Integrated 3D-Stacked Server Designs for Increasing Physical Density of Key-Value Stores

Anthony Gutierrez Michael Cieslak Bharan Giridhar
Ronald G. Dreslinski Luis Ceze[†] Trevor Mudge

Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{atgutier, mrciesla, bharan, rdreslin, tnm}@umich.edu

[†]Computer Science and Engineering Department
University of Washington - Seattle, WA
luisceze@cs.washington.edu

Abstract

Key-value stores, such as Memcached, have been used to scale web services since the beginning of the Web 2.0 era. Data center real estate is expensive, and several industry experts we have spoken to have suggested that a significant portion of their data center space is devoted to key-value stores. Despite its wide-spread use, there is little in the way of hardware specialization for increasing the efficiency and density of Memcached; it is currently deployed on commodity servers that contain high-end CPUs designed to extract as much instruction-level parallelism as possible. Out-of-order CPUs, however have been shown to be inefficient when running Memcached.

To address Memcached efficiency issues, we propose two architectures using 3D stacking to increase data storage efficiency. Our first 3D architecture, Mercury, consists of stacks of ARM Cortex-A7 cores with 4GB of DRAM, as well as NICs. Our second architecture, Iridium, replaces DRAM with NAND Flash to improve density. We explore, through simulation, the potential efficiency benefits of running Memcached on servers that use 3D-stacking to closely integrate low-power CPUs with NICs and memory. With Mercury we demonstrate that density may be improved by 2.9×, power efficiency by 4.9×, throughput by 10×, and throughput per GB by 3.5× over a state-of-the-art server running optimized Memcached. With Iridium we show that density may be increased by 14×, power efficiency by 2.4×, and throughput by 5.2×, while still meeting latency requirements for a majority of requests.

ASPLOS '14, March 1–4, 2014, Salt Lake City, Utah, USA.

http://dx.doi.org/10.1145/2541940.2541951

Categories and Subject Descriptors C.5.5 [*Computer System Implementation*]: Servers

General Terms Design, Performance

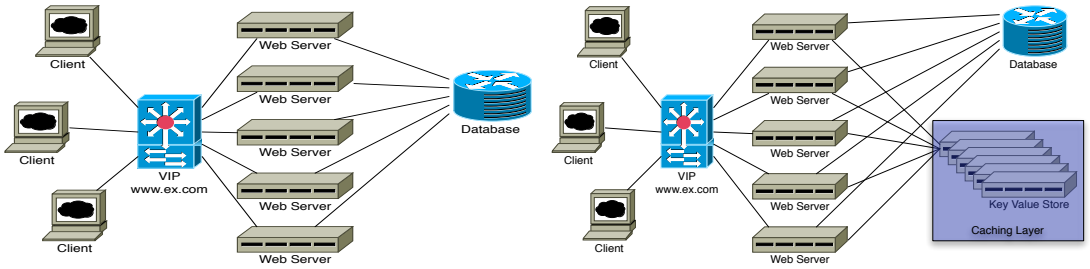
Keywords 3D Integration, Data Centers, Key-Value Stores, Physical Density, Scale-Out Systems

1. Introduction

Since the emergence of the Web 2.0 era, scaling web services to meet the requirements of dynamic content generation has been a challenge—engineers quickly discovered that creating content for each visitor to a web site generated a high load on the back-end databases. While it is easy to scale the number of servers generating HTML and responding to client requests, it is harder to scale the data stores. This two-tier infrastructure becomes increasingly difficult to scale and requires many redundant systems to prevent a single point of failure. In order to meet the performance and reliability requirements of handling such a massive volume of data, highly distributed scale-out architectures are required.

Memcached is one example of a distributed, in-memory key-value store caching system. Memcached is used as the primary piece of scaling infrastructure for many of today's most widely-used web services, such as Facebook, Twitter, and YouTube; although not prepared to go on the record, several industry experts we've spoken with have estimated that approximately 25% of their data center is dedicated to key-value stores. Due to its wide-spread use, and the high cost of data center real estate, it is important that Memcached be run as efficiently as possible. Today, however, Memcached is deployed on commodity hardware consisting of aggressive out-of-order cores, whose performance and efficiency are measured with respect to how well they are able to run CPU benchmarks, such as SPEC [17].

Realizing that aggressive out-of-order cores are not an efficient choice for many classes of server applications, several studies have advocated the use of low-power embedded CPUs in data centers: [2, 23, 28, 38]. There are challenges with this approach as well—embedded cores are un-



(a) **The database handles all reads and writes.**

(b) **The database only handles writes and reads that miss in the cache.**

Figure 1: **Configurations with 2 and 3 layers behind a vip to service web requests.**

able to provide the throughput and latency guarantees that are required to supply responsive dynamic content. More recently, Lim et al. [29] have shown that mobile cores alone are not enough to improve the efficiency of distributed key-value store caching systems. Instead, they create a custom Memcached accelerator, and rely on recent technology scaling trends to closely integrate their accelerator into a system-on-chip (SoC) design they call *Thin Servers with Smart Pipes* (TSSP). These approaches may improve overall energy and performance, however they do not address density. Dense servers, on the other hand, have been shown to provide greater efficiency for CPU-bound electronic commerce workloads in the past [10], however for workloads that required large amounts of memory, traditional servers were found to be a better solution.

We take the notion of an efficient, integrated system one step further to include density as a primary design constraint. We propose two integrated 3D-stacked architectures called Mercury and Iridium[1]. With Mercury we are able to tightly couple low-power ARM Cortex-A7 cores with NICs and DRAM, while maintaining high bandwidth and low latency. Recently, Facebook introduced McDipper [13], a Flash-based Memcached server using the observation that some Memcached servers require higher density with similar latency targets, but are accessed at much lower rates. To address these types of Memcached servers we introduce Iridium, a Flash based version of Mercury that further increases density at the expense of throughput while still meeting latency requirements. These two architectures allow density to be significantly increased, resulting in more effective use of costly data center space.

In summary we make the following contributions:

- Given data center costs, we contend that server density should be considered a first-class design constraint.
- We propose Mercury, an integrated, 3D-stacked DRAM server architecture which has the potential to improve density by 2.9×, making more efficient use of costly data center space.
- By closely coupling DRAM and NICs with low-power Cortex-A7 cores, we show that it is possible to improve power efficiency by 4.9×.
- By increasing density, and closely coupling DRAM, NICs, and CPUs, it is possible to increase *transactions per second* (TPS) by 10× and TPS/GB by 3.5×.
- Finally, we propose Iridium for McDipper [13] style Memcached servers, which replaces the DRAM with NAND Flash to improve density by 14×, TPS by 5.2×, and power efficiency by 2.4×, while still maintaining latency requirements for a bulk of requests. This comes at the expense of 2.8× less TPS/GB due to the much higher density.

The rest of this paper is organized as follows: in Section 2 we motivate the need for efficient Memcached servers; related work is presented in Section 3; our proposed architectures, Mercury and Iridium, are described in Section 4; in Section 5 we outline our experimental methodology; results are presented in Section 6; finally, we provide concluding remarks in Section 7.

2. Background and Motivation

2.1 Cloud Computing Design

Figure 1a shows the design of a standard web server setup. A load balancer typically has one or more virtual-IPs (VIP) configured. Each domain is routed by a DNS to a particular load balancer, where requests are then forwarded to a free server. After the load balancer a fleet of front-end servers service the web request. If needed, these web servers will contact a back-end data store to retrieve custom content that pertains to the user, which means that all servers could connect to the back-end data store.

In Figure 1b a caching layer is added, which can service any read request that hits in the cache. A request will first be forwarded to the caching layer on any read. If the requested data is present, the caching layer returns its copy to the

[1] Mercury, the Roman god, is extremely fast, while the element of the same name is very dense. Iridium, while not a god, is more dense than mercury.

client. If the data is not present, a query will be sent to the back-end data store. After the data returns, the server handling the request will issue a write to the caching layer along with the data. Future requests for that data can then be serviced by the caching layer. All write requests are sent directly to the back-end data store. Updating values in the caching layer depends on how it is configured. The two most common cases are that writes are duplicated to the caching layer or a time-to-live is placed on data in the caching layer.

