ARCHITECTING ENERGY EFFICIENT SERVERS

by

Tae Ho Kgil

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2007

Doctoral Committee:

Professor Trevor N. Mudge, Chairperson
Associate Professor Scott Mahlke
Associate Professor Steven K. Reinhardt
Associate Professor Dennis Michael Sylvester

ACKNOWLEDGEMENTS

I started my graduate program in Michigan with the hope of gaining some experience in architecting computing platforms. To this end, I have gained a lot more than I expected. I would like to first acknowledge my Research Advisor Trevor Mudge for providing me with excellent direction and advice. He has introduced me to interesting problems that need to be solved by today's system architects. My defense committee members have also helped me improve my research abilities. Dennis, I remember the first time we met when I was a fresh graduate student asking for guidance in respect to course work. I recall taking a class you taught in the first semester. That class has inspired me and lead me to believe that I made the right decision coming to graduate school. Steve, your collaboration and guidance in research, especially with system level simulation, has helped me realize issues that I could not identify before. I sincerely appreciate your efforts. And finally Scott, your comments on my work in many aspects have definitely helped me rethink the problem and look at it at a new angle.

I would like to acknowledge my colleagues, Sangwon Yoon, Jisoo Yang, Mark Woh, HyunSeok Lee, Geoff Blake, Nate Binkert, Ali Saidi, Ron Dreslinski and Yuan Lin. The interesting discussions and debates we had, helped me in many ways and will be missed. They have truly made my life better. I wish everyone the very best and hope to work with them in the future. I acknowledge my parents who have gone through a lot to put me through school and raising me. I am a stubborn person who is difficult to persuade yet they somehow guided me this far. I truly appreciate their love.

Finally, I am grateful to my wife Young Bin Baek. She has gone through my highs and lows which I could not even imagine tolerating. Your constant attention and support has made my life a lot better and allowed me to pursue a PhD. Young Bin, I couldn't have finished graduate school without you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

ARCHITECTING ENERGY EFFICIENT SERVERS

by

Tae Ho Kgil

Chairperson: Professor Trevor N. Mudge

This dissertation investigates how energy efficient servers can be architected using current and future technology. We leverage recent trends in packaging and device technology to deliver low power and high throughput. Specifically at the package level, this dissertation looks at 3D stacking technology that has emerged as a promising solution in achieving energy efficiency by delivering high throughput at a low cost. It shows how one would leverage this new technology into a datacenter. 3D stacking technology can be used to implement a simple, low-power, high-performance chip multiprocessor suitable for throughput processing. Our proposed architecture leveraging this technology, PicoServer, employs 3D technology to bond one die containing several simple slow processing cores to multiple memory dies sufficient for a primary memory. The multiple memory dies are composed of DRAM. 3D stacking technology also enables wide low-latency buses between processors and memory. These remove the need for an L2 cache allowing its area to be re-allocated to additional simple cores. The additional cores allow the clock frequency to be lowered without impairing throughput. Lower clock frequency along with the integration of non-volatile memory in turn reduces power and means that thermal constraints, a concern with 3D stacking, are easily satisfied. The PicoServer architecture targets server applications,

which exhibit a high degree of thread level parallelism. An architecture targeted to efficient throughput is ideal for this application domain.

At the memory device level, this dissertation investigates how the system memory could be re-architected to reduce the rising power consumption of system memory and disk drives. Flash memory has emerged as a strong candidate to reduce system memory power while remaining cost effective than conventional system memory. This dissertation discusses how Flash could be integrated at the system level and provides insights on the architectural support for Flash in servers. Our architecture uses a two level disk cache composed of a relatively small DRAM, which includes a primary disk cache, and a Flash based secondary disk cache. Further, based on our observations, we found that the Flash based disk caches should be split into a read optimized disk cache and write optimized disk cache.

CHAPTER I

INTRODUCTION

Datacenters are an integral part of today's computing platforms. It has received much attention by embracing the Internet and successfully connecting today's end users to the World Wide Web. With the growing importance of servers found in internet service providers like Google and AOL, energy efficiency has become a critical task. Low power systems such as blade servers have been introduced to reduce power in conventional power hungry server farms. Server farms based on off-the-shelf general purpose processors are unnecessarily power hungry, require expensive cooling systems and occupy a large space. It has been shown that 25% of the operating costs for these "server farms" can be directly or indirectly attributed to power consumption [74]. This figure has the potential to grow rapidly along with the continuing growth in web services.

Naturally much effort should be invested in reducing power costs and introducing new architectures that are energy efficient in datacenters. Unfortunately, this cannot be achieved using conventional techniques that primarily focussed on the microprocessor. With current advances in technology, it is apparent that performance and power should be handled at the system level. Today's system architects should recognize this trend and focus their attention in improving energy efficiency. Starting from the system memory to the I/O peripheral, an architect should re-consider the cost and benefit of integrating and building platforms that scale well and provide high throughput.

There are many technological solutions that enable us to provide high throughput and possibly provide a better organized interconnection network. The recent introduction of new memory technology may also result in a drastic change in the system memory hierarchy. This dissertation intends to provide several insights that shows how a system architect could build computing platforms for energy efficient datacenters. We will show how advances in packaging technology (3D stacking technology) benefit the datacenter and how recent advances in memory devices could impact the

overall organization of the system memory as well as the storage hierarchy while reducing overall system memory power consumption.

3D stacking technology has emerged as a driving force enabling new chip multiprocessor (CMP) architectures that significantly improve energy efficiency. Our proposed architecture using 3D stacking technology called PicoServer, employs 3D technology to bond one die containing several simple slow processor cores to multiple memory dies that form the primary memory. In addition, 3D stacking enables a memory to processor interconnect that is both very high bandwidth and low latency. As a result the need for complex cache hierarchies is reduced. The die area normally spent on a L2 is better spent on additional processor cores. For example, in our experiments we show that an L2 cache can be replaced by 4 extra cores. The additional cores means that they can be run slower without affecting throughput. Slower cores also allow us to reduce power dissipation and with it thermal constraints, a potential roadblock to 3D stacking.

In addition, because many server workloads require a modest amount of computation power, a large amount of their performance depends heavily on memory, I/O bandwidth along with their access latency. To mitigate I/O latency and bandwidth, especially latency in hard disk drives, server farms typically use large system memories that try to map the entire fileset onto memory, by caching the whole fileset onto DRAM. Unfortunately, as the filesets grow in size and they require a substantial amount of DRAM. Furthermore, large amounts of DRAM consumes a large portion of overall system power. Today's typical servers come with large quantities of main memory—4$\sim$32GB and have been reported to consume as much as 45W in DRAM idle power[23]. 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.

With the availability of low power memory devices like non-volatile Flash, there is an opportunity to reduce overall system power consumption while improving memory throughput. Flash has generally been found to scale well and provide more storage density due to the cell dimensions. With the wide-spread adoption of multi-level cell technology, it is expected to be more than 4 times denser than DRAM. The cost-effectiveness and low power consumption of Flash has introduced many interesting ways of leveraging this on today's datacenters. A good example would be the memory access behavior in system memory found in a web server workload. It shows an access latency of tens to hundreds of microseconds can be tolerated when accessing a large part of a disk cache without affecting throughput. This is due to the multi-threaded

nature of 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 system memory. These characteristics make a strong case for using Flash as a secondary disk cache. Their continued rapid improvement is supported by their growing usage in a wide variety of high volume commercial products. Flash consumes orders of magnitude less idle power and are cheaper than DRAM, especially NAND Flash, making them an excellent component used for energy-efficient computing.

By combining innovative packaging technologies like 3D stacking technology and innovative memory devices like Flash we will show that it is possible to cut power requirements further. They enable the following key improvements:

- **3D stacking**
  - **High bandwidth buses between system memory and L1 caches that support multiple cores—1000's of low latency connections with marginal area overhead between dies are possible.**

    Because much of the memory to processor interconnect traffic using 3D stacking technology is on-chip and able to sustain high traffic at a low cost, we are able to implement scalable wide buses with a relatively lower power budget compared to inter-chip implementations. Implementations not using 3D stacking technology rely on narrower processor to memory interfaces. High memory bandwidth is achieved by customized I/O interfaces that use complex and power hungry drivers.
  - **Modification in the memory hierarchy due to the integration of large capacity on-chip DRAM.**

    It is possible to remove the L2 cache and replace it with more processing cores. The access latency for the on-chip DRAM is also reduced because address multiplexing and off-chip I/O pad drivers [71] are not required. Recent trends in DRAM interfaces have shown that it is becoming more power consuming and time consuming to achieve high frequency for off-chip DRAM interfaces.
  - **Overall reduction in system power primarily due to the reduction in core clock frequency and integrating commonly accessed peripherals**

    The benefits of 3D stacking stated in items 1 and 2 allow us to integrate more cores clocked at a modest frequency—in our work 500-1000MHz—

on chip while providing high throughput. Reduced core clock frequency allows their architecture to be simplified; for example, by using shorter pipelines with reduced forwarding logic. Additional components could also be integrated on-chip. The interface to these components can also be simplified and implemented to be less power hungry compared to off-chip I/O interfaces.

- **Integrating Flash onto system memory**
  - **Overall cost reduction in system memory.**

    Using a hybrid system memory (DRAM + Flash) is a much more cost-effective solution than a DRAM-only based solution. The density of Flash exceeds DRAM by more than 2×, which is reflected in the reduced cost per bit of Flash. Therefore, the total cost of system memory is much less costly than a DRAM-only solution.
  - **Reducing system memory power and disk power** The physical characteristics of Flash significantly reduces idle power in system memory. The storage density of Flash enables more files to be cached on Flash than DRAM. It also results in a reduction in disk power consumption because disks are likely to spin down more often with bigger disk caches.
  - **Quicker startup time compared to conventional DRAM-only platforms.**

    The nonvolatile property in Flash implies disk cache warm up is not necessary after boot up. For database applications, that may potentially use Flash as a log file, it enables quicker recovery time when databases crash.

Of course, there are potential drawbacks in applying these solutions. The potential drawback for 3D stacking, now that the technology has been shown to be feasible, is thermal containment. However, this is not a limitation for the type of simple, low power, cores that we are proposing, as we show in section 5.3. In fact the ITRS projections of Table 2.2 predicts that systems consuming just a few watts do not even require a heat sink. The drawback of Flash is guaranteeing endurance and reliability. However, these problems have existed in disk drives for several decades and many of the techniques used in disk drives can be adopted and applied in Flash.

The dissertation is organized as follows. In the next Chapter we provide background for this dissertation by describing the current state and behavior of servers, a brief overview of 3D stacking technology and advances in memory technology. In

Chapter III, we outline our methodology for the design space exploration. In Chapter IV and V, we provide an overview and details of the PicoServer architecture. Chapter VI shows how we integrate Flash onto system memory to reduce overall system memory power. In Chapter VII, we discuss previous work explored in the areas of 3D stacking, chip multiprocessors, non-uniform memory architectures and investigating the impact of emerging memory devices. A summary and concluding remarks are given in Chapter VIII.

CHAPTER II

BACKGROUND

This chapter discusses in detail the current state of server platforms, 3D stacking technology and memory technology. We first show how servers are currently deployed in datacenters and analyze the behavior of current server workloads. Next, we explain the state of 3D stacking technology and how it is applied in this dissertation. Finally, we show the advances in memory technology. We explain the current and future trends in DRAM that is typically used as system memory. We describe the strengths and weaknesses of Flash.

2.1 Server platforms

2.1.1 3 Tier Server Architecture

Today's datacenters are commonly built following a 3 Tier Server Architecture. Figure 2.1 shows a 3 Tier server farm and how it might handle a request for service. The first tier handles the bulk of the requests from the client. Tier 1 server applications handle events on a per-client basis, which are independent and display high levels of thread level parallelism. Tier 1 servers handle web requests and forwards requests that require intensive computation or database accesses to Tier 2. Tier 2 servers that are commonly known to be computationally intensive, execute user applications that interpret script languages and primarily makes decisions on what objects (typically database objects) should be accessed. Tier 2 servers generates database requests to Tier 3 servers. Tier 3 servers receive database queries and sends the results back to Tier 2 servers.

For example, 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

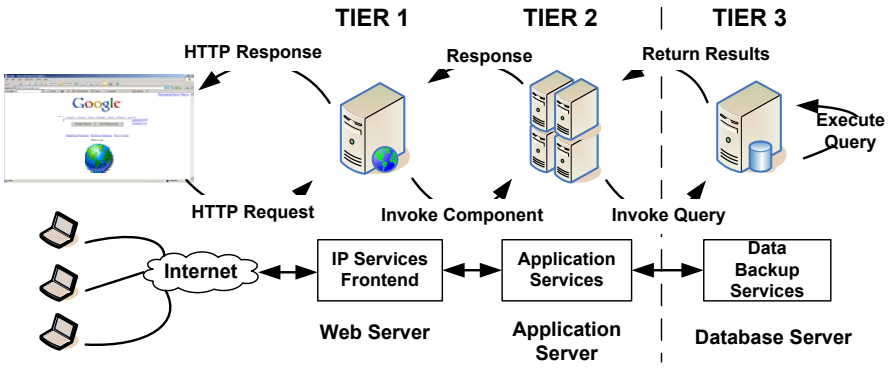


Figure 2.1: A Typical 3 Tier Server Architecture. Tier 1—Web Server, Tier 2—Application Server, Tier 3—Database Server

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.

Another excellent example of a 3 Tier Server Architecture can be found in Google's platform. Google adopts a 3 Tier Architecture to enable quick search response time. The Google Web Server (GWS) acts as a Tier 1 server that directly connects with the client. A load balancer sits in between the GWS and the client to evenly balance and effectively utilize each GWS in Google's datacenter. The GWS is responsible for forwarding client requests to Tier 2, 3 servers and later generating the HTML web page that is delivered to the client. Google's Tier 2 server called an 'Index Server' generates document IDs that are relevant to the search text submitted by the user. It is the most computationally intensive server that is essentially a decision support system in Google's datacenter. Document IDs that are generated from Google's Index Server are sent to Google's Tier 3 'Document Server' that generates document summaries by looking up the database. These results are sent back to the Tier 1 GWS which delivers the HTML result page to the client.

3 Tier Server Architectures are commonly deployed in today's server farm because it is able to optimize each distinctive workload characteristic and provide better energy efficiency by optimizing platforms to meet these characteristics.

2.1.2 Server workload characteristics

This section describes the individual workload behavior of applications commonly found in server farms. It is well-known that server workloads display a high degree of thread-level parallelism (TLP) since connection level parallelism through client

Attribute	Web99	JBOB (JBB)	TPC-C	SAP 2T	SAP 3T DB	TPC-H
Application Category	Web Server	Server Java	OLTP	ERP	ERP	DSS
Instruction Level Parallelism	low	low	low	med	low	high
Thread Level Parallelism	high	high	high	high	high	high
Instruction/Data working-set	large	large	large	med	large	large
Data Sharing	low	med	high	med	high	med
I/O Bandwidth	high (network)	low	high (disk)	med (disk)	high (disk)	med (disk)

Table 2.1: Behavior of Commercial Workloads adapted from [59]

connections can be easily translated into thread level parallelism (TLP). Table 2.1 shows the behavior of commercial server workloads. Most of the commercial workloads display high TLP and low instruction-level parallelism (ILP) with the exception of decision support systems. Conventional general-purpose processors, however, are typically optimized to exploit ILP. These workloads suffer from a high cache miss rate regularly stalling the machine. This leads to a low instructions per cycle (IPC) and poor utilization of processor resources. Our studies have shown that except for computation intensive like PHP application servers, video streaming servers and decision support systems, out-of-order processors have an IPC between $0.21 \sim 0.54$ for typical server workloads requiring modest computation even with a large L2 cache of 2MB. These workloads do not perform well because much of the requested data has been recently DMAed from the disk to main memory, invalidating cached data at that address and initiating a cache miss. Therefore, we can generally say that single-thread optimized out-of-order processors do not perform well on server workloads. Another interesting property of most server workloads including these is the vast amount of time spent in kernel code. Unlike SPECCPU benchmarks, server workloads spend an appreciable amount of time in kernel mode. This section of code is largely comprised of interrupt handling in the NIC or disk driver, packet transmission, network stack processing and disk cache processing.

Finally, 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, disk caching in the system memory plays a critical part in providing sufficient throughput. Without a disk cache, the performance degradation due to the hard disk drive latency

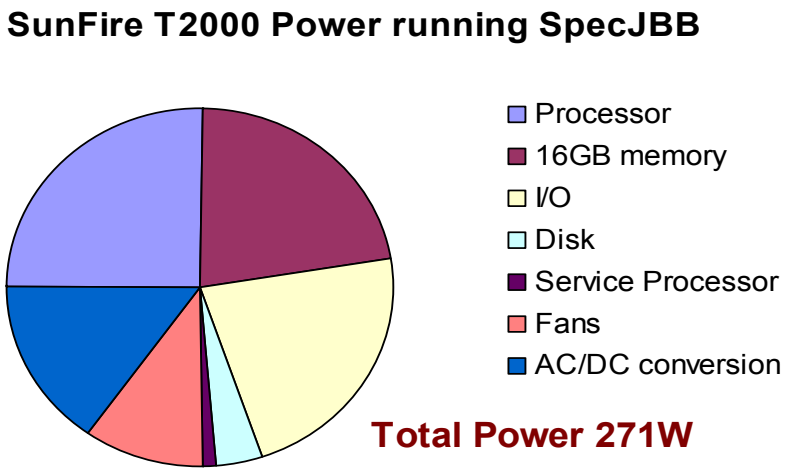


Figure 2.2: Power breakdown of T2000 UltraSPARC executing SpecJBB

would be unacceptable.

To perform well in these types of workloads an architecture must have a great deal of TLP run multiple threads as well as the large amount of processing to decode and encode the requested data. Thus a CMP or SMT architecture should be able to better utilize the processor die area.

2.1.3 Conventional Server power breakdown

Figure 2.2 shows the power breakdown of a server platform available today. This server uses a chip multiprocessor implemented with many simple in-order cores resulting in less power consumption by the processor. The power breakdown shows that 1/4 is consumed by the processor, 1/4 is consumed by the system memory, 1/4 is consumed by the power supply and 1/5 is consumed by the I/O interface. Immediately, we can see that a large amount of system memory consumes a substantial amount of power. This is expected to increase as the system memory clock frequency increases to improve memory bandwidth along with increased system memory size. We also find that despite using simpler cores that are energy efficient, a processor would still consume a noticeable amount of power due to the delivered clock frequency. The I/O interface consumes a large amount of power due to the high I/O supply voltage required in off-chip interfaces. The I/O supply voltage is likely to reduce as we scale

in the future but won't scale as much as the core supply voltage. This implies that system level integration could further reduce power. And finally, we find that the power supply displays some in-efficiency. This is due to the multiple levels of voltage it has to support and the over-provisioning of power supplies that results in less efficient power supplies.

2.2 3D stacking technology

This section provides an overview of 3D stacking technology. In the past there have been numerous efforts in academia and industry to implement 3D stacking technology[33][63][57][68][87]. They have met with mix success. This is due to the many potential challenges that need to be addressed in 3D stacking technology: 1) Achieving high yield in bonding die stacks, 2) delivering power to each stack and 3) managing thermal hotspots due to the silicon dioxide 3D interface. However, the large scale and competition typical of mobile untethered systems have re-initiated a demand for small form factors with very low power. In response, several commercial enterprizes have begun offering reliable low-cost die-to-die 3D stacking technologies.

In 3D stacking technology, dies are typically bonded as face to face or face to back. Face to face bonds provide higher die to die via density and lower area overhead than face to back bonds. The lower via density for face to back bonds attribute to through silicon vias—TSVs that punch through silicon bulk. Using the bonding techniques in 3D stacking technology, we can achieve a synergistic effect of stacking heterogeneous dies together. For example, architectures that stack conventional DRAM and logic manufactured from different process steps. Furthermore, with the added third dimension from the vertical axis, the overall wire interconnect length can be reduced and wider bus width can be achieved at lower area costs. The parasitic capacitance and resistance for 3D vias are negligible compared to global interconnect. We also note that the dimensions and pitches of 3D vias add a modest area overhead. 3D via pitches are equivalent to 22λ for $90nm$ technology, which is equivalent size of an 6T SRAM cell. They are also expected to reduce as this technology becomes mature.

The ITRS roadmap in Table 2.2 predicts deeper stacks being practical in the near future. The connections are by vias that run perpendicular to the dies. The dimensions for a 3D interconnect via varies from $1 \sim 3\mu m$ with a separation of $1 \sim 6\mu m$. Current commercial offerings can support 1,000,000 vias per cm^2 [44].

Overall yield using 3D stacking technology is a product of the yield of each individual die layer. Therefore, much effort must be put in achieving high yield per die.

	2005	2007	2009	2011	2013
Low-cost/handheld #die/stack	6	7	9	11	13
SRAM density Mbits/cm^2	84	138	225	365	589
DRAM density Mbits/cm^2 at production	1,220	1,940	3,660	5,820	9,230
NAND Flash density (SLC/MLC) Mbits/cm^2 at production	2,714/ 5,706	4,835/ 9,243	6,913/ 13,568	10,745/ 41,635	17,306/ 69,225
Max. Power Budget for cost-performance systems(W)	91	104	116	119	137
Max. Power Budget for low-cost/handheld systems with battery(W)	2.8	3.0	3.0	3.0	3.0

Table 2.2: ITRS projection [27] for 3D stacking technology, memory array cells and maximum power budget for power aware platforms. ITRS projections suggest DRAM density exceeds SRAM density by $15 \sim 18\times$ entailing large capacity of DRAM can be integrated on-chip using 3D stacking technology as compared to SRAM.

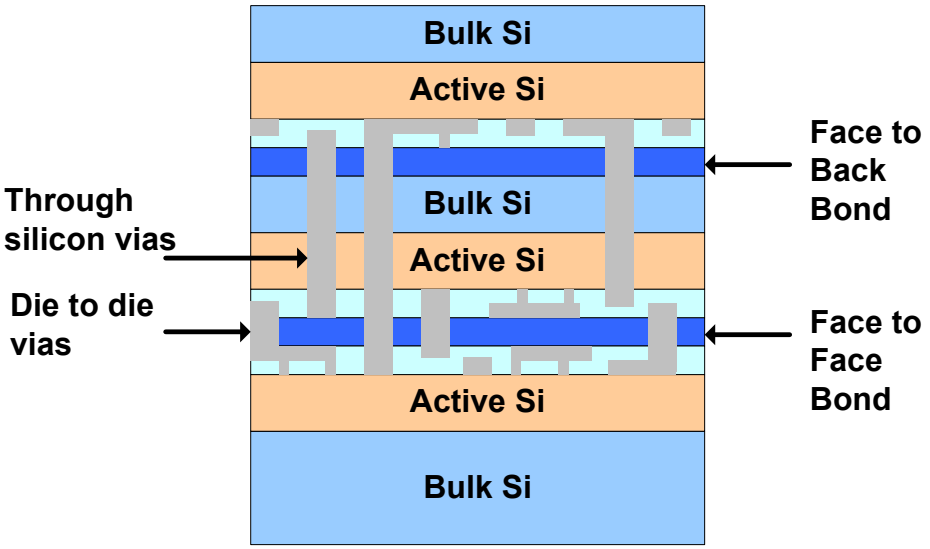


Figure 2.3: Example of a 3 layer 3D IC

	Face-to-Back	Face-to-Face	RPI	MIT 3D FPGA
Size	$1.2\mu \times 1.2\mu$	$1.7\mu \times 1.7\mu$	$2\mu \times 2\mu$	$1\mu \times 1\mu$
Minimum Pitch	$< 4\mu$	2.4μ	N/A	N/A
Feed Through Capacitance	$2 \sim 3$fF	≈ 0	N/A	2.7fF
Series Resistance	$< 0.35\Omega$	≈ 0	≈ 0	≈ 0

Table 2.3: 3D stacking technology parameters[44][29][68]

This implies logic to memory stacking is a better choice than logic to logic stacking. Memory devices are known to produce higher yield through many well-known techniques. For example, refusing extra bitlines to compensate for defect cells, applying single bit error correction logic to memory to compensate poor yield. Several studies including [75] claim that DRAM yields are extremely high suggesting chips built with a single logic layer and several DRAM layers generate yield close to the logic die.

2.3 Memory Technology

This section discusses the technological advances in memory technology. Advances in memory technology potentially produces drastic changes in the overall platform architecture. In particular, we discuss advances in DRAM and Flash memory technology in the next subsections.

2.3.1 DRAM

DRAM technology has improved side by side with logic technology. DRAM today is offered in numerous types, often determined by the application space. For typical desktop and server platforms, DDR2 DRAM has emerged as the primary solution for system memory. FBDIMM DRAM that delivers higher throughput than DDR2 is also emerging as a alternative solution. For graphics workloads, XDRAM, GDDR3

	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	15μs	200μs	1.5ms	Yes

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

Table 2.4: Cost and power consumption for conventional DRAM, NOR, NAND Flash memory. NAND Flash is the most cost-effective while consuming the least amount of power.[78][62]

	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 program/erase 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 2.5: ITRS 2005 roadmap for Flash memory technology. NAND Flash 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.[27]

DRAM that can provide ultra high bandwidth are suitable solutions in this domain. Finally, RLDRAM, NetRAM[3][18] that is appropriate for network workloads is gaining strong acceptance in network applications as well as being used as an L3 cache. Like its logic counterpart, DRAM technology will likely meet steep challenges in the nanometer regime. However, they are expected to be resolved through enhancements in process technology and the adoption of new transistor types like double gate transistors(FinFETs).

