Mercurial Caches: OS Support for Low Power DRAM

Asim Kadav, Rathijit Sen, Michael M. Swift
Computer Sciences Department, University of Wisconsin-Madison
{kadav, rathijit, swift} @cs.wisc.edu

Abstract

DRAM is a significant consumer of power, accounting for a large percentage of total power in servers. While many hardware technologies to improve DRAM power efficiency exist, achieving energy proportionality for memory is still difficult. Modern operating systems lack the mechanisms and abstractions to use low-power technologies. For example, they use idle memory to aggressively cache file data, and they allow physical memory to become fragmented, making it impossible to turn off memory banks without losing data.

In this paper, we describe *mercurial caches*, which are an operating system mechanism to achieving energy-proportionality for memory. Mercurial caches provide a copy-in/out interface to low powered memory and act as a new level between the file cache and storage.

We show how mercurial caches can be integrated in current operating systems and demonstrate that they provide substantial power savings when memory is idle with minimal performance overhead. Through an analytical model, we demonstrate that it can provide DRAM energy savings proportional to the DRAM under active usage.

FIXME [Real results]???

1. Introduction

DRAM consumes significant power, accounting for up to 25–57% in servers [3, 11, 23]. With modern servers provisioned with tens of gigabytes of memory, DRAM power increasingly dominates the power cost of idle servers. As a result, many research projects provide hardware and software support to reduce memory draw [9, 11, 21, 23].

A common goal for reducing idle or off-peak power is through energy proportionality. Energy proportionality strives to provide power efficiency by only using as much power as required by the running applications [2]. This has been possible in modern processors with existing frequency and voltage-scaling techniques [**?**]. However, energy proportionality in DRAM is limited to time multiplexing DRAM usage. This is usually achieved through modifications to DRAM devices or micro-controllers, with support for memory power "naps" but impose latency to transition in and out low-power states [11].

Large compute clusters such as those used by Google [19] and Condor [6] run heterogeneous workloads with different memory requirements. Hence, low power memory saving technologies based on time multiplexing DRAM usage are insufficient when workloads only partially consume available memory.

Power-saving hardware techniques such as Deep Power Down (DPD), Partial Array Self Refresh (PASR) [5], or reducing the refresh rates (Low Refresh) can be used to save energy by selectively applying them to unused memory portions in a system. However, such techniques cannot be directly used with existing operating systems for two reasons. First, most operating systems use idle memory to cache file data to avoid expensive disk accesses. Thus, there is little memory that is completely empty. Second, a operating systems allow physical memory to become fragmented over time because there is little need for contiguity. However, power-saving hardware such as PASR requires contiguous free regions at least at least 1/16th of a DIMM long (up to 1 GB).

We provide energy proportionality for memory by providing support for *mercurial caches*. A mercurial cache is a region of DRAM in a low-power state that can store infrequently accessed clean data. Compared to normal memory, the latency of accessing data from a mercurial cache is higher, and it may be discarded by the cache to save power. Thus, the techniques described above that reduce power can be applied to a portion of memory to achieve energy proportionality: only the fraction of DRAM needed for actively used memory need be fully powered.

We modify the Linux virtual memory system to automatically move unused page cache into a mercurial cache. If these pages are referenced again they are copied back into regular DRAM. We also implement mechanisms that prevent system memory from being fragmented when power savings are desired.

In addition, mercurial caches support unreliable memory technologies. A primary use of power in DRAM is refresh, and reducing refresh rates can reduce power use at the risk of data corruption [20]. When used with such technology, mercurial caches use software checksums to augment hardware ECC.

FIXME [Say there are multiple techniques to save power with different trade offs, including power savings, latency of access and reliability. We do something with them]???.

Using an analytic model, we show that different memory references patterns are best served by different power-saving techniques, and that **FIXME [some other conclusions]???**.

The contribution of this paper is as follows:

- We demonstrate how DRAM power saving technologies can be used in an OS and identify the challenges in providing such support in an operating system.
- We analyze the available technologies and potential power savings through an analytical model. Our model demonstrates that space multiplexing DRAM usage provides power savings even when DRAM usage is high.
- Through a real OS implementation and emulated DRAM controller hardware, we measure the power benefits and performance overheads.

2. Motivation and Background

In this section we describe how the OS manages and consumes memory and describe the existing hardware technologies to save DRAM power.

2.1 OS memory management

Modern operating systems are designed for high performance and the memory consumption pattern is best summarized by the following quote:

"Free memory is bad memory"

– *Attributed to Linus Torvalds [1]*

This quote emphasizes that free memory represents wasted memory because it can be used to improve system performance. Hence, Linux and other modern operating systems are designed to use all available memory. This is because the cost of of disk accesses is significantly more than accessing data from memory. Furthermore, power has only recently been a first order design constraint and most OS decisions are largely guided by performance. Hence, operating systems try to cache as much data as possible to avoid the disk penalty. They cache substantially more pages than required by the running applications' working set. For example, while bringing in new pages, the LRU approximation algorithm [4], which tries to predict future page references, also prefetches additional pages that may be referenced in the future. Similarly, when an application terminates, its pages are not freed immediately but aged out of the cache to avoid re-reading the disk in case of a future reference. Hence, over a period of time the OS occupies all available memory except for a limited amount of reserved memory used to allocate memory in interrupt contexts. Reclaim mechanisms ensure that memory can be quickly reclaimed from these caches when free memory is required by the system by simply discarding cached data. Table 1 represents memory consumption of a system running for 8 hours, while occasionally running desktop workloads (kernel-compile, file reads). We find that memory is dominated with file data and the amount of free memory is very low.

