FlashCache: A NAND Flash Memory File Cache for Low Power Web Servers

Taeho Kgil
Advanced Computer Architecture Laboratory
The University of Michigan
Ann Arbor, USA
tkgil@eecs.umich.edu

Trevor Mudge
Advanced Computer Architecture Laboratory
The University of Michigan
Ann Arbor, USA
tnm@eecs.umich.edu

ABSTRACT

We propose an architecture that uses NAND flash memory to reduce main memory power in web server platforms. Our architecture uses a two level file buffer cache composed of a relatively small DRAM, which includes a primary file buffer cache, and a flash memory secondary file buffer cache. Compared to a conventional DRAM-only architecture, our architecture consumes orders of magnitude less idle power while remaining cost effective. This is a result of using flash memory, which consumes orders of magnitude less idle power than DRAM and is twice as dense. The client request behavior in web servers, allows us to show that the primary drawbacks of flash memory—endurance and long write latencies—can easily be overcome. In fact the wear-level aware management techniques that we propose are not heavily used.

Categories and Subject Descriptors

B.3 [**Semiconductor Memories**]; C.0 [**System architectures**]

General Terms

Design, Experimentation, Performance

Keywords

Low power, web server, application-specific architectures, Flash memory, server platforms, embedded system, full-system simulation

1. INTRODUCTION

With the growing importance of web servers found in internet service providers like Google and AOL, there is a trend towards using simple low power systems as blade servers in power hungry server farms. These suit the modest computation power and high memory throughput required in typical server workloads. As shown in Figure 1, web servers connect directly to the client and are only in charge of delivering the web content page to the client. Since web servers require just a modest amount of computation power, a large amount of their performance depends heavily on memory, I/O bandwidth and access latency. To mitigate I/O latency and bandwidth, especially latency in hard disk drives, server farms typically use large main memories that try to map the entire fileset onto memory, caching the whole fileset onto DRAM. Unfortunately, DRAM consumes a large portion of overall system power. Today's typical servers come with large quantities of main memory—4∼32GB and have been reported to consume as much as 45W in DRAM idle power[8]. If we consider that the Niagara core inside the Sun T2000, a typical server, consumes about 63W, we can clearly see that DRAM idle power contributes to a large portion of overall system power.

We examined the behavior of DRAM main memory in Figure 2. In particular, Figure 2(a) shows the hit rate of a file buffer cache for web server applications. We see marginal improvement in page cache hit rate after a certain point. However, because the access latency to a hard disk drive is in milliseconds, a large amount of DRAM size is still needed to make up for the costly hard disk drive access latency. Otherwise, the server will not be fully utilized and remain idle waiting to get information from the hard disk drive. Surprisingly though, from our experiments done on web server workloads shown in Figure 3, we observe an access latency of tens to hundreds of microseconds can be tolerated when accessing a large part of a file buffer cache without affecting throughput. This is due to the multi-threaded nature of web server applications that allows modest access latency of microseconds to be hidden. The resulting non-uniform memory hierarchy consumes less power while performing equally well as a flat DRAM-only main memory. Furthermore, reads are more frequent than writes in web server applications. These characteristics make a strong case for using flash memories as a secondary file buffer cache. Flash memories have established themselves as storage devices for embedded and mobile platforms. Their continued rapid improvement is supported by their growing usage in a wide variety of high volume commercial products. Flash memories consume orders of magnitude less idle power and are cheaper than DRAM, especially NAND-based flash memory, making them an excellent component used for energy-efficient computing.

In this paper, we propose an architecture called FlashCache that uses a NAND-based flash memory as a secondary

CASES'06, October 23–25, 2006, Seoul, Korea.

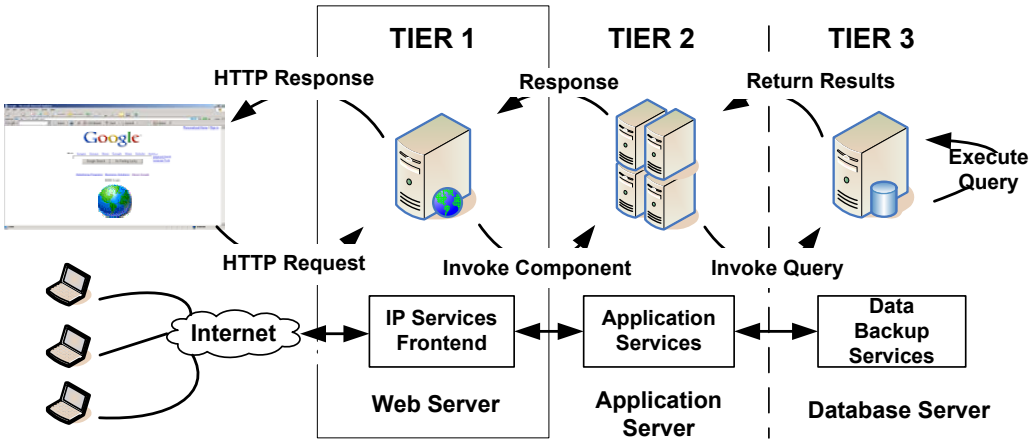


Figure 1: A Typical 3 Tier Server Architecture. Tier 1—Web Server, Tier 2—Application Server, Tier 3—DataBase Server. PicoServer is targeted at Tier 1. An example of an internet transaction is shown. When a client request comes in for a Java Servlet Page, it is first received by the front end server—Tier 1. Tier 1 recognizes a Java Servlet Page that must be handled and initiates a request to Tier 2 typically using Remote Message Interfaces (RMI). Tier 2 initiates a database query on the Tier 3 servers, which in turn generate the results and send the relevant information up the chain all the way to Tier 1. Finally, Tier 1 sends the generated content to the client.

file buffer cache to reduce overall power consumption in the main memory without impacting network bandwidth. Although flash memory has limitations in terms of endurance and bandwidth, we will make the case in this paper that the benefits found in the reduction of overall main memory power outweigh the drawbacks in adopting flash memory.