2.2 Scaling in the Cloud

In general, the traffic to a website varies by the time of day and time of year. Data published by Netflix [35] demonstrates how traffic to their site varies over a three day period. As their data shows, traffic peaks during midday, and is at its lowest point around midnight. They also quantify the corresponding number of front-end servers needed to maintain equal load throughout the day, which tracks closely with the traffic. While this is easy to do for front-end servers, because they maintain little state, back-end data stores are not easily scaled up and down. Netflix overcomes this problem by the extensive use of caching, which is a cheaper solution than scaling back-end data stores.

While turning on and off servers helps save power, it does not help reduce space because the servers must still be physically present in order to meet peak demand. To cope with the physical demands of scaling, new data centers must be built when a given data center is either over its power budget, or out of space. A recent report states Google is spending 390 million USD to expand their Belgium data center [37]. Facebook also has plans to build a new 1.5 billion USD center in Iowa[32]. Because of this high cost, it is critical to avoid scaling by means of simply building new data centers or increasing the size of existing ones. This paper focuses on increasing physical density within a fixed power budget in order to reduce the data center footprint of key-value stores.

2.3 Memcached

In this paper we use Memcached 1.4 as our key-value store. We choose Memcached as our key-value store because of its widespread use in cloud computing. Memcached does not provide data persistence and servers do not need to communicate with each other, because of this it achieves linear scaling with respect to nodes. To date, the largest Memcached cluster with published data was Facebook's, which contained over 800 servers and had 28TB of DRAM in 2008 [45].

The ubiquity of Memcached stems from the fact that it is easy to use, because of its simple set of verbs. Only three details about Memcached need to be understood: first, it is distributed, which means that not every key will be on every server. In fact, a key should only be on one server, which allows the cluster to cache a large amount of data because the cache is the aggregate size of all servers. Second, the cache does not fill itself. While this may seem intuitive, the software using Memcached needs to ensure that after a read from the database, the data is stored in the cache for later retrieval. Entering data in the cache uses a *PUT*, and retrieving data from the cache uses a *GET*. Lastly, there are several options to denote when data expires from the cache. Data can either have a time-to-live, or be present in the cache indefinitely. A caveat to using Memcached is that data will be removed from your cache if a server goes down as Memcached does not have persistent storage.

2.3.1 Versions and Scaling

There are several versions of Memcached: the current stable release is 1.4, and 1.6 is under development. Version 1.6 aims to fix scaling issues caused by running Memcached with a large number of threads. A detailed analysis is presented in [43]. Prior research has shown that Memcached saturates neither the network bandwidth, nor the memory bandwidth [29], due to inefficiencies in the TCP/IP stack. In this work, we distribute the work of the TCP/IP stack among many small cores to provide greater aggregate bandwidth while increasing the storage density. This is possible because 3D stacking provides higher bandwidth and a faster memory interface.

3. Related Work

Prior work has focused on increasing the efficiency or performance with respect to the *transactions per second (TPS)* of Memcached systems, rather than density. As previously mentioned, we believe that density should be studied as a first class design constraint due to the high cost of scaling out data centers.

3.1 Characterizing Cloud Workloads

Ferdman et al. have demonstrated the mismatch between cloud workloads and modern out-of-order cores [11, 12]. Through their detailed analysis of scale-out workloads on modern cores, they discovered several important characteristics of these workloads: 1) Scale-out workloads suffer from high instruction cache miss rates, and large instruction caches and pre-fetchers, are inadequate; 2) instruction and memory-level parallelism are low, thus leaving the advanced out-of-order core underutilized; 3) the working set sizes exceed the capacity of the on-chip caches; 4) bandwidth utilization of scale-out workloads is low.

3.2 Efficient 3D-Stacked Servers

Prior work has shown that 3D stacking may be used for efficient server design. PicoServer [23] proposes using 3D stacking technology to design compact and efficient multicore processors for use in tier 1 servers. The focus of the PicoServer is on energy efficiency—they show that by closely stacking low-power cores on top of DRAM they can remove complex cache hierarchies, and instead, add more low-power

Component	Power (mW)	Area (mm^2)
A7@1GHz	100	0.58
A15@1GHz	600	2.82
A15@1.5GHz	1,000	2.82
3D DRAM (4GB)	210 (per GB/s)	279.00
3D NAND Flash (19.8GB)	6 (per GB/s)	279.00
3D Stack NIC (MAC)	120	0.43
Physical NIC (PHY)	300	220.00

Table 1: **Power and area for the components of a 3D stack.**

DRAM	BW (GB/s)	Capacity
DDR3-1333 [36]	10.7	2GB
DDR4-2667 [36]	21.3	2GB
LPDDR3 (30nm) [4]	6.4	512MB
HMC I (3D-Stack) [36]	128.0	512MB
Wide I/O (3D-stack, 50nm) [25]	12.8	512MB
Tezzaron Octopus (3D-Stack)[1]	50.0	512MB
Future Tezzaron (3D-stack)[14]	100.0	4GB

Table 2: **Comparison of 3D-stacked DRAM to DIMM packages.**

cores. The improved memory bandwidth and latency allow for adequate throughput and performance at a significant power savings. Nanostores [9] build on the PicoServer design and integrate Flash or Memristors into the stack. Both PicoServer and Nanostores could be used in a scale-out fashion to improve density, although this was not addressed in the work. This paper builds on their designs to address density, particularly in the context of Memcached.

More recently, Scale-Out Processors [30] were proposed as a processor for cloud computing workloads. The Scale-Out Processor design uses 3D stacked DRAM as a cache for external DRAM and implements a clever prefetching technique[19]. Our designs differ in that we only use the on-chip DRAM or Flash for storage with no external backing memory. This is possible because we share the Memcached data over several independent stacks in the same 1.5U box.

3.3 Non-Volatile Memory File Caching in Servers

In addition to Nanostores, several prior studies have proposed using non-volatile memory for energy-efficiency in servers [22, 24, 39]. They propose using non-volatile memory (NAND Flash and phase-change memory) for file caching in servers. The use of a programmable Flash memory controller, along with a sophisticated wear-leveling algorithm, allow for the effective use of non-volatile memory for file caching, while reducing idle power by an order of magnitude.

3.4 Super Dense Servers

Super Dense Servers (SDS) [10] have been shown to provide greater performance and efficiency for CPU-bound electronic commerce workloads. However, for workloads that require a large amount of physical memory—such as Memcached—traditional servers provided a better solution. By utilizing state-of-the-art 3D-stacked memory technology, we overcome the limitation of SDS by providing a very high level of memory density in our proposed server designs.

3.5 McDipper

Memcached has been used at Facebook for a wide range of applications, including MySQL look-aside buffers and photo serving. Using DRAM for these applications is relatively expensive, and for working sets that have very large footprints but moderate to low request rates, more efficient solutions are possible. Compared with DRAM, Flash solid-state drives provide up to 20× the capacity per server and still supports tens of thousands of operations per second. This prompted the creation of McDipper, a Flash-based cache server that is compatible with Memcached. McDipper has been in active use in production at Facebook for nearly a year to serve photos [13]. Our Iridium architecture targets these very large footprint workloads which have moderate to low request rates. We further extend their solution by using Toshiba's emerging 16-layer pipe-shaped bit cost scalable (p-BiCS) NAND Flash [21], which allows for density increases on the order of 5× compared to 3D-DRAM.