Total memory	**Free memory**	**Active page cache**	**In-active page cache**
16GB	1.9GB	0.4GB	13.3GB

Table 1. Memory utilization in a quiesced system with 16GB DRAM running Linux 3.4.1.

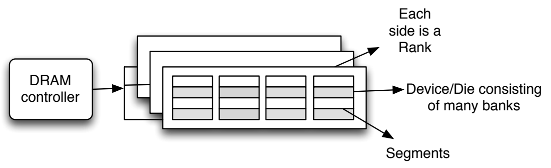


Figure 1. DRAM controller and modern DRAM hierarchical organization into ranks, banks and segments. Low power DIMMS support refresh granularity of ranks or segments (bank-segment selective).

Another characteristic of memory management in modern operating systems is fragmentation of physical addresses. Due to frequent small memory allocations, the physical memory becomes fragmented over time. Since requests to allocate physically contiguous memory are rare (such as super pages) and are often of the order of few MBs, fragmentation is not a problem for most system or application functions. Hence, de-fragmentation is an expensive operation that adds limited value to OS when run periodically. Furthermore, the OS may pin certain pages in memory such as for DMA. Hence, these pages remain locked-in and create holes in memory making de-fragmentation difficult.

The above two problems hamstring using current hardware technologies to save DRAM power, such as turning off unused memory, since consolidation of physical memory becomes difficult.

2.2 DRAM Background

We briefly describe modern DRAM organization and function. Figure 1 illustrates a DRAM organization. CPUs communicate to memory through DRAM controller. Each DRAM controller transfers data to one or more *ranks*. Each rank comprises of multiple DRAM *banks*. Each bank represents a memory arrays which maybe further sub-divided into multiple segments where data is stored in rows and columns. Data from memory cannot be read out directly from the memory arrays. Memory accesses initiate the following sequence of events:

1. *Activate*: The entire selected row is read into row buffers. This is also referred to opening a page (not related to OS pages).

DRAM technology	Data retention	Granularity	Latency	Power savings
ACPI S4	No	All DRAM	> 1 s	100%
Deep power-down	No	Bank	200 μs	95%
Self-refresh	Yes	Bank	100ns	33%
Clock-stop	Yes	All DRAM	200 μs	83%
TCSR	Yes	All DRAM	100 ns	60%
PASR	No	1/16th DIMM	140 ns	25-30%
Low Refresh	Partial	Any	130ns	42%

Table 2. Comparison of different DRAM power saving technologies. Many technologies in the figure can co-exist. TCSR savings are for 40C drop for 64MB DIMM.

2. *Read/Write*: Data is accessed from row buffers and transmitted to the memory controller.
3. *Precharge*: DRAM reads are destructive (the row needs to be written back) and row buffers are of limited capacity. Before another row can be opened, the data needs to be written back from the row buffer to the opened row. For the rest of this paper, we assume a *close page* policy, where the row is precharged immediately after completing the read/write access.
4. *Refresh*: DRAM memory cells leak capacitive charge, and hence lose data over time. To prevent data loss, the memory controller performs periodic *refresh* operations (every 3.2ms) for each row.

The first three operations consume power when the DRAM is in active use. This is called as *active* power. The refresh operation, along with the power consumed by DRAM controller (when idle) constitute *background* power. When idle, DRAM power can be saved by reducing refresh power, as discussed in next section.

2.3 DRAM low power modes

DRAM low power technologies either completely themselves off, losing data or alter DRAM refresh rates.

1. **ACPI S4 state**: The most extreme technique to save DRAM power is to use the ACPI S4 or *hibernation* state which turns all DRAM off and stores memory contents to persistent storage. However, this solution is drastic and only saves DRAM power in completely idle systems [7].

2. **Deep Power Down**: Deep power down cuts off power to DRAM and reduces leakage current independent of the rest of the system. Applications that do not require data retention can utilize this DRAM power state in LPDDR2 DRAM [15].

3. **Self-refresh DRAM**: Self-refresh mode, supported in server DDR2/DDR3 and mobile LPDDR2 class memories, enables refreshing memory contents without involvement the DRAM controller. This saves DRAM power when CPU is idle for short intervals, but imposes short exit penalties for mobile RAMs and long penalties for DDR2/3 class memories [16].

4. **Clock-stop DRAM**: Clock stop DRAM provides power savings by stopping the DRAM clock when there are no memory transactions in progress or when transactions can be processed at lower speeds. Clock stop is a software controlled feature available in LPDDR2 DRAM [16].

5. **TCSR DRAM**: Temperature Control Self-Refresh allows DRAM to adjust DRAM refresh rates based on temperature measured by an on-chip sensor. Hence, DRAM power can be saved by by reducing the refresh rate when ambient temperature is low. This feature is available in LPDDR2 DRAM [16].

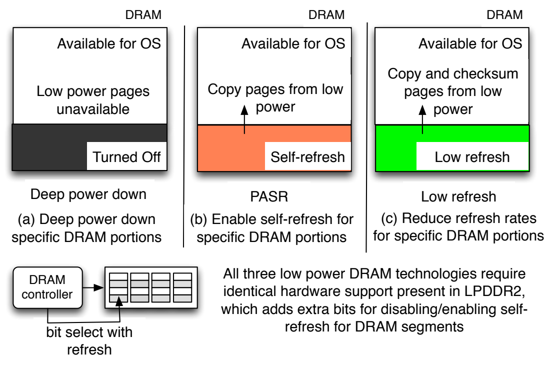
6. **PASR DRAM**: Partial Array Self-Refresh(PASR) is an enhancement to self-refresh which provides the ability to refresh memory at different granularity, upto 1/16th of a bank [15] and is supported in LPDDR2 and DDR3 DRAM specifications. Simple extensions to these controllers can allow arbitrary refresh times to different areas of memory DIMMs [5, 9, 15]. When the system memory consumption is low, PASR can be used with appropriate OS support to reduce DRAM power.