Looking ahead, with Flash memory emerging as a niche market for memory, DRAM is likely to be 1∼2 generations behind Flash memory in terms of process technology. Supply voltage is likely to scale with process technology and fitting the pass gate transistor in DRAM will always remain a challenge. Recently, an alternative DRAM cell structure using SOI technology has received much attention. SOI DRAM[47] uses the floating body effect in SOI to store the state of memory. SOI DRAM is an appealing technology since 1) it can be built on a logic process and 2) the cell dimensions are known to be smaller than than conventional DRAM cells. SOI DRAM has shown strong promise in the IP block sector with companies like AMD announcing to license this technology and use it in their future processors.

In summary, DRAM is still likely to scale and still has a strong case for achieving high storage capacity at relatively acceptable access latency. The following subsections will provide an overview of current and near-future offerings of DRAM.

DDR2 DRAM

Double Data Rate 2 DRAM (DDR2 DRAM) improves DRAM bandwidth by transferring data at the positive and negative edge of a system clock. It has been widely adopted in common desktop platforms. It has reached a point where the data rate has

reached GHz frequencies and raises issues with implementing high frequency off-chip I/O interfaces.

XDR DRAM

XDR DRAM delivers ultra high bandwidth by implementing an aggressive analog based high speed interface. It competes directly with DDR2 and other derivatives of SDRAM. It uses a sensitive differential driver that is able to detect 200mV. This driver enable transfer rates of several GHz. These high speed analog interfaces are known to consume more power.

Fully Buffered DIMM (FB-DIMM) DRAM

FB-DIMM DRAM is a DRAM solution that enables DRAM to scale in bandwidth and size while maintaining an acceptable signal integrity. Conventional memory controllers directly connect to every DRAM module. As we increase the DRAM memory width and the number of memory modules, the quality of the signal degrades substantially limiting the speed and the memory density. With FB-DIMM DRAM, the memory controller does not connect to the memory module directly. Instead, the memory controller connects to the Advanced Memory Buffer (AMB) with a serial interface. Each memory module has a AMB that is either connected to another AMB from another memory module or connected to the memory controller. The AMB can compensate signal deterioration by buffering or resending the signal to the next AMB or memory controller. The drawback however, is an increase in memory access latency and high power consumption. Some applications may display dramatic performance degradation due to the increased memory access latency.

RLDRAM

Reduced Latency DRAM (RLDRAM) reduces the random access latency of DRAM by using a large number of banks and early writebacks to rows[3][18]. The large number of banks enables bank interleaving reducing the wait time to access DRAM. Row writebacks occur immediately after a DRAM read. DRAM reads are destructive and must be accompanied with writeback. Other types of DRAM only writeback when a row is closed. Rows are commonly closed when trying to access another DRAM row. RLDRAM immediately performs a row writeback even when the current row is being accessed. There is a power overhead in applying this aggressive writeback scheme and an area overhead by using large number of banks. The area overhead

of RLDRAM is known to be approximately 10%. RLDRAM is commonly found in routers and switches. It is also considered as a L3 cache in enterprize platforms.

Z-RAM

Z-RAM is a new DRAM cell promoted by Innovative Silicon Technologies [47] [41] [76]. It is built on top of a SOI silicon wafer and takes advantage of the Floating Body effect in SOI. Theoretically, Z-RAM cells have a smaller cell dimension than conventional DRAM cells but have not yet been implemented with this density. Innovative Silicon currently offers them with a storage density of 400Mbits/mm^2 at 65nm process technology[47]. The access latency to Z-RAM is found to be of several nanoseconds. It is a promising technology and will add more density than SRAM. DRAM vendors are also introducing similar DRAM cells and call it Floating Body Gate RAM (FBRAM) at the research level. Successful implementations have been presented in with similar characteristics [79] [75].

2.3.2 Flash memory

Flash memory has emerged as a storage device for mobile and embedded systems. It has also recently been introduced as a way to mitigate disk drive latency. Hybrid disk drives and Robson technology are a good example in showing the potential of Flash as solid state storage devices. 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 2.4 and Table 2.5 summarizes the properties of NAND and NOR along with DRAM. The biggest difference, compared to DRAM, is when writing to a Flash. Flash memories require a preceding erase operation to perform a write operation. The ITRS roadmap projects single-level cell NAND Flash 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. With the potential to support for multi-level cells, storage density is expected to improve even more as shown in the ITRS roadmap. The following subsections explain in detail the types of Flash memory currently available and discuss the wear-out behavior of Flash.

Flash memory type

NOR NOR Flash memory was introduced early in the 1990's. It was primarily used to replace Read Only Memory (ROM) and intended to store critical code in a platform.

The memory is organized in a NOR like structure which enables relatively low read latencies. However, due to the organization, writes and erases take a long time. The read, write and erase latencies make a NOR Flash useful to directly executed code for embedded battery power platforms. It is commonly found in cell phones that typically require a tight power constraint to support long battery operation time. A NOR Flash has a random read access time that is as fast as SDRAM. Since NOR Flash performs relatively well for random read accesses, it is commonly used for applications requiring fast access to code and data. Accordingly, NOR Flash has typically been used to store code and data on handheld devices, PDAs, laptops, cell phones, etc.

NAND As Flash emerged as a storage device, much effort was invested in improving storage density. Due to the compactness of a NAND structure, NAND Flash started to be widely adopted by trading in performance for storage density. For sequential digital content NAND Flash was an attractive solution and became widely deployed with the proliferation of digital media. Compared to a NOR Flash, a random read latency in a NAND Flash takes 10's of microseconds. However, the write and erase latency is less than a NOR Flash. NAND Flashes were originally built using single-level cell technology. With the ability to implement multi-level cells with acceptable reliability, NAND Flash is able to deliver several GBytes storage today at a low cost. In addition to lengthy random access time, NAND Flash also has issues with reliability. NAND Flash is likely to be manufactured with faulty cells. Furthermore, wear-out may cause good cells to become bad cells. NAND Flash 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[24].

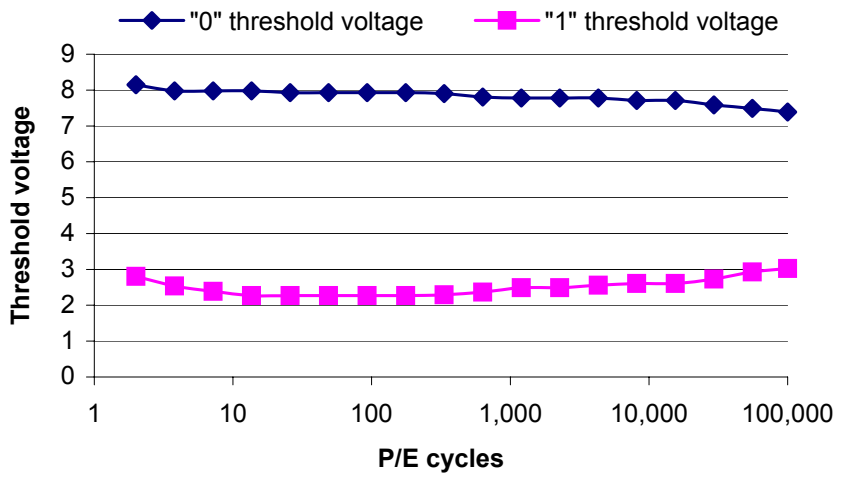
A page in a NAND Flash is typically 2KB in size and used as a basic unit in reading and writing. A block in a NAND Flash is composed of multiple pages—typically 128 to 256 pages. The size of a block varies based on the cell type, where MLC Flash has 256 pages per block whereas SLC Flash has 128 pages.

OneNAND To mitigate the lengthy random read latency in a NAND Flash, many vendors came up with solutions integrating SRAM on-die with Flash to improve overall throughput and average access latency. Samsung's OneNAND is a representative solution for this effort. OneNAND uses a large SRAM cache to temporarily store multiple Flash pages. This SRAM enables NAND Flash to reach access latencies similar to or better than NOR Flash for many code intensive applications.

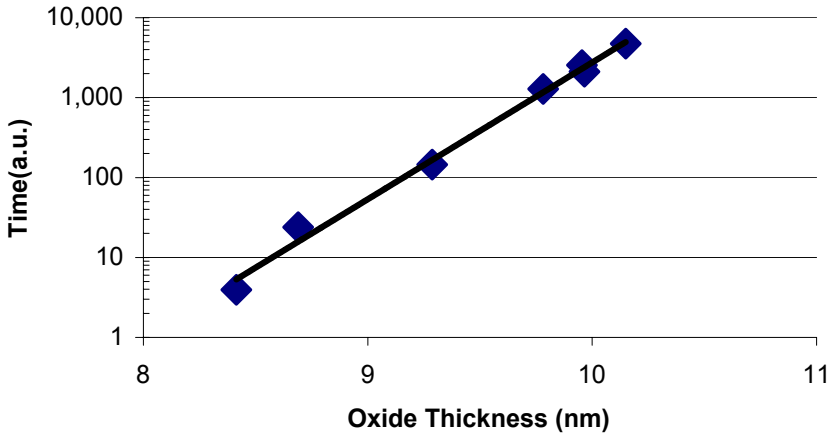
Flash wear-out behavior

Flash wear-out has remained a problem since it was introduced in the early 90's. Wear-outs occur because of program (write) and erase activities to Flash. The error behavior of Flash cells due to wear-out has been well understood and published in several papers. [53] compares a simplified analytical model with empirical results. [73] presents empirical studies on the deterioration of Flash memory cells with program and erase cycles. They have shown empirically that the primary cause of Flash memory wear-out is due to the gradual increase in electrons getting trapped in the floating gate. Figure 2.4(a) shows the change in threshold voltage as we increase the number of program and erase cycles. Due to the gradual increase in trapped electrons, the threshold voltage achieved after erase and program shifts to a different voltage level and as a result the threshold voltage window gap decreases. The decrease in this gap makes it difficult for the sense amplifier to differentiate between memory states.

[73] has stated that the degree of decrease in the threshold window gap is dependent on the oxide thickness of the floating gate. Further, the 100,000 cycles guaranteed by the manufacturers comes from the endurance a Flash memory cell with minimum oxide thickness can achieve. Figure 2.4 (b) shows the relationship between oxide thickness and wear-out. The y-axis shows the amount of time it takes to achieve a certain tailed threshold voltage distribution which directly correlates to wear-out when applying constant program and erase operations. The amount of time can be viewed as the number of program and erase cycles. It shows an exponential relationship with oxide thickness and wear-out. Thus, improving the quality and thickness of oxide directly results in improving Flash endurance.



(a)



(b)

Figure 2.4: (a) Flash threshold voltage behavior as Flash wears out (b) Flash wear-out behavior for varying oxide thickness [73]

CHAPTER III

METHODOLOGY

This chapter explains the methodology applied in modeling and simulating our findings in this dissertation. We relied on a full system simulator to evaluate throughput and analytical and empirical results generated from well-known models, publications and CAD tools for estimating power consumption and die area. They were derived from data published by industry and academia [27][44][82][4][29][33]. Specifically, timing and power for DRAM and Flash were obtained from IBM and Micron technology datasheets [11]. Core, peripheral power is estimated using the well-known cubic law and publications of real implementations. The architectural aspects of our studies were obtained from a full system simulator called M5 [32] that is able to run Linux and evaluate full system-level performance. A server connected to multiple clients is modeled. The client requests are generated from user level network applications. A detailed description of our methodology is described in the following subsections.

	OO4-(small/large) baseline/ w. 3D stacking	Conventional CMP MP(4/8/12) w.o. 3D stacking	PicoMP(4/8/12)-(500MHz/1000MHz)*
Clock Frequency	4GHz	1GHz	500MHz/1GHz
Number of Processors	1	4/8	4/8/12
Processor Type	out-of-order	in-order	in-order
Issue Width	4	1	1
L1 cache	2 way 16/128KB	4 way 16KB per core	4 way 16KB per core
L2 cache	8 way 256KB/2MB 7.5ns unloaded latency	8 way 2MB 16ns unloaded latency	N/A
Memory Bus Width	64bit at 400MHz/ 1024bit at 250MHz	64bit at 250MHz	1024bit at 250MHz
System Memory	512MB DDR2	512MB DDR2	128/192/256MB DRAM

* PicoServer platform using 3D stacking technology. The core clock frequency of PicoServer is typically 500MHz. PicoServer configurations with 1GHz core clock frequency are later used to show the impact of 3D stacking technology.

Table 3.1: Commonly used simulation configurations. System memory latencies are generated from DDR2 DRAM models. L2 cache unloaded latency for single core and multicore configurations differ due to longer global interconnect lengths in multicore platforms[60].

	Server configuration parameters
NAND Flash Memory	256MB/512MB/1GB/2GB/4GB fully associative 128KB logical block size random read latency 25μs write latency 200μs erase latency 1.5ms bandwidth 50MB/s
IDE disk	average access latency 6ms bandwidth 300MB/s

Table 3.2: Flash memory and hard disk drive configurations in our studies.

3.1 Simulation Studies

3.1.1 Full system architectural simulator

To evaluate the performance of our platform, we used the M5 full system simulator. M5 boots an unmodified Linux kernel on a configurable architecture[32]. Multiple systems are created in the simulator to model the clients and server, and connected via an ethernet link model. Peripheral devices (Flash, network interface card) that exist on these systems can also be modeled in M5. The server side executes a server application. The client side executes benchmarks that generate representative requests for Tier 1, 2, 3 servers respectively. For comparison purposes we defined a Pentium 4-like class system [84], and a chip multiprocessor-like system similar to [56]. We also looked at configurations using 3D stacking technology on these platforms to investigate in general the usefulness of this technology. Table 3.1 and Table 3.2 shows commonly used configurations in our simulations.

3.1.2 Server Benchmarks

We use several benchmarks that directly interact with client requests. We used two web content handling benchmarks SURGE[30] and SPECweb99[22] to measure web server performance. Both benchmarks request filesets of more than a 1GB. A web script handling benchmark SPECweb2005[21] using PHP is selected to represent script workloads. A video streaming benchmark—Fenice[17] that uses the RTSP protocol along with the UDP protocol is chosen to measure behavior for on-demand workloads. For a file sharing benchmark we use an NFS server and stressed it with dbench. Finally, we executed two database benchmarks to measure database performance for Tier 2, 3 workloads.

SURGE The SURGE benchmark represents client requests for static web content. 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 12KB in file size is requested 50% of the time in our client requests. We configured the SURGE client to have 20 outstanding client requests. It has been shown in [37] that the number of requests handled per second is consistent after 10 concurrent connections implying 20 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 client request of SPECweb99 to have 20 outstanding requests with the same principle applied from [37].

SPECweb2005 Scripting languages are a popular way to describe web pages. SPECweb2005 offers 3 types of benchmarks. A Banking benchmark that emulates the online banking activity of a user. A E-commerce benchmark that emulates the online purchase activity. A Support benchmark that emulates the online stream activity. All benchmarks require a dynamic web page to be generated from a script interpreter. We use a PHP interpreter to measure the behavior of Tier 2 Servers. We used the simplest version of the Zend optimizing cache[26] that caches interpreted PHP bytecode. The client requests are generated from methods described for SPECweb99 and SURGE clients.

Fenice On-demand video serving is also an important workload for Tier 1 servers. For copyright protection and live broadcasts, the RTSP protocol is commonly used for real-time video playback. Fenice is an open source streaming project [17] that provides workloads supporting the RTSP protocol. We modified it to support multithreading. Client requests were generated with a modified version of 'nemesi', a RTSP supporting MPEG player. 'nemesi' is also from the open source streaming project. Unlike event driven HTTP requests seen in SPECweb99 or SURGE, video streaming servers require timing driven real-time packets to be constantly scheduled such that MPEG video frames are delivered precisely to the client side. Therefore, it is reasonable to assume the number of concurrent connections to be equal to the number of requests handled. We generated multiple client requests that fully utilized the server CPUs for a high quality 16Mbps frame rate, 720×480 resolution MPEG2 standard file.

dbench This benchmark is commonly used to stress NFS daemons. In our tests we used the in-kernel NFS daemon which is multithreaded and available in standard Linux kernels. We generated NFS traffic using dbench on the client side that stressed the file server. dbench generates workloads that both read and write to the file server while locking these files so that a different client could not access it simultaneously.

OLTP On-line transaction processing is a typical workload executed on Tier 2, 3 Servers. The TPC council has described in detail benchmarks for OLTP. We have used a modified version of TPC-C made available by the Open Source Development Lab—OSDL— called DBT2 [14]. DBT2 generates transaction orders. Our database server is MySQL 5.0. MySQL is a widely adopted DB Server. We use the InnoDB storage engine that is known to support transactions and provide a reasonable amount of scalability with multicores. We generated a 1GB warehouse which is commonly

used for small-scale computation intensive databases. We chose a small working-set size due to the limitations in simulation. We selected a buffer pool size accordingly. The query cache is disabled to prevent speedup in query time due to caching.

DSS Decision support is another typical workload used to evaluate Tier 2,3 Servers. We used TPC-H, the current version of a DSS workload. Again a modified version of TPC-H available by OSDL (DBT3) [14] is used in this study. We loaded the TPC-H database onto mySQL and used the defined TPC-H queries to measure performance. The query cache is disabled to prevent speedup in query time due to caching. To reduce our simulation time to a reasonable amount, we only performed multiple Q22 queries out of the many TPC-H queries and measured query time.

3.1.3 Server Disk Traces

Much of our simulations can only capture a small fraction in time. The M5 simulator typically is able to only process 20 seconds of execution time for 1 days simulation. Because many of our simulation benchmarks displayed repetitive behavior, we believe our throughput results were representative. However, when it came to long term disk accesses, this would not be the case. 20 seconds in simulation time may not be representative in disk access behavior because disk accesses occur very infrequently. We overcame this problem by reducing the disk access latency but still found it difficult to generate enough disk accesses. Therefore, we had to resort to publicly available disk traces for server workloads. We found these workloads to be useful when comparing our disk access patterns on M5 with these traces. These disk traces were used extensively when conducting system memory and storage hierarchy analysis. These disk trace are shown as follows:

Microbenchmarks Microbenchmarks are a quick and effective way in understanding how an architecture would behave in different usage models. In many instances, real world benchmarks can later be identified to behave similar to a microbenchmark. We generated I/O traces for microbenchmarks that access files following an 1) exponential access probability distribution function, 2) zipf access probability distribution function and 3) uniform probability distribution function.

WebSearch Three I/O traces from several web search engines. These web search traces are strictly read-only due to the nature of the workload itself. They were measured for intervals of 50 minutes, 4 hours and 3.5 days. The UMass storage repository provides these traces that profile time, logical block address, request size read or write.

Financial Two I/O traces from different Financial institutes. These traces show a notable amount of disk write behavior. They were measured for approximately 0.5 days. They are provided in a similar file format found in WebSearch.

3.2 Modeling the system level components—estimating timing, power and area

Timing, power and die area estimation at the architectural level is difficult to do with great accuracy. To make a reasonable estimation and show general trends, we resorted to industry and academia publications on die area and we compared our initial analytical power models with real implementations and widely used estimates generated from CAD tools. We discuss how each component is modeled in the next subsections.

3.2.1 Processors

We relied to a large extent on figures reported in [36][2][83] for an ARM processor to estimate processor power and die area. The ARM is representative of a simple in-order 32 bit processor. We extrapolated the die area and power consumption for our cores at 500MHz from published data in [36][2][83][15][12][9]. For power consumption at other core clock frequencies, for example 1GHz, we generated a power versus frequency plot by calibrating the well-known cubic law [42] and frequency, voltage plot for 24 FO4—fan out of 4 inverter chain using PTM 90nm process technology. We assumed the logic depth of our PicoServer core to be 24FO4 from [83]. Figure 3.1 shows our plot.

64 bit support for a core used for servers seems inevitable in the future for address sizes above 4GB. We expect the additional area and power overhead for 64 bit support in a core to be modest when we look at the additional area and power overhead for 64 bit support in commercially available cores like MIPS and Xeon. As for the L2 cache, because we noticed numerous problems in power, area models in nanometer technology for caches, we referred to [85] and scaled the area, power number generated from real measurements. We assumed the power numbers in [85] were generated when the cache access rate was 100%. Therefore, we scaled the L2 cache power by size and access rate while assuming leakage power would consume 30% of the total L2 cache power.

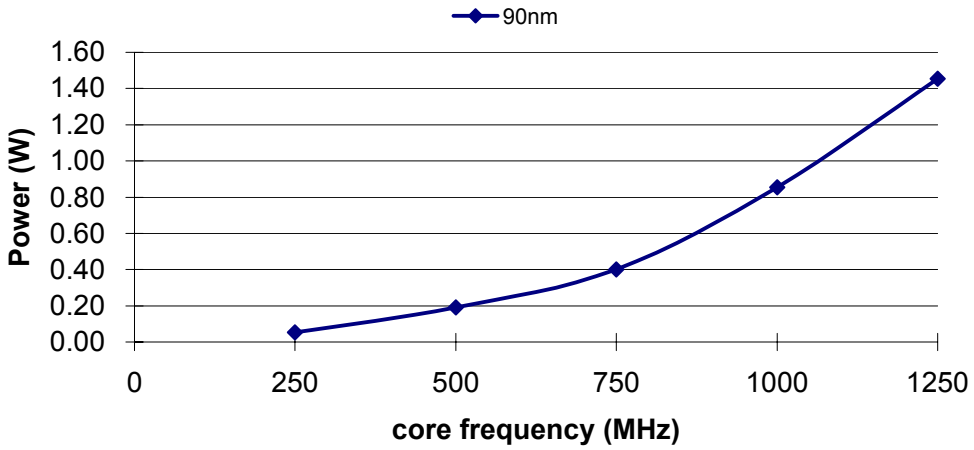


Figure 3.1: Processor power versus frequency plot generated from calibrating the well-known cubic law and voltage, frequency plot for 24FO4 using PTM 90nm process technology[16]

3.2.2 Interconnect considering 3D stacking technology

For the purposes of this study, we have adopted the data published in [27][44][82] as typical of 3D stacking interconnects. In general, we found wafer to wafer—3D via, interconnect capacitance to be below 3fF. We also verified this with extracted parasitic capacitance values from 3D Magic, a tool recently developed at MIT. The extracted capacitance was found to be 2.7fF, which agrees with the results presented in [44]. By comparison with a 2D on-chip interconnect, based on [49], a global interconnect wire was estimated to have capacitance of 400fF per millimeter. Therefore, we can assume that the additional interconnect capacitance in 3D stacking vias are negligible. As for the number of connections that are possible between dies a figure of 10,000 connects per square millimeter is reported. Our needs are much less. From our studies, we just need 1056 connections: 32 bits for our address bus and 1024 bits for the data bus. For estimating the interconnect capacitance on our processor and peripheral layer, we referred to [49] to generate analytical and projected values. We selected a wire length of 12mm to account for 1.3 times the width/height of a 80mm^2 die and scaled the wire length accordingly for smaller die sizes. We assumed we would gain a 33% reduction in wire capacitance compared to a 2D on-chip implementation from projections on interconnect wire length reduction shown in [38]. Based on these initial values, we calculated the number of repeaters required to drive the interconnect range from 250 $\sim$ 400MHz from hspice simulations. We found we needed only a

maximum of 2 $\sim$ 3 repeaters to drive this bus since the frequency of this on-chip wide bus was relatively slow. We measured the toggle rate and access rate of these wires and calculated power using the well-known dynamic power equation to calculate interconnect power.

3.2.3 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 [19][11] to estimate timing and power consumption. For die area estimation, we used published data found in [48][69] and projections found in the executive summary of the ITRS roadmap [27]. We discuss this further in the next subsections.

DRAM

The timing model for DRAM is generated from the Micron datasheets in [10]. 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 [11]. 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 [23]. For die area estimation, we used numbers generated from industry products found in [69].

We made DRAM area estimates for the PicoServer using the data in [69]. Currently, it is reasonable to say that 80mm^2 of chip area is required for 64MB of DRAM in 90nm technology.

Conventional DRAM is packaged separately from the processor and is accessed through I/O pad pins and wires on a PCB. However, for our architecture, DRAM exists on-chip and connects to the processor and peripheral through a 3D stacking via. Therefore, the pad power consumed by the packages, necessary for driving signals off-chip across the PCB, should not be added in our design. Using the Micron DRAM spreadsheet calculator[11], modified to not include pad power, and profile data from M5 including the number of cycles spent on DRAM reads, writes and page hit rates, we generated an average power for DRAM. We compared the estimated power from references on DRAM and especially with the DRAM power values generated from the

SunFire T2000 Server Power Calculator[23]. The Micron spreadsheet uses actual current measurements for each DRAM operation—read, write, refresh, bank precharge etc. We assumed a design with a 1.8V voltage supply.

Flash memory

To understand the timing and power model for NAND Flash, we used several publications found in [19]. We assumed a multiple 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 [48] and we performed comparisons on similar process technologies. Flash memory has begun to outpace DRAM in process technology resulting in Flash to be 1 generation ahead of DRAM. To estimate the power consumption of Flash, 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.

3.2.4 Modeling the disk drive

Disk drives are off-chip components that are connected using cables. The power consumption of a disk drive can be measured empirically and modeled onto a simulator. This method is well-known and prior work [45] has used it to show it agreed with real measurements. We relied on datasheets commonly found in [20] to estimate disk drive power at idle, active and standby mode and measured time spent in idle, active and standby mode when simulating a platform on M5.

