Mercurial Caches: OS Support for Energy Proportional DRAM

Asim Kadav and Michael M. Swift
Computer Sciences Department, University of Wisconsin-Madison
Under Submission

Abstract

DRAM has become one of the significant consumers of power, accounting for a large percentage of total power in servers. While many hardware technologies for improving power efficiency have emerged, energy proportionality of DRAMs cannot be achieved. This is because modern operating systems continuously occupy all available memory as file cache and spread active pages across memory making consolidation for power savings impossible.

In this paper, we propose OS support for energy proportional memory consumption by supporting low-powered unreliable memory in operating systems, with *mercurial caches*. Mercurial caches provide a copy-in/out interface to memory running at low refresh rates and support large system caches such as the clean pages from OS page cache.

We describe our design of how mercurial caches can be integrated in current operating systems. Through an analytical model, we demonstrate that it can provide energy savings proportional to the DRAM under active usage ranging from 1-19%, while requiring no changes to existing applications.

1 Introduction

DRAM consumes significant power, accounting for up to 25 - 57% in servers [3, 10, 18]. With modern servers provisioned with tens of gigabytes of memory, DRAM power is increasingly dominating the power cost of idle servers. As a result, many existing research projects provide hardware and software support to reduce this power draw [9, 10, 17, 18].

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 significant exit penalties [10].

Partial Array Self Refresh (PASR) [5] provides the ability to vary refresh rates of parts of DRAM and save energy by only refreshing actively used memory. However, such techniques cannot be directly used with existing applications. Operating systems manage system wide memory and prevent any power savings for two reasons. First, existing operating systems are designed to use all possible memory for performance reasons. Since most paging algorithms only approximate the application memory consumption patterns, all free memory is used to cache disk pages that may be referenced in future. Second, a running operating system over time fragments physical memory making memory consolidation expensive. Since the OS does not need more than few MBs of contiguous memory, support for de-fragmenting memory is not available inside OS. However, PASR requires contiguous free pages adding up to at least 1/16th of a DIMM (up to 1 GB) to reduce power in DRAMs.

In this paper, we provide energy proportionality for main memory by providing support for *mercurial* caches. A mercurial cache is a low power cache with support for software error detection to survive corruption from low refresh DRAM. Mercurial cache stores clean pages from OS caching subsystems such as the page cache in low power DRAM. As pages age out in the page cache, they are moved into low power DRAM. If these pages are referenced again they are copied back into regular DRAM and refreshed at regular rates. Since low power caches are inherently error prone (some cells may lose data when running at low refresh rates [16]), mercurial caches use software checksums for error detection each time pages are copied from mercurial cache. Hence, a mercurial cache enables operating systems to use all available memory but performs caching in low power DRAM. Furthermore, mercurial caches also provide policies that decide which pages in the OS should move to the mercurial cache and provides mechanisms that prevent system memory from being fragmented.

In this paper, we describe our proposed design and evaluate it using an analytical model. We find the potential cost savings resulting from supporting mercurial caches range from 1-19% of DRAM power, depending on the amount of memory being used.

2 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 best 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 service page requests quickly 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.

Another characteristic of memory management in modern operating systems is fragmentation of physical addresses. Due to frequent small memory allocations, the physical memory gets fragmented over time. Since requests to allocate physically contiguous memory are rare (like super pages) and are often of the order of few MBs, fragmentation is not a problem for any existing system or application functions. Hence, defragmentation adds little value to OS and only adds up as a cost if it is performed 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.

DRAM technology	**Data retention**	**Granularity**	**Latency**	**Power savings**
ACPI S4	No	All DRAM	> 1 s	100%
Deep power-down	No	All DRAM	200 μs	95%
Self-refresh	Yes	All DRAM	1 μs	95%
Clock-stop	Yes	All DRAM	200 μs	83%
TCSR	Yes	All DRAM	10 ns	60%
PASR	Flaky	1/16th DIMM	10 ns	22%

Table 1: **Comparison of different DRAM power saving technologies. Many technologies in the figure can co-exist. TCSR savings are for 40C drop for 64MB DIMM.**

2.2 Available hardware technologies

In this sub-section, we review available techniques to reduce DRAM power in server class DDR2/DDR3 memories and also mobile class LPDDR2 DRAM. DRAMs store data in the form of capacitive charge. Since capacitors leak charge over time, this charge must be periodically refreshed or else the data stored is slowly lost. DRAM refresh times are dependent on external factors such as temperature, capacity. 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, 16, 17]. Hence, many DRAM vendors offer technologies to lengthen the refresh times by exposing this control knob to applications.

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 [6].

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 [11].