7. **Low Refresh DRAM**: Low refresh DRAM saves power by reducing the frequency of refresh. Recent work has shown that most DRAM configurations are programmed with significantly higher refresh rates than required and a refresh time of the order of seconds only marginally affects error rates [9, 20, 21]. Reducing refresh rates can lower the error rates in DRAM. For example, there is a BER of $4.0x108$ when lowering refresh rates from 3.2ms to 1 second and 3.2 x 10^-6 when refresh rate is reduced to 5 second [22].

Comparing these technologies as shown in Table 2, we argue selectively applying refresh altering power saving techniques to reduce DRAM power on a rank basis. We use DPD, PASR and Low Refresh techniques to expose control knobs to the OS to reduce DRAM power based on memory usage and can be used to provide energy-proportional RAM usage. In the next section, we describe the DRAM organization required to support such a design and analyze its benefits.

3. Mercurial Cache Hardware Support

In order to space multiplex DRAM usage, we examine three different DRAM usage models that are enabled by PASR as described in Figure 2; we intend to use the existing PASR and DPD technologies supported in LPDDR and LPDDR2 DRAMs [12, 15, 16]. Additionally, we propose a low-refresh (LR) mode that requires support for a programmable refresh rate in the memory controller, along the lines of prior work [9, 21]. We assume that these three power-saving modes are applicable at the granularity of ranks.



DRAM
Available for OS
Low power pages unavailable
Turned Off
Deep power down
(a) Deep power down specific DRAM portions
DRAM
Available for OS
Copy pages from low power
Self-refresh
PASR
(b) Enable self-refresh for specific DRAM portions
DRAM
Available for OS
Copy and checksum pages from low power
Low refresh
Low refresh
(c) Reduce refresh rates for specific DRAM portions
DRAM controller
bit select with refresh
All three low power DRAM technologies require identical hardware support present in LPDDR2, which adds extra bits for disabling/enabling self-refresh for DRAM segments

Figure 2. Hardware support using existing LPDDR2 technology to support low powered caches in OS.

We recommend that regions of memory that are not actively used be transitioned to power-saving mode. DPD offers maximum power savings for the target memory regions but may incur additional performance penalty due to increased disk accesses. All data in the target memory devices are lost during DPD. In contrast, during self-refresh (or PASR(1)), all the data is retained but power savings are lower. PASR(n) offers further power savings during self-refresh by stopping refresh to selected banks or parts of banks within the device. LPDDR2 offers both bank-selective and segment-selective masking to support PASR(n) [12]. The portions of the device that are refreshed under PASR(n) are done so at the regular rate of once in 3.2 msec for each row. This guarantees error-free data retention. Our proposed LR scheme lowers refresh rate to further save power but there may be data corruption due to leakage of capacitive charge. We correct for this by doing error-detection and correction when accessing the page. Note that for read/write accesses to happen in DPD/PASR(n), the rank needs to be transitioned into working mode. This has a non-zero exit latency of 140 ns for self-refresh/PASR(n) and ~200 μsec for DPD [16**?**]. New refresh-rate settings for LR take effect after one refresh interval.

3.1 Analytical Model

We construct an analytical model to understand the power savings from above DRAM hardware in different low power modes, as a function of the memory access rate and percentage of total memory that is placed in low-power mode. The analytical model helps us answer the following questions:

1. *Power Savings*: How much power do we save from different technologies for different amounts of DRAM? Specifically, active DRAM power dominates the total DRAM power. Are savings from refresh power substantial when the total DRAM power usage is taken into account?
2. *Reliability Costs*: For different memory reference rates, does the extra copy power between low power and regular DRAM offset any refresh power savings?

We model three different configurations (i) DPD, where a portion of memory is turned off (ii) PASR, where a portion of memory is in self-refresh and (iii) Low Refresh, where a portion of memory is in self-refresh at low refresh rates. For power calculations, we use the methodology outlined in MICRON technical reports [13, 14] and values for operating currents and command timings from the LPDDR2-S4 datasheet [12]. To the best of our knowledge, this is the first model to evaluate low power techniques that considers both *active* and *refresh* power.

3.2 Memory Organization

We describe our DRAM memory organization that switches DRAM to low power states on a rank basis and is the basis for our analytical model.

Each memory device in our study is a 256MB (2Gb) x32 (32-pin data interface) LPDDR2-S4 SDRAM. Memory cells within each device are logically arranged in arrays of rows and columns. Using connectors similar to recent work [10], we group multiple devices into ranks such that all devices within the same rank share addressing, data and command interfaces. The number of devices in each rank and the total number of ranks depend on the total memory capacity to be supported and the total number of ranks that each memory channel can support without significant loss in signal integrity. For modeling purposes, we assume that two devices are grouped together to form an x64 (64-bit wide data interface) rank of capacity 512MB. The total memory capacity of 16GB is formed with 32 ranks, spread over the available memory channels. Each rank can operate in either working or power-saving mode (Section 3.2.3). The total memory power is given by

$$P_{MEM} = \sum_{i}^{n_R} P_{Ri} + P_{extra} \qquad (1)$$

where n_R is the number of ranks (32), P_{Ri} is the power of the i^{th} rank, and P_{extra} includes power for additional overhead operations such as extra copying of pages or checksum recovery.

3.2.1 Access Timings

Each memory access (read/write) occurs in units of 64 bytes (cache line size) and is serviced by all devices in a rank. Each device provides 8 bytes over 8 memory clocks. Since the memory is DDR, the memory bus is occupied for 4 clocks for each cache line transfer. We will refer to the memory bus cycle time as tCK.