3.2.5 Network Interface Controller—NIC

Network Interface Controller power was difficult to model analytically due to lack of information on the architecture of NICs. For our simulations, we looked at the National Semiconductor 82830 gigabit ethernet controller. This chip implements the MAC layer of the ethernet card and interfaces with the physical layer—PHY using the Gigabit Media Independent Interface—GMII interface. For power, we analyzed the datasheet and found the maximum power consumed by this chip to be 743mW[13]. This power number is for 180nm technology. Because much of our work is based on 90nm process technology, we had to estimate power for 90nm. Fortunately, [1] datasheet listed power values for 90nm technology. We used this value to estimate NIC

power. Although, our functional simulations are based on a different NIC, we believe [1] would generate similar throughput. We assumed maximum power is consumed when all the input and output pins were active. By doing so, we calculated the number of bytes written and read from the chip and normalized it to the maximum case. We assumed static power consumed 30% of the maximum chip power.

When estimating die area, we synthesized an open source version of a gigabit ethernet controller. Our synthesized results using physical compiler showed that 1.2mm^2 at 0.13μm process technology. We compared these results to scaled numbers from [66] and found them to agree.

CHAPTER IV

OVERVIEW OF A PICOSERVER ARCHITECTURE

This chapter and the next chapter provides the architecture of a system using 3D stacking technology. We will discuss in detail how 3D stacking can be leveraged to implement extreme levels of integration and how the vast amount of throughput between the processor and much of the memory and the I/O devices can produce an energy efficient architecture. Our resulting architecture is called PicoServer.

Figure 4.1 shows the overall architecture of a platform leveraging 3D stacking technology. For the purpose of our study we called this platform PicoServer. It is composed of a single logic die stacked to 4, 8 layers of system memory. Our PicoServer architecture uses face to face bonds for the logic die to memory die and face to back bonds for memory to memory die stacking. The 3D vias function as interconnect and thermal pipes. For our studies, we assume that the logic-based components—the microprocessor cores, the network interface controllers (NICs), disk controllers and peripherals—are on the bottom layer and conventional capacity-oriented DRAMs. We are conservative in what we assume about stacking depth. For the purposes of this study we assume a stack of 5, 9 dies. To understand the design space and potential benefits of this new technology, we explored the trade-offs of different bus widths, number of cores, frequencies, and the memory hierarchy in our simulations.

	SDRAM	DDR2 DRAM	XDR DRAM	L2 Cache at 1.2GHz	On-chip DRAM 3D IC
Bandwidth (GB/sec)	1.0	5.2	31.3	21.9	31.3
Average access latency(ns)	30ns	25ns	28ns	16ns	25ns

Table 4.1: Bandwidth and latency suggest on-chip DRAM can easily provide enough memory bandwidth compared to an L2 cache noted in [60][85]. Average access latency for DRAM is estimated to be t_{RCD}+t_{CAS} where t_{RCD} denotes RAS to CAS delay and t_{CAS} denotes CAS delay. For, XDRAM t_{RAC-R} is used where t_{RAC-R} denotes the read access time.

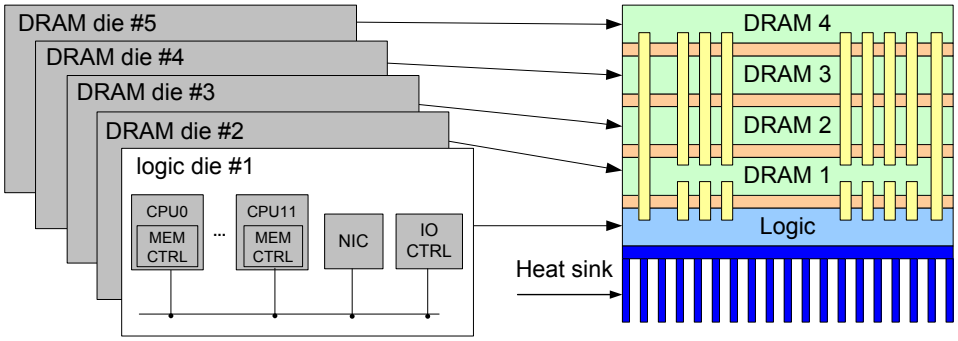


Figure 4.1: A diagram depicting the PicoServer: a CMP architecture connected to a conventional DRAM using 3D stacking technology with an on-chip NIC to provide low-latency high-bandwidth networking.

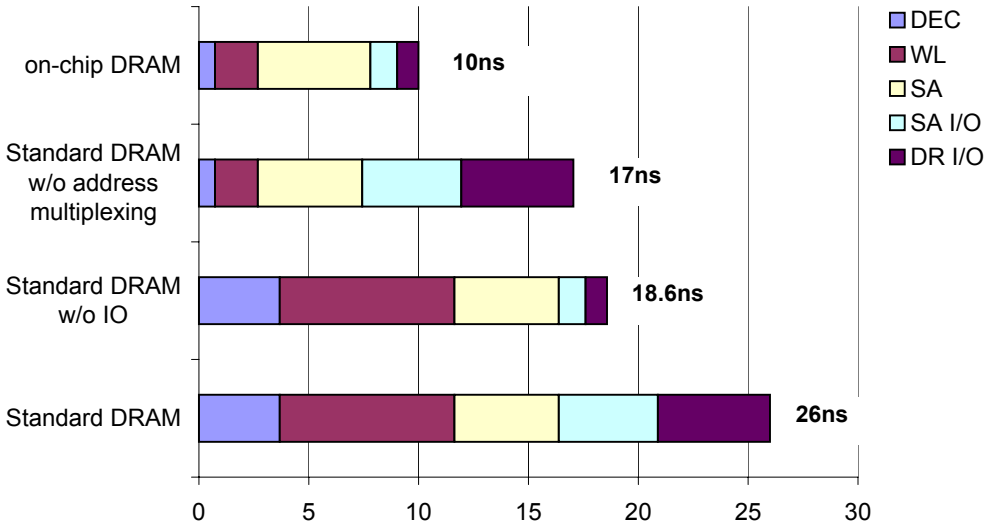


Figure 4.2: Breakdown in DRAM latency for DEC:decode, WL:wordline, SA:sense amplifier, SA I/O:sense amplifier I/O and DR I/O: driver I/O. Clearly shows a reduction in DRAM latency bringing DRAM on-chip.

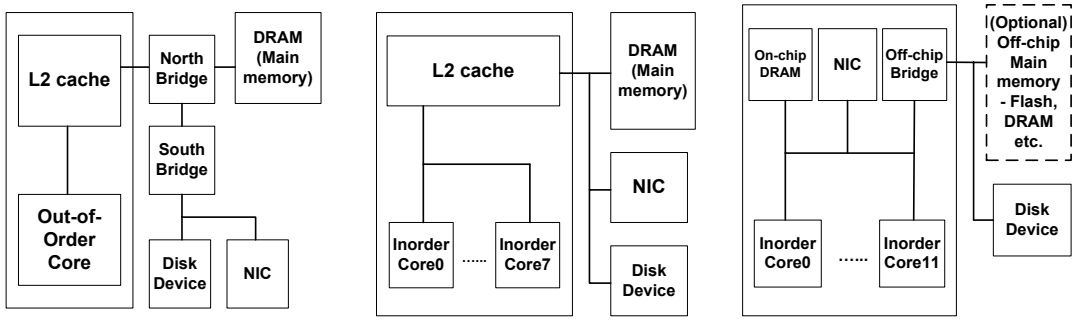


Figure 4.3: Block diagram of two conventional platforms and PicoServer. (a) general purpose processor platform, (b) conventional CMP platform without 3D stacking, (c) PicoServer platform using 3D stacking

We found bus widths of 1024 bits with a latency of 2 clock cycles at 250 MHz to be reasonable in our architecture. In addition, we aim for a reasonable area budget constraining the die size area to be below 80mm^2 at 90nm process technology. Our 12 core PicoServer configuration which occupies the largest die area is conservatively estimated to be approximately 80mm^2. The die areas for our 4, 8 core PicoServer configurations are respectively 40mm^2 and 60mm^2.

To understand the impact of bringing DRAM on-chip, Table 4.1 shows the latency and bandwidth achieved for conventional DRAM, XDR DRAM, L2 cache and on-chip DRAM using 3D stacking technology. Conventional DRAM and high throughput off-chip DRAM uses high frequency to deliver bandwidth due to the limited amount of externals pins available off-chip. But with 3D stacking using a 1024 bit wide bus, the memory latency and bandwidth achieved in an on-chip DRAM can be comparable to an L2 cache and XDR DRAM. This is primarily due to the wide bus width and additional performance optimizing efforts that could be applied when removing the die area invested in I/O related circuitry. This suggests an L2 cache is not needed if stacking is used. Furthermore, the removal of off-chip drivers in conventional DRAM reduces access latency by more than 50% [71] shown in Figure 4.2. This strengthens our argument that on-chip DRAM can be as effective as an L2 cache. Therefore, our PicoServer architecture does not have an L2 cache and the on-chip DRAM is connected through a shared bus architecture to the L1 caches of each core. The role of this on-chip DRAM is a primary system memory.

The commonly held belief of memory walls existing in DRAM, is a result of DRAM vendors making more effort in delivering high bandwidth and high storage

density while neglecting to implement architectural techniques that reduce overall access latency. We note that the access latency to DRAMs could be reduced when applying well-known techniques in SRAMs that reduce latency. One example is the reduced access latency found in RLDRAMs. As we have discussed in Chapter II, it uses well-known architectural techniques such as subbanking to reduce the overall access latency.

The PicoServer architecture is comprised of single issue in-order processors that together create a chip multiprocessor which is a natural match to applications with a high level of TLP [56]. Each PicoServer CPU core is typically clocked at 500MHz and has an instruction and data cache, with the data caches using a MESI cache coherence protocol. Our studies showed the majority of bus traffic is generated from cache miss traffic, not cache coherence. This is due to the properties of the target application space and the small cache—16KB size per core. With current densities, the capacity of the on-chip memory stack in PicoServer is between hundreds of megabytes to several gigabytes. In the near future this will rise to tens of gigabytes as noted in the Table 2.2. Other components such as the Network Interface Controller (NIC), DMA controller, and additional peripherals that are required in implementing a full system are integrated on the CPU die.

The next chapter discusses in detail the architecture of PicoServer. It provides data supporting our design decisions.

CHAPTER V

DETAILED DESCRIPTION OF PICOSERVER

This chapter describes in detail, the rational in architecting PicoServer which leverages 3D stacking technology. Our design space exploration was conducted using the M5 simulation environment. We first describe the architectural details of the logic components and later describe the on-chip memory architecture.

5.1 Logic architecture of PicoServer

There are many logic components in PicoServer. The processing cores that are a critical element in processing client requests and execute the operating system, I/O peripherals like the Network Interface Card (NIC), Disk Controller that are critical in delivering the content to system memory and the interconnect network that must be able to handle the interactions between the cores, peripherals and memory. We discuss them in the next subsections.

5.1.1 Using simple cores

PicoServer is made up of simple scalar in-order cores. A 32bit machine is assumed for each core. Although, we expect it to increase to a 64bit machine in the near

Workload	Branch Prediction Rate
SURGE	98.70%
SpecWeb99	97.06%
fenice	99.42%
dbench	97.92%
SpecWeb2005-bank	95.13%
SpecWeb2005-ecommerce	95.04%
SpecWeb2005-support	95.81%
dbt2-TPCC	96.16%
dbt3-TPCH	96.71%

Table 5.1: Branch Prediction Rates for various server workloads

	L1 cache	L2 cache	clock frequency	total power	total die area(mm^2)
Pentium 4 90nm	16KB	1MB	3.4GHz	89-103W	112
ARM11 MP core 90nm	16KB	N/A	620MHz	267mW	2-3
Xscale 90nm	32KB	N/A	1.5GHz	850mW	6-7*
MIPS32 34K 90nm†	16KB	N/A	500MHz	310mW	2-3
PowerPC405 90nm	16KB	N/A	400MHz	76mW	2
Leon3 MP core 130nm‡	16KB	N/A	400MHz	N/A	8-9
OpenRISC 130nm‡	16KB	N/A	154MHz	N/A	4-5
PicoServer MP core 90nm§	16KB	N/A	500MHz	190mW	4-5

* Die area for a 90nm Xscale excludes L2 cache[83]

† MIPS32 34K supports multi-threading (2 kernel threads, 4 user threads)

‡ Values generated from Synopsys physical compiler

§ For the PicoServer core, we estimated our power to be in the range of an ARM11, Xscale

Table 5.2: Published and synthesized power consumption and die size for various microprocessors[36][2][15][12][9][83][84]

future, 32bit machines seemed to work well with our simulation environment and power/area estimates. Branch prediction is still useful in a server workload. The processor has a typical hybrid branch predictor with a 1KB history table. Our studies found in 5.1 showed the accuracy of the branch predictor for server workloads over 95%. We assume no support for multithreading. We will later describe the impact of multithreading in later sections.

Each core also includes architectural support for a shared memory protocol and a memory controller that is directly connected to DRAM. The memory controller responds to shared bus snoops and cache misses. On a request to system memory, the memory controller delivers the address, data—for memory writes—and cpu ID—for memory reads—which is used later for routing where the data should go on memory reads. A die area analysis on the expected die area per core was conducted. We collected several die area numbers available from ARM, MIPS, PowerPC and other comparable scalar in-order processors. Table 5.2 lists these estimates along with values listed in [2][83][15][12][9] and a Pentium 4 core for comparison. We also synthesized several 32bit open source cores that are computationally comparable to a single PicoServer cores. We synthesized them using the Synopsys Physical compiler toolset. Unfortunately, we did not have access to 90nm and below standard cell libraries. We apply well-known scaling rules to estimate die area and power consumption. The power values listed in Table 5.2 include static power. Our estimates for a 500MHz PicoServer core are conservative compared to the ARM core values, especially with respect to [83]. Considering 850mW is consumed at 1.5GHz and 1.3V for the Xscale core, a power consumption of 190mW at 500MHz for our 90nm PicoServer core is conservative when applying the 3× reduction in clock frequency and the additional

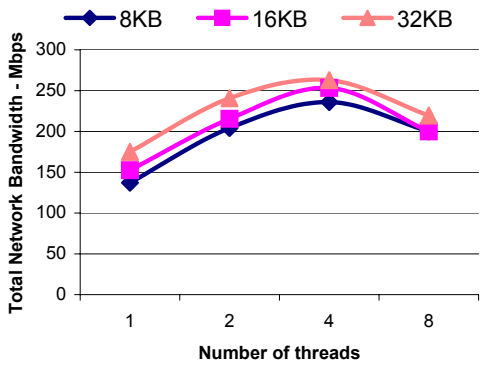
opportunities to scale voltage. From our synthesized results and related publication survey, we believe a single PicoServer core would occupy a die area of 5mm^2 and consume less than 200mW at 500MHz.

5.1.2 Impact of multi-threading

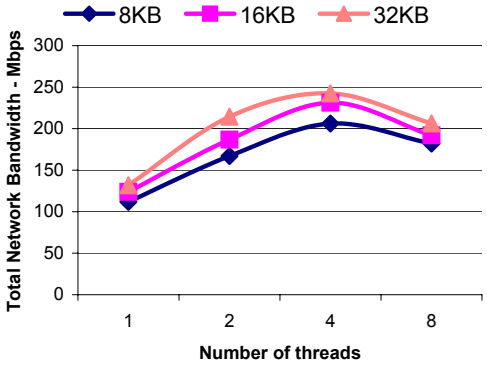
Multithreading improves overall throughput by switching thread contexts during lengthy stalls to memory. To study the impact of multithreading in PicoServer, we assume multithreading support that includes an entire thread context—register file, store buffer and interrupt trap unit. An additional pipeline stage is required to support multithreading. The added stage schedules threads to be executed. In the study conducted, we varied the number of threads supported and the unloaded access latency to memory in a single core and measured the network bandwidth (a metric for throughput) delivered by this core. We did our analysis running SURGE because the simulation time of SURGE is relatively shorter than other workloads and it displayed the highest L1 cache miss rate which implies SURGE would benefit the most from multithreading. Our metrics used in this study are total network bandwidth and network bandwidth/mm^2. We varied the cache size to see the impact of threading.

Figure 5.1, 5.2 shows our simulated results. From our studies, we are able to conclude threading indeed helps improve overall throughput, however only to a limited extent when considering the area overhead and the impact of 3D stacking. 3D stacking reduces the access latency to memory by simplifying the core to memory interface and reducing the transfer latency. This implies 3D stacked memory to be accessed in 10's of cycles which corresponds to plots shown in Figure 5.1(b), 5.2(b). It shows in respect to area efficiency and throughput, limiting it to only support 2 threads may be best. [60] predicted a 20% die area overhead for a single core in their architecture. Considering the size of a single Niagara core—16mm^2, we believe the area overhead for our PicoServer cores would be more than 20%. We also find that an increase in throughput stresses the interconnection network suggesting that for conventional CMPs to scale would require high throughput low latency interconnection networks or a drastic increase in cache size per core. A system must be able to deliver sufficient I/O bandwidth, memory bandwidth to accommodate the additional threads. If any of the above is not scalable, then the impact of threading is detrimental to overall system throughput. I/O bandwidth impacts processor utilization and memory bandwidth.

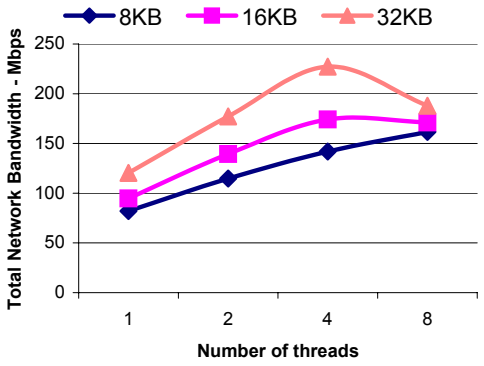
Another support required in fully utilizing threaded cores is system software support. Making the operating system scalable to the number of cores has been known to be challenging due to conventional OSes written to be optimized for single threaded



(a) memory latency = 1

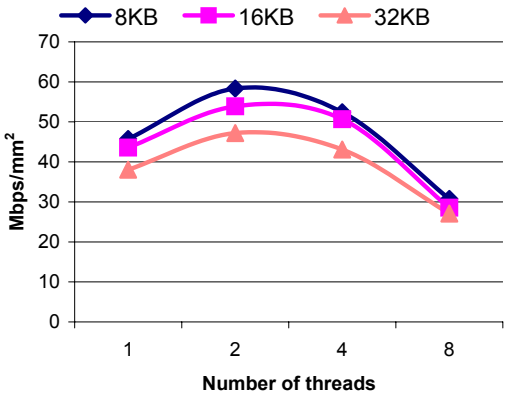


(b) memory latency = 10

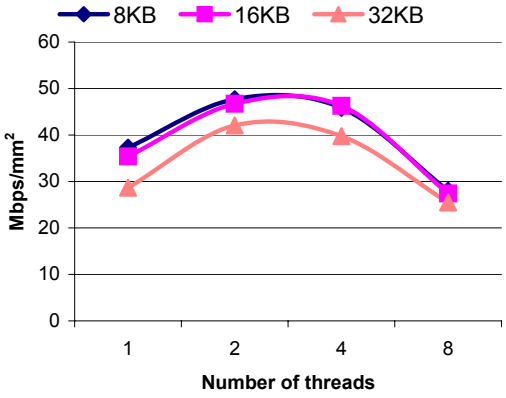


(c) memory latency = 100

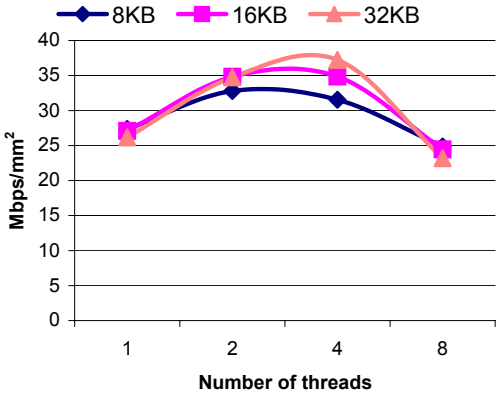
Figure 5.1: Impact of multi-threading for varying memory latency on SURGE for varying 4 way set associative cache sizes(8KB, 16KB, 32KB) and varying number of threads. We assume the core is clocked at 500MHz



(a) memory latency = 1



(b) memory latency = 10



(c) memory latency = 100

Figure 5.2: Impact of multi-threading for Mbps/mm^2 when varying memory latency on SURGE. The same setup and assumptions in 5.1 are applied.

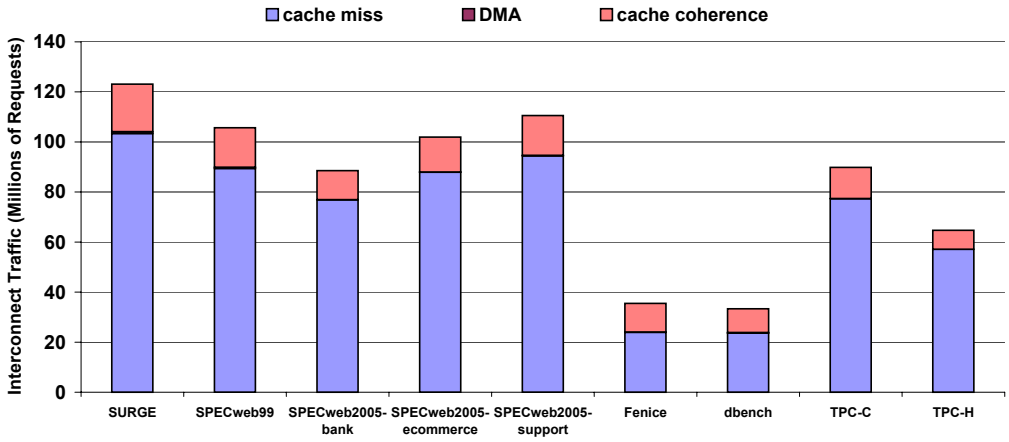


Figure 5.3: Interconnect traffic measured for 4 way 16KB 128 byte L1 cache

	130nm	90nm
on-chip 2D 12mm	5.6nF	5.4nF
on-chip 3D 8mm	3.7nF	3.6nF
off-chip 2D	16.6nF	16.6nF

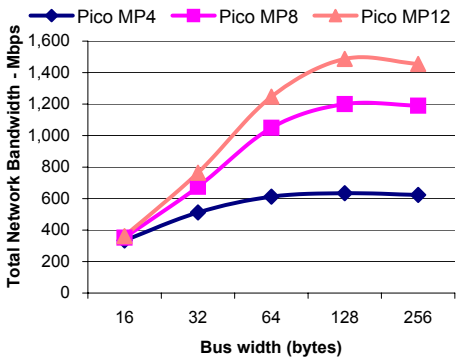
Table 5.3: Parasitic interconnect capacitance for on-chip 2D,3D and off-chip 2D for a 1024 bit bus

performance and single core platforms. Threading requires code modification to allow each core to be fully utilized and not remain idle. This has been presented as a challenge in many aspects and is an area that is beyond the scope of this dissertation.

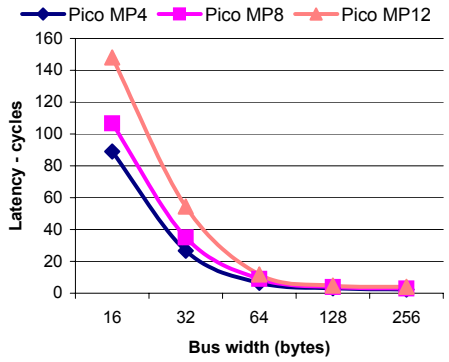
5.1.3 Wide shared bus architecture

PicoServer adopts a wide shared bus architecture that provides high memory bandwidth and fully utilizes the benefits of 3D stacking technology. Interconnect architectures are dependant on the traffic pattern injected. From our simulation studies, we found that our static web server benchmark—SURGE—generated high cache traffic as well as high I/O traffic shown in Figure 5.3 and displayed modest simulation time. Our static web server benchmark generated a high cache miss rate per core. Our bus architecture was determined from static web server benchmark runs.

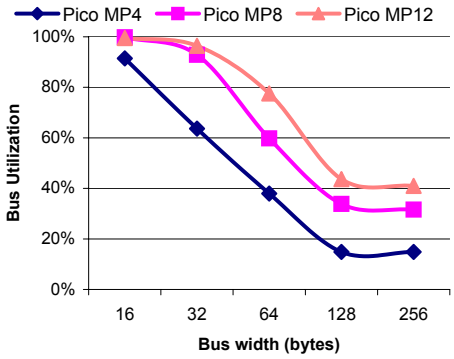
To explore the design space of our bus architecture, we first ran simulations for varying the bus width on a single shared bus—ranging from 128 bits to 2048 bits. Network performance is measured to determine the impact of bus width on the PicoServer. As shown in Figure 5.4(a), a relatively wide data bus is necessary to achieve scalable network performance to satisfy the outstanding cache miss requests. This



(a) Network bandwidth vs. Bus width



(b) Overall loaded latency vs. Bus width



(c) Bus utilization vs. Bus width

Figure 5.4: Network performance for various shared bus architectures based on our L1 cache size—16KB on SURGE. We assumed a CPU clock frequency of 500MHz for these experiments. Our bus architecture must be able to handle high bandwidths as the number of processors increase.