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 [12].

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 [12].

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 [12].

6. **PASR DRAM**: Partial Array Self-Refresh(PASR) is an enhancement to self-refresh which provides the ability to refresh memory at different granularities, upto 1/16th of a bank [11] 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, 11]. When the system memory consumption is low, PASR can be used with appropriate OS support to reduce DRAM power.

Comparing these technologies as shown in Table 1, the last technique, PASR DRAM, looks promising. It exposes control knobs to reduce DRAM power based on memory usage and can be used to provide energy-proportional RAM usage. In the next section, we describe how one can provide OS support for PASR DRAM.

3 Proposed solution: Mercurial Caches

To reduce power consumption proportional to memory usage without reducing the amount of available memory, we introduce *mercurial caches*. These caches provide the ability to cache clean pages at low refresh rates while other pages such as OS and applications use memory refreshed at normal rates. Mercurial caches use PASR support that allows DRAM in multiples of 1/16th of DIMM to be refreshed at lower rates. However, this requires the support for physically contiguous pages in memory and software error detection.

PASR DRAM support can also be provided in OS by constraining memory usage by using existing memory stealing techniques such as memory hotplug and ballooning [15] or compression [8]. This memory can be turned off to save power. However, this technique reduces the amount of memory available to the OS and will degrade the overall performance of the system. Mercurial caches adopt a middle ground by using slightly more power than turning off the DRAM but do not degrade performance. They tradeoff power for reliability (instead of performance), and verify correctness using software checksums.

Supporting mercurial cache requires addressing three challenges. First, mercurial caches need to provide an interface to store and retrieve pages in a fail safe manner. Second, they need to identify what pages in memory management code should be served by mercurial caches. Finally, mercurial cache support needs to provide mechanisms that will help consolidate memory to move it to a low power state.

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. However, since the low powered 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. This behavior is similar to transcendental memory in Linux [14], 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. Hence, we intend to re-use the existing transcendental memory interface in Linux (our choice of OS for implementation) to provide support for Mercurial caches. The cost of copying pages can be reduced by using hardware support [7] and we evaluate the software cost of copy/checksum in the next sub-section.

2. **OS support to migrate pages to Mercurial caches**: Mercurial cache support requires modification to the OS memory management algorithms in Linux. We identify the page cache as one of the largest consumers of memory. We intend to modify the LRU approximation algorithm in Linux. The LRU algorithm stores pages in active and in-active lists depending on the frequency of page usage. Pages are aged out from the active to in-active list and finally to the disk. We intend to move clean pages from the in-active list into the mercurial caches. If the mercurial cache returns a corrupt page, the page fault handler can re-read the page from the disk. Hence, we introduce additional transitions in the LRU algorithm where clean inactive pages are moved to mercurial caches and are moved to the active list (upon reference) or evicted (after timeout).

While the page cache represents one of the largest consumers of idle memory, we also intend to investigate other system caches that can provide power savings from using the mercurial cache. For example, the swap cache, which stores a portion of swap space in memory. Another use case can be proving a memory allocation zone for the user to directly allocate memory from mercurial caches. A common use of memory is as a cache for file system contents: for example, the memcached service is used as a distributed cache of read-only data. The contents of the cache, though, are always stored persistently on disk to survive a power failure and can be re-fetched from storage.

Model variables	
Total memory	8 GB/DIMM
CPU power/core	16.25W
Power per DIMM	3W
Low power/DIMM	2.34W
Page copy time	0.00000097 sec
Page checksum time	0.000000027 sec

Table 2: **I/P parameters for analytical model. DRAM PASR savings are from Flikker [9] and page copy/checksum times are measurements on a Pentium-D 3.0Ghz machine.**

3. **Coalescing memory to accomodate Mercurial Cache**: In order to provide power savings, the memory allocated for a mercurial cache needs to be physically contiguous. To use the existing PASR hardware and move pages in/out off mercurial cache dynamically, mercurial caches need to allocate and release physically contiguous memory dynamically. This memory allocation needs to be in chunks of minimum DRAM size that can be partially refreshed (currently 1/16th of DIMM size). To ensure fragmentation does not become a bottleneck, 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 (such as by using `GFP_MOVABLE` flag). This ensures that we do not have holes that can prevent coalescing of memory. Second, when the amount of free contiguous memory is low, we re-map pages and de-fragment the physical address space. This ensures that we can dynamically enable/disable mercurial caches.

Hence, mercurial caches save memory by moving clean memory pages into low power memory. Low power memory saves power because it is refreshed at long intervals. However, it also introduces additional cost of checksum and copy-in to regular memory during reference. In the next section, we evaluate the benefits of using PASR for caching in OS.