Our model assumes LPDDR2-S4 with a data rate of 1066M/s having average memory clock cycle time, tCK, of 1.875 ns. The datasheet [12] specifies the following latency values: read latency, RL = 8*tCK, is the time when the first read data is available after activate; minimum row active time, tRAS = 42ns, is the minimum time between activate to precharge; precharge command period, tRP = 18ns; row cycle time, tRC = tRAS+tRP, is the minimum time between back-to-back activates to the same memory bank. Furthermore, inour baseline system, each row is refreshed every 32 msec, and time between two consecutive refresh operations is 3.9 μsec.

3.2.2 Access Distribution Assumptions

1. Memory accesses are mostly uniformly distributed over available ranks (512MB bins). The average number of accesses per rank is inversely proportional to the number of available ranks.
2. Accesses to the same rank are serialized. While current devices do allow parallel accesses to different banks within a rank (with some delay), such analysis would require a more sophisticated model that includes fine-grained address conflict modeling.

3.2.3 Operating Modes

Devices in a rank can be either in working or power-saving modes. Ranks can be accessed for memory reads/writes only in working mode. Accesses to ranks in power-saving mode will cause devices in the rank to transition to working mode after a non-zero latency. Energy consumption in working mode has three components:

1. Active energy: This consists of activate, read/write, transmit, and precharge energy.
2. Background energy: This is the energy consumed by the DRAM circuitry.
3. Refresh energy: This is the energy used during refresh operations.

When accesses are not needed to some ranks, they can be put in low power mode. We consider the following power-saving modes:

1. DPD: all data in the devices in the rank is lost.
2. PASR(n=1,1/2,1/4,1/8): Only fraction n of each device in the rank holds valid data. The data in the remaining (1-n) fraction of the device is lost.
3. Low Refresh (LR): The memory arrays are refreshed at a lower rate than normal resulting in power savings as well as occasional data retention errors.

3.3 Power Calculations

3.3.1 Working Mode Power Calculation

We calculate power per rank in working mode by calculating energy consumed over a 1-second interval. This gives average power consumed over this interval. As discussed in subsection 3.2.3, energy consumption in this mode has four components, each of which is discussed in detail below. Table 3 lists parameter values used in the calculations.

1. **Active energy/rank**: Active energy consumed by a rank for each cache line access is the sum of activation, read/write, transmit and precharge operations per device in the rank. Assuming that reads and writes are in the ratio 2:1, we have

LPDDR2-S4 Parameters	
Number of devices per rank (n_{DR})	2
Voltage(V_{DD2})	1.2 V
Activate-Precharge Current(I_{DD02})	65 mA
Active Standby Current(I_{DD3N2})	24 mA
Pre-Charge Standby Current (I_{DD2N2})	16 mA
Pre-Charge Power-Down Current (I_{DD2P2})	1.6 mA
Burst Read Current (I_{DD4R2})	220 mA
Burst Write Current (I_{DD4W2})	185 mA
Burst Refresh Current (I_{DD52})	130 mA
Deep Power Down Current (I_{DD82})	30 μA
No. of DQs (n_{DQ})	18
Capacitive Loading/DQ (C_{DQ})	20 pF

Table 3. Parameters used to calculate power in LPDDR2-S4 DRAM. Current and DQ values are per device.

$$E_R(Active) = n_{DR} *$$
$$\left(E(ACT + PRE) + \frac{2}{3}E(RD) + \frac{1}{3}E(WR) + E(DQ)\right)$$

The individual components are calculated as follows:

- **E(ACT+PRE)**: This is the energy consumed by activate and precharge operations per device in each rank for a cache line access.

$$E(ACT + PRE) = V_{DD2} *$$
$$(I_{DD02} * tRC - I_{DD3N2} * tRAS - I_{DD2N2} * tRP)$$

- **E(RD)**: This is the energy consumed due to reads per device in each rank for a cache line read.

$$E(RD) = V_{DD2} * (I_{DD4R2} - I_{DD3N2}) * 4 * tCK$$

- **E(WR)**: This is the energy consumed due to writes per device in each rank for a cache line write.

$$E(WR) = V_{DD2} * (I_{DD4W2} - I_{DD3N2}) * 4 * tCK$$

- **E(DQ)**: This is the energy used to drive the output pins (DQs) per device in each rank for a cache line access.

$$E(DQ) = n_{DQ} * C_{DQ} * V_{DD2}^2 * \left(2 * \frac{1}{tCK}\right) * 4 * tCK$$

2. **Background energy/rank**: A rank consumes different background energy when it is accessed from when it is not. The total background energy of a rank is thus

$$E_R(BG) = n_{DR} * (t_{acc} * E(BG_{acc}) + \overline{t_{acc}} * E(BG_{\overline{acc}}))$$

The different components in the above equation are calculated as follows:

- t_{acc} is the time, measured in units of tRC, that the rank had a read/write access.
- $t_{\overline{acc}}$ is the time, measured in units of tRC, that the rank had no accesses.

  Assuming non-overlapping accesses in close-page mode, total tRC cycles/second/rank = $\frac{1}{tRC}$. In that case, $t_{acc} =$ #accesses per second per rank, and $t_{\overline{acc}} = \frac{1}{tRC} - t_{acc}$.