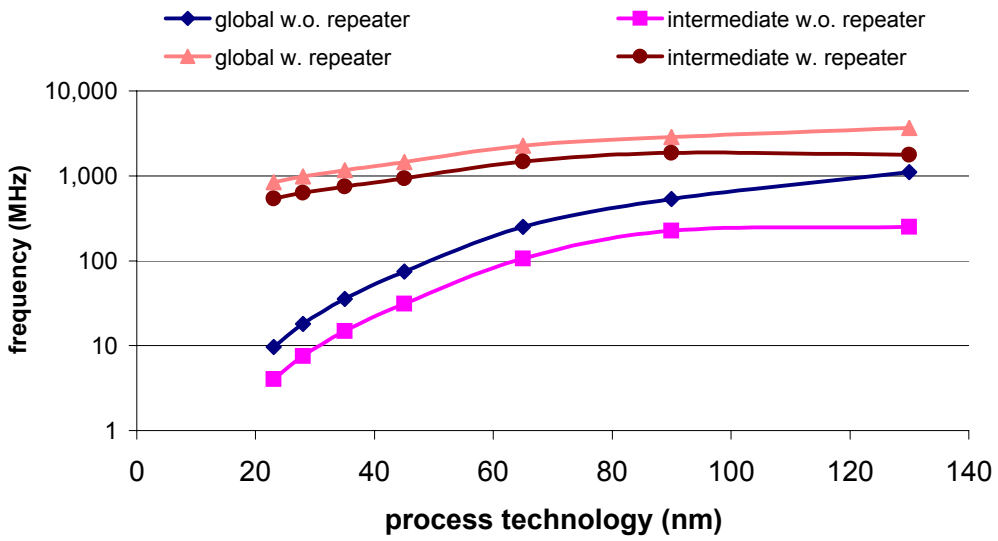


Figure 5.5: Maximum interconnect clock frequency roadmap for global and local wires with wire lengths of 10mm

is because of the high bus contention on the shared data bus for high bus traffic that is generated for narrow bus widths as shown in Figure 5.4(b) and Figure 5.4(c). As we decrease the bus width, the bus traffic increases resulting in superlinear increase in latency. Reducing bus utilization implies reduced bus arbitration latency, thus improving network bandwidth. Wide bus widths also help speedup NIC DMA transfers by allowing a large chunk of data be copied in one transaction. A 1024 bit bus width seems reasonable for our typical PicoServer configurations—4, 8, 12 multiprocessors, since network performance saturated after that point. We also looked at interleaved bus architectures but found with our given L1 cache miss rates, a 1024 bit bus is sufficient enough to handle the bus requests. For architectures and workloads that generate higher bus requests by increasing the number of cores to 16 or more with higher L1 cache miss rates—more than 10%—then interleaving the bus is more effective.

Using our analytical models described in Chapter III, Table 5.3 shows the expected interconnect capacitance for 1024bits in the case of 2D on-chip, 3D stacking, and 2D off-chip implementations. Roughly speaking, on-chip implementations have at most 33% capacitance of an off-chip implementation. Furthermore, since the supply voltages in IO pads—typically $1.8 \sim 2.5$V, are generally higher than the core supply voltage, we find the overall maximum power for an off-chip implementation consumes an order of magnitude more power than an on-chip and off-chip average interconnect

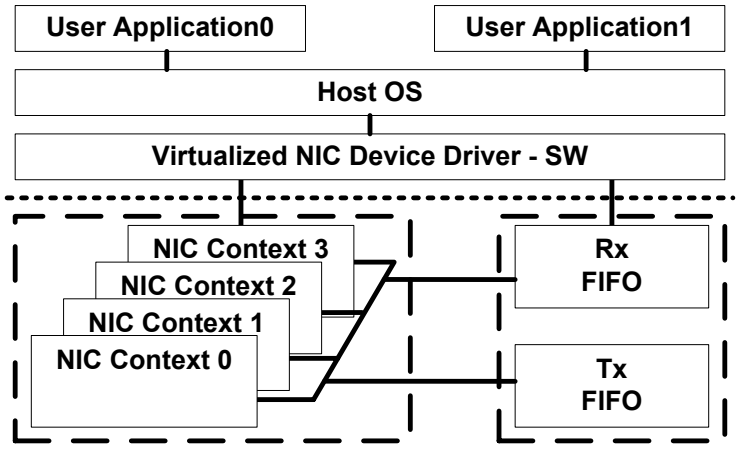


Figure 5.6: Virtualized NIC architecture

power. With modest toggle rates, small to modest access rates for typical configurations found in our benchmarks and modest bus frequency—250MHz, we conclude that on-chip interconnect power contributes very little to overall power consumption.

As we can see in Figure 5.5, a wire length of 10mm of global wires without repeaters can easily be clocked at 500MHz.

5.1.4 The need for Multiple NICs on a CMP architecture

A common problem of servers with large network pipes is handling the hundreds of thousands of packets that might arrive each second. Interrupt coalescing is one method of dealing this problem. This method works by starting a timer when a non-critical event occurs. Any other non-critical events that occur before the timer expires are coalesced into one interrupt and thus the number of interrupts can be reduced. Even with this technique however the number of interrupts received by a relatively low frequency processor, such as one of the PicoServer cores, can overwhelm it. In our simulations we get around this difficulty by having multiple NICs each having their interrupt lines routed to a different processor. Although this method would be valid, a smarter single NIC that could route interrupts to multiple CPUs, each with separate DMA descriptors and TX/RX queues, could be built. For an 8 chip-multiprocessor architecture with 1 NIC and an on-chip DRAM integrated by using 3D stacking technology, we found the average utilization per processor to be below 60% as one processor could not manage the NIC by itself. To fully utilize each processor in our multiple processor architecture, we inserted 1 NIC for 2 processors. For example, a 4

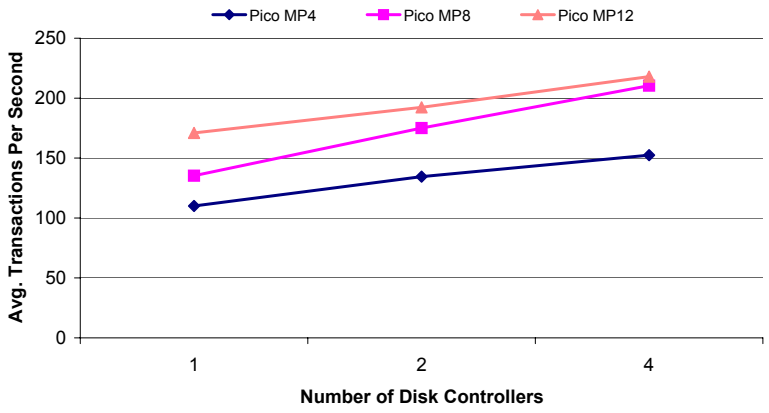


Figure 5.7: Adding multiple disk controllers to improve overall throughput

CMP architecture would have 2 NICs, a 8 CMP architecture would have 4 NICs and so forth. However, as mentioned above, this could be one NIC either with multiple interface IP addresses or an intelligent method of load balancing packets to multiple processors. Such a NIC would need to keep track of network protocol states at the session level. We present an example diagram in Figure 5.6. There have been previous studies of intelligent workload balancing on NICs to achieve optimal throughput on platforms[37]. TCP splicing and handoff are also good examples of intelligent load balancing at higher network layers[70].

5.1.5 The need for Multiple Disk Controllers on a CMP architecture

Similar to what is observed in NICs, multiple disk controllers may be required for chip multiprocessor based platforms executing disk intensive server workloads. Tier 3 servers that generate lots of disk I/O requests fall into this category of workloads. The disk pipes that depend on the disk access latency and the disk interface must be able to provide enough requests to fully utilize all the cores. We found in our simulations that a 12 core system with a single disk controller could not keep up with the disk requests generated per core. Because the OS views a hard disk drive as a block device, it transfers large blocks that are several kilobytes in size using DMA. In addition the OS tries to reduce the amount of disk access latency by aggregating requests that are adjacent into a single request. As a result, I/O requests typically occur asynchronously and in large chunks which may block threads from executing. The effect of thread blocking is evident in server workloads because many of the

disk requests in server workloads are random accesses. By observing the average bytes transferred per disk request and considering the read ahead window size, we found this behavior to be true for most of our server workloads. Aside from video streaming workloads that are spatially contiguous, other server workloads randomly accessed disk resulting in finding no benefit in aggregating disk requests. By using multiple disk controllers the I/O bottleneck can be reduced since multiple requests to disk can occur simultaneously thus improving I/O throughput. Figure 5.7 shows how much improvement we see from adding additional disk controllers and disks to a 12 multicore conventional CMP. We see an 15% improvement in overall throughput. This not only improves the throughput but indirectly leads to a reduction in energy due to the reduction in idle time.

5.2 On-chip memory architecture of PicoServer

PicoServer stacks multiple DRAM dies on top of the logic die. We describe the role of this on-chip DRAM memory in the following subsections.

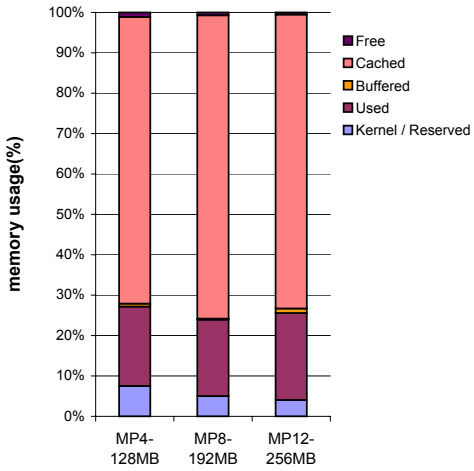
5.2.1 Role of on-chip DRAM

Based on the logic die area estimates, we projected the DRAM die size for a 12 way PicoServer to be a die size of 80mm^2 and respectively, 40mm^2, 60mm^2 for a 4 way, 8 way PicoServer. Table 5.5 shows the projected amount of on-chip memory for our PicoServers. For example, to obtain a total DRAM size of 256 MB, we assume DRAM is made up of 4 layers of the final die allowing us to integrate aggressive capacity for on-chip stacked DRAM. 8 layers of DRAM die is assumed for Tier 3 servers which rely heavily on system memory size. With current technology—90nm, it is feasible to create a 4 layer stack containing 256MB of physical memory for a die area of 80mm^2. Although a large amount of physical memory is typical in server farms (4 to 16GB), with the short sample time of a simulator, it is difficult for a benchmark

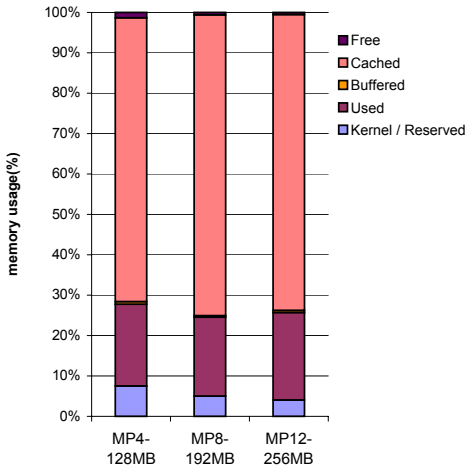
	Density	Area(mm^2)	Process
Samsung	64MB	49	80nm
Samsung	64MB	71	90nm
Infineon - A*	64MB	87	110nm
Infineon - B	128MB	176	N/A
Micron	128MB	180	110nm

* Infineon A, B are 2 different types of DRAM chips

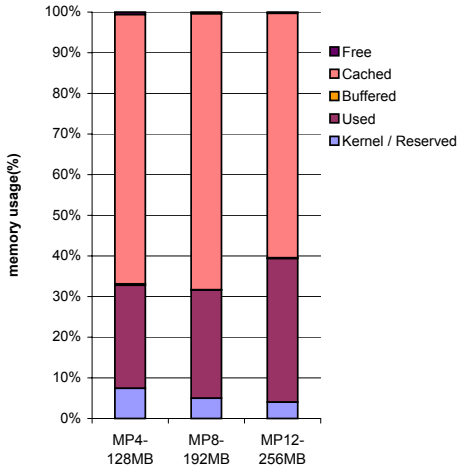
Table 5.4: DRAM die size from various vendors noted in Semiconductor SourceInsight 2005 [69]



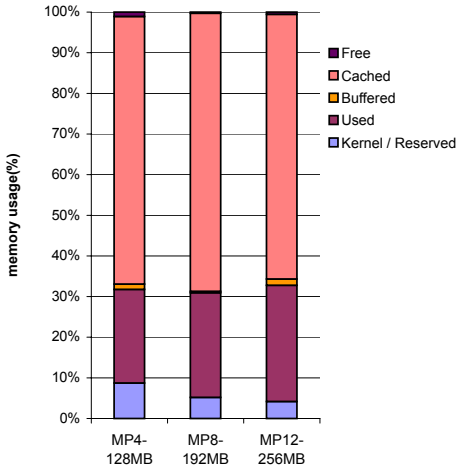
(a) SURGE



(b) SPECWeb99

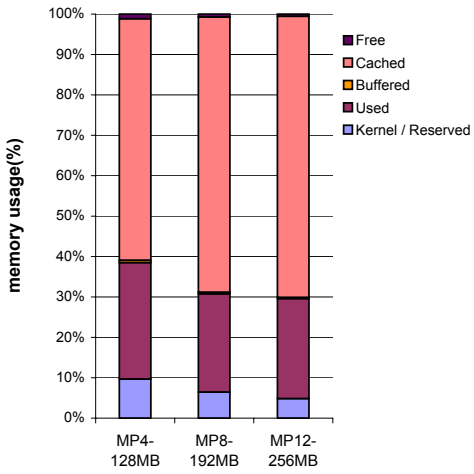


(c) Fenice

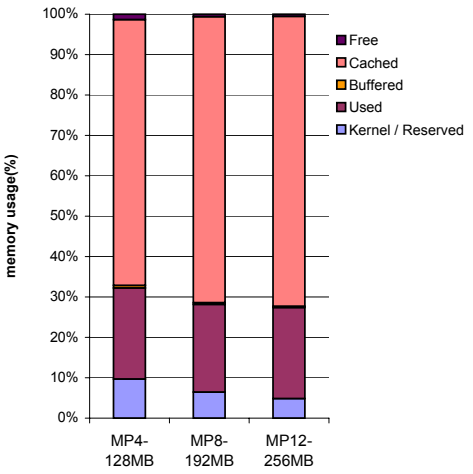


(d) dbench

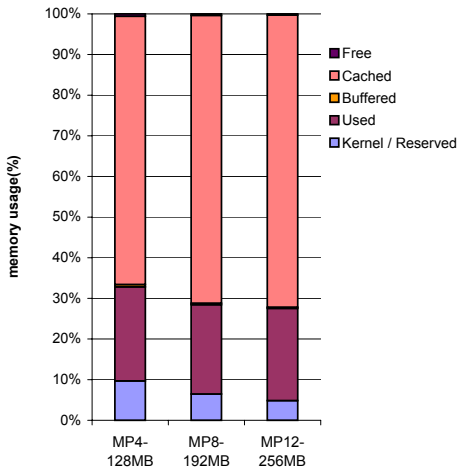
Figure 5.8: Breakdown in memory for server benchmarks (SURGE, SPECWeb99, Fenice, dbench)



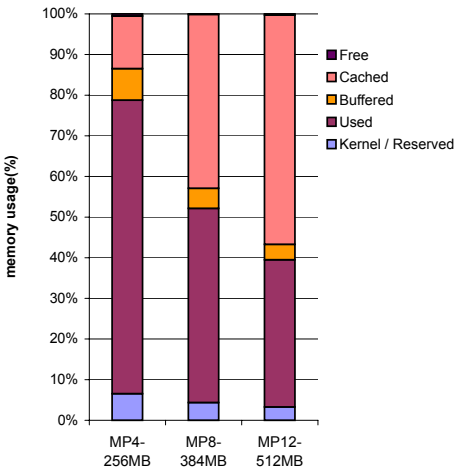
(a) SPECWeb2005-bank



(b) SPECWeb2005-ecommerce



(c) SPECWeb2005-support



(d) TPC-C

Figure 5.9: Breakdown in memory for server benchmarks (SPECWeb2005, TPC-C) TPC-H is excluded because it displayed similar memory usage as TPC-C.

	130nm	110nm	90nm	80nm
DRAM stack 4 layer each layer 40mm^2	64MB	96MB	128MB	192MB
DRAM stack 8 layer each layer 40mm^2	128MB	192MB	256MB	384MB
DRAM stack 4 layer each layer 60mm^2	96MB	144MB	192MB	288MB
DRAM stack 8 layer each layer 60mm^2	192MB	288MB	384MB	576MB
DRAM stack 4 layer each layer 80mm^2	128MB	192MB	256MB	384MB
DRAM stack 8 layer each layer 80mm^2	256MB	384MB	512MB	768MB

Table 5.5: Projected on-chip DRAM size for varying process technologies. Area estimates are generated based on Table 5.4. 80mm^2 of die size is similar to that of a Pentium M at 90nm.

to touch such a large memory space. Depending on the workload of each PicoServer, the estimated physical memory may be enough. From our measurements on memory usage for server applications shown in Figure 5.8 and 5.9, we found a modest amount—around 64MB—of system memory is occupied by the user application, data and the kernel OS code. The remainder of the memory is either free or used as a disk cache. Considering the fact that 256MB can be integrated on-chip for 4 die layers, we project a large portion of on-chip DRAM to be used as a disk cache. Therefore, for applications that require small/medium filesets, a on-chip DRAM of 256MB is enough to effectively handle client requests.

For large filesets, there are several options to choose from. First, we could add additional on-chip DRAM by stacking additional DRAM dies. From the ITRS roadmap in Table 2.2, we recall that the number of stacked dies we assume is conservative. With aggressive die stacking, we could add more die stacks to improve on-chip DRAM capacity—ITRS projects more than 11 layers to be possible in the next $2 \sim 4$ years. This is possible because our power density in the logic layer is extremely small—less than $5\text{W}/cm^2$. Another alternative is to add a secondary system memory which functions as a disk cache. From our simulations, we found that the access latency of this secondary system memory could be as slow as hundreds of μs and still generate equal network bandwidth. The multithreaded nature of server applications hide thread stalls due to the long access latency to memory through interleaved thread execution. An access latency as slow as hundreds of μs implies that Flash memory that consumes less active/standby power can be used as secondary system memory. Therefore, for workloads requiring large filesets, we could build a non-uniform memory architecture with fast on-chip DRAM and relatively slower off-chip secondary system memory.

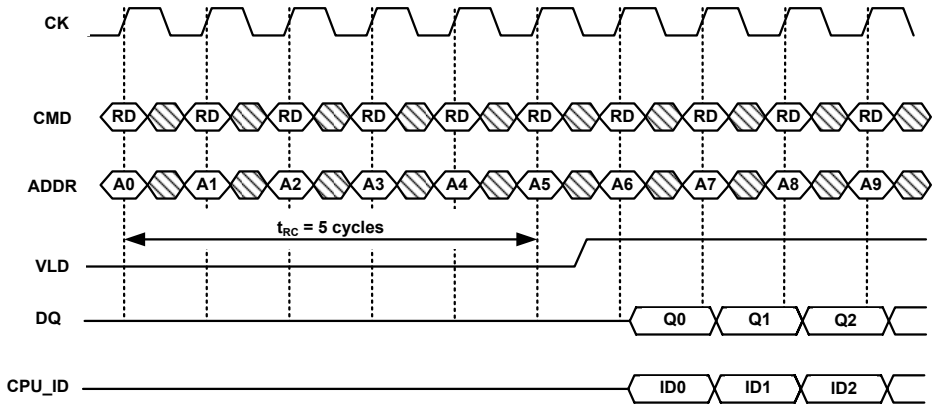


Figure 5.10: on-chip DRAM read timing diagram

The fast on-chip DRAM would primarily hold code, data and a small disk cache and the slow system memory would function as a large disk cache device.

5.2.2 On-chip DRAM interface

To maximize the benefits of 3D stacking technology, the conventional DRAM interface needs to be modified for PicoServer's 3D stacked on-chip DRAM. Conventional DDR2 DRAMs are designed assuming a small pin count and use address multiplexing and burst mode transfer to make up for the limited number of pins. With 3D stacking technology, there is no need to use narrow interfaces and furthermore address multiplexing which requires multi-phase commands such as a RAS followed by a CAS. Instead, the additional logic required in latching and muxing narrow address/data can be removed. The requested addresses can be sent as a single command while data can be driven out in large chunks. Further, conventional off-chip DRAMs are offered as DIMMs made up of multiple DDR2 DRAM chips. The conventional off-chip DIMM interface accesses multiple DDR2 DRAM chips per request. For 3D stacked on-chip DRAM, only one subbank needs to be accessed per request. As a result 3D stacked on-chip DRAM consumes much less power per request than off-chip DRAM. Figure 5.10 shows an example of a read operation. DRAM vendors already provide interfaces that do not require address multiplexing such as Reduced Latency DRAM from Micron [18] and NetDRAM[3] from Samsung. This suggests the interface for 3D stacked on-chip DRAM can be tailored with minor tweaks. Additional die area made available through the simplification of the interface can be used to speed up

the access latency to on-chip DRAM. By investing more die area on subbanking the on-chip DRAM, DRAM latencies that are approximately $10ns$ can be achieved.

5.2.3 Impact of on-chip DRAM refresh on throughput

DRAM periodically requires each cell to be refreshed. Each individual cell has a retention time of 64ms in an industry standard temperature and decreases to 32ms in harsh environments. But refresh circuits incur an additional area overhead forcing a DRAM bank to share the refresh circuit. This results in frequent refresh circuit activity. When we assume we share a refresh circuit per bank, the average DRAM refresh interval is approximately 7.8125μs and requires approximately 200ns to complete. This implies a DRAM bank cannot be accessed for a duration of 100's of CPU clock cycles every 1000's of CPU clock cycles. To measure the impact of this effect, we modeled the refresh activity of DRAM on M5 and observed the CPI overhead. The access frequency to on-chip DRAM is directly correlated to the amount of L1 cache misses observed. We found for a 5% L1 cache miss rate and 12 cores clocked at 500MHz, this would incur a CPI overhead of 0.03 CPI due to refresh. This is because many of the L1 cache misses do not occur when a refresh command is executed resulting in a marginal performance overhead. For many core (more than 16) PicoServers, we believe the L1 cache size that impacts the L1 cache miss rate should be sized such that many L1 cache misses don't overlap with a DRAM refresh cycle. Careful provisioning of the number of DRAM banks along with the L1 cache size and the number of cores should be performed to marginalize the impact of DRAM refresh. The overhead of DRAM refresh time would only become significant if the interconnect between DRAM and the cores is a bottleneck. Interconnect interfaces that are near saturated states are very sensitive to small changes in latency.

	Thermal Conductivity (W/m·K)	Heat Capacity ($J/m^3 \cdot K$)
Si	148	1.75×10^6
SiO_2	1.36	1.86×10^6
Cu	385	3.86×10^6
Air at $0C^o$	0.024	1.25×10^3

Table 5.6: Thermal parameters for commonly found materials in silicon devices

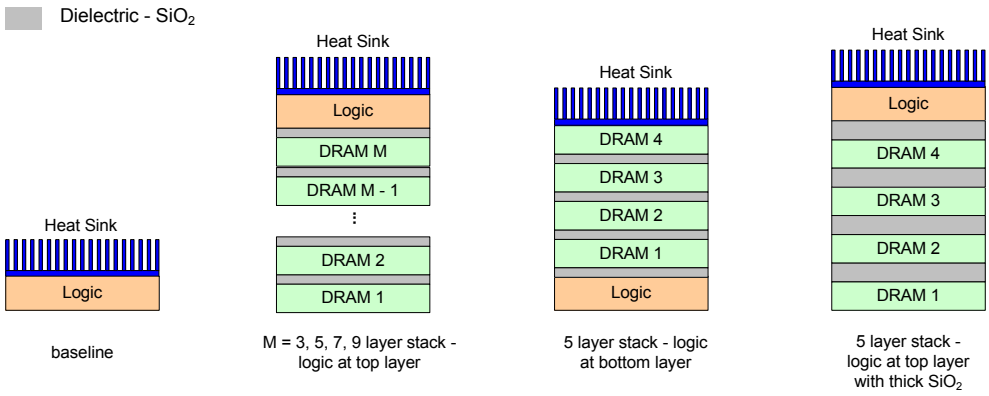
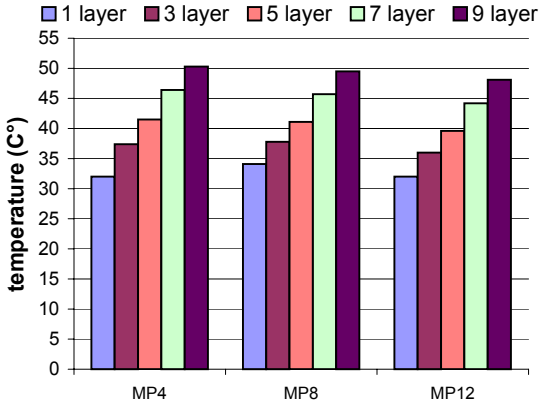


Figure 5.11: A diagram depicting the thermal analysis performed on architectures using 3D stacking technology.

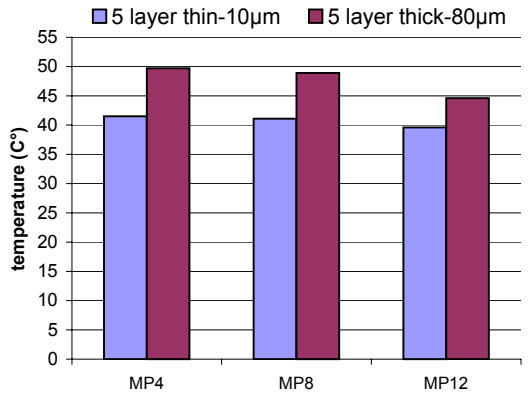
5.3 Thermal concerns in 3D stacking

A potential concern with 3D stacking technology is heat containment. To address this concern, we investigated the thermal impact of 3D stacking on the PicoServer architecture. Since we could not measure temperature directly on a real 3D stacked platform, we modeled the 3D stack onto the grid model in Hotspot [52]. Mechanical thermal simulators such as FLOWTHERM and ANSYS were not considered in our studies due to the limited information we could obtain from the 3D stacking process. We believe Hotspot's RC equivalent heat flow model is adequate to show trends and potential concern in 3D stacking. Because this work describes the usefulness of integrating 3D stacking into the server space, instead of describing the details in heat transfer, we present general trends. We leave detailed studies to future work and published references that cover heat in 3D stacking as a primary topic.

