Mercurial Caches: OS Support for Energy Proportional DRAM

Asim Kadav (student) and Michael M. Swift
Computer Sciences Department, University of Wisconsin-Madison

DRAM has become one of the significant consumers of power, accounting for a large percentage of total power in servers. Energy proportionality of DRAM usage cannot be achieved because modern operating systems continuously occupy all available memory as page cache and spread active pages across memory making consolidation for power savings impossible. In this poster, we propose OS support for energy proportional memory by providing OS support for low-powered unreliable memory. *Mercurial caches* provide a copyin/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 the OS abstractions and preliminary results from an analytical model in supporting mercurial caches in existing operating systems.

Problem: With modern servers provisioned with tens of gigabytes of memory, DRAM power is increasingly dominating the power cost of idle or partially used servers. Processors can save power in partially used servers with existing frequency and voltage scaling techniques. However, energy proportionality in DRAM is limited to time multiplexing DRAM use but they impose significant exit penalties.

Partial Array Self Refresh (PASR) [1] provides the ability to vary refresh rates of parts of DRAM and saves energy by only refreshing actively used memory. However, such techniques cannot be directly used with existing applications since the OS manages system wide memory and prevents power savings for two reasons. First, modern OS's 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, the physical address space is not de-fragmented periodically by the OS. However, PASR requires contiguous pages up to at least 1/16th of a DIMM to reduce power in DRAMs.

Solution: To reduce power consumption proportional to memory usage without reducing the amount of memory, we introduce *mercurial caches*. These caches provide the ability to cache clean pages at low refresh rates (using PASR with low refresh support), while other pages such as OS and applications use memory refreshed at normal rates. 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 which pages should be served by mercurial caches. Finally, we need to provide mechanisms that will help consolidate 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 [2] and we evaluate the software cost of copy/checksum.

VM changes to support mercurial caches: Mercurial cache support requires modification to VM to move clean pages from page cache. We modify the LRU approximation algorithm in Linux and move clean pages from the in-active list into the mercurial caches. If mercurial cache returns a corrupt page, the page fault handler re-reads this 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). Furthermore, the mercurial cache interface is invoked when the kernel needs to shrink the page cache. We also plan to investigate power savings from using the mercurial cache directly from user-mode applications.

Coalescing memory to accomodate 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.

Evaluation: We construct an analytical model to understand the power savings from mercurial cache and demonstrate, that it (1) provide energy savings proportional to the DRAM under active usage ranging from 1-19%, (2) can sustain a reference rate of around 1.2 million words/second before the cost of checksum/copy dominates over power savings and (3) is better turning off memory or swapping to SSD, which affect performance.

References

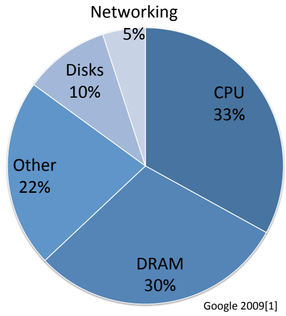
[1] ELPIDA Inc. Low Power Function of Mobile RAM Partial Array Self Refresh (PASR). `http://www.elpida.com/pdfs/E0597E10.pdf`, 2005.

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

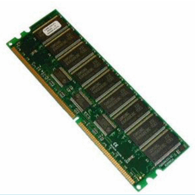
Mercurial Caches: Operating System Support for Energy Proportional DRAM

Asim Kadav and Michael M. Swift, University of Wisconsin-Madison

DRAMs contribute significantly to system power



Google 2009[1]



Servers are provisioned with 10s of GBS of memory and consume 30-57% of total power for a system provisioned with 128 GB DDR3 memory[2]

How can we make DRAM power consumption energy proportional ?

Problem: DRAM is one of the significant consumer of power in modern systems. Unlike CPUs which provide voltage and frequency scaling techniques, DRAM techniques to save power upon partial use are limited.

Goal: Provide abstractions in existing operating systems to utilize low powered memory states in modern operating systems using *mercurial caches*.

DRAM Power-saving Technologies

DRAMs store data in the form of capacitive charge. Since capacitors leak charge over time, this charge must be periodically refreshed else the data stored is slowly lost. Server class DDR2/DDR3 DRAM and mobile class LPDDR2 DRAM save power by lowering the refresh rate or turning off the DIMM completely.