Aside from the cost-effectiveness and low idle power of flash memory, our FlashCache architecture can also take advantage of flash memory's persistence. Integrating flash memory into the conventional system-level memory hierarchy enables quick warm up of file buffer caches, which allows quicker deployment of web servers. We can quickly image the cached file content onto the flash memory and reduce the warm-up time of the file buffer cache.

Our FlashCache architecture provides:

- **Overall reduction in system level power.** The physical characteristics of flash memory reduces by a significant amount the idle power in main memory. The overhead resulting from the increased latency in accessing items in the file buffer cache can be minimized because of the access behavior of the files.
- **Overall cost reduction in main memory.** A cost-effective solution with multi-chip main memory compared to a DRAM-only based solution. The density of flash memory exceeds DRAM by more than 2×, which is reflected in the reduced cost per bit of flash memory. Therefore, the total cost of a main memory is much less costly than a DRAM-only solution.
- **Quicker startup time compared to conventional DRAM-only platforms.** The nonvolatile property in flash memory means file buffer cache warm up is not required after boot up.

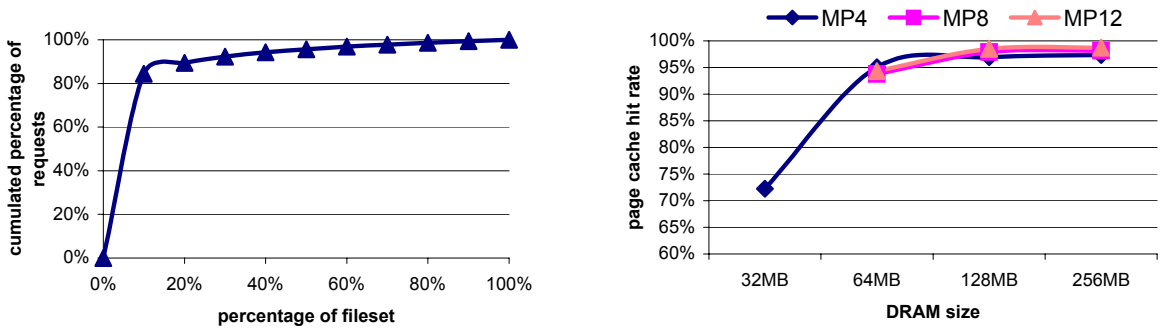
In the process of describing our architecture, we interchangeably use *page cache* and *file buffer cache*. This is because after Linux kernel version 2.4 the file buffer cache has been merged into a page cache.

The paper is organized as follows. In section 2, we provide background on flash memory, web server workloads, and related work. Section 3 and 4 provide a detailed description of our FlashCache architecture and explains how it works. Section 5 and 6 present our experimental setup and results. And finally we present our conclusions and future work in Section 7.

2. BACKGROUND

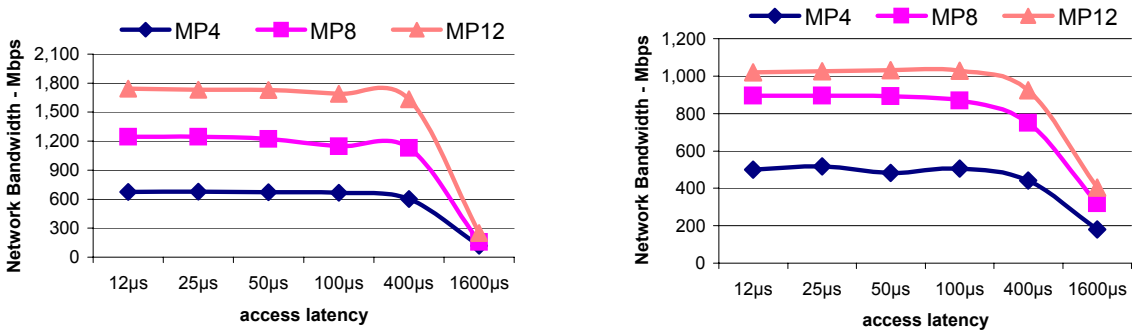
2.1 Flash memory

There are two types of flash memory—NAND and NOR—that are commonly used today. Each type of memory has been developed for a particular purpose and has its own pros and cons. Table 1 and Table 2 summarizes the properties of NAND and NOR along with DRAM. The biggest difference, compared to DRAM, is when writing to a flash memory. Flash memories require a preceding erase operation to perform a write operation. The ITRS roadmap projects NAND flash memory cell sizes to be 2∼4× smaller than DRAM cell sizes and NOR flash memory cell sizes to be similar or slightly bigger than DRAM cell sizes. A NAND flash memory uses a NAND structure memory array to store content. The small cell size for a NAND flash memory results in higher storage density. With the potential to support multi-level cells, storage density is expected to improve even more as shown in the ITRS roadmap. However, the drawback is that the random read access time for NAND flash memory is lengthy due to the nature of the NAND structure. Sequen-



(a) (b)

Figure 2: (a) File buffer cache access behavior on the server side for client requests. We measured for 4, 8, 12 multicore configurations and varied the DRAM size. (b) A typical cumulative distribution function of a client request behavior. 90% of requests are for 20% of the web content files.



