh Energy

Zoran Jakšić
Dept. of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
zjaksic@ac.upc.edu

Ramon Canal
Dept. of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
rcanal@ac.upc.edu

***Abstract*—Recent technology trends has turned DRAMs into an interesting candidate to substitute traditional SRAM-based on-chip memory structures (i.e. register file, cache memories). Nevertheless, a major problem to introduce these cells is that they lose their state (i.e. value) over time, and they have to be refreshed. This paper proposes the implementation of coherent caches with DRAM cells. Furthermore, we propose to use the coherence state to tune the refresh overhead. According to our analysis, an average of up to 57% of refresh energy can be saved. Also, comparing to the caches implemented in SRAMs total energy savings are on average up to 39% depending of the refresh policy with a performance loss below 8%.**

***Index Terms*—FinFETs, 6T SRAM, 3T DRAM, retention time, cache coherence**

I. Introduction

The introduction of FinFET technology has prolonged Moore's Law by solving the problem of process variability caused by random dopant fluctuations (RDF). However, in deep sub 20nm technologies threshold variability caused by Line Edge Roughness (LER) and Metal Gate Granularity (MGG) significantly increases [1]. Also, besides the fabrication process, devices change their characteristics over time due to temporal fluctuation of voltage and temperature due to aging (NBTI, PBTI, and HCI).

On the other hand, novel multi-core processor architectures demand more on-chip caches for effectively sharing information across parallel processing units. Traditionally, on chip memories are based on SRAMs with the 6T (and recently 8T) cells as major building units. Since memory structures occupy the biggest part of the chip area, it is of crucial importance that these cells are scaled to the minimal dimensions. This, in turn, makes them most vulnerable to the effects of process variability (i.e. read noise margin is reduced, stability is compromised and static power consumption shoots up).

Many literature papers propose state of the art techniques to reduce cache leakage and increase cell stability. One of those ideas that has drawn great attention lately is to use multiple transistor DRAM cells to replace classical SRAM cells in different cache levels. Comparing to the conventional destructive-read 1T1C memory cell, these cells typically use the stored charge to control the transistor in the read-out path. Thus, they have a non-destructive read [2]. The area and leakage of these cells are smaller than that of the classical 6T SRAMs while read access time is not significantly degraded. There are many papers that analyze gain cells and suggest optimization techniques for their implementation at different cache levels. [3]–[5] This paper explores DRAM-based coherence caches. On top of that, we perform our analysis on future 10nm SOI FinFETs. We analyze memory power consumption and overall system performance when classical 6T SRAM cells are replaced by dynamic gain cells. Our analysis also includes the proposal of novel refresh techniques based on the coherence state of each cache line. As far as we know, this is the first paper that uses the cache coherence state to reduce refresh energy. In short, the main contributions of this paper are:

- The proposal and evaluation of DRAM-based L1 and L2 coherent cache structures;
- Extensions to the MESI coherence protocol to support DRAMs refresh implementation;
- Performance figures based on future SOI FinFET technology analysis.

When compared to the 6T baseline, our proposal reduces energy by an average of 39% for the PARSEC and SPLASH benchmarks with a performance loss below 8% on average.

The rest of the paper is organized as follows. In section II, we present the related work. In section III, we describe MESI cache coherence protocol and its adjustment for DRAM cache implementation. In section IV, we present simulation methodology and simulation results. Finally, in section V, we draw the conclusions.

II. Related Work

Gain cells have drawn more attention since Luk et. al [6] proposed a novel 3T1D DRAM cell (Figure 1a). Its access time is comparable to the 6T SRAM [6]. The gated diode acts as a storage device and an amplifier for the cell voltage. Before a read occurs, RBL is precharged to V_{dd}. When a logic "1" (high voltage) is stored in the cell, the gate of the PD transistor is strongly biased during the read process. High current flows through the PD transistor, and the output capacitance of the RBL is strongly discharged to ground. On the other hand, when the opposite situation occurs (low voltage stored in the cell), the bias on the gate of the PD transistor is lower, the current trough the PD transistor is reduced and the output capacitance is discharged more slowly.