3.6 Enhancing the Scalability of Memcached

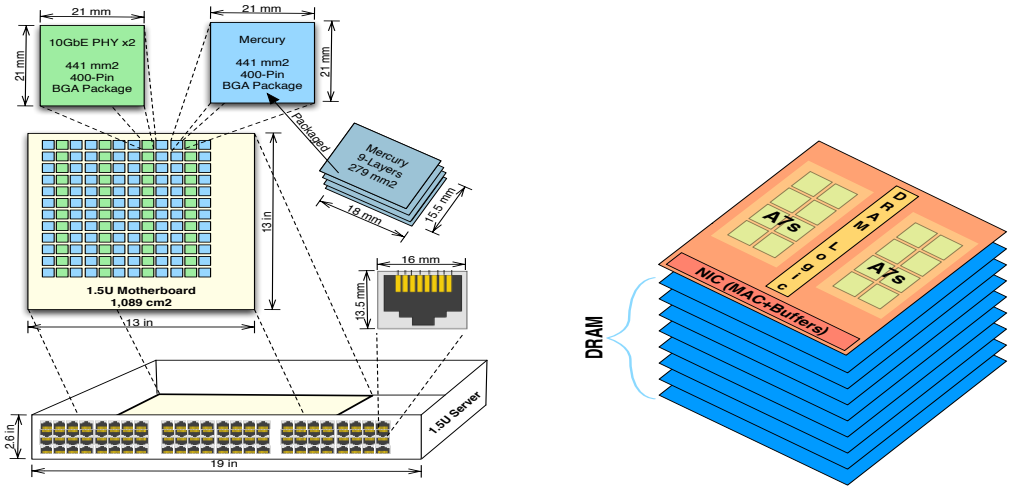
Wiggins and Langston [43] propose changes in Memcached 1.6 to remove bottlenecks that hinder performance when running on many core systems. They find that the locking structure used to control access to the hash table and to maintain LRU replacement for keys hinders Memcached when running with many threads. To mitigate the lock contention they use fine grain locks instead of a global lock. In addition, they modify the replacement algorithm to use a pseudo LRU algorithm, which they call Bags. Their proposed changes increase the bandwidth to greater than 3.1 MTPS, which is over 6× higher than an unmodified Memcached implementation.

3.7 TSSP

Lim et al. propose TSSP [29], which is an SoC including a Memcached accelerator to overcome the need for a large core in Memcached clusters. They find that, due to the network stack, Intel Atom cores would not be able to replace Intel Xeons in a standard cluster configuration. TSSP is able to offload all GET requests from the processor to the accelerator. The offload is accomplished by having a hash table stored in hardware and having a smart NIC that is able to forward GET requests to the accelerator. After data for a key is found, the hardware generates a response. Because little work needs to be done by software, an ARM Cortex-A9 is a suitable processor. The TSSP architecture achieves 17.63 KTPS/Watt.

3.8 Resource Contention in Distributed Hash Tables

Distributed hash tables (DHT) can suffer from issues that arise because there is not a uniform distribution of requests across resources. Keys in a key value store are assigned a



(a) **1.5U server with 96 Mercury stacks**

(b) **The 3D-stacked Mercury architecture**

Figure 2: **A Mercury server and a single stack.** A Mercury stack is made up of 8 DRAM layers, each 15.5mm×18mm, stacked with a logic layer containing the processing elements, DRAM peripheral logic, and NIC MAC & buffers. These stacks are then placed in a 400-pin BGA package. The 1.5U enclosure is limited to 96 Ethernet ports, each of which will be connected to a Mercury stack. Therefore, 96 Mercury stacks and 48 Dual NIC PHY chips are placed in the 1.5U enclosure. Due to space limitations a separate diagram of Iridium is omitted, however the high-level design is similar—the only difference being that we use a single layer of 3D-stacked Flash for the Iridium design.

resource by mapping a key onto a point in a circle. From this circle each node is assigned a portion of the circle, or arc. A server is responsible to store all data for keys that map onto their arc. Prior work dealing with resource contention in DHTs shows that increasing the number of nodes in the DHT reduces the probability of resource contention, because each node is responsible for a smaller arc [20, 41]. Typically, increasing the number of nodes has been accomplished by assigning one physical node to several virtual nodes. These virtual nodes are then distributed around the circle, which results in a more uniform utilization of resources. Because we increase the number of physical cores with Mercury and Iridium, resource contention should be minimized.

3.9 TILEPro64

Berezecki et al. [6] focus on adapting Memcached to run on the *TILEPro64*. In this work they cite power consumption of data centers as an important component for the success of a web service. With this in mind, they aim to improve the efficiency of Memcached by running it on a TILEPro64. They compare their implementation to both Opteron and Xeon processors running Memcached, and report an TPS/W of 5.75KTPS/W, which is an improvement of 2.85× and 2.43× respectively.

3.10 FAWN

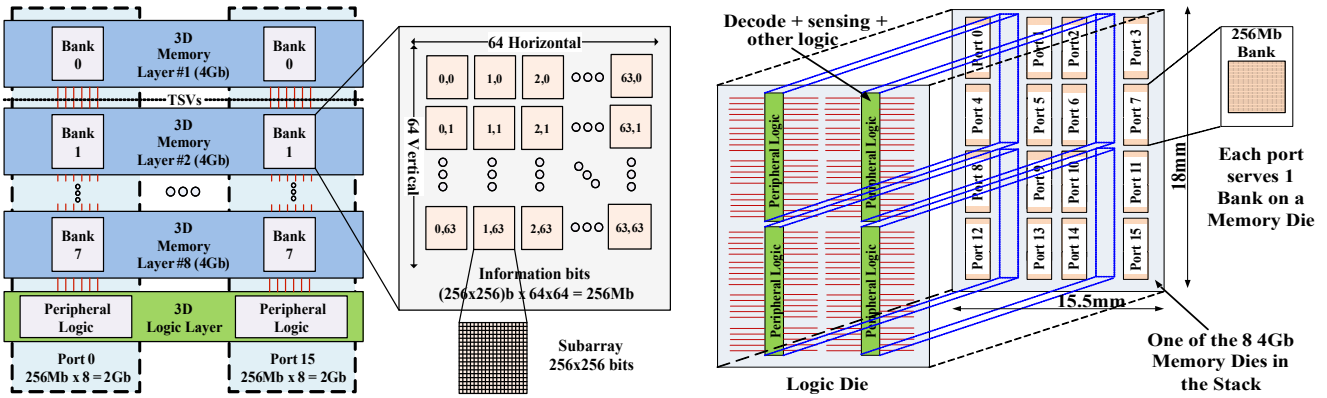
Andersen et al. design a new cluster architecture, called FAWN, for efficient, and massively parallel access to data [2]. They develop FAWN-KV—an implementation of their FAWN architecture for key-value stores that uses low-power embedded cores, a log-structured datastore, and Flash memory. With FAWN-KV they improve the efficiency of queries by two orders of magnitude over traditional disk-based systems. The FAWN system focuses on the key-value and filesystem design, whereas our work focuses on designing very dense servers.

4. Mercury and Iridium

The Mercury and Iridium architectures are constructed by stacking ARM Cortex-A7s, a 10GbE NIC, and either 4GB of DRAM or 19.8GB of Flash into a single 3D stack. The MAC unit of the NIC, which is located on the 3D stack, is capable of routing requests to the A7 cores. A conceptual representation is presented in Figure 2a. To evaluate the use of 3D stacks with key-value stores, we vary the number of cores per stack and measure the throughput and power efficiency for each server configuration. We designate the different architectures as Mercury-n or Iridium-n, where n is the number of cores per stack. We estimate power and area requirements based off of the components listed in Table 1.

4.1 Mercury

While the primary goal of Mercury and Iridium is to increase data storage density, this cannot be done at the expense of latency and bandwidth—the architecture would not be able to meet the *service-level agreement* (SLA) requirements that are typical of Memcached clusters. Thus we utilize low-power, in-order ARM Cortex-A7 cores in our 3D-stacked architecture, and as we will show in section 6, we are able to service a majority of requests within the sub-millisecond range.



(a) **Logical organization of the 4GB 3D-stacked DRAM.**

(b) **Physical floor plan of the logical die and a 512MB memory die in the 3D stack.** The top-down view of the stack shows that DRAM banks are arranged around spines of peripheral logic.