4 Evaluation

We construct an analytical model to understand the power savings from mercurial cache. We model our results on PASR enabled RAM for memory such as LPDDR2, which are used in mobile systems. Recent research projects have also demonstrated that they can be reliably used in servers [18]. To model PASR behavior, we use PASR values from Flikker [9], and pick a refresh rate of 1 second, which gives us 22% reduction in power and error rate probability of $4x10^{-8}$ per bit. Other parameters of the analytical model are given in Table 1.

We calculate the total power consumed by a single DIMM at any instant as the sum total of power consumed by memory under normal refresh rate and the power consumed by memory under low refresh rate (mercurial cache) and the additional power spent on copying

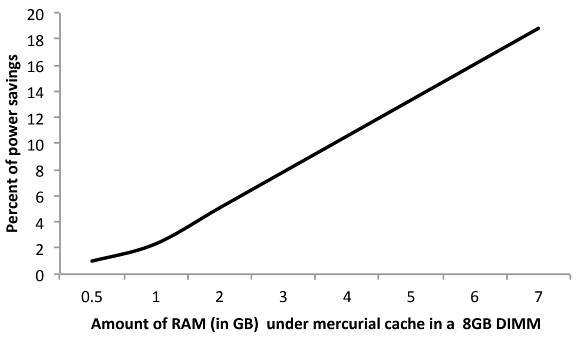

Figure 1: **Power savings for different amounts of DRAM at 1 million words/sec reference rate.**

the page in/out of mercurial cache and checksum when pages in mercurial caches are referenced.

$P_{DIMM} = P_{refresh\_mcache} + P_{refresh\_other} + P_{checksum\&copy} x References/sec$

We calculate the CPU power consumed for calculating checksum and copy operations as the product of the time to execute these operations with the CPU power and the mercurial cache reference rate.

$P_{DIMM} = RAM_{mcache}/RAM_{total} * P_{PASRfraction} * P_{DIMM} + (RAM_{total} - RAM_{mcache})/RAM_{total} * P_{DIMM} + 2 * Time_{checksum\&copy} * P_{CPU} * Ref_{pagecachesec}$

We model the above equation in Figure 1 for different amounts of DRAM under mercurial cache for an 8GB DIMM. The figure represents power savings for a single DIMM when different amounts of memory are moved into a mercurial cache at a reference rate of 1 million words/second. We see that using a mercurial cache results in memory savings even when a small amount of memory is being unused (˜1% savings for 500MB out of 8GB total) and offers proportional savings as more memory is moved into mercurial cache (19% savings for 7 out of 8GB). 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 in Figure 2. We find that mercurial caches can sustain a reference rate of around 1.2 million words/second before the cost of checksum/copy dominates over the savings from DRAM power.

Reliability tradeoff: Mercurial caches serves 4K pages with error probability of $1.6x10^{-4}$ per page. If instead of using caches, we just turned off the cache, the amount of available memory to OS is reduced, and the reference rate drops to zero. This will result in excessive page faults and may cause thrashing.

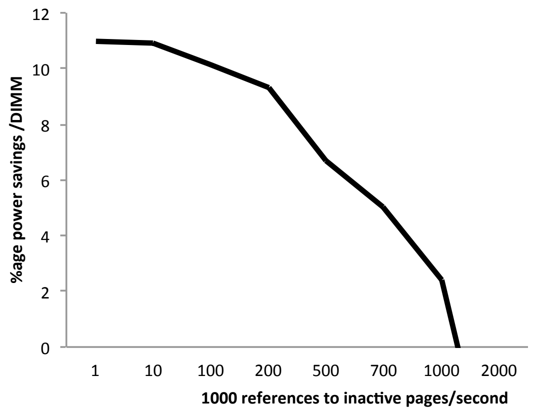

Figure 2: **Power savings at 50% DRAM in low power for different mercurial cache reference rates.**

We are in the process of building our OS subsystem to support mercurial cache. Through an OS implementation, we will be able to quantify benefits and compare different policies for caching OS pages and memory reservation.

5 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 [16], 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 [13] 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.

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

[3] L.A. 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] Richard W. Carr and John 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] 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`, December 2011.

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

[8] Jonathan Corbet, Linux Weekly News. zcache: a compressed page cache. `http://lwn.net/Articles/397574`, 2010.

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

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

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

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

[13] 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.

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

[15] J.H. Schopp, K. Fraser, and M.J. Silbermann. Resizing memory with balloons and hotplug. In *Proceedings of the Linux Symposium*, volume 2, pages 313–319, 2006.

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

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

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