The main issue in these cells is retention time (τ). Due to the leakage currents (mostly through the WR transistor) the cell loses its state after some time, and it has to be refreshed. In [3], [7] authors show that 2% of processor performance is lost when a 3T1D with retention time of $0.8\mu s$ is used in the L1 cache.

978-3-9815370-2-4/DATE14/

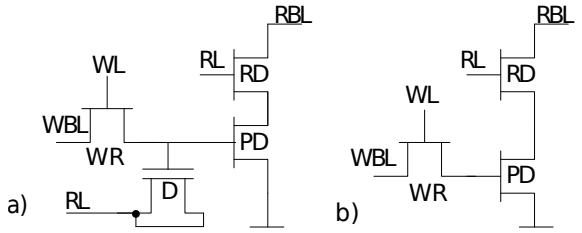

Fig. 1. Gain Memory Cells: a) Dynamic 3T1D; b) Dynamic 3T

Using a slightly different dynamic gain cell, the authors in [4], [8] analyze the performance and power of DRAM-based last-level shared L3 cache. They conclude that refresh energy is the biggest contributor to overall consumption. On the other hand, if no refresh is applied in the L3 cache performance is reduced almost by 40%. Moreover, they assume a cell retention time of $20\mu s$ which might be challenging to achieve in sub 20nm technologies for high performance devices, especially at high temperatures. For instance, in [9] Jaksic and Canal characterize a 3T cell at 10nm. Although they report retention times greater than $20\mu s$, even for higher temperatures, that comes at the cost of having two additional voltage sources to maintain retention time at this level.

In [10], authors present refresh technique for L2 and L3 caches in order to reduce refresh energy. They based their line refresh proposal on the last access moment, and its state (clean, dirty, idle-dirty lines not being accessed for long time). They state 56% energy reduction comparing to classical SRAM implementation with 6% increasing in execution time. However, they based their work on low power devices and nominal system frequency of 1GHz in order to sustain retention time up to $100\mu s$.

In this paper, we simulated cache memories implemented with the 3T DRAM cell (Figure 1c). This cell is smaller (1 transistor less) than the 3T1D. Plus, as shown in [9], the read access time is comparable to that of the 6T cell when they are implemented with 10nm FinFETs. In this paper, we analyze the performance and the power consumption of coherent caches (typically L1 and L2 caches nowadays) when they are implemented with dynamic cells. We also explore how the cache coherence protocol can help in tuning the refresh energy while keeping the performance untouched. We also show how to extend the MESI cache coherence protocol to direct DRAM refresh policies.

III. Cache Coherence

A. MESI protocol

The MESI protocol [11]–[15] is a widely-used cache coherency and memory coherency protocol. Every cache line is marked as one of the following states:

- **Modified (M)**: Cache line exists only in that cache and it is dirty (it is not consistent with the lower level memory value). It should be written to lower memory level before it is invalidated or replaced.
- **Exclusive (E)**: Cache line exists only in that cache and it is clean (it is consistent with the value in the lower level memory).
- **Shared (S)**: Cache line may exist in other caches and it is clean.
- **Invalid (I)**: Cache line doesn't hold valid data.

A cache reads can be serviced from any cache state besides Invalid. On the other hand, cache writes can take place only if the line is in Modified or Exclusive state. If the cache line is in Shared state, first we have to triger a request for ownership, and -consequently- all shared lines will be invalidated first.

A cache line that is in Modified or Exclusive state has to "snoop" all other caches accesses to intercept any request to the same address in the lower memory level. If this is the case, the line changes to the Shared state, and if the line was previously in the Modified state, it is writen back to the lower cache before moving into the Shared state and sent to the requester.