Figure 3: **Logical organization and physical floorplan of the 3D DRAM.**

4.1.1 3D Stack

The proposed Mercury architecture relies on devices from Tezzeron's 3D-stacking process [40]. This process allows us to stack 8 memory layers on a logic die through finely-spaced (1.75µm pitch), low-power through silicon vias (TSV). The TSVs have a feedthrough capacitance of 2-3fF and a series resistance of $< 3\Omega$. This allows as much as 4GB of data in each individual stack.

The 4GB stack's logical organization is shown in Figure 3a. Each 4GB 3D chip consists of eight 512MB DRAM memory dies stacked on top of a logic die. The organization of the DRAM die is an extension [14] of Tezzaron's existing Octopus DRAM solution [1]. Each 3D stack has 16 128-bit data ports, with each port accessing an independent 256MB address space. Each address space is further subdivided into eight 32MB banks. Each bank, in turn, is physically organized as a 64×64 matrix of subarrays. Each subarray is a 256×256 arrangement of bit cells, and is 60µm×35µm.

Figure 3b shows the physical floor plan of each DRAM memory die and the logic die. The logic die is fabricated in a 28nm CMOS process and consists of address-decoding logic, global word line drivers, sense amplifiers, row buffers, error correction logic, and low-swing I/O logic with pads. Each memory die is partitioned into 16 ports with each port serving 1 of the 16 banks on a die. The memory die is fabricated in a 50nm DRAM process and consists of the DRAM subarrays along with some logic, such as local wordline drivers and pass-gate multiplexers.

While there are more advanced DRAM processes (e.g. 20nm), TSV yield in existing 3D-stacked prototypes has only been proven up to the 50nm DRAM process node [25, 36]. All subarrays in a vertical stack share the same row buffer using TSVs, and at most one row of subarrays in a vertical stack can have its contents in the row buffer, which corresponds to a physical page. Assuming an 8kb page, a maximum of 2,048 pages can be simultaneously open per stack (128 8kb pages per bank × 16 banks per physical layer). The device provides a sustained bandwidth of 6.25GB/s per port (100GB/s total).

4.1.2 Address Space

The 3D stack DRAM has 16 ports for memory access, this segments the 4GB stack into 256MB chunks. Each core is allocated one or more ports for memory access, which prevents Memcached processes from overwriting each other's address range. If the Mercury or Iridium architectures increase past 16 cores, additional ports would need to be added, or cores would need to share ports.

4.1.3 Memory Access

Other studies have shown that small cores alone are not able to provide the needed bandwidth to be useful in key-value stores [29]. Mercury differs from this research because it is coupled with a memory interface that provides higher bandwidth at a lower latency through 3D integration. For comparison, Table 2 shows the bandwidth and capacity of several current and emerging memory technologies. Coupling cores on the 3D stack allows Mercury to forego using an L2 cache, which prior research has shown to be inefficient [29], and issue requests directly to memory. The 3D DRAM has a closed page latency of 11 cycles at 1GHz. By having a faster connection to memory we mitigate the cache thrashing by the networking stack reported in [29].

4.1.4 Request Routing

To simplify design and save power we do not use a router at the server level. Instead, the physical network port is tied directly to a 3D stack. This allows for each stack to act as a single node, or multiple nodes, without having contention from other stacks. At the stack, we base our design off of the integrated NIC on the Niagra-2 chip [5, 26]. The

integrated NIC is capable of buffering a packet and then forwarding it to the correct core. Cores on the same stack will need to run Memcached on different TCP/IP ports. This simplifies request routing on the stack. The physical port (PHY) portion of the 10GbE is based on the Broadcom design [8] and is not located on the stack.

4.2 Iridium

Due to the high cost of server real estate, physical density is first-class design constraint for key-value stores. Facebook has developed McDipper [13], which utilizes Flash memory to service low-request-rate transactions while improving density by up to 20×. McDipper has been in use for over a year to service photos at Facebook. With Iridium we target these low-request-rate applications and explore the tradeoff between physical density and throughput by replacing the DRAM in a Mercury stack with NAND Flash memory. This comes at the expense of throughput per GB of storage, but is still applicable to low-request-rate high-density Memcached clusters, such as Facebook's photo server.

4.2.1 3D Stacked Flash

McDipper reported an increase in density by 20× when moving from DRAM DIMMs to Solid-State drives. Because the Mercury design already improved density through 3D-DRAM we will not see as significant a gain. To quantify the improvement in the density of the stack we estimate our Flash density using the cell sizes of Toshiba's emerging pipe-shaped bit cost scalable (p-BiCS) NAND Flash [21]. Flash cells are smaller than DRAM cells, offering a 2.5× increase in density. Because p-BiCS has 16 layers of 3D Flash[2], as opposed to 8 layers for 3D-stacked DRAM, this leads to an overall 4.9× increase in density for Iridium stacks. For our access organization we maintain Mercury's 16 separate ports to DRAM by including 16 independent Flash controllers. The read/write latency values and energy numbers used for simulation are drawn from [15], which are conservative for 3D-stacked Flash. The power and area numbers for Flash are shown in table 1. In addition, because the Flash latency is much longer, an L2 cache is needed to hold the entire instruction footprint. Our results in Section 6.2 will confirm this assumption.

5. Methodology

The following sections describe our Memcached setup, as well as our simulation framework and power models.

5.1 Memcached

For our experiments we utilize the latest stable version of Memcached cross-compiled for the ARM ISA. We modify our simulation infrastructure, which will be described in the next section, to collect timing information. We do this, as opposed to using library timing functions such as *gettimeofday()*, because they do not perturb the system under test. All experiments are run with a single Memcached thread.

For our client we use an in-house Java client that is based on the work by Whalin [42]. Because we measure performance from the server-side, the overhead of running Java is not included with the measurements.

5.2 Simulation Infrastructure

To calculate the request rates that Mercury and Iridium are able to achieve, we used the gem5 full-system simulator [7]. With gem5, we are able to model multiple networked systems with a full operating system, TCP/IP stack, and Ethernet devices. We use Ubuntu 11.04 along with version *2.6.38.8* of the Linux kernel for both the server and client systems. For our client-side Java VM, we use *Java SE 1.7.0_04-ea* from Oracle.

While Mercury and Iridium both utilize ARM Cortex-A7 cores, we also explore the possibility of using the more aggressive, out-of-order Cortex-A15 core by modelling both cores in our simulation infrastructure. Simulations use a memory model with a memory latency varied from 10-100ns for DRAM and 10-20μs for Flash Reads. Our memory model represents a worst-case estimate as it assumes a closed-page latency for all requests. To measure the TPS we vary request size from 64B to 1MB, doubling request size at each iteration. We do not consider values greater than 1MB for the following reasons: 1) Memcached workloads tend to favor smaller size data; 2) prior work [3, 29, 43], against which we compare, present bandwidth per watt at data sizes of 64B and 128B; and, 3) requests that are 64KB or larger have to be split up into multiple TCP packets.

5.3 TPS calculation

To calculate the TPS we collect the round-trip time (RTT) for a request, i.e., the total time it takes for a request to go from the client to the server, then back to the client. Because we use only a single thread for Memcached, the TPS is equal to the inverse of the RTT. The RTT for each request is obtained by dumping TCP/IP packet information from gem5's Ethernet device models. The packet trace is run through TShark [44] to parse out timing information. After timing information is obtained from gem5 for a single core, we apply linear scaling to calculate TPS at the stack and server level. Linear scaling is a reasonable approach in this context because each core on a stack is running a separate instance of Memcached. Running separate instances avoids the contention issue raised by the work of Wiggins and Langston[43]. For the Mercury-32 and Iridium-32 configurations we use the same approach, however we assume two cores per memory port (as mentioned previously we can fit a maximum of 16 memory ports on a stack), because Memcached has been shown to scale well for two threads [43].

[2] The 16 Flash layers are contained in a single monolithic layer of 3D Flash, and are not 3D die-stacked.