- **E**(BG_{acc}): This is the background energy consumed by a rank over a time period of tRC when there was a read/write access to the rank.

$$E(BG_{acc}) = V_{DD2} * (I_{DD3N2} * tRAS + I_{DD2N2} * tRP)$$

- **E**($BG_{\overline{acc}}$): This is the background energy consumed by a rank over a time period of tRC when there was no access to the rank. We assume that the rank is in precharge powerdown mode when not accessed.

$$E(BG_{\overline{acc}}) = V_{DD2} * I_{DD2P2} * tRC$$

3. **Refresh energy/rank**: Since the interval between consecutive refreshes is 3.9 μsec, each device in a rank is refreshed (1sec/3.9μsec) times per second. The refresh energy per rank is calculated as:

$$E_R(refresh) = n_{DR} * \frac{1}{3.9 * 10^{-6}} * E(refresh)$$

- **E(refresh)**: This is the refresh energy per row per device in each rank for a single refresh operation.

$$E(refresh) = V_{DD2} * (I_{DD52} - I_{DD3N2}) * tRFCab$$

3.3.2 Power-Saving Mode Power Calculation

Config.	Current(mA)	Power/device (mW)	Power/rank (mW)
PASR(1)	2.5	3.00	6.00
PASR(1/2)	2.0	2.40	4.80
PASR(1/4)	1.7	2.04	4.08
PASR(1/8)	1.5	1.80	3.60

Table 4. PASR power calculation. Voltage is 1.2V.

1. **DPD power/rank**: Deep Power Down offers maximum savings. The rank under DPD consumes negligible power ($n_{DR} * V_{DD2} * I_{DD82}$).

2. **PASR power/rank**: We calculate PASR power from LPDDR-S4 datasheet as shown in Table 4. We estimate self-refresh power with PASR(1) power.

3. **LR power/rank**: We calculate Low Refresh power similar to Flikker [9] by linearly interpolating PASR values to determine the constant power consumed by the DRAM controller circuitry at any refresh rate. From Table 4, PASR(1)-PASR(1/8)=2.4 mW/rank for $1 - \frac{1}{8} = \frac{7}{8}$ part shutdown ($\sim$infinitely slow refresh cycle). Thus, constant power consumed by control circuitry = $6.00 - \frac{8}{7} * 2.40 =\sim 3.26$ mW/rank. With a different refresh interval of *r* msec with LR (as compared to 32 msec interval with the baseline), LR power/rank = $3.26 + \frac{r}{32} * (6.00 - 3.26)$ mW/rank.

PASR/LR extra copy power: Compared to the baseline with only working memory, PASR/LR memory incurs extra overhead in occasionally copying pages to working memory. We note that 1 page = 4096 bytes = 64 cache lines. So, the extra power includes 1 activate + 64 reads from PASR/LR memory (brought out from PASR) + 64 writes to working memory - 64 reads from working memory (the reads would anyway have happened in the baseline) + 1 precharge.

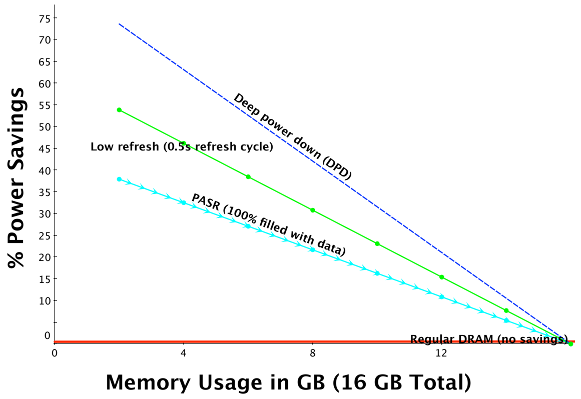


Figure 3. Power savings for different amounts of DRAM at 3.46 million words/sec reference rate.

3.4 Results

We calculate total memory power using Equation 1 where the extra power term is non-zero only for PASR/LR modes. The baseline system has all ranks in working mode. For the results in this section, we assume that the number of extra page-copy operations for PASR/LR modes is 1% of the memory access rate.

3.4.1 Power savings based on DRAM usage

Figure 3 shows percent power savings for different amounts of DRAM under mercurial cache, with total memory capacity being 16GB. The figure represents power savings when different amounts of memory are moved into a mercurial cache with a total memory access rate of 3.46 million words/second. The access rate was measured using hardware performance counters on a Intel Quad Core i5 1.6 Ghz machine with 16GB DRAM when running *filebench* with the fileserver workload. We find using our model that all three techniques, DPD, LR, and PASR save power depending on the amount of DRAM being used. The power savings results from consolidation of memory ranks under use which only consume refresh (or self-refresh) power. Apart from savings resulting from reduction in refresh power, active power is consumed for fewer banks.

3.4.2 Sustainable Reference Rates

We also calculate the number of references to mercurial cache can sustain to provide power savings before the cost of checksum takes over the savings from DRAM. We alter the mercurial cache reference rate while keeping the mercurial cache size constant at 50% of total DRAM size in Figure 4. We find that mercurial caches can sustain very high reference rates of around 1.2 million words/second while still providing memory savings.

4. Design Overview

To support the low powered memory hardware, we introduce *mercurial caches* in operating systems. Based on the *do no*

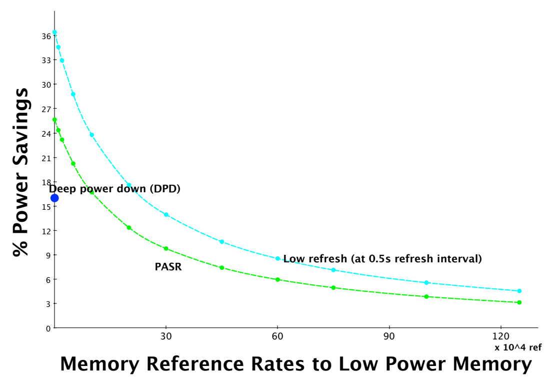


Figure 4. Power savings at 50% DRAM in low power for different mercurial cache reference rates.

harm principle, we list four goals of supporting mercurial caches:

1. *Reliable Interface*: Mercurial caches should expose reliable software interface. Even if data is stored at low refresh rates, mercurial caches should be able to return data in a fail-safe manner.
2. *Non-interfering*: Mercurial caches should not interfere the VM's prefetch and caching behavior. They should also not affect the working set of running programs.
3. *Energy Proportional*: Mercurial caches should be able to identify unused or low activity portions of physical memory and put them in low power state.
4. *Little/no-overhead*: Mercurial caches should have low performance overhead and perform much better than simply turning off DRAM.
5. *Hardware Interface*: Mercurial caches should expose the memory controller hardware physically contiguous low power chunks to be put to low power state.