B. MESI extensions for DRAM support

The main drawback of using dynamic cells is that they loose state over time and they have to be refreshed. In general, memory refresh reduces system performance (since access to the memory is blocked during that time) and increases dynamic energy (refresh energy). Many techniques for reducing DRAM refresh energy have been presented in the past [3], [7], [16]. Although the block refresh is the simplest solution for implementation (refreshing whole memory block after a predefined time period) it doesn't leave room for any further optimization. On the other hand, techniques that assume line refresh are slightly complicated to implement but a significant amount of refresh energy can be saved.

In this section, we show how the cache coherence state can be exploited to find an energy-delay optimal refresh line policy. Since different programs access cache in a different way, we propose coherency-aware refresh. In other words, we consider refreshing a cache line depending on its coherence state. In order to make this possible, we extend the MESI protocol the following way:

- **Modified (M)**: In case that a Modified line expires, and depending of on the specific refresh scheme, it should be either refreshed which keeps the line in Modified state or it must be evicted to a lower memory level before it is tagged as Invalid.
- **Exclusive (E)**: If the line is refreshed, it will stay in Exclusive. Otherwise, it must be invalidated (no eviction is needed since the correct data exist in lower cache).
- **Shared (S)**: If a cache line in this state is refreshed it stays in Shared. In case of no refresh, the line must be invalidated (same as Exclusive).
- **Invalid (I)**: Since this line doesn't hold valid data it should never been refreshed since that would be unnecessarily energy consumption.

According to the combination of the line states that can be refreshed or not, 8 refresh policies can be defined: NONE, M, E, S, ME, ES, SM, MES (letter identify that states the states that lines should be in to be considered for refreshed, NONE means that no line is being refreshed.

Figure 2 presents the state diagram of the modified MESI protocol. Dashed lines present modified transitions. One more state is added to handle expired lines in Modified state. Standard line definitions are used to describe state transitions ("PrRd" - Processor Read (Read request from processor), "PrWr" - Processor Write (Write request from processor), "BusRd" - Bus Read (Read request from the bus without intent to modify), "BusRdX" - Bus Read Exclusive (Read request from the bus with intent to modify)). Label "Exp" is used to define expired line transition.

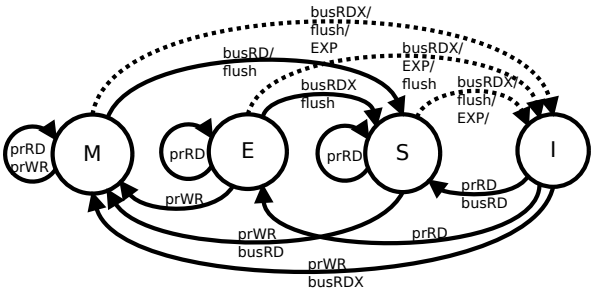

Fig. 2. MESI protocol with extensions (dashed lines) to handle expired lines

IV. Simulation Results

A. Methodology

In this section, we present the simulation results of the dynamic coherent cache memory. The configuration parameters can be found in Table I. For simulation we used marssx86, a full-system simulator for x86-64 CPUs. [17], [18]

In order to simulate dynamic memories with marssx86, we enhanced it with additional refresh controller logic. Since the access time of gain cells is similar to SRAMs, memory refreshing is the only reason that can influence the system performance and energy consumption [9]. We implemented line (or block) counters that measure the lifetime of each line in the cache (i.e. time since last write -or refresh). Counters are 10 bits wide. Due to the area savings achieved by using 3T over 6T cells, the counter overhead is neglected. When the counter reaches a certain limit (minimum retention time), the refresh controller is triggered and it reacts according to the configuration (refresh policy). If the line is meant to be refreshed, it blocks a cache access port to complete the refresh (2 cycles in this paper). In case the line is not refreshed, the controller acts accordingly (as explained in section III) in order to preserve data consistency.