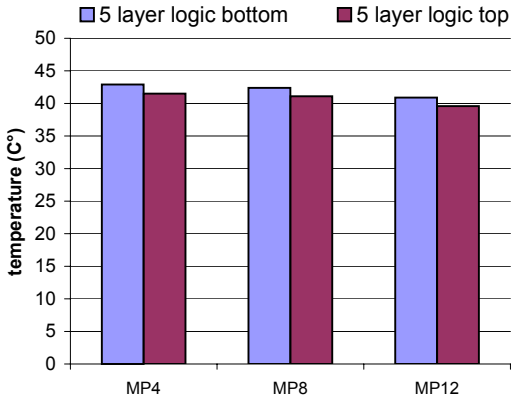
The primary thermal issue in devices utilizing 3D stacking is heat containment due to the interface material—SiO_2—and the free air interface between silicon and air as can be seen in Table 5.6. Silicon and metal conduct heat much more efficiently. We first configured our PicoServer architecture for various scenarios by 1) varying the amount of stacked dies, 2) varying the location of the primary heat generating die—the logic die on our platform, 3) varying the thickness of the SiO_2 insulator that is typically used in between stacked dies. Figure 5.11 shows the configurations used in this study. Our baseline configuration assumes a logic die directly connected to a heat sink assuming 27C° room temperature. We assumed a naive floorplan of our PicoServer architecture and varied the number of processors. Hotspot requires input



(a)



(b)



(c)

Figure 5.12: Maximum junction temperature for sensitivity experiments on Hotspot. (a)varying the number of layers, (b)varying 3D interface thickness, (c)varying location of logic die. A core clock frequency of 500MHz is assumed in calculating power density. We varied the size of on-chip memory based on the number of layers stacked. 1 layer assumes no on-chip memory at all.

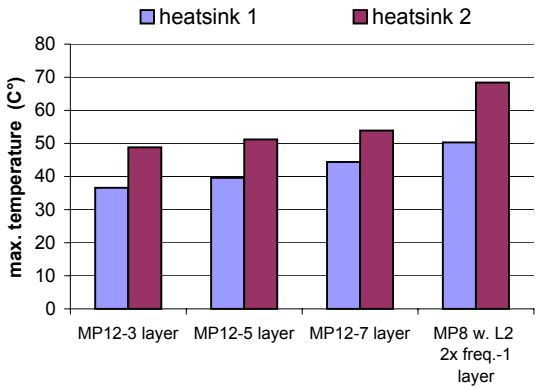


Figure 5.13: Maximum junction temperature for heatsink quality analysis.

for properties in material and power density to generate steady state temperature throughout the platform. We extracted 3D stacking properties from [57][68][87] and assigned power density at the component level based on area and power projections for each component. Components were modeled at the platform-level—processor, peripheral, global bus interconnect, etc. We generated maximum junction temperature in our PicoServer architecture shown in Figure 5.12.

Figure 5.12(a) shows the sensitivity to the number of stacked layers. We find roughly a $2 \sim 3C^{\circ}$ increase in maximum junction temperature for each additional layer stacked. Figure 5.12(b) shows the sensitivity to the 3D stacking dielectric interface. We compared the effect of the SiO_2 thickness (the interface material) for $10\mu m$ and $80\mu m$. In [33][57][68][87] we find the maximum thickness of the interface material does not exceed $10\mu m$ for 3D stacking. The $80\mu m$ point is selected to show the impact of heat containment as the thickness is increased substantially. It results in a 6 degree increase in junction temperature. While notable this is not a great change given the dramatic change in material thickness. Figure 5.12(c) shows the sensitivity to placement in the stack—top or bottom layer. We find the primary heat generating die is not sensitive to the geographic location of the heat sink.

We also conducted an analysis on the impact of heatsink quality. We varied the heatsink configuration to model a high cost heatsink (heatsink 1) and a low cost heatsink (heatsink 2). Figure 5.13 shows the impact of 3D stacking technology has on heatsink quality. It clearly suggests that a low cost heatsink can be used on platforms using 3D stacking technology.

We believe heat containment for having multiple stacked layers is not a major limitation in the PicoServer platform. The power density is relatively low for our

architecture. It does not exceed 5W/cm^2. As a result, the maximum junction temperature does not exceed 50C°. 3D vias can also act as heat pipes, which we didn't take into account in our analysis, however this is expected to improve the situation. An intelligent placement would assign the heat generating layer (the processor layer) adjacent to the heat sink resulting in a majority of the heat being transferred to the heat sink. There is independent support for our conclusions in [34][43].

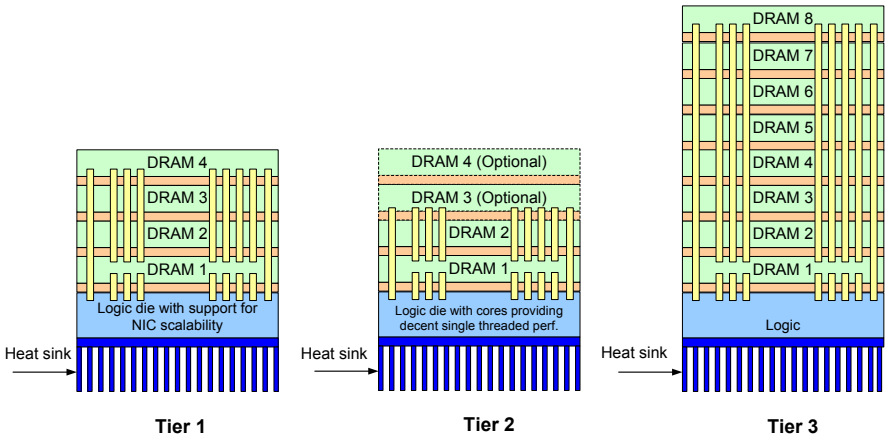
5.4 Customizing Energy Efficient Servers in a Datacenter

As we had shown in Chapter II, in a 3 Tier Server architecture, the workload characteristics of each Tier varies quite a bit resulting in different requirements for each Tier. All workloads in common have an abundance of thread level parallelism. However, Tier 1 stresses the network I/O, Tier 2 requires a decent amount of single threaded performance and Tier 3 stresses the disk I/O. To architect an energy efficient datacenter, one should architect each platform by satisfying the different requirements. Some server workloads are CPU bound, others are NIC bound and finally they could also be disk bound. Based on these properties, it suggests servers should be architected to satisfy the specific needs of the application. For example, Tier 2 and 3 servers that are not network intensive should not spend much effort in providing intelligent load balancing techniques for NICs and providing high speed NICs. In fact, a 100Mbps or 10Mbps may be enough for NICs used in Tier 2,3 servers.

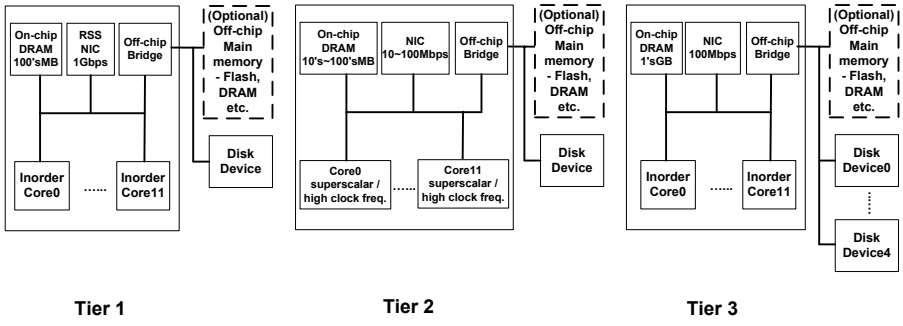
With respect to an architecture like PicoServer, the number of layers invested in on-chip DRAM and the die area invested in logic functionality will have to vary slightly depending the Tier PicoServer is targeted for.

Tier 1 servers require intelligent built-in NIC load balancers that interface with the high level global load balancers(Layer 7 switches) should be included in the overall platform. Disk I/O is less of a concern for Tier 1 servers suggesting off-chip system memory may not be required at all. This is possible since web page requests display a short-tailed zipf like distribution.

In contrast, much effort must be invested in improving disk I/O throughput for Tier 3 servers. Tier 3 servers do not generate a large network bandwidth implying less efforts could be invested in improving overall network bandwidth. In fact, it would be wise to support lower network throughput to reduce overall NIC power. Because the size of a database gradually grows and the access behavior to database systems are known to be long-tailed, much effort must be put in provisioning memory requirements



(a)



(b)

Figure 5.14: System architecture of datacenter using PicoServers, (a) 3D stacking block diagram of PicoServers, (b) Platform level block diagram of PicoServers

that mitigates disk I/O. It is highly likely that Tier 3 PicoServers require several disk controllers as well as stacking more layers of on-chip DRAM. Tier 3 PicoServers are expected to require the most storage density.

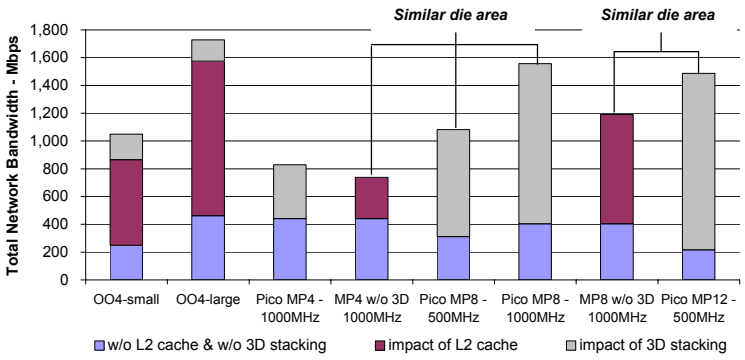
Tier 2 servers require a substantial amount of computation power. The type of computation required in Tier 2 workloads are decision support, interpretation of scripting languages etc. These application display more ILP than TLP. Unless ILP applications can be transformed into TLP applications using an ILP based superscalar core is desirable for Tier 2 Servers. Since ILP based high performance superscalar cores are known to be power hungry, software optimization techniques that cache previous computations are necessary when deploying superscalar cores on Tier 2 Servers. Tier 2 servers though occupy a small memory footprint (If we choose to neglect the heap size used in java applications)

5.5 Evaluation

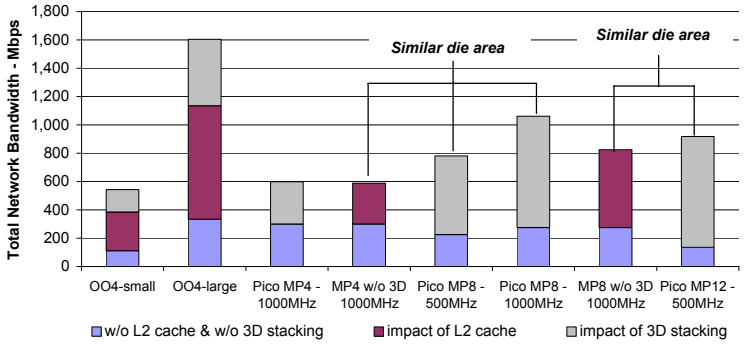
To evaluate the PicoServer architecture two metrics are important—throughput and power. Throughput is measured as network bandwidth, transactions per second and queries per minute based on the server workload. This is a good indicator of overall system performance because it is a measure of how many requests were serviced. In this section, we compare various PicoServer configurations to other architectures first in terms of achievable network bandwidth and then in terms of power. We compare PicoServer to CMP architectures without 3D stacking and conventional high performance desktop architectures which we call OO4—Pentium 4-like. Since the PicoServer has not been implemented, we use a combination of analytical models and published data to make a conservative estimate about the power dissipation of various components. Finally we present a pareto chart showing the energy efficiency of the PicoServer architecture.

5.5.1 Server Throughput for various configurations—overall performance

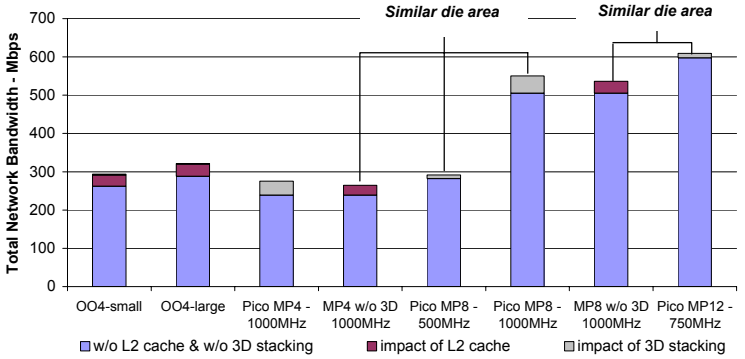
Figure 5.15, 5.16, 5.15 shows the throughput for our Tier 1, 2, and 3 workload runs. We breakdown the contribution to throughput with respect to a baseline with no L2 cache and a narrow (64bit) bus width, having an L2 cache, and implementing the benefits of 3D stacking technology. Due to the lengthy simulation time, TPC-H was executed on a real machine for out-of-order configurations. For simple multicore architectures, we recall in Table 4.1 the L2 cache unloaded latency is similar to the



(a) SURGE

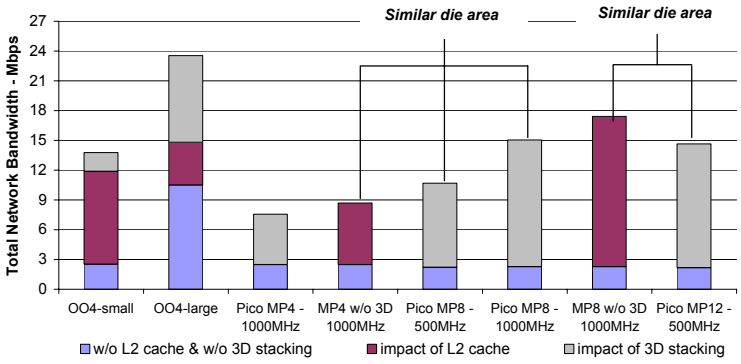


(b) SPECweb99

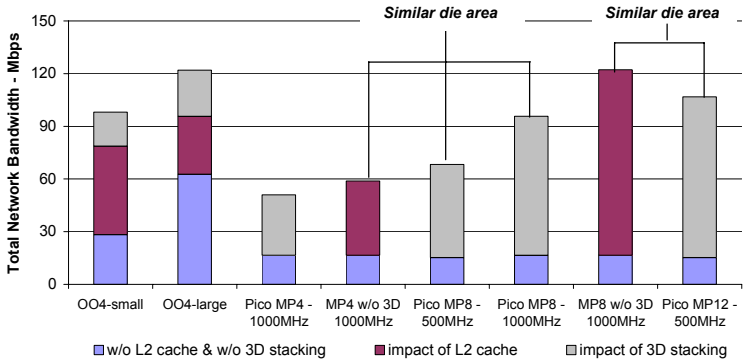


(c) Fenice

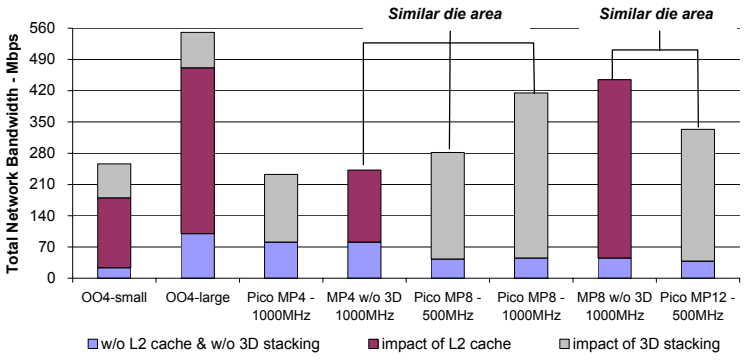
Figure 5.15: Throughput measured for varying processor frequency and processor type. For PicoServer CMPs, we fixed the on-chip data bus width to 1024bits and bus frequency to 250MHz. For a Pentium 4-like configuration, we placed the NIC on the PCI bus and assumed the memory bus frequency to be 400MHz. For a MP4, MP8 without 3D stacking configuration, to be fair we assumed no support for multithreading and a L2 cache size of 2MB. The external memory bus frequency was assumed to be 250MHz. (SURGE, SPECweb99, Fenice)



(a) SPECweb2005-bank

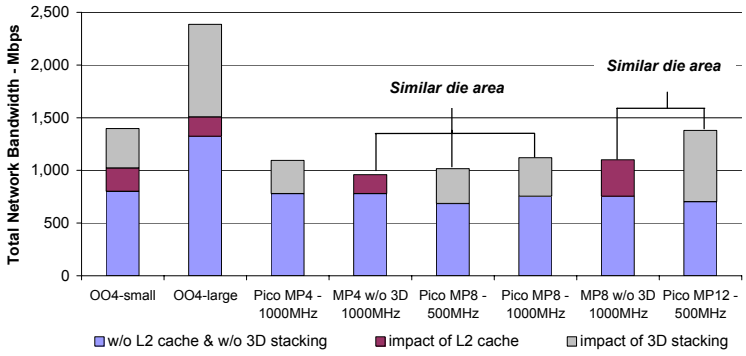


(b) SPECweb2005-ecommerce

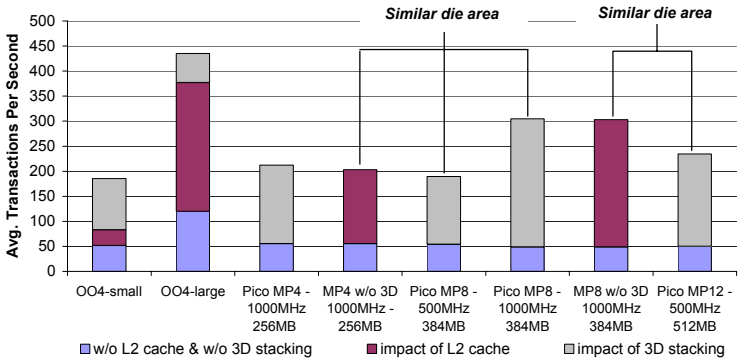


(c) SPECweb2005-support

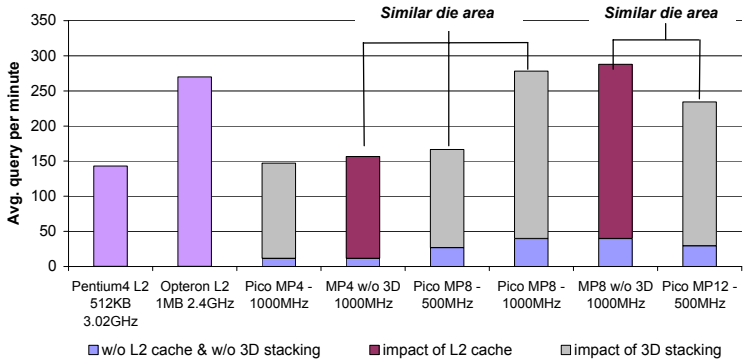
Figure 5.16: Throughput measured for varying processor frequency and processor type. (SPECweb2005), we applied the same assumptions used in Figure 5.15



(a) dbench



(b) TPC-C



(c) TPC-H

Figure 5.17: Throughput measured for varying processor frequency and processor type. (dbench, TPC-C, TPC-H), we applied the same assumptions used in Figure 5.15. For TPC-H, out-of-order core performance was measured on a real machine because the simulation time would be weeks.

DRAM access latency. Hence, we are able to make comparisons that differentiate the impact of 3D stacking technology with the impact of having an L2 cache. We show in separate bars the impact of having an L2 cache or adopting 3D stacking technology. It shows that 3D stacking technology alone improves overall performance equal to or more than having an L2 cache. A fair comparison for a fixed number of cores, for example, would be a Pico MP4-1000MHz versus a conventional CMP MP4 without 3D-1000MHz. In general, workloads that generated modest to high cache miss rates (SURGE, SPECweb99, SPECweb2005, TPC-C, TPC-H and dbench), showed dramatic improvement from adopting 3D stacking technology. Fenice that is bounded by video stream computation, generated low cache miss rates resulting in marginal improvement with adding an L2 cache or adopting 3D stacking technology. Since video streaming workloads inherently support many client connections, we found TLP friendly architectures perform well for this benchmark—the more cores you have, the higher the network bandwidth. Surprisingly, the script language based Tier 2 like benchmark—SPECweb2005 displayed fair performance compared to OO4 configurations built for single threaded performance.

For OO4 configurations, we combine the impact of having an L2 cache and 3D stacking since the unloaded L2 cache latency on a uniprocessor is likely to be smaller than the access latency to a large capacity DRAM making it less appealing to only have a high bandwidth on-chip DRAM implemented from 3D stacking. We find that 3D stacking improves performance by 15% on OO4 configurations. When we compare a OO4 architecture without 3D stacking with our PicoServer architecture, a PicoServer MP8 operating at 500MHz performs better than a 4GHz OO4 processor with a small L1 and L2 cache of 16KB and 256KB respectively. For a similar die area comparison, we believe comparing PicoServer MP8 and a OO4-small architecture is a fair comparison considering the additional die area required for a OO4-large that has a L1 cache size of 128KB and a L2 cache size of 2MB.

When we assume that the area occupied by the L2 cache in our conventional CMP MP4/8 without 3D stacking technology is replaced with additional processing cores—a benefit made possible by using 3D stacking technology—a comparison in network performance for similar die area can be conducted on Pico MP8-500MHz versus a conventional MP4 without 3D-1000MHz and Pico MP12-500MHz versus a conventional MP8 without 3D-1000MHz—for Fenice, compare with Pico MP12-750MHz. Our results suggest that on average, additional processing elements and reducing core clock frequency by half on our workloads improve throughput and significantly save on power—shown in Section 5.5.2. Our estimated area for adding

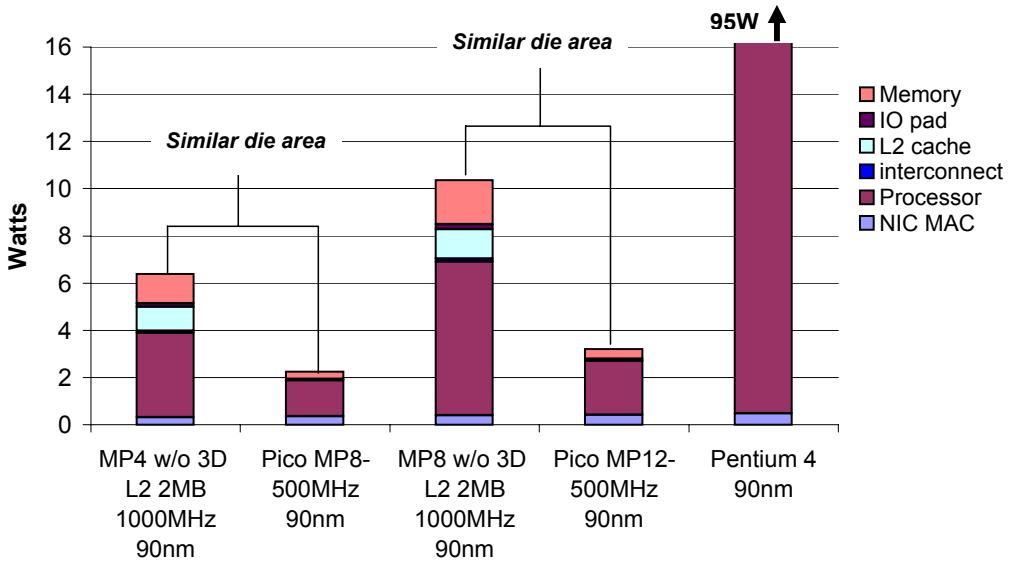


Figure 5.18: Breakdown of average power for 4, 8, 12 PicoServer architectures using 3D stacking technology for 90nm process technology. Estimated power per workload does not vary by a lot because the cores contribute to a significant portion of power. We expect $2 \sim 3$W to be consumed at 90nm. An MP8 without 3D stacking operating at 1GHz is estimated to consume 8W at 90nm.)

extra cores are extremely conservative suggesting more cores could be added thereby resulting in even more improvement in throughput.

5.5.2 Breakdown in overall power consumption

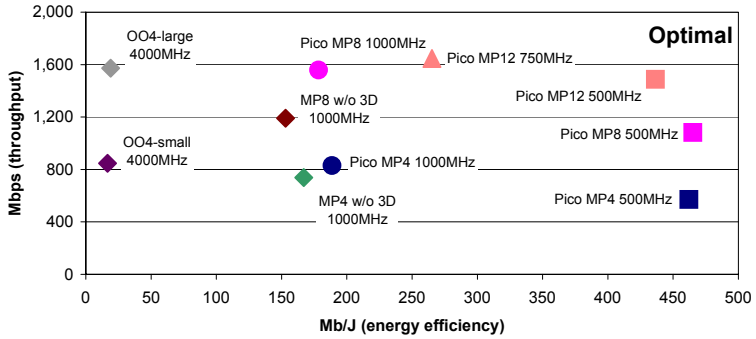
Processor power still dominates overall power in PicoServer architectures. Figure 5.18 shows the average power consumption based on our power estimation techniques for server application runs. We find PicoServer with a core clock frequency of 500MHz is estimated to consume somewhere between $2 \sim 3$ Watts for 90nm process technology depending on the design points (optimal power or network bandwidth). Overall power is primarily used by the simple in-order cores. NIC power consumed a considerable amount due to the increase in number of NICs when increasing the number of processors, however as described in section 5.1.4 an intelligent NIC designed for this architecture could be more power efficient as you would only need one. An appreciable amount of DRAM power reduction is also observed due to 3D stacking. The simplified on-chip DRAM interface enables less DRAM subbanks to be simultaneously accessed per request. Other components such as the interconnect make marginal contributions to overall system power due to the modest access rates and toggle rates of these components.