Mercurial caches appropriately size virtual memory to switch unused memory in low memory state. Depending on the type of hardware used, these caches provide the ability to either switch off memory refresh (DPD), cache clean pages (PASR) or store them at low refresh rates (LR) while other pages such as OS and applications use regular memory refreshed at normal rates to achieve energy proportionality. Supporting mercurial caches require addressing three challenges. First, mercurial caches need to provide an interface to store and retrieve pages in a fail-safe manner. Second, we need policies to identify *automatically* the amount of memory being used and move unused pages to low power state. Finally, we need to provide mechanisms that will help consolidate physical memory to move it to a low power state.

Mercurial Cache Interface: Mercurial caches provide an interface to store and retrieve 4K pages into the low powered memory (`mcache_get_page` and `mcache_put_page`). These operations require pages to be copied to/from regular memory. However, since the low power memory is unreliable, mercurial caches compute the page checksum during store and retrieve operations. If the checksums do not match, indicating that the page has become corrupt, then the retrieve operation (`mcache_get_page`), may fail. The cost of copying pages can be reduced by using hardware support [8] and we evaluate the software cost of copy/checksum.

Make mercurial caches transparent to VM: Mercurial cache support requires modification to VM to address three challenges. First, how do we ensure that mercurial caches dynamically occupy memory depending on application needs for best performance. We solve this problem similar to hypervisors by using the free page information to detect applications working set. Second, mercurial caches should not appear as allocated memory so that system functions such as prefetching and allocations do not fail. We modify the VM to account for mercurial caches when calculating free memory for all system actions. Third, when mercurial caches are used to store data, we need policies on what data to store. Currently, we use mercurial caches as a third level eviction cache and used evicted page cache information in self-refresh or at low refresh rates, when applications become finish.

Coalescing memory to accommodate Mercurial Cache: Mercurial caches support dynamic creation and freeing of pools. This requires sufficient physically contiguous memory in the system in chunks of minimum DRAM size that can be partially refreshed. To restrict fragmentation, we modify the OS memory management to ensure such an allocation is possible. First, we mark pages in mercurial cache as non-pinnable for long term usage (`GFP_MOVABLE`). This ensures that we do not have holes that can prevent coalescing of memory. Second, when the available contiguous memory is low, we migrate pages and de-fragment the physical address space. This ensures that we can dynamically enable/disable mercurial caches.

5. Mercurial Cache Implementation

We implemented mercurial cache in Linux 3.4.1.

5.1 Mercurial Cache Interface

Mercurial caches provide an interface to store and retrieve 4K pages into the low powered memory (`mcache_get_page` and `mcache_put_page`). These operations require pages to be copied to/from regular memory to low power memory. Both reads and writes are destructive i.e. release the source when copied over to the destination.

Furthermore, low powered memory can be unreliable(Low Refresh). Hence, mcache compute the page checksum during store and retrieve operations. The cost of copying pages can be reduced by using hardware support in DRAM or other hardware [8]. If the checksums do not match, indicating that the page has become corrupt, then the retrieve operation (`mcache_get_page`), may fail. This behavior is similar to transcendental memory in Linux [18], which provides memory consolidation for virtual machines where memory used by a virtual machine using the transcendental API may be re-allocated to another virtual machine. We implement a backend driver for Linux transcendental memory support, an additional kernel module to manage mercurial cache pools. The driver using transcendental API to manage and store evicted pages from the page cache in low powered memory pools (of 128MB) managed as ring buffer. The circular ring buffer allocates memory at page granularity using bitmaps. Since mcache requires extra hardware support that is not available, as memory is moved in/out of low power, we appropriately set the specific DRAM segment in low power using a fake DRAM controller. We use modified PASR patches in Linux that map physical DRAM locations to physical memory address chunks [].

5.2 Virtual Memory Integration

Mercurial cache support requires modification to the OS memory management algorithms in Linux. We have three challenges of mercurial cache integration in Linux:

1. Mercurial cache sizes should dynamically move from low power to DRAM memory depending on applications being run. This requires accurately identifying the application's working set requirements.
2. Mercurial caches should not appear as missing memory or allocated memory to the VM subsystem. We need to identify VM functions that use memory accounting for various decisions and account for mercurial caches.
3. Mercurial caches can be used to store data instead of being turned off. This data is sparingly used but is still faster than going to disk when we are not able to accurately predict the working set. We need appropriate policies on when a page should be moved to disk.

5.2.1 Dynamically moving DRAM to low power

To identify idle times of memory usage and provide power benefits, mercurial cache should dynamically grow and shrink in memory and actively move memory between low power and normal states.