We assume a baseline retention time of $3\mu s$. For 10nm FinFET technology this value can be achieved just with one additional voltage source. A small negative voltage in WL when the cell is on hold mode is needed to achieve this value at higher temperature. [9]

For our analysis, we used 2 parallel benchmark suites - PARSEC and SPLASH-2 [19]–[21]. We configured them as single process, 2 thread workloads. We used simmedium configuration for PARSEC. SPLASH benchmarks are configured according to [22]. Initialization phase is skipped and only each application's region of interest is simulated. All workloads run on top of Ubuntu 9.04 (Linux 2.6.31). The full list of workloads that marssx86 supports can be found in [17]. All of them are simulated. We report 7 individual benchmarks plus the average (of all benchmarks in both suites). This sample was chosen in order to present extreme cases as well as the diversity in behavior for the different refresh policies proposed. In other words, we tried to avoid plotting benchmarks that show similar behavior.

B. System performance

Figure 3 shows normalized system performance. For a system performance metric, we present execution time. Results are normalized to the baseline multiprocessor which has SRAM cache memories. Figure 3 also includes a bar for the simple block refresh technique. This technique is the simplest for DRAMs (i.e. when the counter reaches the limit it refreshes the line regardless of its coherent state). Also, it assumes whole memory refresh (even the invalid lines) that would needlessly block the cache access and consume refresh energy.

TABLE I
Base System Architecture

Processor	2 core Out of Order, 4-wide issue width
L1 Data Cache	2x32KB (organized in 32KB blocks), MESI
L1 Instruction Cache	2x32KB (organized in 32KB blocks), MESI
L2 Cache	2x256KB (organized in 32KB blocks), MESI
L3 Cache	1MB (organized in 32KB blocks) Shared
Main Memory	4GB (50ns delay) 1 channel
Technology	10nm FinFETs
Retention Time	3us
System Frequency	2.5GHz

In general, the MES technique delivers the same performance as the baseline SRAM system. Nevertheless, the other schemes perform within a 8% of the baseline which is a minor performance degradation (given the energy benefits, as we will see in short). The individual behavior of the benchmarks can be quite different among them (e.g. "facesim" achieves high performance when the Exclusive state is refreshed while "blackscholes" achieves this for the Shared). In terms of system performance, "blackscholes" and "facesim" show the greatest performance lost.

C. Energy Consumption

In order to estimate energy, we run HSPICE simulations to characterize the different cells (6T SRAM and 3T DRAM). We simulated a memory column when it is exposed to process and environmental variation as shown in [9]. For this simulation, we used 10nm SOI FinFET model card developed by the University of Glasgow, Device Modeling Group [1]. For process variation, we assumed 16% standard deviation of T_{fin} and 12% variation of H_{fin} as in [9], [23]. We assumed random variation as the main cause of it is LER and MGG [1]. Chip temperature is 50C. In order to minimize statistical error, we used the Monte Carlo method and we simulated 1000 instances. We, then, calculated the average static and dynamic power consumption of one memory block with 64 lines 512 bit wide each.

After the estimation of the power at the circuit level, full system simulation was done with the Marssx86 simulator. We used total number of cache accesses (number of total read hits, read miss, write miss, write miss) and cache "snooping" communication in order to estimate dynamic energy. We also extracted the total number of refresh lines in order to evaluate refresh energy and the total number of cycles in order to evaluate leakage. Combining the access/cycle counts and the energy per access/cycle we obtained the final energy numbers.