(a) SURGE (b) SPECWeb99

Figure 3: Measured network bandwidth for full system simulation while varying access latency to a secondary page cache. We assumed a 128MB DRAM with a slower memory of 1GB. We measured bandwidth for 4, 8, 12 multicore configurations. The secondary page cache can tolerate access latencies of hundreds of microseconds while providing equal network bandwidth.

tial read accesses are not as slow as random reads, making it a good candidate for applications that have streaming locality. As a consequence, NAND flash memory has become popular for storing digital content—audio and video files. In addition to lengthy random access time, NAND flash memory also has issues with reliability.

NAND flash memory is likely to be manufactured with faulty cells. Furthermore, wear-out may cause good cells to become bad cells. NAND flash memory comes with built-in error correction code support to maintain reliability and extend the lifespan of flash memory. There have been implementations of file systems that extend the endurance of NAND flash memory to more than 5,000,000 cycles using BCH (Bose,Chaudhuri and Hocquenghem) block codes[9]. A NOR flash memory, in contrast to NAND flash, has a random read access time that is faster. Since NOR flash memory performs relatively well for random read accesses, it is commonly used for applications requiring fast access to code and data. Accordingly, NOR flash memory has typically been used to store code and data on handheld devices, PDAs, laptops, cell phones, etc.

2.2 Web Server workload

Web Server workloads are known to have high levels of thread level parallelism (TLP), because connection level parallelism can be translated into thread level parallelism. In addition to that, they cover a working set size determined by the client web page request behavior. Client web page requests follow a Zipf-like distribution as shown in Figure 2(b). A large portion of requests are centered around the same group of files. These file accesses translate into memory and I/O accesses. Due to the modest computation, memory and I/O latency are critical to high performance. Therefore, file caching in the main memory plays a critical part in providing sufficient throughput. Without a page cache, the performance degradation due to the hard disk drive latency would be unacceptable. To work well on this particular workload, the preferred architecture is a simple multicore architecture with enough page cache to reduce the amount of access to the hard disk drive[18].

	Density– Gb/cm^2	$/Gb	Active Power*	Idle Power*	Read Latency	Write Latency	Erase Latency	Built-in ECC support
DDR2 DRAM	0.7	48	878mW	80mW†	55ns	55ns	N/A	No
NOR	0.57	96	86mW	16μW	200ns	200μs	1.2s	No
NAND	1.42	21	27mW	6μW	25μs	200μs	1.5ms	Yes

* Power consumed for 1Gbit of memory
† DRAM Idle power in active mode. Idle power in powerdown mode is 18mW

Table 1: Cost and power consumption for conventional DRAM, NOR, NAND-based flash memory. NAND flash memory is the most cost-effective while consuming the least amount of power.[22][20]

	2005	2007	2009	2011	2013	2016
Flash NAND Cell size –SLC/MLC*(μm^2)	0.0231/0.0116	0.0130/0.0065	0.0081/0.0041	0.0052/0.0013	0.0031/0.0008	0.0016/0.0004
Flash NOR Cell size(μm^2)†	0.0520	0.0293	0.0204	0.0117	0.0078	0.0040
DRAM Cell size(μm^2)	0.0514	0.0324	0.0153	0.0096	0.0061	0.0030
Flash erase/write cycles	1E+05	1E+05	1E+05	1E+06	1E+06	1E+07
Flash data retention	10-20	10-20	10-20	10-20	20	20

* SLC - Single level Cell, MLC - Multi Level Cell
† We assume a single level cell with smallest area size of 9F^2 stated in the ITRS roadmap

Table 2: ITRS 2005 roadmap for flash memory technology. NAND flash memory is projected to be upto 7∼8× as dense as DRAM. Flash memory endurance improves by an order of magnitude approximately every 5∼6 years. Data retention is over 10 years which is a long time for server platforms.[10]

2.3 Related Work

There has been considerable work on using flash memory as part of the memory hierarchy. In [22], it was shown that flash memory could be used directly for high performance code execution by adding an SRAM. In [2], the authors proposed to integrate flash memory into hard disk drives to improve their access latencies in mobile laptops. The prototype used the flash memory as a boot buffer to speedup boot time and as a write buffer to hide write latencies to the hard disk drive. The issue of wear-level awareness has also been studied. For example, wear-level aware file systems have been outlined in [3] to extend flash memory endurance. [15] has shown error correction codes extend the lifespan of flash memory with marginal overhead.

In the case of non-uniform main memory, [14] showed the potential of having a non-uniform main memory architecture. They examined using a slow secondary main memory as a swap cache and measured the performance overhead. The overhead was shown to be negligible. [17][19] showed that a considerable amount of main memory power can be reduced with power aware page allocation. They reduced DRAM idle power by putting non-recently accessed pages to sleep. Our work extends the work from [14][2] and integrates flash memory as a secondary file buffer cache for web server applications. We will show that this architecture reduces operating costs due to an order of magnitude reduction in main memory power consumption, while giving a significant cost advantage compared to a DRAM-only architecture. By adapting and simplifying the methods in [3] and incorporating them onto a cache replacement algorithm, we mitigate any wear-out problems.