		1.5U Mercury Server						1.5U Iridium Server					
	Number of Cores per Stack	1	2	4	8	16	32	1	2	4	8	16	32
A15 1.5GHz	Area(cm^2)	635	635	576	331	179	86	635	635	635	363	185	93
	Power(W)	449	569	749	750	750	720	328	449	690	740	737	730
	Density(GB)	384	384	348	200	108	52	1,901	1,901	1,901	1,089	554	277
	Max BW(GB/s)	27	55	99	114	123	118	1	3	6	7	7	7
A15 1GHz	Area(cm^2)	635	635	635	496	278	139	635	635	635	595	304	152
	Power(W)	401	473	618	745	742	728	281	353	498	750	740	728
	Density(GB)	384	384	384	300	168	84	1,901	1,901	1,901	1,782	911	455
	Max BW(GB/s)	27	54	108	169	189	189	2	3	6	11	11	11
A7 1GHz	Area(cm^2)	635	635	635	635	635	616	635	635	635	635	635	635
	Power(W)	341	353	378	429	529	749	221	233	258	309	410	612
	Density(GB)	384	384	384	384	384	371	1,901	1,901	1,901	1,901	1,901	1,901
	Max BW(GB/s)	19	37	75	149	299	578	1	3	6	12	22	44

Table 3: **Power and area comparison for 1.5U maximum configurations.** For each configuration we utilize the maximum number of stacks we can fit into a 1.5U server, which is 96, or until we reach our power budget of 750W. The power and bandwidth numbers are the maximum values we observed when servicing requests from 64B up to 1MB.

5.4 Power Modeling

Table 1 has a breakdown of power per component, and Table 3 has the cumulative power totals for each configuration of Mercury and Iridium at their maximum sustainable bandwidths. To calculate the power of an individual stack we add together the power of the NIC, cores, and memory. The integrated 10GbE NIC is comprised of a MAC and buffers. The MAC power estimates come from the Niagra-2 design [5, 26], and the buffers are estimated from CACTI [34]. The ARM Cortex-A7 and Cortex-A15 power numbers are drawn from [16], and the 3D DRAM is calculated from a technical specification obtained from Tezzaron [14]. Because DRAM active power depends on the memory bandwidth being used, we must calculate the stack power for the maximum bandwidth that a given number of cores can produce. Similar calculations are used to obtain NAND Flash power, which use read and write energy values drawn from [15]. Finally, each stack requires an off-stack physical Ethernet port. Power numbers for the PHY are based on a Broadcom part [8].

5.4.1 Power Budget for 1.5U Mercury System

In calculating the power for the 1.5U server system we multiply the per stack power by the number of stacks. To determine how many stacks can fit in a 1.5U power budget, we start with a 750W power supply from HP [18]. First 160W is allotted for other components (disk, motherboard, etc.), after this we assume a conservative 20% margin for miscellaneous power and delivery losses in the server. This results in a maximum power of $(750 - 160) \times 0.8 = 472$W for Mercury or Iridium components.

5.4.2 TPS/Watt calculation

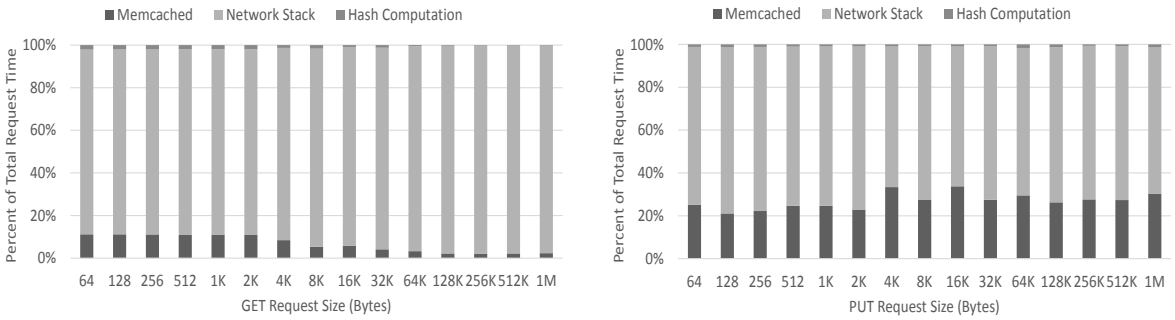
When calculating the maximum number of 3D stacks that fit in a system, we used the maximum memory bandwidth that Mercury and Iridium can produce. However, for proper TPS/Watt calculation, we estimate power by using the GB/s power consumption of DRAM and Flash at the request size we are testing.

5.5 Area

Area estimates for the 10GbE NIC were obtained by scaling the Niagra-2 MAC to 28nm and the buffers were obtained from CACTI. The area for an ARM Cortex-A7 chip in 28nm technology is taken from [16]. DRAM design estimates for the next generation Tezzaron Octopus DRAM were obtained from Tezzaron [14]. Given the available area on the logic die of the Tezzaron 3D-stack, we are able to fit >400 cores on a stack. However, the memory interface becomes saturated if there are ≥ 64 cores in a stack. We are further limited, by the number of DRAM ports in a stack, to 16 cores per stack unless cores share the memory controller. In each 1.5U server, multiple 3D stacks are used to increase the bandwidth and density. Once packaged into a 400-pin 21mm×21mm ball grid array (BGA) package, each stack consumes 441mm^2. Each NIC PHY chip is also 441mm^2 and contains 2 10GbE PHYs/chip. If 77% of a 1.5U, 13in×13in motherboard [27] is used for Mercury or Iridium stacks and associated PHYs, then the server can fit 128 Mercury or Iridium stacks. However, only 96 Ethernet ports can fit on the back of a 1.5U server [31]. Therefore, we cap the maximum number of stacks at 96.

5.6 Density

We define density to be the amount of DRAM that we can fit in the system. We then maximize density within three constraining factors: power, area, and network connections. As more cores are added to the system, the power requirement for both core power and DRAM bandwidth will increase, because of this there is a tradeoff between throughput and density. Each stack can fit 4GB of DRAM or 19.8GB of Flash and each server can fit up to 128 3D stacks, which gives a maximum density of 512GB of DRAM for Mercury or 2.4TB of Flash for Iridium. Each server can fit a maximum of 96 network connections, capping the number of stacks to 96 and density to 384GB of DRAM for Mercury or 1.9TB of Flash for Iridium.



(a) **GET execution time**

(b) **PUT execution time**

Figure 4: **Components of GET and PUT requests.**

6. Results

We evaluate several different 3D-stacked server configurations, which are differentiated based on the number (and type) of cores per stack, and on the type of memory used. DRAM-based configurations we call Mercury, while Flash-based configurations are called Iridium (because Iridium is more dense).

6.1 Request Breakdown

We first explore the different components of GET and PUT requests. Figure 4 shows a breakdown of execution time for both GET and PUT requests; execution is broken down into three components: hash computation time (*Hash Computation*), the time in metadata processing to find the memory location of the data (*Memcached*), and time spent in the network stack and data transfer (*Network Stack*). These experiments were run using a single A15 @1GHz, with a 2MB L2 cache, and DRAM with a latency of 10ns. We ran these experiments for various configurations, however the results were similar.

6.1.1 GET and PUT requests

Figure 4a shows the breakdown of execution during a GET request. For requests up to 4KB roughly 10% of time is spent in Memcached, 2-3% is spend in hash computation, and a vast majority of the time (87%) is spent in the network stack. At higher request sizes nearly all of the time is spent in the network stack, which is in agreement with prior research [29].

Figure 4b shows the breakdown of execution during a PUT request. As expected, hash computation takes up the same amount of time for a PUT request as it does for a GET request, however it represents a much smaller portion of the time: only around 1%. Also as expected, Memcached metadata manipulation takes up more computation for a PUT request: up to 30% in some cases. Network processing is still the largest component at nearly 70% for some request sizes.