Comparing our PicoServer architecture with other architectures, we do very well. For a similar die area comparison, we use less than half the power when we compare Pico MP8/12-500MHz with a conventional MP4/8 without 3D stacking with an L2 cache at 1000MHz. We also recall in section 5.5.1 that performance-wise for a similar die area, our PicoServer architectures perform on average 10 $\sim$ 20% better than conventional CMP configurations. Furthermore, we use less than 10% of the power of a Pentium4 processor and as in the previous section perform comparably. At 90nm technology, it can be projected that the power budget for a typical PicoServer platform satisfies mobile/handheld power constraints noted in ITRS projections. This suggests the potential of implementing server-type applications in ultra small form factor platforms. We generally find that our PicoServer architecture provides excellent performance while using energy efficiently.

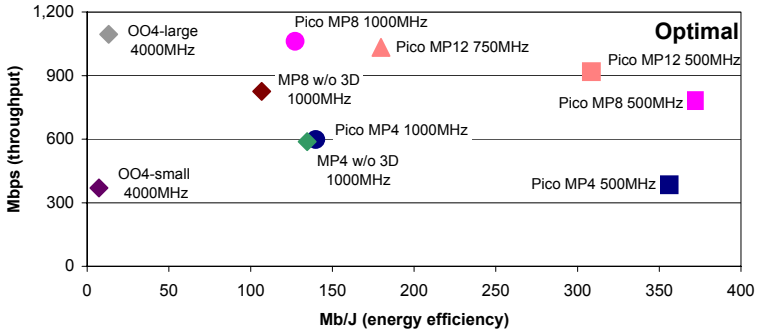
5.5.3 Energy efficiency, Throughput Pareto Chart

In Figure 5.19, 5.20, 5.21, we present a pareto chart for PicoServer depicting the energy efficiency and network performance. The points on this plot show the large out-of-order cores and the conventional CMP MP4/8 without 3D stacking processors we have described up to this point as well as our PicoServer with 4, 8, or 12 cores. On the y-axis we present Mbps and on the x-axis we show Mb/J. From Figure 5.19, 5.20, 5.21, it is possible to find the optimal configuration of processor number and frequency for a given energy efficiency/throughput constraint.

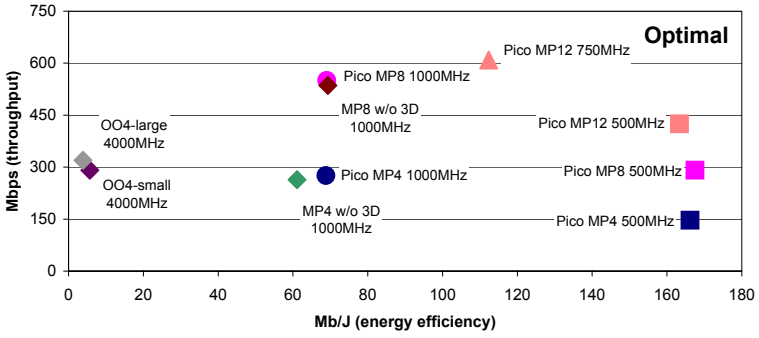
Additionally from Figure 5.19, 5.20, 5.21, we find our PicoServer architectures clocked at modest core frequency—500MHz are 2 $\sim$ 4$\times$ energy efficient than conventional chip-multiprocessor architectures without 3D stacking technology. The primary powersavings can be attributed to 3D stacking technology that enables a reduction in core clock frequency while providing high network bandwidth. A sweetspot in system-level energy efficiency for our plotted datapoints can also be identified within our PicoServer architectures when looking at Pico MP4/8/12-500MHz. These sweetspots in energy efficiency come from diminishing return in throughput improvement for increasing parallel processing width. The increase in parallel processing width—adding additional cores, raises many issues related to inefficient interrupt balancing, kernel process/thread scheduling and resource allocation that result in diminishing return.



(a) SURGE

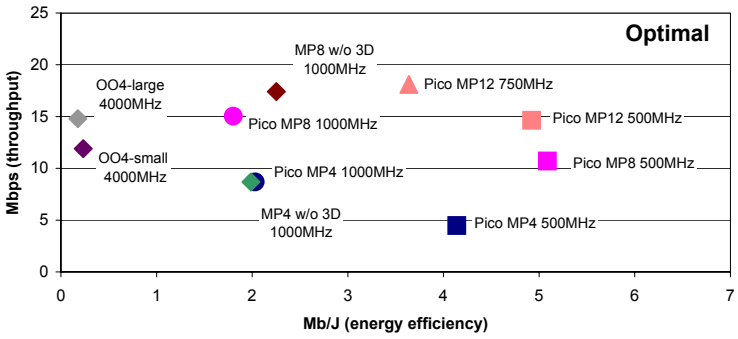


(b) SPECweb99

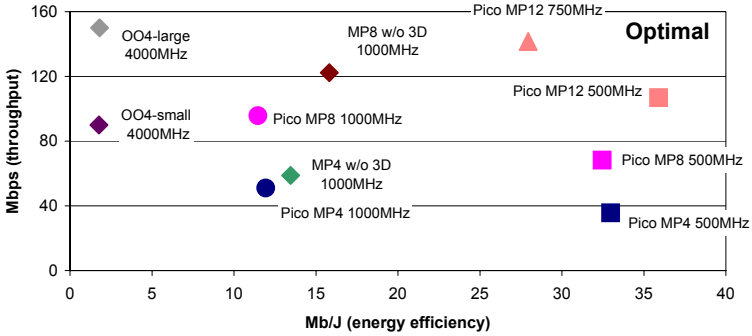


(c) Fenice

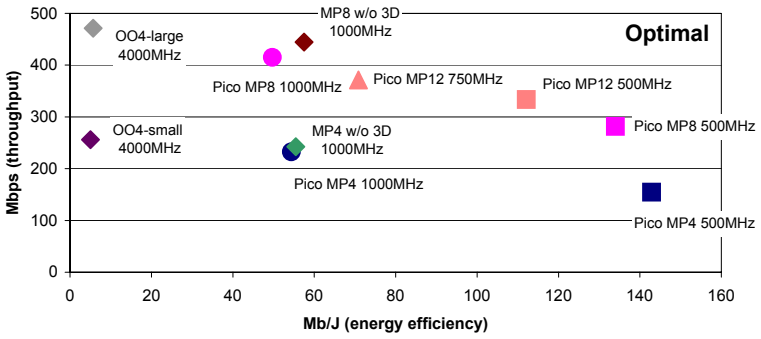
Figure 5.19: Energy efficiency, Performance pareto chart generated for 90nm process technology. 3D stacking technology enables new CMP architectures that are significantly energy efficient. (SURGE, SPECWeb99, Fenice)



(a) SPECweb2005-bank

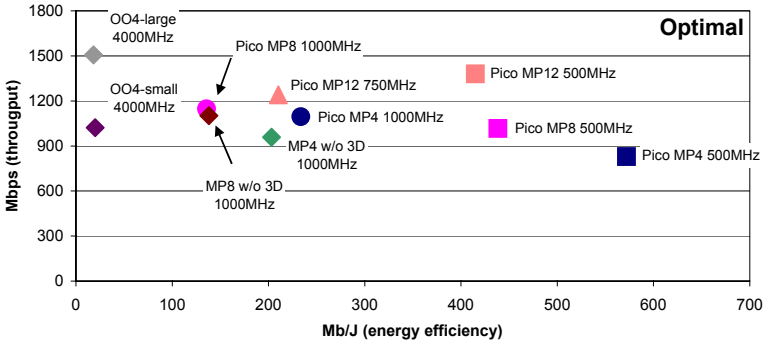


(b) SPECweb2005-ecommerce

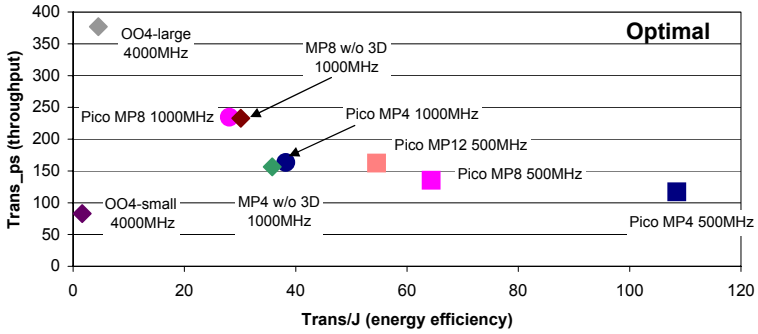


(c) SPECweb2005-support

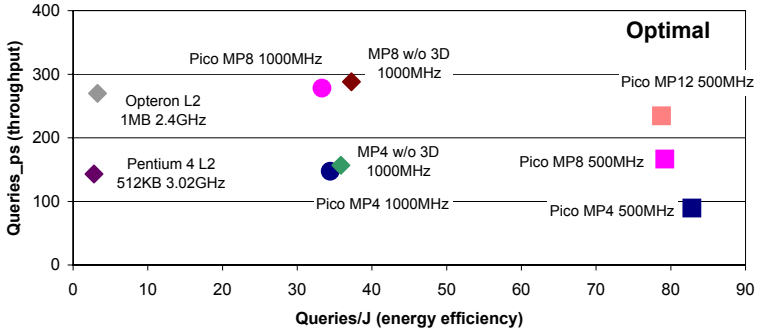
Figure 5.20: Energy efficiency, Performance pareto chart generated for 90nm process technology. (SPECWeb2005)



(a) dbench



(b) TPC-C



(c) TPC-H

Figure 5.21: Energy efficiency, Performance pareto chart generated for 90nm process technology. 3D stacking technology enables new CMP architectures that are significantly energy efficient. (dbench, TPC-C,TPC-H)

CHAPTER VI

INTEGRATING FLASH ONTO THE SYSTEM MEMORY ARCHITECTURE

In the previous chapter, we discussed how 3D stacking technology could reduce the power consumption of logic components and I/O interfaces in a server. This chapter discusses how the system memory hierarchy and disk drives can consume less power while delivering similar throughput when leveraging emerging memory devices like Flash. We discuss how the off-chip system memory could be architected to consume a reasonable amount of power. Off-chip memory is especially important for Tier 3 workloads that display long-tailed access behaviors.

6.1 Off-chip DRAM

This section explains the off-chip system memory hierarchy of current server architectures and emphasizes the role of system memory as disk cache—also known as page cache in Linux. Aside from storing code and data, server workloads use a large amount of system memory as disk cacheto mitigate long disk access latencies. Disk caches are an important way of fully utilizing a server and delivering high throughput. We recall that the memory usage analysis in the previous chapter showed that a large amount of system memory is used as disk cache in server workloads. For conventional architectures that do not have a portion of system memory on-chip and use the on-chip memory as a cache, the disk cache would occupy a significant amount of total off-chip system memory. For platforms using 3D stacking technology like PicoServer which have a large on-chip memory, off-chip system memory will function only as a disk cache.

We start off by describing the uniform memory architecture (UMA) commonly found in conventional memory systems. We discuss the problems with UMAs in terms of power consumption. We then show the off-chip memory architecture in PicoServer—it is a non-uniform memory architecture (NUMA). We introduce the

potential importance of NUMAs to build energy efficient architectures like PicoServer. Finally we build a case for using NUMAs in general to reduce power consumed by system memory. We show that Flash can potentially be used in NUMAs.

6.1.1 Off-chip DRAM in conventional platforms

Conventional platforms that use off-chip DRAM as system memory and on-chip SRAM memory as caches primarily assume all accesses to off-chip DRAM is uniform (UMA). This is not entirely true but given a single platform is built on top of a single socket processor, we can typically say in most cases it is a UMA. With UMAs, it isn't difficult to allocate and deallocate memory from the software's perspective. Conventional UMA platforms spend much less time and effort in memory management when writing software compared to a non-uniform memory architecture (NUMA). In fact, many applications today are written on top of a UMA model and require additional NUMA APIs to scale with NUMA platforms. UMA platforms also don't require memory to be allocated in a contiguous manner. This also applies to memory allocated in a disk cache. Because the allocated memory in a disk cache in a UMA is distributed, one drawback with UMAs is the difficulty in setting some portions of memory in low power state. This is particularly important for servers with 10's$\sim$100's of gigabytes of system memory. System memory power in platforms for these memory sizes consume an appreciable amount of overall system power. To set a particular memory chip or module to low power state, the OS must be modified to allocate and deallocate memory contiguously and at times perform frequent migration similar to garbage collection. Any request to allocate memory space would result in power-aware memory management.

6.1.2 Off-chip DRAM in PicoServer

Because PicoServer stacks hundreds to thousands of megabytes of memory on-chip, off-chip DRAM in PicoServer is likely to function only as a disk cache. Since the on-chip DRAM comprises of code, data and a small amount of disk cache, adding external memory to this platform would imply increasing the disk cache size. It essentially becomes a NUMA platform, which will require a rewrite of the OS kernel to properly utilize the off-chip DRAM. To minimize this effort, it would be desirable to only fix code that manages the disk cache. As we had shown in Chapter V, PicoServers are much more energy efficient than conventional platforms. To bring out PicoServer's full potential, NUMA platforms would definitely be a better choice.

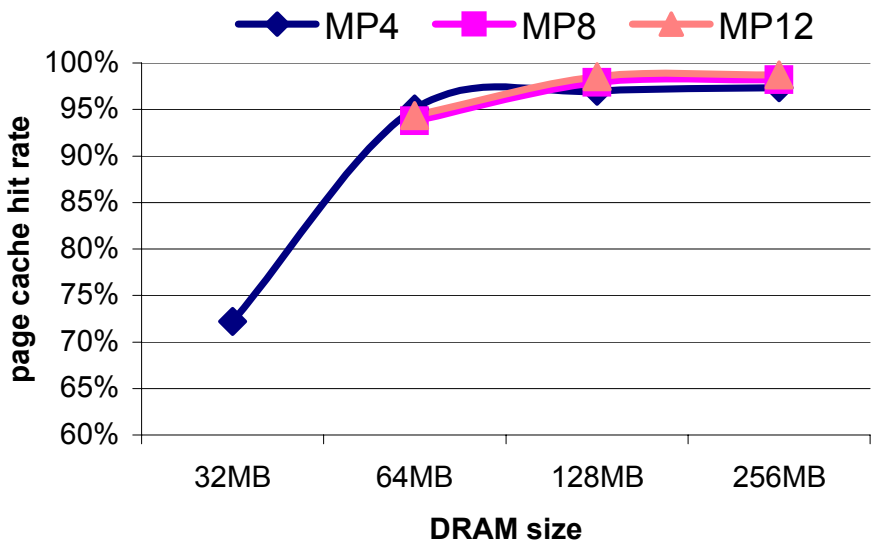
It will increase the disk cache size of PicoServers a lot and enable us to fully utilize the cores in PicoServer.

6.1.3 A case for non-uniform memory architectures to reduce power

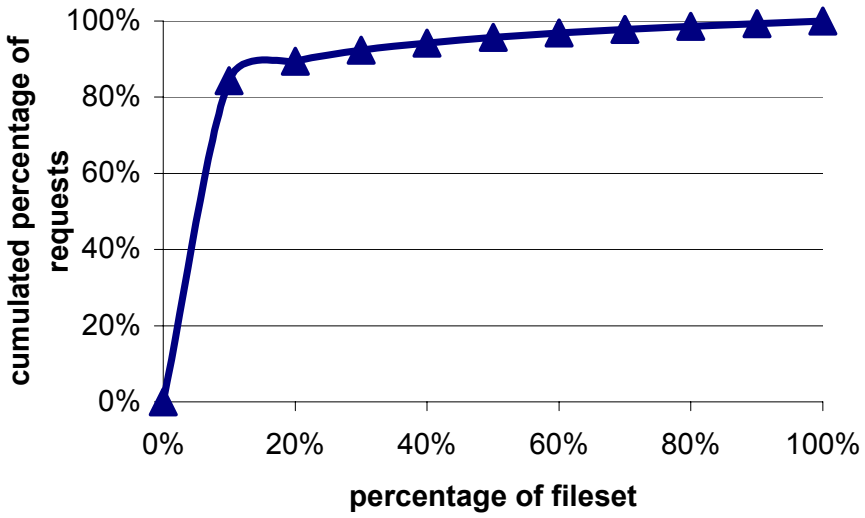
This section explains how NUMAs are more energy efficient than UMAs in respect to recent trends in DRAM. Conventional system memory (DRAM) has improved its storage density by scaling process technology. Throughput though has not improved at a faster rate than density from generation to generation. With the introduction of the DDR interface and deviations of this interface, DRAM delivers higher throughput— up to several gigabytes per second today. These improvements though have not come at a cheap cost. Power consumption in DRAM with high throughput has grown appreciably and in some cases require heat sinks. For system memories that are built for servers, larger memory sizes must also be considered. For example, future generations from the Ultrasparc T2000 series expect to be equipped with 100's of gigabytes of memory. This translates into power budgets that easily exceed 100Watts. To reduce the operation costs of system memory and potentially leverage alternative memory technologies, we look at how the off-chip system memory could be rearchitected.

By analyzing how the OS fills and empties the disk cache, one could identify potential slacks in access latency that the system memory could take advantage of especially for server workloads. Figure 6.1 shows one example of how the disk cache in system memory can be optimized. It shows that due to the access behavior to files in web server workloads—they display a short tailed distribution—a large portion of disk cache can tolerate access latencies of ten's to hundreds of microseconds. The lengthy access latency can be leveraged to look at other memory devices that display lower power and are built with higher storage density. One way to take advantage of this is to supply a lower clock frequency to the DRAM that is used as a disk cache. It will require a change in the memory management policy by the OS and force some sections in memory to be contiguously mapped. However, since lowering the clock frequency brings significant powersavings, it is worth the effort.

Taking a step further, one can consider integrating Flash memory which has recently caught much attention on server platforms—especially NAND Flash. NAND Flash is likely to be a cost effective solution than DRAM because NAND Flash cell's are less than 1/2 the size of DRAM. In a server platform, this enables a server to have much more system memory when integrating Flash onto a server. As we noted

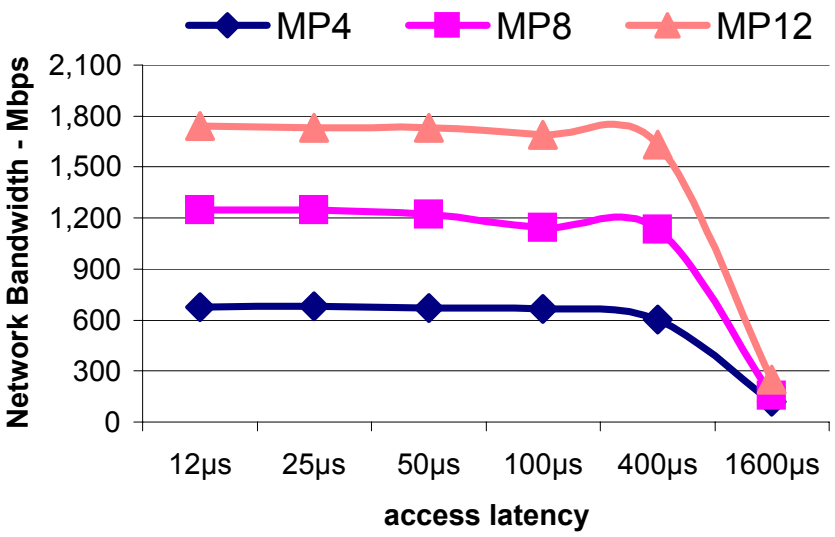


(a)

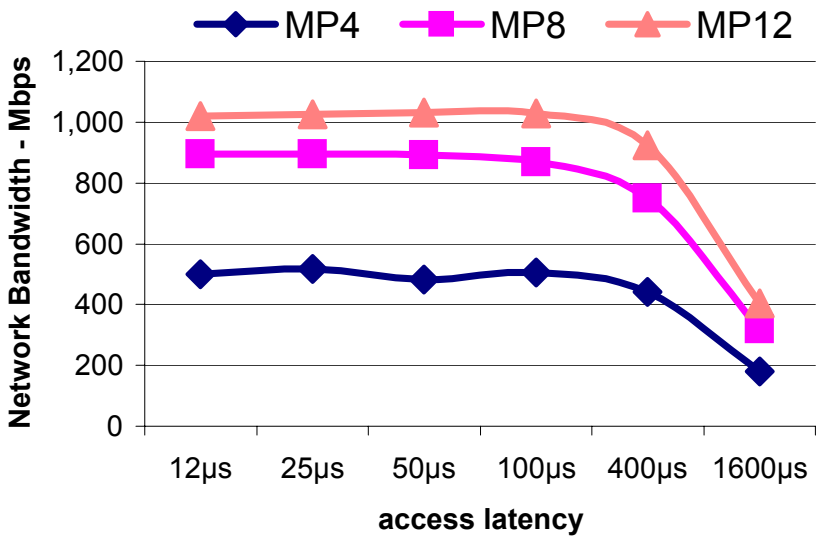


(b)

Figure 6.1: (a) Disk 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 6.2: Measured network bandwidth for full system simulation while varying access latency to a secondary disk cache. We assumed a 128MB DRAM with a slower memory of 1GB. We measured bandwidth for 4, 8, 12 multicore configurations. The secondary disk cache can tolerate access latencies of hundreds of microseconds while providing equal network bandwidth.

bigger disk caches potentially allow disks to be spun down longer. Due to the low idle power and high density, it is indeed a promising solution in reducing both disk and system memory power consumption. We believe Flash's are a better solution than slower DRAM or software managed power-aware memory, because it is likely to be cheaper and simpler to manage while maintaining a reasonable access latency. The next section discusses how Flash could be integrated onto a server platform.

6.2 Integrating Flash memory onto a server platform

Despite the benefits Flash can potentially have on a server platform, it requires additional support to guarantee reliability. Unlike DRAM, error correction code (ECC) is a mandatory requirement. We first describe how Flash memory is managed at the system level and how it can reduce power consumption while remaining reliable. We then, describe the architectural support for a programmable Flash memory controller to improve average Flash access latency while extending the overall lifetime of Flash. This is accomplished by observing the workload behavior and adopting dynamic ECC support, SLC to MLC or MLC to SLC mode changes and increasing write, erase time. Finally, we explain why a Flash based disk cache is more suitable than a Flash storage device in a server platform.

6.2.1 The FlashCache architecture

As we described in Chapter II, Flash memory is much denser than DRAM and likely to consume much less power. NAND Flash consumes less power due to lower clock frequencies and the NAND device structure. By observing the system memory usage behavior in server workloads, we realized a NUMA based system memory architecture can be beneficiary and consumes much less power while delivering equal throughput or even higher throughput.

Figure 6.3 shows an overview of how Flash could be used in a server platform. Compared to a conventional DRAM-only architecture, Figure 6.3 shows a hybrid system memory architecture composed of a relatively smaller DRAM and very dense Flash. It is a non-uniform system memory architecture with DRAM functioning as a level 1 disk cache and Flash functioning as a level 2 disk cache. A Flash controller is required to interface with Flash. We will provide a detailed description of our Flash controller in later sections. Additional data structures required to manage the Flash memory are placed in DRAM. Our Flash memory is called FlashCache.

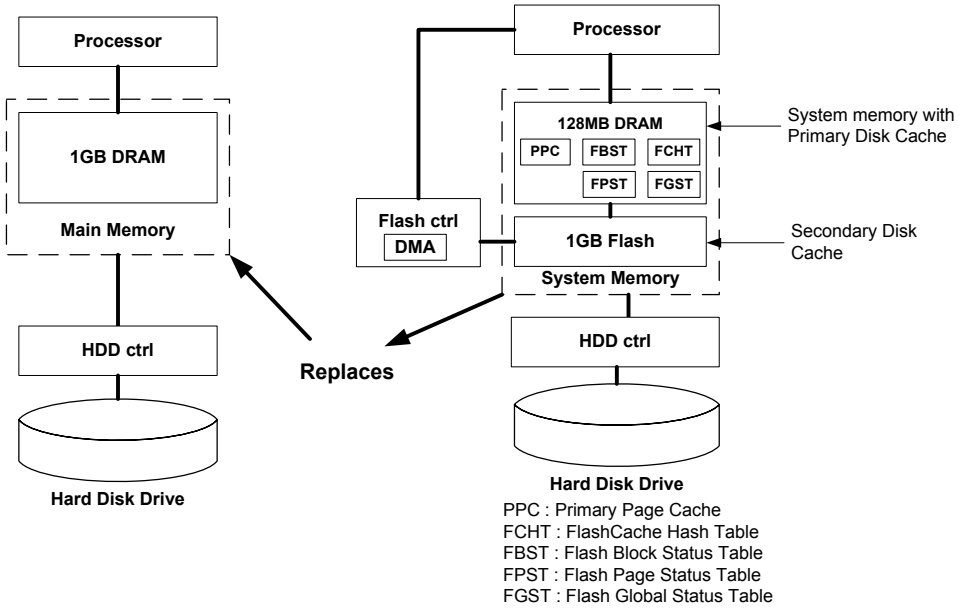


Figure 6.3: We show an example of a 1GB DRAM replaced with a smaller 128MB DRAM and 1GB NAND Flash memory. Additional components are added to control the Flash memory. The total die area required in our multichip memory is 60% the size of a conventional DRAM-only architecture.