3. FLASHCACHE ARCHITECTURE

Figure 4 shows an overview of our proposed architecture using a FlashCache. Compared to a conventional DRAM-only architecture, our proposed FlashCache assumes a two level page cache. It requires a non-uniform main memory architecture with a small DRAM that holds the primary page cache and flash memory that functions as the secondary page cache. A flash memory controller is also required. Additional data structures required to manage the FlashCache are placed in DRAM.

Our FlashCache architecture uses a NAND-based flash memory rather than a NOR-based flash memory due to the compactness and faster write latency found in NAND flash memory. The random read access latency for a commercial NAND is 25μs and the write latency is 200μs with an erase latency of 1.5ms per block [6]. We employ flash memory as a page cache rather than a generic main memory or a storage device. Flash is unsuitable as a generic main memory, because of the frequent writes and associated wear-out. It is unsuitable as a storage device, because of the large memory overhead required in a file system where wear-out is a concern. A page cache only requires cache tags for each block, implying less memory overhead than a file system requiring a tree structure filled with file location nodes. Data structures used in FlashCache management are read from the hard disk drive or the flash memory and loaded to DRAM to reduce access latency and mitigate wear-out.

In conventional DRAM-only architectures that use a single level DRAM for file caching, a fully associative page cache table is managed in software and probed to check if a certain file is in DRAM. The search time is speed up by using tree structures. A considerable amount of search time can still elapse, but is sustainable because DRAM access latency for transferring file content is in nanoseconds. This is not the case for a flash memory-based page cache, where the read access latency for transferring file content is 10∼100 mi-

croseconds. Although, we found search time to be 300∼400 nanoseconds, for conservative reasons, we employ a hash table to reduce search time.

3.1 FlashCache Hash Table for tag lookup

The FlashCache Hash Table (FCHT) is a memory structure that holds tags associated with the flash memory blocks. This table exists in DRAM. A tag is made up of a page address field and a flash memory address field. The page address field points to the location in the hard disk drive and is used for determining whether the flash memory holds this particular location in the hard disk drive. The corresponding flash memory address field is used to access flash memory. If a hit occurs in the FCHT, the corresponding flash memory address field is used to access flash memory. More than 99% of the time a hit occurs for our FlashCache system and the flash memory location containing the requested file is sent to the primary page cache existing in DRAM. If a miss occurs, the FlashCache management scheme determines which block to evict based on a wear-level aware replacement algorithm and schedules that block to be evicted.

The FCHT is partitioned into a set associative table. In this work, we assume 16 way set associativity. Our studies suggested that there was a marginal improvement in the hit rate for higher associativity. Wear-level awareness is managed both at the block level—fine grain—and at the set level—coarse grain. Each set is made up of 16 *logical blocks* that are in turn made up of 4 of the write/erase blocks in the flash memory. A wear-level status table is required to profile the number of writes and erases performed on a particular block. In the following subsections, we describe each component in an architecture using the FlashCache.

3.2 Wear-level aware cache replacement

The drawback of using flash memory is wear-out. Compared to DRAM, flash memory can only be written a fixed amount of times. The endurance for flash memory is expected to improve in the future [10]. Table 2 shows the ITRS projection for flash memory endurance. A 10× improvement in endurance is expected every 5∼6 years. Endurance improvement is attributed to the use of better material. Although endurance is expected to improve to the point where it is no longer a concern, we have chosen to be conservative and make our FlashCache replacement algorithm wear-level aware. The wear-level status table exists to assist in wear-level management.

3.2.1 Wear-level status table

The wear-level status table located in DRAM maintains the number of erases and writes performed on a logical block. The number of erases and writes is equal to the number of evictions. A wear level status counter exists for each logical block in the flash memory. Every time a logical block is selected for eviction its wear-level status table entry located in DRAM is incremented. The wear level status table is used for both coarse grain and fine grain level wear-level management. The wear-level status table determines a *hot set* or a *hot block*. A set or block is considered hot if its status table entry exceeds a certain threshold.

3.2.2 Management at the block level and set level

Wear-level management for the FlashCache is performed on FlashCache misses. At the block level, we initially select a logical block to be evicted using an LRU policy. However, if this block is identified to be a *hot block* by observing the difference between the eviction count of the logical block selected for eviction chosen from the LRU policy and the minimum eviction count from the other logical blocks belonging to the same set exceeds a certain threshold, then the logical block corresponding to the minimum eviction count is evicted to balance the wear level.

At the set level, we determine if the currently accessed set is a *hot set* by comparing it to the maximum eviction count for the logical blocks belonging to the currently accessed set and whether this number exceeds the eviction count of other sets by a certain threshold. A *hot set* is swapped with a *cold set* to balance the wear-level. The temporary staging area for swaps is located in flash memory where a temporary buffer is used in the swap process.

3.3 Flash memory controller with DMA support

The flash memory controller handles the unique interface required in accessing flash memory. The flash memory controller supports DMA to handle DMA transfers from DRAM to flash memory and vice versa. The procedure required in transferring flash memory data is simple in principle—similar to accessing DRAM. However, there are two potential problems in performing reads and writes in flash memory. The first potential problem is bandwidth. Usually, a flash memory can read and write only a single byte or word per cycle. In addition, today's NAND flash memories have a slow transfer clock— 50MHz. Therefore, a large amount of time is spent in reading and writing data. This becomes a problem when we access the flash memory frequently. The limited bandwidth in flash memory potentially becomes a bottleneck in providing high performance. The other potential problem is blocking writes. Today's NAND flash memory suffer from blocking writes and do not support Read While Write (RWW). A NAND flash memory is busy during writes, blocking all other accesses that can occur. Without RWW, a blocking flash memory write could also become a potential bottleneck in providing high performance.