Because the network stack takes up a significant portion of the time, utilizing aggressive out-of-order cores is inefficient. As we will demonstrate, by closely integrating memory with many low-power cores, we can achieve a high-level of throughput much more efficiently than traditional servers.

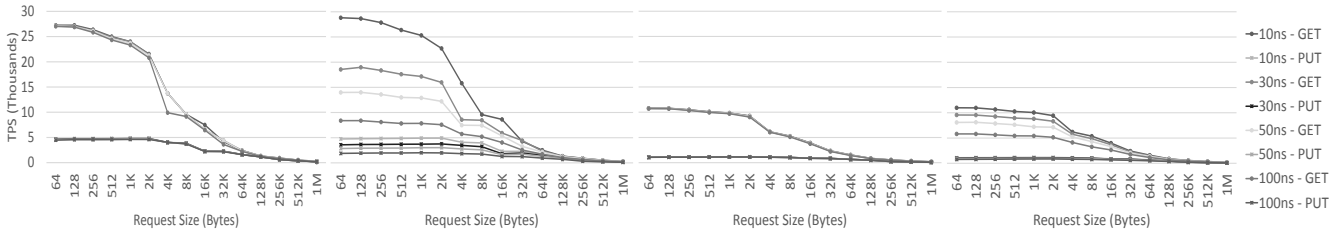
6.2 3D-Stack Memory Access Latency Sensitivity

While the focus of this work is on improving density for Memcached servers, this cannot be provided at the expense of latencies that would violate the SLA. To explore the effect memory latency and CPU performance have on overall request RTT we measure the average TPS, which is the inverse of the average RTT for single-core Mercury and Iridium stacks. The lower the RTT the higher the TPS, thus a higher TPS indicates better overall performance for individual requests.

Figure 5 shows the TPS sensitivity to memory latency, with respect to GET/PUT request size, for a Mercury-1 stack. We evaluate a Mercury-1 stack for an A15 and an A7, both with and without an L2 cache. We also explored using an A15 @1.5GHz, which is the current frequency limit of the A15, however we do not report these results because they are nearly identical to an A15 @1GHz. For each configuration we sweep across memory latencies of 10, 30, 50, and 100ns. Similarly, figure 6 demonstrates the throughput sensitivity to memory latency for an Iridium-1 stack. For Iridium-1 we sweep across Flash read latencies of 10 and 20μs; write latency is kept at 200 μs.

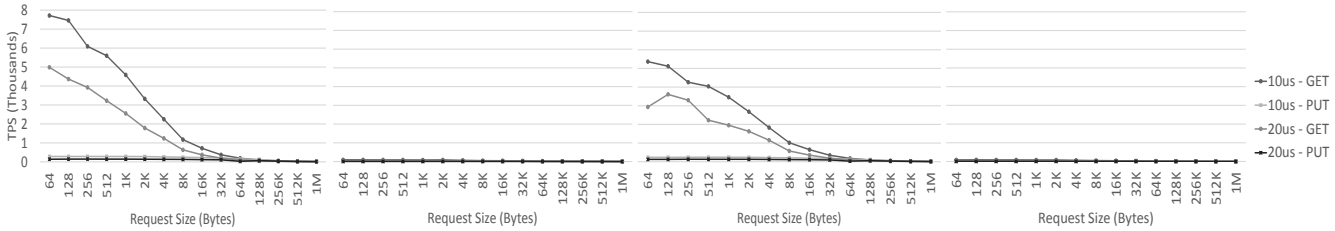
Figures 5a and 5b show the average TPS for an A15-based Mercury-1 stack with and without an L2 cache respectively. As can be seen, at a latency of 10ns the L2 provides no benefit, and may hinder performance. Because the DRAM is much faster than the core, the additional latency of a cache lookup, which typically has poor hit rates, degrades the average TPS. However, at the higher DRAM latencies the L2 cache significantly improves performance; while the stored values are typically not resident in the L2 cache, instructions and other metadata benefit from an L2 cache.

Similarly, figures 5c and 5d report average TPS for an A7-based Mercury-1 stack with and without and L2 cache.



(a) **A15 @1GHz with a 2MB L2** (b) **A15 @1GHz with no L2** (c) **A7 with a 2MB L2** (d) **A7 with no L2**

Figure 5: **Transactions per second for a Mercury-1 stack for the different CPU configurations and DRAM latency.**



(a) **A15 @1GHz with a 2MB L2** (b) **A15 @1GHz with no L2** (c) **A7 with a 2MB L2** (d) **A7 with no L2**

Figure 6: **Transactions per second for a Iridium-1 stack for the different CPU configurations and Flash latency.**

Because of the less aggressive core, the L2 cache makes less of a difference for an A7-based Mercury-1 stack. The A15-based Mercury-1 stack significantly outperforms an A7-based Mercury-1 stack by about 3× at the lower request sizes. If the L2 cache is removed, the A15 only outperforms the A7 by 1-2× at the lower request sizes.

Finally, figure 6 reports the average TPS for an Iridium-1 stack. Because of the relatively slow read and write latencies of Flash, an L2 cache is crucial for performance; removing the L2 cache yields average an average TPS below 100 for both the A7 and A15, which is not acceptable. However, with an L2 cache both the A15 and A7 can sustain an average of several thousand TPS for GET requests, with a bulk of the requests being serviced under 1ms. For PUT requests the average TPS is below 1,000, however GET requests have been shown to make up a bulk of the requests for typical Memcached applications [3]. Because of Flash's relatively slow speed, the A15 outperforms an A7 by around 25% on average.

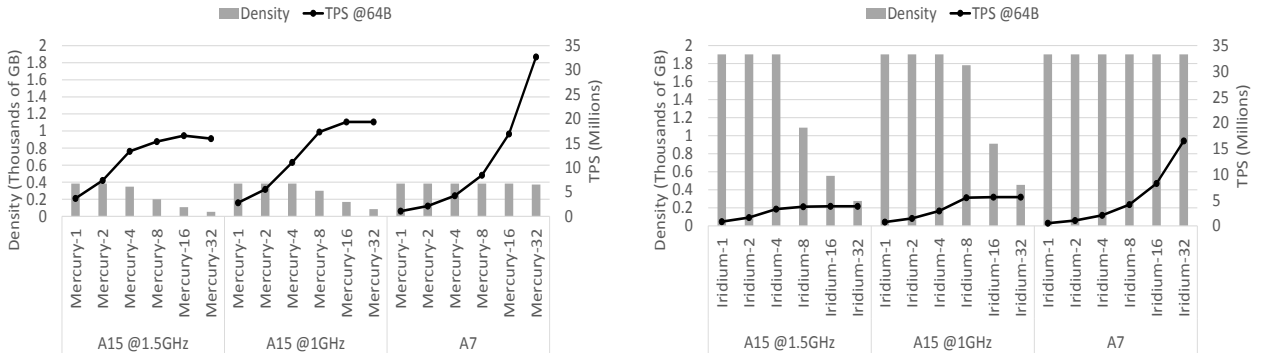
These results demonstrate the tradeoff between core and memory speed and performance. If very low response time is required for individual requests, faster memory and cores are desired. If, however, throughput and density are first-class constraints, then less aggressive cores and memory may be used without violating the SLA; as we will show in the next sections, density and efficiency may be vastly improved.

6.3 Density and Throughput

We define the density of a stack to be the total amount of memory it contains. Figure 7 illustrates the tradeoff between density and total throughput for different Mercury and Iridium configurations. Throughput is measured as the average TPS for 64B GET requests; prior works have shown that small GET requests are the most frequent requests in Memcached applications, and base their evaluations on such requests [3, 43]. Each configuration is labelled Mercury-n or Iridium-n, where n is the number of cores per stack. For all Mercury configurations we use a DRAM latency of 10ns, and for all Iridium configurations we use Flash read and write latencies of 10 and 200μs respectively. In each configuration the core contains a 2MB L2 cache. Table 3 lists the details of each separate Mercury and Iridium configuration.