Mercurial caches use the number of free pages and other VM signals to identify memory demands in a system in the following ways. First, when the system boots mercurial caches spawns a thread every 10 seconds and tries to allocate as much memory as possible for in mcache pools in 128MB chunks. It then establishes a high watermark for the number of free pages that can be safely allocated without triggering reclamation and a low watermark which indicates the system is facing memory pressure. The thread periodically wakes up and checks if there are free pages above this watermark to safely allocate a mcache pool. Also, if the number of free pages begin to drop mercurial cache releases a pool. Second, we modify the VM subsystem to release pools, when allocations may trigger reclamation (and subsequently retry), even for page caches. This ensures that spikes of memory demand are immediately satisfied. As the applications continue to run and demand more memory, mercurial cache relinquishes more pools. Finally, as applications finish executing (as indicated by large influx of free pages), mercurial caches attempt to recover memory back into low power state. Furthermore, as physical pages are allocated and de-allocated frequently, physical memory fragmentation can make it harder to allocate mercurial cache pools. Mercurial caches automatically inflates the low watermark to accommodate fragmentation. When the fragmentation levels are very high, as indicated by the low watermark, mercurial cache invokes coalescing which we describe later in this section.

5.2.2 Pre fetching and allocations

The VM subsystem rapidly dynamically adapts to provide best performance for a given amount of memory. For example, the VM subsystem, prefetches a number of pages in memory depending on the size of the available memory. The VM subsystem is modified such that mercurial caches is seen as free memory, and prefetching decisions are not adversely affected by mercurial caches.When the system attempts to allocate memory for page cache or makes large allocations, mercurial caches release memory to ensure these allocations are satisfied without triggering a reclamation. We modify free memory accounting done by Linux VM by modifying `vmstat`, which monitors free memory on a system and Linux memory zone wide basis. Other VM functions such as allocation, prefetching and other system caching (such as `dentry` caching) query the `vmstat` to obtain this information and balance the system. Furthermore, we also modify allocation routines, such that mcache pools are released if an allocation would trigger page reclamation to avoid slowing down memory allocation.

5.2.3 VM changes/When to push pages in mcache

When mercurial caches are used to store data (PASR, LR), we need to decide which pages to move to mercurial cache. We identify the page cache as one of the largest consumers of memory. As pages age out from page cache, file pages are evicted out as clean pages. We modified the virtual memory(VM) system to store pages in low power memory. When the page is referenced again, its read from the low power cache instead of going to the disk. Having a third level cache allows us to improve performance, if the VM system is inaccurate or mercurial cache is not precise enough in calculating the free space requirements. Furthermore, when applications finish, mcache re-allocates memory and drains out the page cache. When running in DPD mode, subsequent runs of application will hit the disk, but when running PASR or LR modes, applications can still access data cached in page cache.

Using these pools as a third level cache does not affect the existing working set. Since mercurial caches uses its pools similar to L2ARC [**?**]. The pools are maintained as ring buffer, allocated using a bitmap allocating in fixed size page granularity. As soon as pages in LRU list are purged and moved to disk, they are moved to mercurial caches. Whenever there is memory pressure, mercurial caches are purged on a pool basis instead of evicting the oldest blocks to maximize memory savings. While these may favor purging of newer blocks over older blocks, the hottest blocks are already maintained in the LRU list.

5.3 Physical Memory Fragmentation

In order to provide power savings, the memory allocated for a mercurial cache needs to be physically contiguous. This is because the hardware power saving techniques such as Deep Power Down and PASR can only be used at minimum granularity of physically contiguous pages (such as a rank).

To ensure fragmentation does not become a bottleneck, we modify the OS memory management to ensure such an allocation is possible. Mercurial caches uses two complimentary techniques to ensure this is possible - (i) Anti-fragmentation and (ii) Coalescing.

The OS can allocate pages in system memory that are pinned (unmovable, usually kernel allocations), reclaimable (such as file mapped pages) or movable (user space pages). Mercurial caches uses anti-fragmentation by ensuring that pages are not pinned for long term storage across physical memory. It marks pages in physical memory as non-pinnable for long term usage (such as by using `GFP_MOVABLE` flag).

Second, when the amount of free contiguous memory is low, mercurial cache re-maps pages and de-fragments the physical address space. This ensures that we can dynamically enable/disable mercurial caches. As mentioned in section **FIXME [5.2]???**, when the system has large number of free pages and yet mercurial cache is unable to allocate memory for low power pools, it triggers coalescing of physical memory. This is done by scanning physical memory blocks in memory in fixed size chunks, isolating and marking them offline and migrating pages away from these blocks and releasing the memory block as free space to the zone allocator.

6. Evaluation

6.1 Hardware PASR model

The evaluation examines the following aspects of mcache:

1. *Performance.* What is the performance overhead of using mercurial caches? We report the performance cost of providing OS support for each of the hardware power saving techniques.
2. *Power Savings.* What is the power savings obtained from using mercurial caches? We evaluate the power saved in idle situations and while using different workloads.
3. *Migration and Reservation Time.* How well do our de-fragmentation techniques work? We evaluate time required and performance overhead of coalescing chunks of memory.

Workload	Native	Hotplug	DPD	PASR	LR
ImageMagick	36.5	-	35.6	34.0	0
Spin	56.8	-	57.96	57.0	0
SpecJBB	0	-	0	0	0

Table 5. Execution times for different workloads.

Workload	Native	Hotplug	DPD	PASR	LR
fileserver	225.0	31.3	221.0	222.0	222.0
webserver	82.6 MB/s	0	82	82.9	83.3%

Table 6. Throughput for different workloads. Mercurial caches automatically adjust size of memory as the workload executes. PASR, DPD and LR use the same workload prediction component.

Workload	Native	Hotplug	DPD	PASR	LR
ImageMagick	36.5	-	35.6	34.0	0
Spin	56.8	-	57.96	57.0	0
SpecJBB	0	-	0	0	0

Table 7. Power savings for different workloads.

Unless otherwise specified, we compare mcache against an unmodified 3.4.1 Linux kernel.