Figure 4 shows the total energy of the coherent cache. Data is normalized to the SRAM baseline. It can be seen that DRAM memory with block refresh consumes, on average, 18% more energy than the one implemented in SRAMs. However big part of this energy is consumed on refresh. Applying our proposed refresh techniques, we get savings of 21-57% (comparing to block refresh) of the total energy with a performance loss of 1-8% depending of the refresh policy. It should be noted that leakage may vary between configurations as it is a directly proportional to the execution time. In other words, due the performance lost, the total execution time is increased which proportionally affects total leakage energy. Highest energy saving can be achieved when just "Shared" state is refreshed (57%). For "S" refresh policy performance

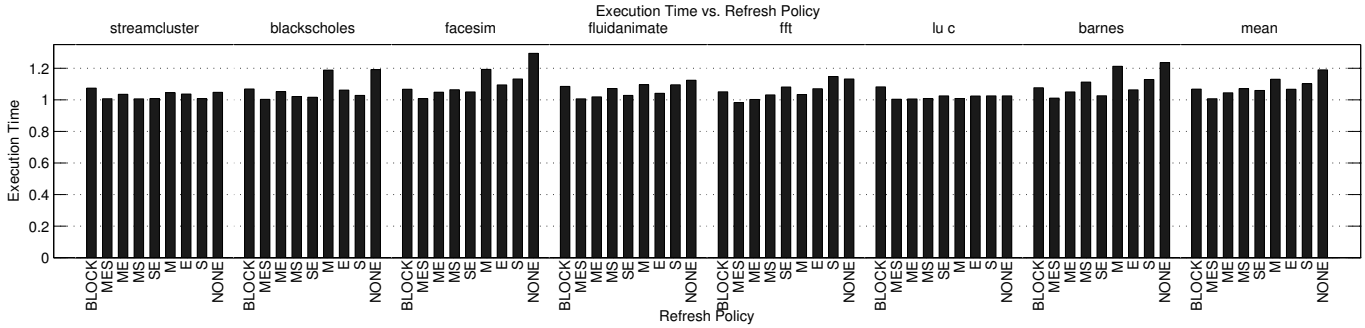

Fig. 3. System Performance for Different Refresh Policies Normalized SRAM Coherent Cache

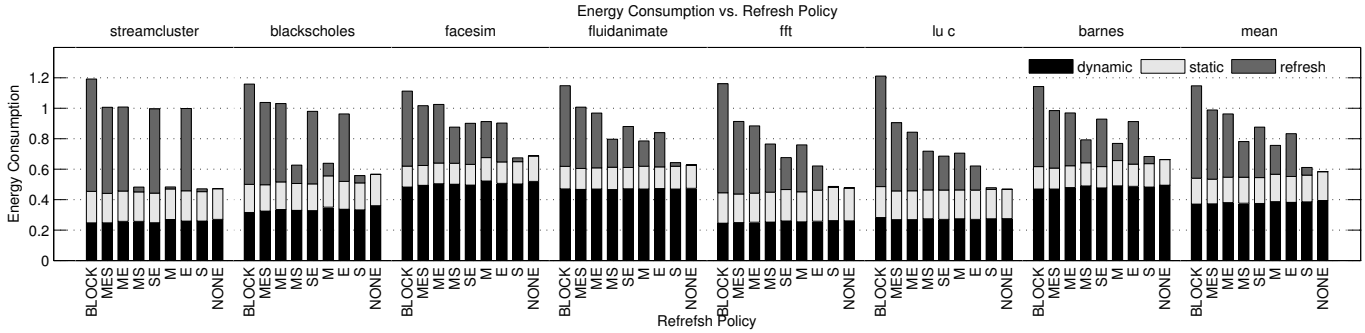

Fig. 4. Energy Consumption For Different Refresh Policies Normalized to SRAM Coherent Cache