Fortunately, these problems can currently be dealt with. From our studies shown in Figure 3, we know that we can tolerate an access latency of hundreds of microseconds. This relieves the limited bandwidth problem. The blocking property of flash memory writes can be solved by distributing writes. Because flash memory writes do not occur frequently, we can schedule flash memory reads to have priority over writes, by using a lazy writeback scheme. By managing a writeback buffer in DRAM that stores blocks that need to be updated in the flash memory, we can prevent a flash memory write from occurring when flash memory reads are requested. The lazy writeback scheme allows writebacks to occur mostly when the flash memory is not accessed for reads.

In the long term we expect to see more direct solutions—improve bandwidth and non-blocking features. Adopting emerging technologies like 3D stacking technology[16] to improve flash memory bandwidth and implementing multi-banked flash memory that supports RWW are possible solutions.

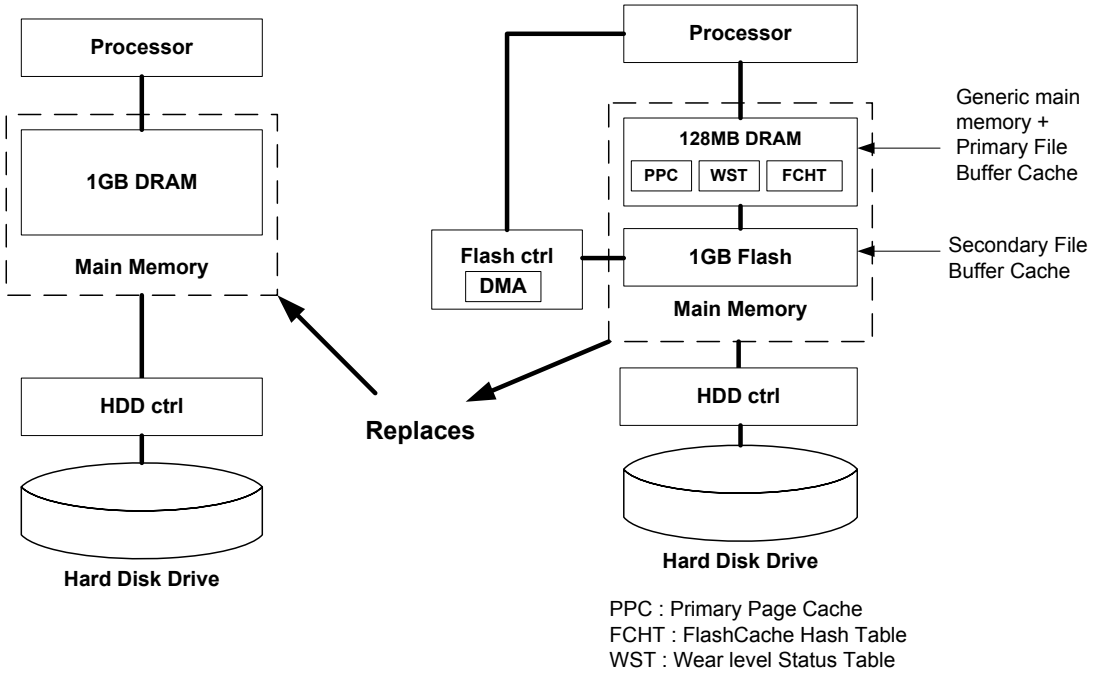


Figure 4: General overview of our FlashCache architecture. We show an example of a 1GB DRAM replaced with a smaller 128MB DRAM and 1GB NAND-based flash memory. Additional components are added to control the flash memory. The total die area required in our multichip memory is 60% the size of a conventional DRAM-only architecture.

4. HOW IT WORKS

In this section we discuss how FlashCache hits and misses are handled.

When a file I/O is performed at the application level, the OS searches for the file in the primary page cache located in DRAM. On a primary page cache hit in DRAM, the FlashCache is not accessed at all, and the file content is accessed directly from the primary page cache. On a primary page cache miss in DRAM, the OS searches the FCHT to determine whether the requested file currently exists in the secondary page cache. If the requested file is found, then a flash memory read is performed and a DMA transaction is scheduled to transfer flash memory content to DRAM. The requested address to flash memory is obtained from the FlashCache Hash Table.

If a miss occurs in the FlashCache Hash Table search process, a logical block is selected for eviction. The selection process first considers wear-level at the fine grain block level. If the selected logical block has been evicted more times than a certain threshold—a *hot block*, we evict the least evicted logical block belonging to the same set instead of applying an LRU policy (noted above). Furthermore, if the set which the selected logical block belongs to is evicted frequently—a *hot set*, a set swap is performed, where a *hot set* is swapped with a *cold set*. Fortunately this does not occur frequently as a result of the behavior of the workload. After a logical block is selected for eviction and whether or not we decide to do a set swap, an erase operation is performed on the flash memory for that logical block. Concurrently, a hard disk drive access is scheduled using the device driver interface. The hard disk drive content is copied to the primary page cache in DRAM. The corresponding tag in the FCHT is also updated. Finally, we schedule a writeback of the hard disk drive content to flash memory through our lazy writeback scheme using DMA.