The core power (listed in table 1) is the limiting factor when determining how many total stacks our server architecture can support. As figure 7 shows, the A15's higher power consumption severely limits the number of stacks that can fit into our power budget. At 8 cores per stack we see a sharp decline in density, while performance levels off. The A15's peak efficiency comes at 1GHz for both Mercury-8 and Iridium-8 stacks, where we are able to fit 75 stacks (600 cores) and 90 stacks (720 cores) respectively; Mercury-8 can sustain an average of 17.29 million TPS with 300GB of memory, while Iridium-8 can sustain an average of 5.45 million TPS with approximately 2TB of memory.

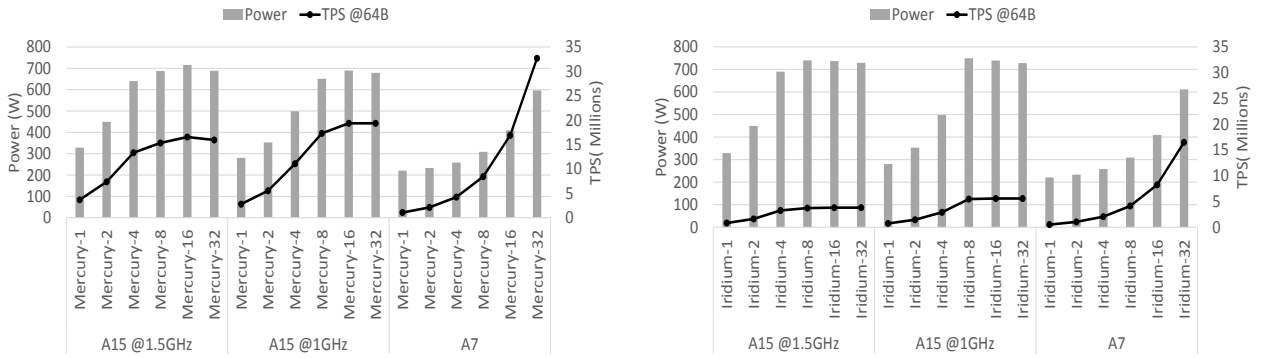
The A7's relatively low power allows us to fit nearly the maximum number of stacks into our server, even at 32 cores per stack. Because of this, the A7-based Mercury and Iridium designs are able to provide significantly higher performance and density than their A15-based counterparts. Mercury-32 can sustain an average TPS of 32.7 million with



(a) **Mercury density vs. TPS**

(b) **Iridium density vs. TPS**

Figure 7: **Density and throughput for Mercury and Iridium stacks servicing 64B GET requests.**



(a) **Mercury power vs. TPS**

(b) **Iridium power vs. TPS**

Figure 8: **Power and throughput for Mercury and Iridium stacks servicing 64B GET requests.**

372GB of memory, while Iridium-32 can sustain an average TPS of 16.48 with approximately 2TB of memory.

The A7 provides the most efficient implementation for both Mercury and Iridium stacks, however Mercury-32 provides around 2× more throughput when compared to Iridium-32; Iridium-32, on the other hand, provides nearly 5× more density. If performance is a primary concern Mercury-32 is the best choice. If high density is needed, Iridium-32 provides the most memory and is still able to satisfy the SLA requirements.

6.4 Power and Throughput

The power and throughput tradeoff is shown in figure 8. Again, we measure the throughput of our system while servicing 64B GET requests. Because we are limited to a maximum of 96 stacks for a single server, the configurations that contain 1, 2, or 4 cores are well under our maximum power budget of 750W, however as we add more cores per stack we come close to saturating our power budget. As power becomes a constraint, the number of stacks we are able to fit into our power budget is reduced. Thus, there is a tradeoff between total throughput and overall power. We seek to maximize throughput while staying within our power budget, therefore we always opt for a system with the maximum number of stacks if possible.

Figure 8a shows that, for Mercury, the A15's power quickly becomes prohibitive, limiting the number of cores we can fit into a server given our 750W power budget. The best A15 configuration is a Mercury-16 system that uses A15s @1GHz. An average of 19.36 million TPS can be sustained at a power of 678W. The max throughput for a Mercury-32 system using A15s @1GHz uses slightly less power than the Mercury-16 system, while delivering nearly the same throughput, because less stacks are used. Using A7s we are able to fit nearly the maximum number of stacks, while staying well below our power budget. A Mercury-32 system using A7s is the most efficient design, delivering 32.7 million TPS at a power of 597W.

For Iridium, the A15 is even less efficient, as seen in figure 8b—the A15's extra power does not help performance because it often waits on memory. The throughput for an A7-based Iridium-32 system is half that of an A7-based Mercury-32 system at roughly the same power budget. However, as noted above the Iridium-32 system provides 5× more density. The power of Iridium is slightly higher than Mercury because the relatively low power of Flash allows for more stacks.

		Mercury			Iridium			Memcached			TSSP
	Version	n=8	n=16	n=32	n=8	n=16	n=32	1.4	1.6	Bags	—
1.5U Server @64B Request	Stacks	96	96	93	96	96	96	1	1	1	1
	Cores	768	1,536	2,976	768	1,536	3,072	6	4	16	1
	Memory(GB)	384	384	372	**1,901**	**1,901**	**1,901**	12	128	128	8
	Power(W)	309	410	597	309	410	611	143	159	285	16
	TPS(millions)	8.44	16.88	32.70	4.12	8.24	16.49	0.41	0.52	3.15	0.28
	TPS(thousands)/Watt	27.33	41.21	**54.77**	13.35	20.13	26.98	2.9	3.29	11.1	17.6
	TPS(thousands)/GB	21.98	43.96	**87.91**	2.17	4.34	8.67	34.2	4.1	24.6	35.3
	Bandwidth(GB/s)	0.54	1.08	2.09	0.26	0.53	1.06	0.03	0.03	0.20	0.04

Table 4: **Comparison of A7-based Mercury and Iridium to prior art.** We compare several versions of Mercury and Iridium (recall n is the number of cores per stack) to different versions of Memcached running on a state-of-the-art server, as well as TSSP. The bold values represent the highest density (GB), efficiency (TPS/W), and accessibility (TPS/GB).

6.5 Cooling

The TDP of a Mercury-32 server is 597W (the same as today's 1.5U systems) and is spread across all 96 stacks; in contrast to a conventional server design where all the power is concentrated on a few chips. This yields a TDP for an individual stack of 6.2W. Thus, we expect the power of each Mercury chip to be within the capabilities of passive cooling, and an active fan in the 1.5U box can be used to extract the heat. Prior work on the thermal characterization of cloud workloads also supports this notion [33].

6.6 Comparison to Prior Work

To gauge the performance of Mercury and Iridium we compare their efficiency, density, and performance to modern versions of Memcached running on a state-of-the-art server. Table 4 lists the pertinent metrics for the best performing Mercury and Iridium configurations, and compares them to prior art. As can be seen in this table both Mercury and Iridium provide more throughput, 10× and 5.2× more than Bags respectively, primarily due to the massive number of cores they can support. At the same time, the use of low-power cores and integrated memory provides greater efficiency: 4.9× higher TPS/W for Mercury and 2.4× more for Iridium. Because of the speed of memory used in Mercury systems, it can even make better use of its density yielding an average TPS/GB that is 3.4× higher than Bags. Iridium has 2.8× less TPS/GB on average due to its much higher density. Overall, Mercury and Iridium provide 2.9 × and 14.8× more density on average, while still servicing a majority of requests within the sub-millisecond range.

The table also reports the same metrics for TSSP, which is an accelerator for Memcached. While this work aims to improve the efficiency of Memcached by using specialized hardware, the 3D-stacked Mercury and Iridium architectures are able to provide 3× and 1.5× more TPS/W respectively.

7. Conclusion

Distributed, in-memory key-value stores, such as Memcached, are so widely used by many large web companies that approximately 25% of their servers are devoted solely to key-value stores. Data centers are expensive, a fact which requires that each unit be used as efficiently as possible. While previous works have recognized the importance of key-value stores, and the fact that they are inefficient when run on commodity hardware, the approach has typically been to try to improve the performance and efficiency of existing server systems.