DRAM Technology	Data Retention	Granularity	Latency	Power Savings
ACPI S4	No	All DRAM	>1s	100%
Deep power down	No	All DRAM	200µs	95%
Self-refresh	Yes	All DRAM	100ns	95%
Clock stop	Yes	All DRAM	200µs	83%
Temperature Controlled Self Refresh	Yes	All DRAM	100ns	60%
Partial Array Self Refresh (PASR)	No	1/16th DIMM	100ns	20%

Low power mode in LPDDR or DDR3 DRAM systems. Many technologies in the figure can co-exist. TCSR savings are for 40C drop for 64MB DIMM.

PASR allows turning off or lowering refresh rates of portions of DRAM to save power. How can we support this in modern operating systems ?

OS memory management

We identify characteristics of OS memory management that hinders adoption of PASR.

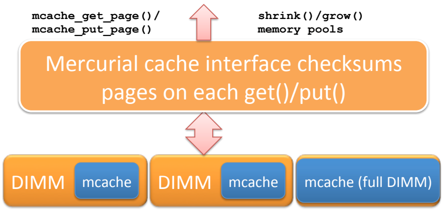
OS Policy	Policy Rationale	Problem
Occupy all memory in the system (as page cache)	OS uses all available memory to improve system performance	There is never any available "free" space to be put to low power state
Fragments physical address space since address space is virtualized.	Contiguous memory requirement is limited (few MBs), keeping address space fragmented is only an overhead.	Memory needs to be put in low powered state in contiguous segments to save power

"Free memory is bad memory" philosophy and physical memory fragmentation hurt any memory consolidation for power savings

Mercurial Caches

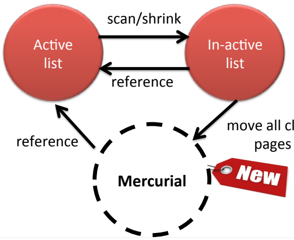
Mercurial caches provide OS abstractions to use low power DRAM. It uses a portion of DRAM in low power state to cache clean data (such as clean page cache). It also uses software checksums to ensure correctness in the absence of reliable hardware. Its goal is to save power with little performance loss during low memory utilization

1. A cache interface to get/put 4K pages reliably and respond to memory pressure



Low power mercurial cache pool per DIMM

2. OS modifications to use mercurial caches: Identify and move memory that can be safely stored in low powered DDR memory



We introduce additional LRU transitions to accommodate mercurial caches. We also provide mercurial cache API to other caches such as swap caches and user space allocations

3. OS support to facilitate PASR granularity pool allocations: Pre-reserve and migrate pages

Problem	Solution
Fragmentation of physical address space	Aggressively migrate all movable pages in system
Migration of pages not possible due to pinned pages	Mark specific segment boundaries dedicated to movable pages

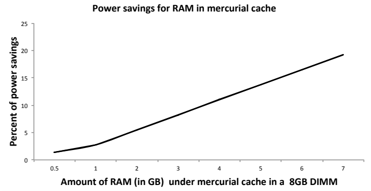
Mercurial caches use reservation policies for movable data and migration mechanisms to switch contiguous memory into low power state

Preliminary Evaluation

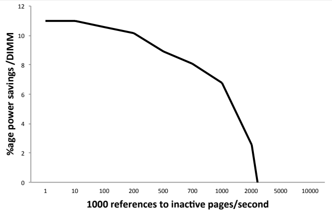
We construct an analytical model to understand the power savings from mercurial cache. We model for standard server hardware with Quad core machine with 8GB DIMMs that support partial refresh at different rates. We measure the software costs of checksum and copy on real hardware.

Model equation: Power(DIMM) =
RAM_mcache/RAM_total ∗ Power_PASRfraction) ∗ Power(DIMM)
+ (RAM_total−RAM_mcache)/RAM_total∗Power(DIMM)
+ 2∗(Time_checksum+copy)∗ Power(CPU) ∗ ReferenceRate_mcache/sec
+ Pagefault_PowerCost*Page_error_Rate

Result 1: Mercurial caches provide energy savings proportional to the DRAM under active usage (in the working set) ranging from 1-19%



Result 2: Mercurial caches can sustain a reference rate of around 1.2 million words/ second before the cost of checksum/copy dominates over power savings



Result 3: a) Our solution is better than completely turning off memory, which reduces the amount of available memory, reducing performance. b) It is also better than swapping out to a low power SSD, where the reference rate that can be sustained is only 10K pages/ sec, which too reduces performance

References

[1] L.A. Barroso and U. Hö̈lzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 2009.

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