5. EXPERIMENTAL SETUP

The architectural aspects of our studies are obtained from a microarchitectural simulator called M5 [12] that is able to run Linux and evaluate full system level performance. A web server connected to multiple clients is modeled. The client requests are generated from user level network application programs. To measure and model behavior found in DRAM and flash memory, we extracted information from [6][5] to add timing and power models to M5. A more detailed description of our methodology is described in the following subsections.

5.1 Simulation Studies

5.1.1 Full system architectural simulator

M5 is a configurable architecture that runs an SMP version of Linux 2.6. The clients and server are all modeled with distinct communicating instances of M5 connected through a point-to-point ethernet link. The server side executes Apache—a web server. The client side executes benchmarks

	Server configuration parameters
Processor type	single issue in-order
Number of cores	4, 8, 12 core
Clock frequency	1GHz
L1 cache size	4 way 16KB
L2 cache size	8 way 2MB
DRAM	64MB∼1GB t_{RC} latency 50ns bandwidth 6.4GB/s
NAND Flash Memory	1GB 16 way 128KB logical block size random read latency 25μs write latency 200μs erase latency 1.5ms bandwidth 50MB/s
IDE disk	average access latency 3.3ms bandwidth 300MB/s
Ethernet Device (NIC)	1Gbps Ethernet NIC
Number of NICs	2, 4, 6

Table 3: Server configurations in our studies.

that generate representative requests for dynamic and static page content. For comparison, a chip multiprocessor-like system is created from [18]. Configurations of 4, 8, 12 processors are used in this study. We assumed a simple in-order 5 stage pipeline core which is similar to, but simpler than the Niagara configuration. A detailed breakdown of our server architecture is shown in Table 3. The clients are modeled just functionally. We use total network bandwidth in the ethernet devices as our performance metric. It is the sum of transmitted and received bandwidth.

5.1.2 Server Benchmarks

We use two web server benchmarks that directly interact with client requests, SURGE[11] and Specweb99[7] to measure client web page requests. Both benchmarks request filesets of more than a 1GB. The fileset size for SURGE is 1.5GB and for Specweb99 is 1.8GB.

SURGE The SURGE benchmark represents client requests for static web content[11]. SURGE is a multi-threaded, multi-process workload. We modified the SURGE fileset and used a Zipf distribution to generate reasonable client requests. Based on the Zipf distribution a static web page which is approximately 16KB in file size is requested 50% of the time in our client requests. We configured the SURGE client to have 24 outstanding client requests. It has been shown in [13] that the number of requests handled per second saturates after 10 concurrent connections implying 24 concurrent connections is more than enough to fully utilize the server.

SPECWeb99 To evaluate a mixture of static web content and simple dynamic web content, we used a modified version of SURGE to request SPECWeb99 filesets. We used the default configuration for Specweb99 to generate client requests. 70% of client requests are for static web content and 30% are for dynamic web contents. We also fixed the number of connections of Specweb99 to be 24 as in the SURGE case.

5.2 Modeling DRAM and Flash memory

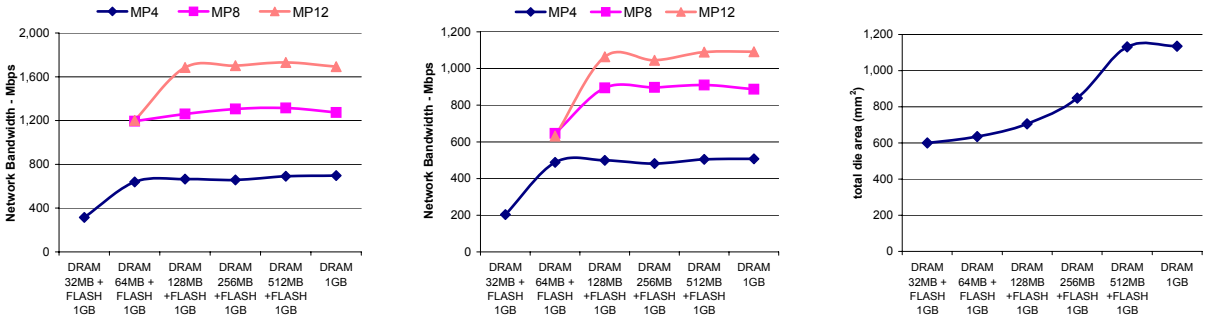
Timing, power, and die area estimation at the architectural level is difficult to estimate with great accuracy. To make a reasonable estimation and show general trends, we relied on industry and academia publications on die size area, power and performance. We used published datasheets found in [6][5] to estimate timing and power consumption. For die area estimation, we used published data found in [20][21]. We discuss this further in the next subsections.

5.2.1 DRAM

The timing model for DRAM is generated from the Micron datasheets in [4]. Our timing model also considers the DRAM command interfaces including address multiplexing, DRAM precharge, etc. This timing model is integrated onto the M5 simulator. The Micron DRAM spreadsheet calculator generates DRAM power based on inputs of reads, writes, and page hit rates [5]. From the platform simulator, we profile the number of cycles spent on DRAM reads and writes, and page hit rates to obtain average power. Our power estimates correlate well with numbers from [8]. For die area estimation, we used numbers generated from industry products found in [21].