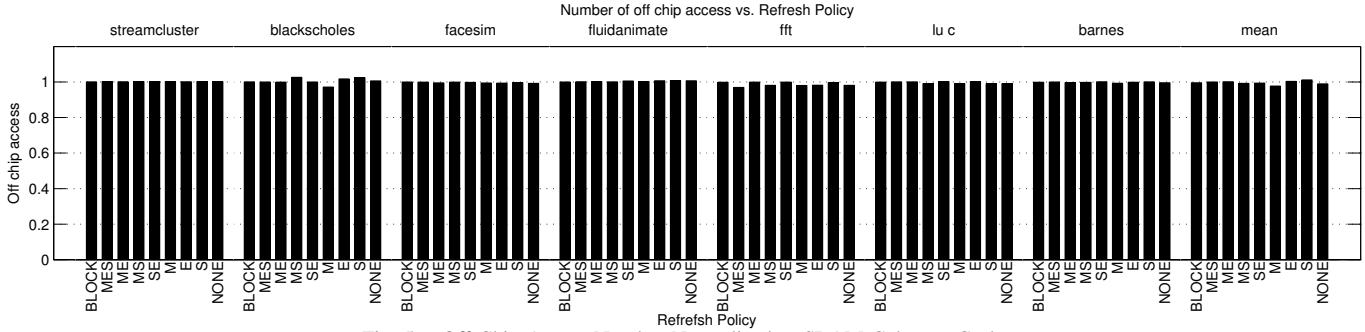

Fig. 5. Off Chip Access Number Normalized to SRAM Coherent Cache

loss is 8%. On the other hand 29% of the energy can be saved (compared to block refresh) when "Exclusive and Shared" states are refreshed, and performance loss is 3%.

D. Off Chip Communication

Given that DRAM-based caches may induce a higher number of accesses to the lower levels of the memory hierarchy, Figure 5 shows the number of access to the external memory. It can be observed that external memory communication is not significantly changed. In the worst case, the number of accesses increases 2% comparing to SRAM baseline.

V. Conclusion

This paper has described the implementation of DRAM-based coherent caches and the integration with the MESI coherence protocol. In addition, we propose to use cache coherence state to define different refresh policies which is discussed for the first time. We performed thorough analysis in terms of performance and energy of multiprocessor system implementing these novel refresh policies. Our results show that cache refresh energy of the coherent cache can be reduced by 39% with performance loss below 8% depending of the refrefresh policy.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Education and Science under grant TIN2010-18368, the Generalitat of Catalunya under grant 2009SGR1250 and Intel Corporation.

References

[1] X. Wang et. al, in *IEDM, 2011*.
[2] W. Luk et. al, in *IEEE journal of solid-state circuits*, April, 2005.
[3] X. Liang et. al, in *MICRO, 2007*.
[4] M.-T. Chang et. al, in *HPCA, 2013*.
[5] E. Amat et. al, in *Device and Materials Reliability, IEEE Transactions on*, January, 2013.
[6] W. Luk et. al, in *VLSIC, 2006*.
[7] X. Liang et. al, in *Micro, IEEE, 2008*.
[8] M.-T. Chang et. al, in http://hdl.handle.net/1903/13296, 2013.
[9] Z. Jaksic et. al, in *ICCD, 2012*.
[10] A. Agrawal et. al, in *HPCA, 2013*.
[11] J. H. Papamarcos et. al, in *ISCA, 1984*.
[12] M. Monchiero et. al, in *ICPP, 2009*.
[13] A. Patel et. al, in *ISLPED, 2008*.
[14] X. Qin et. al, in *DATE, 2012*.
[15] T. Suh et. al, in *DAC, 2005*.
[16] M. Ghosh et. al, in *MICRO, 2007*.
[17] "http://marss86.org/ marss86/index.php/home," *marssx86 home page*.
[18] A. Patel et. al, in *DAC, 2011*.
[19] C. Bienia et. al, in *IISWC, 2008*.
[20] "http://www.capsl.udel.edu/splash/," *SPLASH-2 Benchmark Suite, 2013*.
[21] "http://parsec.cs.princeton.edu/," *PARSEC Benchmark Suite, 2013*.
[22] I. Choi et. al, in *Computer Architecture Letters*, February, 2011.
[23] Z. Jaksic et. al, in *Electron Devices, IEEE Transactions on*, January, 2013.