6.2 Workloads

We evaluate performance with four memory-intensive application workloads with varying working set sizes:

1. *filebench*, running the fileserver, web server and oltp workloads.
2. ImageMagick 6.7.5, resizing a large (385MB) TIF image by 150%.
3. Remote file copy, remotely copies xGB data over xx files.
4. Spin 5.2 [31], an LTL model checker for testing mutual exclusion and race conditions with a depth of 10 million states,
5. pseudo-SpecJBB, a modied SpecJBB 2005 benchmark to measure execution time for 16 concurrent data warehouses with 1 GB JVM heap size using Sun JDK 1.6.0.

6.3 Performance

6.4 Power Savings

- sustained power benefits?

- power savings vs energy policies

- we do not just save refresh power, if pages are distributed throughout memory, we also save active power

6.5 Benefits of PASR

6.6 Migration and Reservation

We evaluate the overhead and time required to de-fragment different chunk sizes of memory. The de-fragmentation is done by a separate background thread. Table 9 gives the time required to perform de-fragmentation after running Im-

Workload	Cold cache	Page cache	DPD	PASR	LR
Read File (2GB)	5.5s	0.22s	5.4s	0.76s	0s
Kernel Compile	12m		22m	17m	0m

Table 8. PASR stores data in low power.

Size	Time	Throughput
128MB	6.4 ms	100%
256MB	13.3ms	100%
512MB	36 ms	100%

Table 9. Throughput for different workloads.

ageMagick workload to resize a 400MB image by 120% and 200%. We record the time required to de-fragment first chunk of 128MB or more. We also verify that after defragmentation, mcache/ is able to allocate and put memory in low power state.

We also execute

7. Related work

Mercurial caches build on existing work in application and OS support to provide energy efficient DRAM. Flickker [9], proposes application changes to split applications into critical and fault tolerant components and stores fault tolerant data (such as images) into low refresh DRAM. However, Flickker requires application changes and qualitatively effects application output. Furthermore, Flickker models memory savings on application behavior which is not representative of system wide memory consumption in environments such as servers. Mercurial caches propose OS level changes to reduce power, with no changes to applications or their outputs and reduces power consumption from idle/inactive memory. RAPID [20], proposes saving DRAM power by prioritizing low refresh DRAM rows while allocating OS pages since different DRAM rows have different refresh thresholds for safely storing data. However, once all low refresh DRAM rows are filled, RAPID is not able to offer any further power saving benefits. RAPID, also does not address complications due to fragmentation and pinning of OS pages in its trace based evaluation. Superpages [17] deals with fragmentation issues in memory in order to provide pages of large sizes to mitigate TLB pressure. Superpages dealt with rather smaller chunks of contiguous pages (upto 4MB) while mercurial caches require large size contiguous memory and will need more aggressive memory reservation policies.

8. Conclusion

In this paper, we introduce OS support for memory energy proportionality based on DRAM utilization. We identify that current operating systems utilize all available memory for caching purposes and propose a low power memory consolidation technique that provides an abstraction for low powered caches called mercurial caches. Mercurial caches store clean disk pages at low refresh rates that retains performance gains from caches and provides energy proportional power savings.

References

[1] Andi Kleen, SUSE Labs. Where is the memory going? memory usage in the 2.6 kernel. `http://halobates.de/memory.pdf`.

[2] L. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[3] L. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.

[4] R. W. Carr and J. L. Hennessy. WSCLOCKa simple and effective algorithm for virtual memory management. In *SOSP*, 1981.

[5] ELPIDA Inc. Low Power Function of Mobile RAM Partial Array Self Refresh (PASR). `http://www.elpida.com/pdfs/E0597E10.pdf`, 2005.

[6] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[7] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification, version 5.0. `www.acpi.info/spec.htm`, Dec. 2011.

[8] Intel Corporation. Accelerating high-speed networking with intel i/o acceleration technology. `http://download.intel.com/support/network/sb/98856.pdf`, 2006.

[9] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn. Flikker: Saving refresh-power in mobile devices through critical data partitioning. *ASPLOS*, 2011.

[10] K. Malladi, F. Nothaft, K. Periyathambi, B. Lee, C. Kozyrakis, and M. Horowitz. Towards energy-proportional datacenter memory with mobile dram. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, june 2012.

[11] D. Meisner, B. Gold, and T. Wenisch. PowerNap: eliminating server idle power. *ISCA*, 2009.

[12] Micron Corporation. 2gb: x16, x32 mobile LPDDR2 SDRAM s4 features.

[13] Micron Corporation. Tn-41-01: Calculating memory system power for DDR3.

[14] Micron Corporation. Tn-46-12: Mobile DRAM power-saving features/calculations.

[15] Micron Corporation. Mobile dram power-saving features and power calculations. `http://www.micron.com/support/dram/~/media/Documents/Products/TechnicalNote/DRAM/184tn4612.ashx`, 2005.

[16] Micron Corporation. Low-power versus standard ddr sdram. `http://download.micron.com/pdf/technotes/DDR/tn4615.pdf`, 2007.

[17] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, transparent operating system support for superpages. *ACM SIGOPS Operating Systems Review*, 36(SI):89–104, 2002.

[18] Oracle Corp. Project: Transcendent Memory. `https://oss.oracle.com/projects/tmem`, 2010.

[19] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of the 3nd ACM Symposium on Cloud Computing, SOCC*, volume 12, 2012.

[20] R. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile dram. In *HPCA 2006.*

[21] J. Veras and O. Mutlu. RAIDR: Retention-aware intelligent dram refresh. In *ISCA*, 2012.

[22] Vimal Bhalodia. SCALE DRAM subsystem power analysis. Master's thesis, Massachusetts Institute of Technology, 2005.

[23] D. Yoon, J. Chang, N. Muralimanohar, and P. Ranganathan. BOOM: Enabling mobile memory based low-power server DIMMs. In *ISCA*, June 2012.