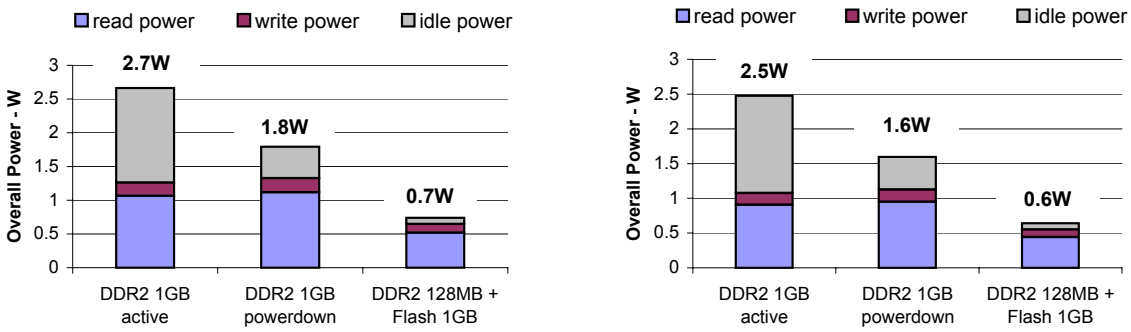
5.2.2 Flash memory

To understand the timing and power model for NAND flash memory, we used several of the publications found in [6]. We assumed a single bit cell in this study and expect density and bandwidth to continue to improve due to the high demand of flash memory in many commercial sectors. Our die area estimates that are used to compare with DRAM are from [20][21] and we performed comparisons on similar process technologies. Although flash memory has begun to outpace DRAM in process technology, we made conservative estimates. To estimate the power consumption of our Flash-Cache architecture, we used measurements from datasheets. Published numbers in datasheets represent maximum power consumption. Therefore, our estimates are expected to be conservative compared to real implementations. The idle power of flash memory is typically 3 orders of magnitude less than that of DRAM.



(a) SURGE (b) SPECWeb99 (c) total die area

Figure 5: (a),(b) show a network bandwidth comparison for various main memory configurations using the FlashCache. The rightmost points are for a DRAM-only system. (c) depicts the total die area.



(a) SURGE (b) SPECWeb99

Figure 6: Overall memory power consumption breakdown. Our FlashCache architecture reduces idle power by several orders of magnitude. For powerdown mode in DRAM, we assume an oracle powerdown algorithm.

6. RESULTS

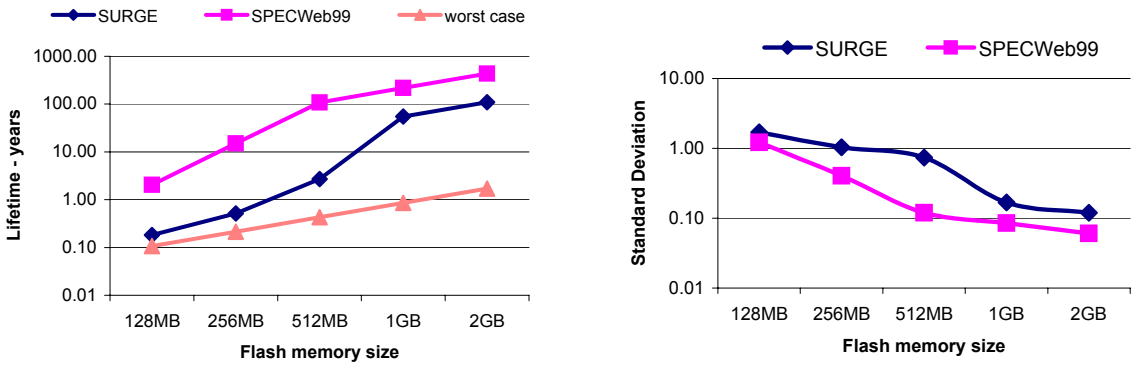
The feasibility of this architecture can be measured by performance, power reduction, and wear-out. In the following subsections, we present performance in terms of network bandwidth, an important metric in server platforms. The power reduction results mostly from the reduction in idle power. We also show that our FlashCache architecture does not require frequent wear-level rebalancing.

6.1 Network Performance

Figure 5 depicts the overall network performance as DRAM size is varied and flash memory is fixed at 1GB. Our baseline comparison is a configuration with no flash and 1GB of DRAM. Our optimal multichip configuration requires less die area than our baseline configuration. From our early observations that large portions of the page cache can tolerate access latencies of tens to hundreds of microseconds, we find our optimal configuration to be 128MB of DRAM and 1GB of flash memory. In terms of area, this configuration requires 40% less die than our baseline of no flash and 1GB of DRAM as shown in Figure 5 (c).

6.2 Overall main memory power

With respect to overall power consumed in main memory, our primary power savings come from the reduction in idle power from using flash memory. It is several orders of magnitude. We compare our multichip configuration with DRAM configurations that have a) no power management—*active* mode and b) ideal power management—*powerdown* mode. Intelligent power management reduces DRAM idle power. Our ideal power management scheme assumes the DRAM is set to a *powerdown* mode whenever possible. These modes were derived from [17][19]. Figure 6 shows our results. As shown in Figure 6, the FlashCache architecture reduces overall main memory power by more than a factor of 2.5, even compared to an oracle power management policy implemented in software for DRAM. The memory power for an architecture using a FlashCache includes the flash memory controller power consumption along with the extra DRAM accesses to manage the flash memory. Since flash memory is accessed only thousands of times per second, the overall average contribution from additional DRAM accesses and the flash memory controller is negligible.