In this work we propose using state-of-the-art 3D stacking techniques to develop two highly-integrated server architectures that are not only able to allow low-power, embedded CPUs to be used without sacrificing bandwidth and performance, but are also able to drastically improve density. This is a crucial component to keeping the cost of ownership for data centers down. Through our detailed simulations, we show that, by using 3D stacked DRAM (Mercury), density may be improved by 2.9×, efficiency by 4.9×, TPS by 10×, and TPS/GB by 3.5× over a current state-of-the-art server running an optimized version of Memcached. By replacing the DRAM with Flash our Iridium architecture can service moderate to low request rate servers with even better density while maintaining SLA requirements for the bulk of requests. Iridium improves density by 14×, efficiency by 2.4×, TPS by 5.2×, with only a 2.8× reduction in TPS/GB compared to current Memcached servers.

Acknowledgments

We would like to than the anonymous reviewers for their feedback. This work was supported by a grant from ARM Ltd.

References

[1] Octopus 8-Port DRAM for Die-Stack Applications . Technical report, Tezzaron, 2012. Accessed: May 29, 2013.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *the Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, pages 1–14, 2009.

[3] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-Scale Key-Value Store. In *Proceedings of the*

12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, pages 53–64, 2012.

[4] Y.-C. Bae, J.-Y. Park, S. J. Rhee, S. B. Ko, Y. Jeong, K.-S. Noh, Y. Son, J. Youn, Y. Chu, H. Cho, M. Kim, D. Yim, H.-C. Kim, S.-H. Jung, H.-I. Choi, S. Yim, J.-B. Lee, J.-S. Choi, and K. Oh. A 1.2V 30nm 1.6Gb/s/pin 4Gb LPDDR3 SDRAM with Input Skew Calibration and Enhanced Control Scheme. In *Proceedings of the 2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 44–46, 2012.

[5] J. Barreh, J. Brooks, R. Golla, G. Grohoski, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, and M. Shah. Niagara-2: A Highly Threaded Server-on-a-Chip. In *Proceedings of the 18th Hot Chips Symposium*, 2006.

[6] M. Berezecki, E. Frachtenberg, M. Paleczny, and K. Steele. Many-Core Key-Value Store. In *Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC)*, pages 1–8, 2011.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 Simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[8] Broadcom. Broadcom Introduces Industry's Lowest Power Octal 10GbE PHY. `http://www.broadcom.com/press/release.php?id=s706156`, 2012. Accessed: May 29, 2013.

[9] J. Chang, P. Ranganathan, T. Mudge, D. Roberts, M. A. Shah, and K. T. Lim. A Limits Study of Benefits from Nanostore-Based Future Data-Centric System Architectures. In *Proceedings of the 9th Conference on Computing Frontiers (CF)*, pages 33–42, 2012.

[10] W. Felter, T. Keller, M. Kistler, C. Lefurgy, K. Rajamani, R. Rajamony, F. Rawson, B. A. Smith, and E. Van Hensbergen. On the Performance and Use of Dense Servers. *IBM Journal of Research and Development*, 47(5.6):671–688, 2003.

[11] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 37–48, 2012.

[12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Quantifying the Mismatch Between Emerging Scale-Out Applications and Modern Processors. *ACM Transactions on Computing Systems*, 30 (4):15:1–15:24, 2012.

[13] A. Gartrell, M. Srinivasan, B. Alger, and K. Sundararajan. McDipper: A Key-Value Cache for Flash Storage. `http://www.facebook.com/notes/facebook-engineering/mcdipper-a-key-value-for-flash-storage/10151347090423920`, 2013. Accessed: July 1, 2013.

[14] B. Giridhar, M. Cieslak, D. Duggal, R. G. Dreslinski, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, and D. Blaauw. Exploring DRAM Organizations for Engery-Efficient and Resilient Exascale Memories. In *Proceedings of the 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 23:1–23:12, 2013.

[15] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing Flash Memory: Anomalies, Obervations, and Applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33, 2009.

[16] L. Gwennap. How Cortex-A15 Measures Up. *Microprocessor Report*, May 2013.

[17] J. L. Henning. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[18] *HP Common Slot Platinum Power Supplies*. HP, 2010.

[19] D. Jevdjic, S. Volos, and B. Falsafi. Die-Stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013.

[20] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 1997.

[21] R. Katsumata, M. Kito, Y. Fukuzumi, M. Kido, H. Tanaka, Y. Komori, M. Ishiduki, J. Matsunami, T. Fujiwara, Y. Nagata, L. Zhang, Y. Iwata, R. Kirisawa, H. Aochi, and A. Nitayama. Pipe-Shaped BiCS Flash Memory with 16 Stacked Layers and Multi-Level-Cell Operation for Ultra High Density Storage Devices. In *Proceedings of the 2009 Symposium on VLSI Technology*, pages 136–137, 2009.

[22] T. Kgil and T. Mudge. FlashCache: A NAND Flash Memory File Cache for Low Power Web Servers. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 103–112, 2006.

[23] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy Efficient Chip Multiprocessor. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 117–128, 2006.

[24] T. Kgil, D. Roberts, and T. Mudge. Improving NAND Flash Based Disk Caches. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, pages 327–338, 2008.

[25] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H.-R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S.-K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. Lee, K. Kyung, J.-S. Choi, and Y.-H. Jun. A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking. In *Proceedings of the 2011 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 496–498, 2011.

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

[27] H. Li and A. Michael. *Intel Motherboard Hardware v1.0*. Open Compute Project, 2013.

[28] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, pages 315–326, 2008.

[29] K. Lim, D. Meisner, A. Saidi, P. Ranganathan, and T. Wenisch. Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013.

[30] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi. Scale-Out Processors. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 500–511, 2012.

[31] *SX1016 64-Port 10GbE SDN Switch System*. Mellanox Technologies, 2013.

[32] C. Metz. Facebook Data Center Empire to Conquer Heartland in 2014. `http://www.wired.com/wiredenterprise/2013/04/facebook-iowa-2014/`, 2013. Accessed: May 31, 2013.

[33] D. Milojevic, S. Idgunji, D. Jevdjic, E. Ozer, P. Lotfi-Kamran, A. Panteli, A. Prodromou, C. Nicopoulos, D. Hardy, B. Falsari, and Y. Sazeides. Thermal Characterization of Cloud Workloads on a Power-Efficient Server-on-Chip. In *Proceedings of the IEEE 30th International Conference on Computer Design (ICCD)*, pages 175–182, 2012.

[34] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 3–14, 2007.

[35] G. Orzell and J. Becker. Auto Scaling in the Amazon Cloud. `http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html`, 2012. Accessed: May 29, 2013.

[36] J. Pawlowski. Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem. In *Proceedings of the 23rd Hot Chips Symposium*, 2011.

[37] A. Press. Google to invest $300m in data center in Belgium. `http://finance.yahoo.com/news/google-invest-390m-data-center-081818916.html`, 2013. Accessed: May 31, 2013.

[38] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, pages 314–325, 2010.

[39] D. Roberts, T. Kgil, and T. Mudge. Using Non-Volatile Memory to Save Energy in Servers. In *Proceedings of the 2009 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 743–748, 2009.

[40] T. Semiconductor. `http://www.tezzaron.com`. Accessed: May 29, 2013.

[41] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.

[42] G. Whalin. Memcached Java Client. `http://github.com/gwhalin/Memcached-Java-Client`. Accessed: May 29, 2013.

[43] A. Wiggins and J. Langston. Enhancing the Scalability of Memcached. Technical report, Intel, 2012.

[44] Wireshark. `http://wireshark.org`. Accessed: July 1, 2013.

[45] M. Zuckerberg. Facebook and memcached. `http://www.facebook.com/video/video.php?v=631826881803`, 2008. Accessed: May 29, 2013.