FlashCache uses a NAND Flash rather than a NOR Flash due to the compactness and faster write latency found in NAND Flash. Write latency is critical for some server workloads like OLTP. 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 [19]. A NAND Flash is accessed in units of pages and blocks. A typical Flash page is 2KB in size and a Flash block is made up of 64 Flash pages (128KB). Random Flash reads and writes are performed on a page basis and Flash erasures are performed per block. A Flash must perform an erase on a block before it can write to a page belonging to that block. The fact that Flash erasures occur in blocks contributes to the high erase latency.

We employ Flash as a disk cache rather than a generic system memory or a storage device. Flash is unsuitable as a generic system 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. The case for using Flash as a disk cache and not as a storage device for servers will be made in later sections in this chapter. A disk cache only requires cache tags for each Flash page, suggesting a lower memory overhead than a file system. File systems require

Type	Field	Description
`unsigned long long`	`disk_addr`	`LBA location in disk drive`
`unsigned long long`	`flash_addr`	`location in Flash`

Table 6.1: The fields of the FlashCache tag structure which are entries to the FCHT

a tree structure filled with file location nodes. Data structures used in a FlashCache block and page management are read from the hard disk drive and loaded to DRAM to reduce access latency and mitigate wear-out.

In a conventional DRAM-only architecture that uses a single level DRAM for disk 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. We found the search time to be 300∼400 nanoseconds, which suggested, we could leave the block management tables for Flash to be fully-associative. The next subsections describe how Flash is managed.

FlashCache Hash Table(FCHT) for tag lookup

The FlashCache Hash Table (FCHT) is a memory structure that holds tags associated with Flash pages. This table exists in DRAM. A tag is made up of a LBA field and a Flash address field shown in Table 6.1. The LBA field points to the location in the hard disk drive and is used for determining whether the Flash holds this particular location in the hard disk drive. The corresponding Flash address field is used to access Flash. If a hit occurs in the FCHT, the corresponding Flash address field is used to access Flash. In many instances, a hit occurs in our FlashCache and the Flash location containing the requested file is sent to the primary disk 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 fully associative table accessed quickly by performing a hash. Wear-level awareness is managed at the block level. A Flash block status table(FBST) is used to profile the number of writes and erases performed on a particular block.

Type	Field	Description
`unsigned long long`	`erases`	`number of block erases`
`unsigned int`	`wear_out`	`degree of block wear-out`
`unsigned char`	`erase_time`	`erase time of block`

Table 6.2: The fields of the flash_block_status structure which are entries to the FBST

Flash block status table(FBST)

The Flash block status table(FBST) located in DRAM maintains the current status of a Flash block. The fields existing in a single entry in the FBST is described in 6.2. It holds the wear-out degree of a Flash block. It also stores the number of writes and erases performed on this block as well as other metrics which are used by the Flash controller. The degree of wear-out is a cost function generated from observing the erase count as well as the overall status of a Flash page (stored in a Flash page status table) belonging to a Flash block. In this study, we defined a degree of wear-out for Flash block i as follows based on the studied behavior of a Flash cell due to wear-out.

$$wear\_out = N_{erase,i} + k_1 \times Total_{ECC} + k_2 \times erase\_time_i + k_3 \times Total_{SLC\_MLC}$$

where $N_{erase,i}$ is the number of erases to block i, $Total_{ECC}$ is the total ECC code strength of a block, which is the sum of ECC code strength for each page in a block, $erase\_time_i$ is the erase time for block i, $Total_{SLC\_MLC}$ is the total number of pages in SLC mode due to wear-out not performance. k_1, k_2, k_3 are positive weight factors. The weight factors in this study were determined by observing the wear-out behavior of Flash. k_2 is the largest weight factor since erase time is the most sensitive to wear-out which is shown in later sections. k_3 is the second largest weight factor with k_1 being the smallest.

The number of erases and writes is related to the number of FlashCache evictions and FlashCache writes. We will later explain how the FBST as well as other data structures are applied to the Flash controller. Every time a block erase is performed its Flash block status table entry located in DRAM is updated. There are other instances when a FBST table is updated in respect to the Flash controller which is described later. The FBST determines whether a block is *hot* or *cold*. The `wear_out` field of the block determines the temperature.

Wear-level aware cache replacement—managing FlashCache misses

The drawback of using Flash is wear-out. Compared to DRAM, Flash can only be written a fixed amount of times. Table 2.5 shows the ITRS projection for Flash endurance. A 10× improvement in endurance is expected every 5∼6 years. Endurance improvement is attributed to the use of better material. Our FlashCache replacement algorithm is wear-level aware. The Flash block status table(FBST) exists to assist in wear-level management.

Wear-level management for the FlashCache is performed on FlashCache misses. At the block level, we initially select a block to be evicted using an LRU policy. However, if this block uses a temperature that exceeds that of the coldest block by a predetermined threshold, then the block corresponding to the minimum wear-out (*coldest block*) is evicted to balance the wear-level. Before we evict the *coldest block*, its content is migrated to the *hot block*. The degree of wear-out of a Flash block is determined by observing the entries in the Flash Block Status Table (FBST).

Splitting FlashCache into read optimized and write optimized—Isolating FlashCache writes

Flash writes also need to be wear-level aware to guarantee reasonable Flash lifetime. Due to this constraint, *out-of-place* writes are commonly used to mitigate wear-out. Out-of-place writes treat Flash as a log and append writes to the end of the log. However, naively managed out-of-place writes hurt the performance of a Flash based disk cache. They tend to increase garbage collection(GC), incurring a significant performance overhead. It also increases the overall Flash based disk cache miss rate due to invalid Flash pages that are generated from out-of-place writes. Because the read access latency to a Flash based disk cache is critical in utilizing the overall system, it is desirable to limit GC and maintain a reasonable access latency to Flash.

To make this possible, it is apparent that dividing the Flash into a read-optimized disk cache and a write-optimized disk cache seems vital in improving disk cache performance. Splitting the disk cache into a read and write disk cache, means a read cache is less susceptible to out-of-place writes, which reduce the read cache capacity while increasing the risk of garbage collection. Read critical Flash blocks are located in the read-optimized disk cache that only evict Flash blocks and pages on disk cache misses. The write optimized disk cache captures writes to the Flash based disk cache and supports out-of-place writes as well as garbage collection. Wear-leveling is applied globally to all regions in the Flash based disk cache. Because writes in a Flash are typically managed with a log-structure, it is also possible to use this write

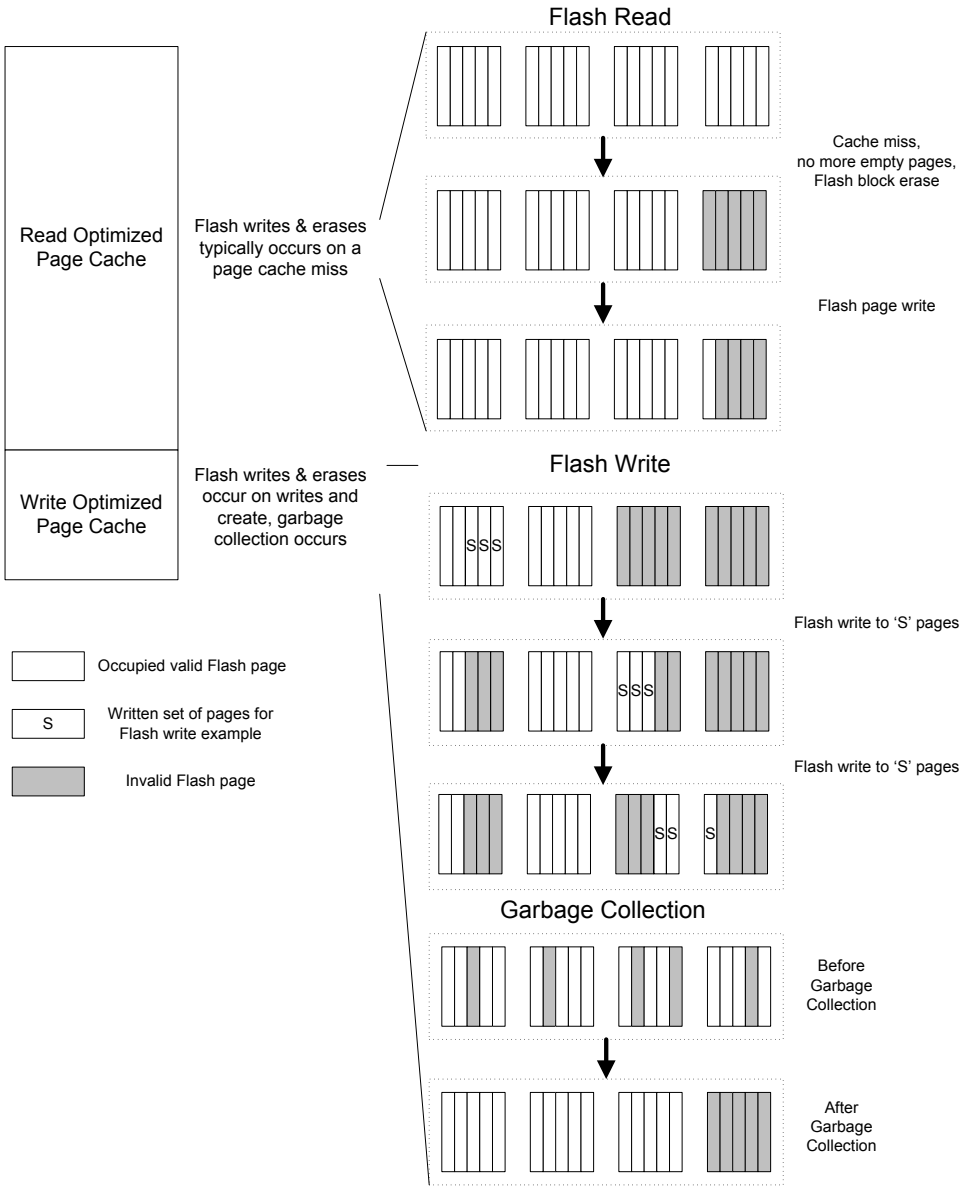


Figure 6.4: Splitting FlashCache into a read optimized and write optimized cache.

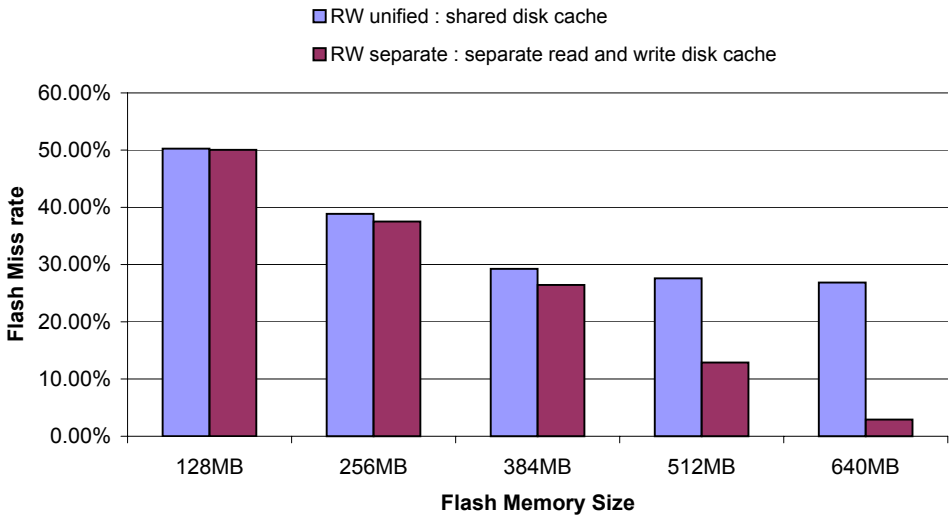


Figure 6.5: A miss rate comparison for a unified FlashCache and a read, write separated FlashCache. Based on the observed write behavior, 90% of Flash is dedicated as a read optimized cache and 10% of Flash is dedicated as a write optimized cache

optimized disk cache for fast recovery. This comes in handy for database workloads that support a log file to reduce the update frequency to disk and allow recovery using log files. Figure 6.5 shows the miss rate improvement obtained by splitting into a read optimized and write optimized disk cache. The disk cache miss rate indirectly shows the improvement in average access latency and indeed shows the benefit of splitting the Flash based disk cache.

Accessing the FlashCache

In this section, we discuss how hits, misses and updates are handled in a Flash based disk cache. When a file read is performed at the application level, the OS searches for the file in the primary disk cache located in DRAM. On a primary disk cache hit in DRAM, the file content is accessed directly from the primary disk cache (no access to Flash related data structures). On a primary disk cache miss in DRAM, the OS searches the FCHT to determine whether the requested file currently exists in the secondary disk cache. If the requested file is found, then a Flash read is performed and a DMA transaction is scheduled to transfer Flash content to DRAM. The requested address to Flash is obtained from the FCHT.

If a read miss occurs in the FCHT search process, a block is selected for eviction. The selection process considers wear-level at the block level. Wear-leveling is

performed based on section 6.2.1. Concurrently, a hard disk drive access is scheduled using the device driver interface. The hard disk drive content is copied to the primary disk cache in DRAM. The corresponding tag in the FCHT belonging to the read optimized disk cache is also updated.

File writes are more complicated. If we update a file, we update/access the page in the primary page cache and this page is later scheduled to be written back to the secondary disk cache and later to the disk drive. When writing back to Flash, we first determine whether it exists on Flash by searching the FCHT. If it is found in the write optimized cache, we update the page by doing an *out-of-place* write to the write optimized cache. If it is found in the read optimized cache, then we invalidate this page and allocate a page in the write optimized cache. If it is not found in the Flash, we just allocate a page in the write optimized cache. The OS then determines whether garbage collection (GC) of the write optimized Flash disk cache is necessary. If GC is required, it is performed in the background. When erasing a block during GC, wear-level management is performed on all blocks in a Flash based disk cache. The disk is eventually updated by flushing the values on the write optimized disk cache. By separating the Flash based disk cache into a read optimized and write optimized disk cache, we are able to reduce the amount of blocks that have to be considered when doing write triggered garbage collection.

6.2.2 Architecting a programmable Flash memory controller

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

Fortunately, these problems can currently be dealt with or are expected to be resolved through interleaving and improved interfaces. The blocking property of Flash writes can be solved by distributing writes. We can schedule Flash 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, we can prevent a Flash write from occurring when Flash reads are requested. The lazy writeback scheme allows writebacks to occur mostly when the Flash is not accessed for reads. The bandwidth limitation is addressed through improved Flash clock frequencies similar to that found in DRAM interfaces. In the long term we expect to see more direct solutions—non-blocking features. Implementing multibanked Flash memory that supports RWW are possible solutions. Flash vendors are making efforts to support interleaving by banking Flash memory and introducing interface roadmaps that are expected to satisfy the bandwidth requirements. For example, DDR2 interface for Flash is projected to be available around 2011 based on roadmap projections by Samsung[54]. 3D stacking technology[44] also reduces the transfer latency to one bus cycle resulting in an improvement in Flash memory bandwidth.

In addition to providing acceptable throughput and latency, Flash controllers need to be programmable. It is primarily due to wear-out. Flash memory wear-out can be more directly visible to the user and software and hardware techniques must be available to mitigate wear-out. We provide a solution that is programmable and extends the lifetime of Flash significantly.

Figure 6.6 shows a high-level block diagram of a programmable Flash controller. It is composed of 3 main components 1)an encoder and decoder for error correction and detection; 2)a density controller; and 3)a Flash program/erase time controller. Software based descriptors that exist in the Flash Block Status Table (FBST), Flash Page Status Table (FPST) and Flash Global Status Table (FGST) (see section 6.2.1) are used to program the Flash controller parameters. The cost function in determining what parameters to use for the Flash memory controller is overall average access latency. Overall access latency in a Flash memory can be written as follows:

$t_{access} = t_{hit} \times (1 - p_{miss}) + t_{miss} \times p_{miss}$

where t_{access} is the Flash memory average access latency, t_{hit} is the average hit latency of the Flash memory cell configuration and error correction code configuration, t_{miss} is the miss latency depending on the access latency to the hard disk drive and the Flash program/erase timing constraint, finally p_{miss} is the miss rate due to unrecoverable errors or capacity misses in the Flash. p_{miss} is related to the density of a Flash memory and the working set size of an application. In a nonprogrammable Flash

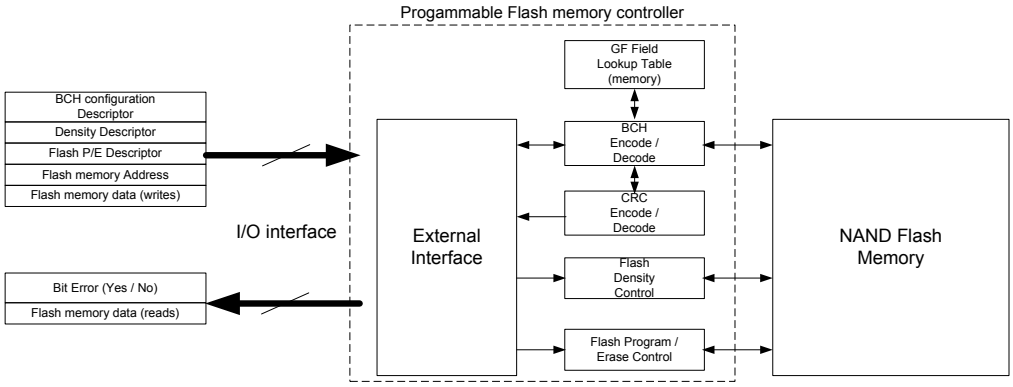


Figure 6.6: High-level block diagram of a programmable Flash memory controller

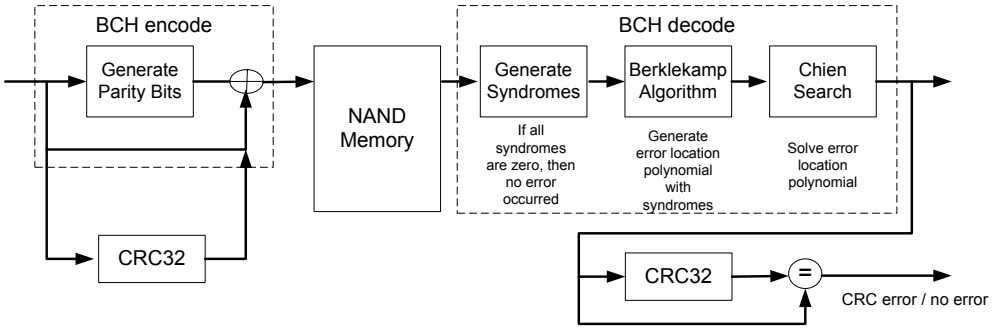


Figure 6.7: High-level block diagram of a BCH and CRC encoder/decoder interfacing with NAND Flash memory.

memory controller t_{hit}, t_{miss} would be fixed and p_{miss} would gradually increase as Flash memory wears out (faulty blocks are simply disabled).

Error Correction Code support

One way to recover from errors in Flash is to use error correction code. This section describes the error correction and detection scheme used in a Flash controller. We also show that a programmable Flash controller can control the error correction code strength to improve overall access latency to Flash while extending overall Flash lifetime.

Error correcting codes (ECC) are widely employed in digital storage devices to mitigate the effects of hard (permanent) and soft (transient) errors in storage cells. It has been a well-established area in disk drives and studied for several decades. Flash uses linear block codes like Bose, Ray-Chaudhuri, Hocquenghem (BCH) code due to the strong code strength and acceptable decode/encode latency. For example, a

recent paper [72] implements a 5-bit BCH error correcting code on words of size 2102 bytes. To effectively detect errors, CRC codes have been commonly used in addition to BCH error correction. Our architecture shown in Figure 6.7 uses a BCH encoder and decoder to perform error correction and a 32 bit CRC checker to perform error detection. We provide a detailed description in the next subsections.

BCH encoder and decoder

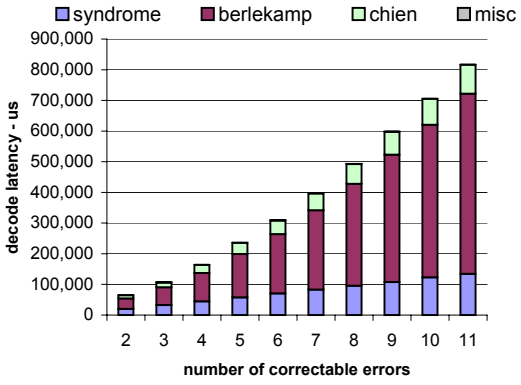
BCH code is a well established coding technique to recover from random errors. Hamming codes were found to be a special case for BCH code which corrects single errors, Reed-Solomon codes have been recognized as a non-binary code for BCH code. Therefore, much of the insight gained from BCH can be applied to implementing Hamming or Reed-Solomon codes which are known alternative channel codes applied to memory devices. A t-error-correcting BCH code must agree with the following parameters:

Block length: $n = 2^m - 1$,
Number of parity-check digits: $n - k \leq mt$,
Minimum distance: $d_{min} \geq 2t + 1$
For any positive integer $m (m \geq 3)$ and t $(t < 2^{m-1})$

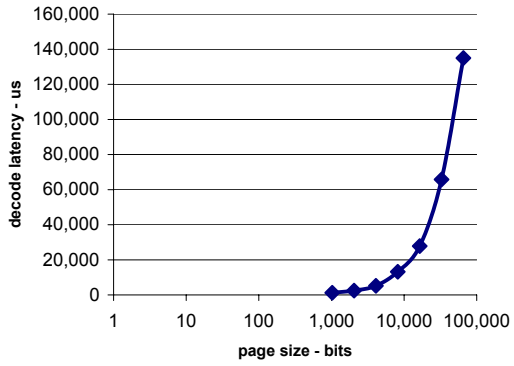
For example, in a 2 error-correcting BCH code of length 255, 239 bits used as the message bits (the actual data) and 16 bits are used as the parity check bits. Roughly speaking, the number of parity check bits increase linearly for a fixed code length.

Figure 6.7 shows an implementation of our BCH encoder/decoder using well-known algorithms. The Berlekamp and Chien Search algorithms in the decoder are widely used due to their simplicity. These methods have been proven to be an effective iterative technique in decoding BCH code [67]. We conducted a bottleneck analysis to understand whether an accelerator is required to satisfy reasonable error correction latency. We implemented the BCH encoder and decoder in C and measured the amount of time spent encoding and decoding BCH code on a 3.4 GHz Pentium 4 system. Although not highly optimized, the source code was sufficient to determine the approximate clock frequency necessary to provide a certain word access latency. This delay is in addition to the intrinsic read or write delay of a Flash cell.

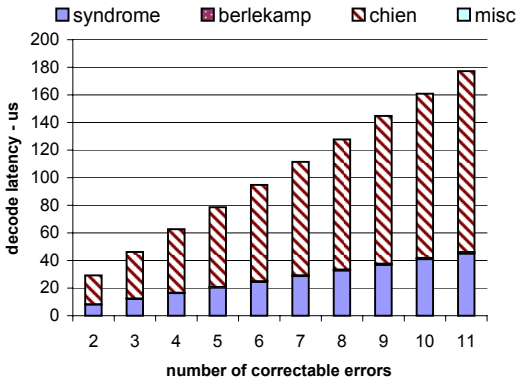
Figure 6.8 shows access latencies for a general-purpose CPU platform, plus that of a projected general-purpose CPU with an accelerator. Our projected latencies were based on the circuit design in [72]. Figure 6.8(b) also shows the impact of varying block size to latency on a general-purpose CPU platform. It is clear that these penal-



(a)



(b)



(c)

Figure 6.8: BCH decode execution time on (a) x86 assuming page size of 2KB, (b) shows the execution time for varying block size on a 2 error correcting BCH decode executed on a x86. (c) embedded processor (inorder 100MHz) with highly parallelized modular arithmetic support, Berlekamp acceleration engine and highly parallelized Chien search engine. Figures clearly suggest there should be an accelerator for ECC.

ties can severely compromise performance, thus an accelerator is necessary. These accelerators primarily improve the modular arithmetic and memory alignment found in BCH code. Fortunately, for BCH decoders can highly parallelized with ease resulting in accelerator engines to be multiple components simultaneously executing similar operations. To support programmability and adaptive ECC strength, a combination of a general-purpose CPU and a specialized functional unit for ECC acceleration is required in our BCH encoder/decoder.

Our implementation uses a 2^{15} entry finite field lookup table as well as 16 finite field adders and multipliers as accelerators to implement the Berlekamp and Chien Search algorithm (16 instances of Chien search engines). BCH code uses finite field operators, which are sufficiently different from standard arithmetic operators that they cause a bottleneck in a general purpose CPU without an accelerator. We limit the programmability to a fixed block size (2KB) to avoid memory alignment with different block sizes and limit the maximum number of correctable errors to 12. A programmable BCH encoder/decoder that supports varying block size and varying correctable number of errors requires careful memory alignment and additional finite field lookup tables and reconfigurable multipliers. This suggests there is a large design space to explore when investigating the trade-off between memory alignment and BCH encode/decode efficiency. We leave this to future work that this area covers a large space.

CRC checksum

One of the drawbacks of BCH code is the inability to perfectly detect more than the correctable number of errors. In many cases, if the number of errors exceeds the amount a BCH code can handle, the Chien search would generate no roots, implying that more errors occurred. Unfortunately, false positives may occur, where the Chien's Search algorithm generates root results that cause a decoding error. As a result, CRC codes are necessary to improve error detection capability.