(a) (b)

Figure 7: Flash memory endurance and spatial variation for a 8 core 128MB DRAM configuration while varying the flash memory size in the FlashCache architecture (a) temporal endurance in years, assuming flash memory endurance of 1,000,000 cycles (b) standard deviation for the number of evicted flash memory blocks spatial variation

6.3 Wear level aware behavior

Figure 7 shows the predicted lifetime for varying flash memory sizes assuming the lifecycle of a flash memory to be a million cycles. These simulations assumed a 128MB DRAM with 8 multicores and FlashCache sizes from 128MB to 2GB. From our simulation results, we found our FlashCache is accessed less than 2000 times a second. This implies a flash memory size of 1GB has a lifetime close to 100 years when assuming a 1,000,000 cell lifecycle. Worst case analysis assuming the hard disk drive is accessed all the time yielded a lifetime of 2 years for a 2GB flash memory. With ECC support for multiple error correction, worst case lifetime can be extended even more. The ECC overhead in latency has been found to be in nanoseconds[15][1]. Applying a 100,000 cell of lifecycle, which is the current endurance figure, we expect 1GB flash memory to have an lifetime of 10 years. We also found that the spatial variation is small implying our architecture naturally levels out wear. This is largely due to the set associativity of the cache. In our simulations, wear-level management routines were seldom invoked—only twice for the whole duration of our simulation.

7. CONCLUSIONS AND FUTURE WORK

Our FlashCache architecture reduces idle power by several orders of magnitude while maintaining cost effectiveness over conventional DRAM-only architectures both in terms of operating cost and energy efficiency. From our observations, a typical web server architecture can sustain a file access latency in the tens to hundreds of microseconds without noticeable loss in throughput. We also observe that the organization of a cache, especially set associativity, inherently displays wear-level aware properties. This strengthens the case for using a flash based file buffer cache instead of a conventional DRAM. Our simulations show more than a 2.5× reduction in overall main memory power with negligible network performance degradation. Our future work will investigate the bandwidth requirements and endurance requirements for adopting flash memory for other types workloads found in other application domains.

8. ACKNOWLEDGEMENTS

This project is supported by the National Science Foundation under grants NSF-ITR CCR-0325898. This work was also supported by gifts from Intel.

9. REFERENCES

[1] Error Correction Code in Single Level Cell NAND Flash Memories. `http://www.st.com/stonline/products/literature/an/10123.pdf`.

[2] Hybrid Hard Drives with Non-Volatile Flash and Longhorn. `http://www.samsung.com/Products/HardDiskDrive/news/HardDiskDrive_20050425_0000117556.htm`.

[3] JFFS: The Journalling Flash File System. `http://sources.redhat.com/jffs2/jffs2.pdf`.

[4] Micron DDR2 DRAM. `http://www.micron.com/products/dram/ddr2/`.

[5] The Micron system-power calculator. `http://www.micron.com/products/dram/syscalc.html`.

[6] Samsung NAND Flash memory datasheet. `http://www.samsung.com/products/semiconductor/NANDFlash/SLC_LargeBlock/8Gbit/K9K8G08U0A/K9K8G08U0A.htm`.

[7] SPECweb99 benchmark. `http://www.spec.org/osg/web99/`.

[8] Sun Fire T2000 Server Power Calculator. `http://www.sun.com/servers/coolthreads/t2000/calc/index.jsp`.

[9] TrueFFS. `http://www.m-systems.com/site/en-US/Support/DeveloperZone/Software/LifespanCalc.htm`.

[10] ITRS roadmap. Technical report, 2005.

[11] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Measurement and Modeling of Computer Systems*, pages 151–160, 1998.

[12] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 simulator:

Modeling networked systems. *IEEE Micro*, 26(4):52–60, Jul/Aug 2006.

[13] E. L. Congduc. Packet classification in the NIC for improved SMP-based internet servers. In *Proc. Int'l Conf. on Networking*, Feb. 2004.

[14] M. Ekman and P. Stenstr. A cost-effective main memory organization for future servers. In *Proc. of the Int'l Parallel and Distributed Processing Symp.*, Apr 2005.

[15] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli. On-chip error correcting techniques for new-generation flash memories. 91(4), Apr 2003.

[16] S. Gupta, M. Hilbert, S. Hong, and R. Patti. Techniques for producing 3D ICs with high-density interconnect. `www.tezzaron.com/about/papers/ieee_vmic_2004_finalsecure.pdf`.

[17] H. Huang, P. Pillai, and K. G. Shin. Design and Implementation of Power-Aware Virtual Memory. In *USENIX Annual Technical Conference*, pages 57–70, 2003.

[18] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded Sparc processor. *IEEE Micro*, 25(2):21–29, Mar. 2005.

[19] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proc. Int'l Conf. on Arch. Support for Programming Languages and Operating Systems*, pages 105–116, 2000.

[20] J. Lee, S.-S. Lee, O.-S. Kwon, K.-H. Lee, D.-S. Byeon, I.-Y. Kim, K.-H. Lee, Y.-H. Lim, B.-S. Choi, J.-S. Lee, W.-C. Shin, J.-H. Choi, and K.-D. Suh. A 90-nm CMOS 1.8-V 2-Gb NAND Flash Memory for Mass Storage Applications. 38(11), Nov 2003.

[21] G. MacGillivray. Process vs. density in DRAMs. `http://www.eetasia.com/ARTICLES/2005SEP/B/2005SEP01_STOR_TA.pdf`.

[22] C. Park, J. Seo, S. Bae, H. Kim, S. Kim, and B. Kim. A Low-cost Memory Architecture With NAND XIP for Mobile Embedded Systems. In *Proc. Int'l Conf. on HW-SW Codesign and System Synthesis(CODES+ISSS)*, Oct 2003.