Using CRC code with error correction codes is a common way of guaranteeing reliability in many applications. CRC codes are found in applications such as Ethernet 802.3, MPEG2 and ATM protocols. CRC codes are capable of covering a wide range of error patterns. For example, CRC32 codes are able to detect up to 32bits of bursty errors and even more distributed errors. Therefore, we found it unnecessary to architect a programmable CRC and instead use a well-optimized hardware implementation of a CRC32 functional block. Our design compiler results showed it occupied an insignificant amount of die area and performance overhead (unit in tens

of nanoseconds). This agrees with other implementations such as [50].

Impact of BCH code strength to Flash lifetime

Because ECC comes at such a high cost, we investigated the relationship between ECC strength and Flash lifetime. This section provides an analytical analysis on the impact of code strength versus Flash lifetime. As it was shown in Chapter II, Flash lifetime depends on the thickness of oxide. We begin by assuming the probability distribution function(pdf) $f_{tox}(X)$ of oxide thickness on a silicon die to follow a gaussian distribution. This assumption has been commonly used in many studies related to process variation and bases it's principles on the central limit theorem.

$$f_{t_{ox}}(X) = \frac{1}{\sqrt{2\pi}\sigma_{ox}} e^{-\frac{(x-m_{ox})^2}{2{\sigma_{ox}}^2}} \tag{6.1}$$

where m_{ox} is the mean oxide thickness and σ_{ox} is the oxide standard deviation, Further, we assume the $3\sigma_{ox}$ value of oxide is 15% of the mean oxide thickness.

Next, we identify a relationship between the Flash lifetime and oxide thickness. In other words, represent Flash lifetime as a function of oxide thickness. From Figure 2.4(b) in Chapter II, we can see for a fixed portion of worn out cells, oxide thickness t_{ox} displays an exponential relationship with Flash lifetime. Using this relationship we can define Flash lifetime W as follows:

$$W = 10^{C_1 \cdot t_{ox}} \tag{6.2}$$

where C_1 is a constant. We will call this the *exponential model*. For comparison reasons, we define a second model that assumes

$$W = C_2 \cdot t_{ox} \tag{6.3}$$

where C_2 is a constant. We call this the *linear model*.

Up to this point, spatial variation is not modeled. But, adding a spatial correlation coefficient into the pdf function is difficult to analyze. Luckily, because error correction is handled in pages (a group of horizontal bits) that are several kilobytes, the horizontal variation can be excluded and we only need to consider vertical variation. Because we handle a Flash in units of pages, when analyzing the impact of ECC, we are only concerned about the number of errors not where the errors occur in a page. We introduce another random variable E that represents the number of bad cells per Flash page at wear-out w. This variable allows us to capture the values that are required in analyzing the impact of ECC. If there were no spatial variation

at all, the variation of random variable E would be 0. Thus, we can model the spatial variation as the variation of random variable E. Further, random variable E can be viewed as a random variable of mean $N_{page} \cdot p_w$ denoted as m_E, where N_{page} is the Flash page size plus additional header bits and p_w is the probability a cell can fail at wear-out w. The spatial variation can be related to σ_E, the standard deviation of random variable E. The probability p_w can be expressed as $p_w = F_W(w) = P(W \leq w)$. $P(W \leq w)$ is as follows:

linear model : $P(W \leq w) = P(C_2 \cdot t_{ox} \leq w) = P(t_{ox} \leq \frac{w}{C_2})$

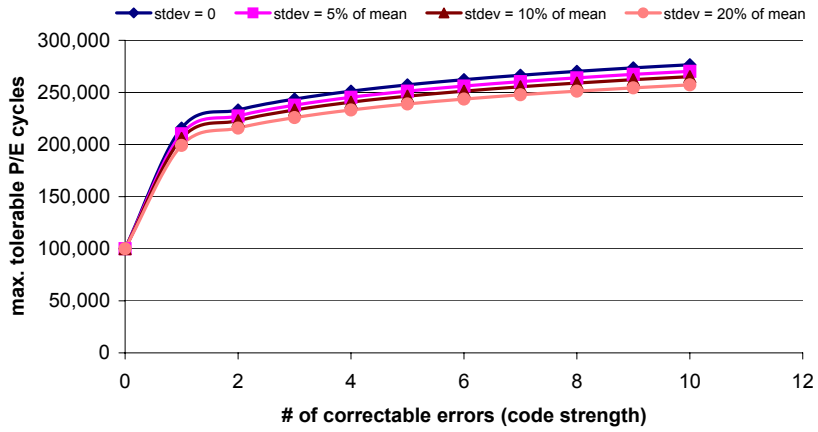
exponential model : $P(W \leq w) = P(10^{C_1 \cdot t_{ox}} \leq w) = P(t_{ox} \leq \frac{log_{10}(w)}{C_1})$

Variation of random variable E can vary based on the spatial correlation. Thus, we used multiple variations that are products of the mean to model spatial correlation. We selected standard deviations of 0, 5%,10%,20% of the mean.

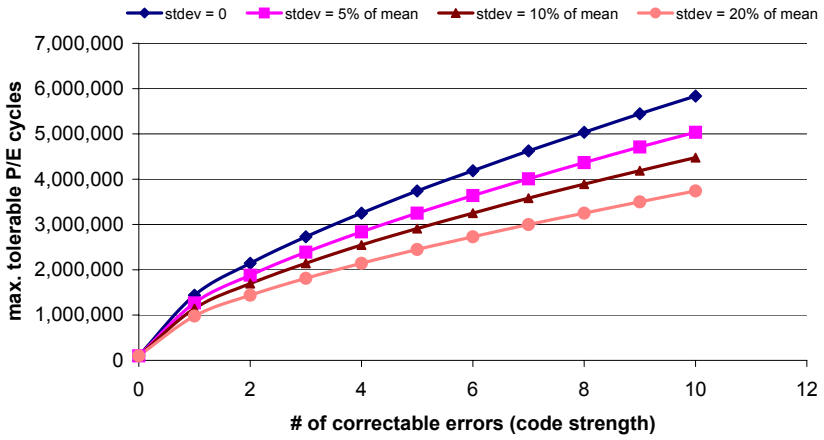
Using the *exponential model* and *linear model*, we plotted the relationship ECC strength versus Flash P/E cycles and available Flash pages versus Flash P/E cycles. Figure 6.9, 6.10 plots these results. As we can see, in both models, ECC strength extends lifetime and available number of Flash pages by quite a bit. When directly relating ECC strength to wear-out, we find that the exponential model wears out more gracefully. Spatial variation negatively impacts code strength because our assumptions in plotting Figure 6.9 assumed all Flash pages had to be recoverable for a certain ECC. As bad Flash cells display a higher spatial correlation, bad cells are located in groups resulting in an increasing number of pages that cannot recover using a weak ECC. Figure 6.10 shows the portion of available Flash pages for a ECC strength of 4. In this case, we find spatial variation helping graceful wear-out of storage capacity.

Density control

This section explains the need to provide dynamic density control in a Flash memory controller to improve overall Flash memory latency and lifetime. With the introduction of multi-level cells (MLC), Flash memory is able to improve storage density without the help of process scaling. However, due to the added complexity of an MLC Flash memory cell, it results in reduced read/write throughput and reduced endurance. MLC Flash memory cells take longer to program and read. Because the Flash memory state is stored using different threshold voltage levels, MLC Flash requires multiple threshold voltage levels. Multiple threshold voltage levels imply that the threshold gap between different logical values is reduced and is the main cause of reduced endurance. To mitigate this drawback, there has been work on enabling

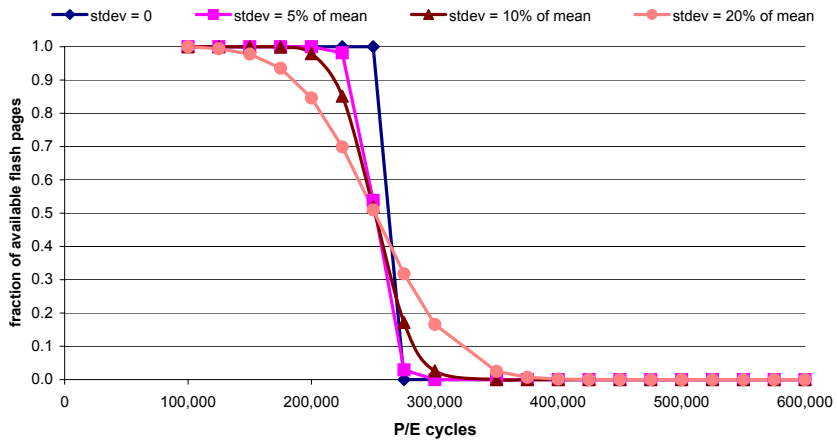


(a) linear

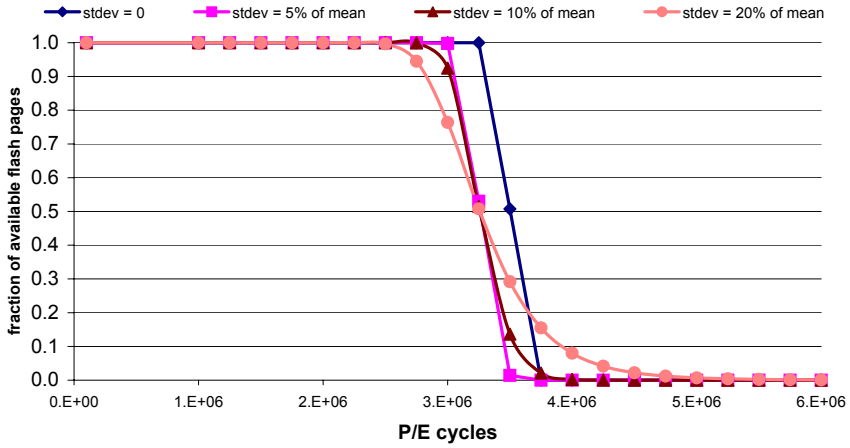


(b) exponential

Figure 6.9: Maximum tolerable Flash P/E cycles for varying code strength. Linear and exponential analytical models are considered. We assume Flash page size to be 2KB and first point of failure to occur at 100,000 P/E cycles.

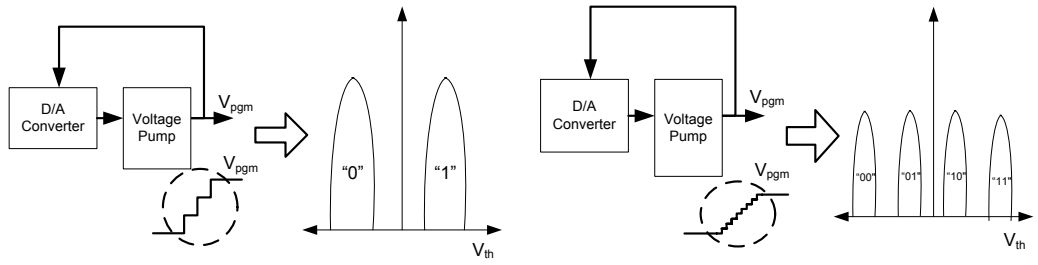


(a) linear



(b) exponential

Figure 6.10: Available Flash page versus Flash P/E cycles. Linear and exponential analytical models are considered,We assume Flash page size to be 2KB and first point of failure to occur at 100,000 P/E cycles.



(a) Single Level Cell (SLC) (b) Multilevel Cell (MLC)

Figure 6.11: Dual mode Flash memories are possible for MLC by controlling the program voltage level

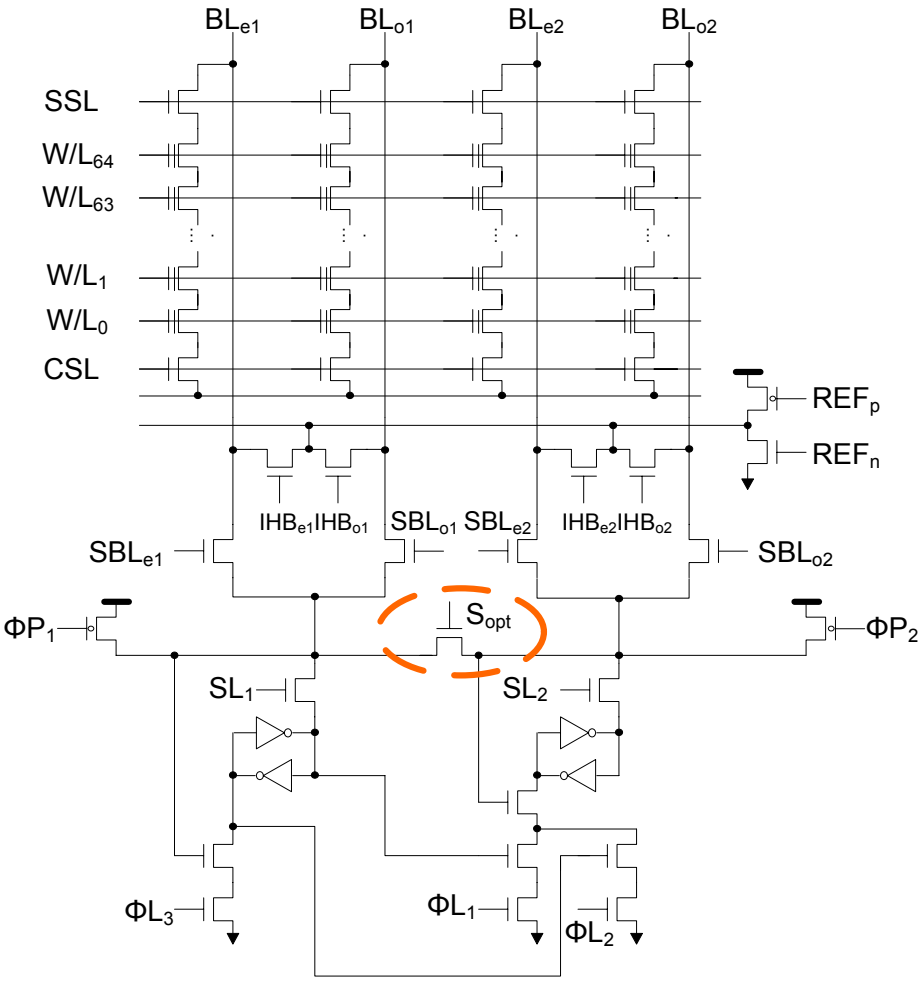
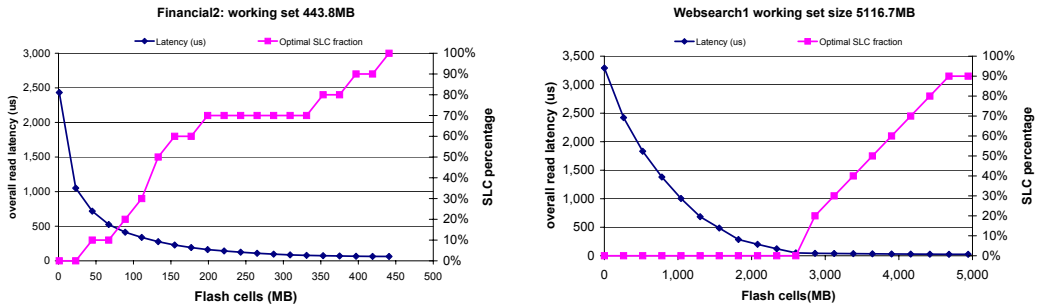


Figure 6.12: Simplified schematic diagram of sense and latch buffer with single-bit-per-cell option transistor[35]. The circled transistor is the single-bit-per-cell option transistor



(a) Financial2 (b) Websearch1

Figure 6.13: Optimal FlashCache access latency and partition

MLC to operate in SLC mode [35] [64] to improve Flash memory endurance. Our programmable Flash memory controller assumes that one can dynamically control the density of a Flash memory at the page level by slightly modifying the sense amplifier circuitry found in a MLC [35] [64] providing a density selection control signal. Therefore, the primary role of a density controller is to indicate the mode of the requested page. The density configuration of the requested page can be found in the density descriptor, in the operating system software. Density control benefits Flash memory performance and endurance, because we are able to reduce access latency for frequently accessed pages and possibly improve endurance for aging Flash memory pages by changing MLC pages into SLC pages on demand.

To show the potential improvement of Flash memory performance by controlling density, we present a study using real disk traces. Using disk activity traces from [25] for financial and web search applications, we analyzed the average access latency for different SLC/MLC partitions, for several Flash cache sizes. Table 3.2 shows the latencies we assumed. For minimum latency, the most frequently accessed data is allocated to SLC storage. At this point, wear-out effects are ignored.

Because the higher density MLC cells exhibit a higher read and write latency than SLC, simply using MLC is not the most effective solution. We show that a hybrid allocation of SLC and MLC provides minimum access latency to files for equal area (number of Flash cells). Figure 6.13 shows the optimal average access latencies achieved for an optimal partition between SLC and MLC. As expected, when the size of the cache approaches the working set size, latency reaches a minimum using only SLC. Intuitively, this is because frequently accessed hotspots should reside in low-latency but low density cells. The optimal partition is dependent on the nature of the workload.

Program/Erase time control

This section shows the need to control the program and erase time using the program/erase controller. As it was shown in Chapter II, Flash wear-out causes a change in threshold voltage. Instead of using error correction code like BCH, to mitigate and improve Flash memory endurance one can increase the program and erase time so that some of the trapped electrons can be removed. Figure 6.14 shows an example of how this may be mitigated by increasing the erase and program time as one writes more to Flash. Longer erase and write time enables the worn out cells to be re-used again. After a certain point, we can see an exponential increase in program and erase time.

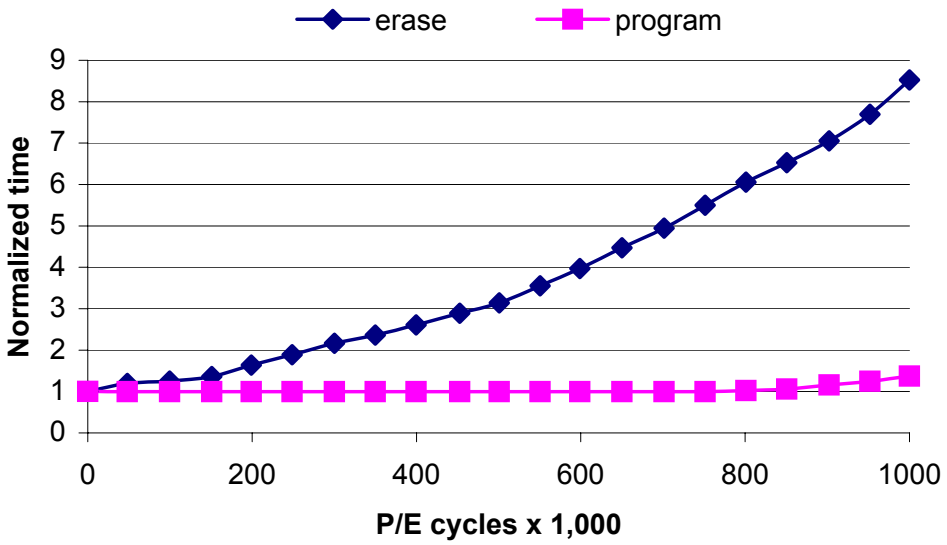


Figure 6.14: Program/Erase time control to mitigate Flash wear-out [73]

The erase time for a Flash memory block and the program time for a Flash memory page can be increased to slow the gradual decrease in the threshold voltage window gap. By configuring the program and erase time at the page and block level, it is possible to slow wear-out. Therefore, the program/erase time controller receives instructions through the program/erase descriptor to control this delay.

We only discuss the implications in Flash memory endurance due to program and erase cycles. The other gradual degradation found with Flash memory wear-out is the reduction in Flash memory retention time. The reason for retention time reduction comes from damage to the potential wall between the body and the floating gate. Retention time is less of a concern when viewing Flash as a disk cache. In later sections, along with many other reasons, this supports the case for using Flash as a disk cache for server applications.

Operating System support for Flash controllers

Operating System (OS) support for integrating the Flash controller must exist to deliver the proper Flash descriptor. We revisit the FCHT and FBST described in section 6.2.1. Two more data structures are introduced to assist in generating descriptors for the Flash controller. The Flash page status table(FPST) and the Flash global status table(FGST). They can be described in Table 6.3 and 6.4.

Type	Field	Description
unsigned int	access_counter	access frequency of page
unsigned char	valid	validity of page
unsigned char	ecc_strength	ECC code strength
unsigned char	slc_mlc	SLC/MLC mode of page
unsigned char	write_time	write time of page

Table 6.3: The fields of the flash_page_status structure which are entries to the FPST

Type	Field	Description
unsigned float	access_lat	overall average access latency
unsigned float	read_lat	overall average read latency
unsigned float	write_lat	overall average write latency
unsigned float	miss_rate	overall average miss rate

Table 6.4: The fields of the flash_global_status structure which are entries to the FGST

We will focus more on the fields found in the FPST and FGST entries. Block level data structures in the FBST manage the degree of wear-out of blocks by factoring the number of erases found in the FBST as well as overall page wear-out for pages belonging to this block. It also maintains the erase time configuration of a block. Page level data structures in the FPST includes the validity, bit error behavior and access frequency of a page. The overall status of a Flash (FGST), which we define as overall average access latency and disk cache miss rate, is also used in making decisions on how to configure a Flash page. The end result of these status entries is to generate a set of descriptors that are sent to the controller for reading, writing or erasing a Flash page. Of course, to determine whether the requested content is located inside Flash, the FCHT (a tag data structure) lookup must occur.

As we described in previous sections, wear-level management uses block wear-out which contain varying error correction code strength, density and program timing constraints. We discuss how the programmable Flash controller interfaces with the Flash when performing wear-leveling. We take advantage of the fact that typical manufacturer guaranteed erase cycle lifetimes of 10,000/100,000 are extremely conservative.

Wear-leveling is primarily performed at the block level as we described in the previous sections and attempts to ensure that small groups of blocks are not worn out quicker than others. For example, when the degree of wear out for block 'A' selected for erase exceeds a threshold compared to that of the degree of the least worn out block 'B', blocks are swapped and the least worn out block 'B' is chosen for erase instead.

To consider density control, the Flash controller must use the following algorithm when performing block replacement.

- Block 'A' is erased and the contents of block 'B' are migrated to block 'A'. Not all page configuration settings (ECC strength, density and latency controls) are migrated to block 'A'. Only page configurations from pages in block 'B' that have set the cell to SLC to improve performance are migrated to pages in block 'A'.

- Block 'B' is erased and pages due to a FlashCache miss are loaded into 'B' from the hard disk.

There may be more effective techniques to estimate the wear-out of a block. However, many of these techniques are time consuming and may not be worthwhile. Our philosophy behind wear-leveling is to equally wear-out each Flash block using a low-complexity scheme and handle wear-out of individual Flash blocks using error correction code strength, cell density and program/erase time.

How it works

This section describes how descriptors are generated for a programmable Flash controller and how we choose configurations. Since we have outlined how a FlashCache is accessed in section 6.2.1, we illustrate the sequence of events involved in interfacing with the Flash controller. On a FCHT read hit, the descriptors for that page are generated and sent to our programmable Flash memory controller to access the Flash memory. On a FCHT read miss, empty pages are allocated on FlashCache to cache data returning from disk. An empty page is a page which underwent an erase operation and requires programming. Descriptors for these empty pages are sent to the Flash controller when writing data. If there are no empty pages available, a Flash block is erased using the LRU and the wear-leveling policy maintained in the block-level data structures in FBST.

When accessing a Flash through the Flash controller,the page and block descriptors existing in the FPST, FBST can be modified. Based on the observed behavior of a Flash page, the descriptors can be updated to maximize overall performance and reliability. If it is updated, the updated characteristics of a page are applied on the next write access to that page and next erase to the block that the page belongs to.

There are two main events that trigger updates to descriptors for a Flash page and block located in the FPST, FBST. One is Flash memory endurance, which can

be determined by looking at the bit error consistency and overall access latency. The other is access frequency.

In regard to endurance, if page errors are observed and found to occur frequently which implies it is a page error due to wear-out, we reconfigure the page. This is achieved by enforcing a stronger error correction code, reducing cell density or controlling the program/erase time. The setting is determined by observing the current configuration and using a heuristic to calculate the expected cost. We note that cell density reduction provides the best reliability improvement, but greater loss of capacity than increased ECC strength.

To explain how the expected cost is calculated, we revisit the overall access latency equation described in section 6.2.2. The three techniques affect the overall access latency equation in different ways. Code strength controls the hit latency t_{hit}, program/erase time controls the miss latency t_{miss} and cell density affects both t_{hit} and p_{miss}. Specifically, a sensitivity analysis of the overall access latency equation reveals the following costs:

$Cost_{code\_strength} = p_{access,i} \times \Delta code\_delay$

$Cost_{program\_erase} = p_{miss} \times p_{erased,i} \times \Delta program\_erase$

$Cost_{density} = \Delta miss \times (t_{miss} - t_{hit} - p_{access,i} \times \Delta SLC)$

where $p_{access,i}$ is how frequently the i-th page is accessed and $p_{erased,i}$ is how frequently the i-th block is erased. The latency increase for enforcing a stronger error correction code is ($\Delta code\_delay$), increasing program/erase time is ($\Delta program\_erase$), increasing miss rate is ($\Delta miss$) and reducing hit latency is due to switching from MLC to SLC is (ΔSLC).

The technique with the minimum cost is selected and is influenced by workload, Flash memory age and the current state of the Flash memory cell.

We favor modifying the code strength since it does not change Flash capacity. To prevent unnecessary memory alignment complexity, we assumed that we are able to correct up to 12 errors using BCH, causing a performance overhead of hundreds of microseconds. The BCH check bit storage overhead is marginal considering the high capacity of today's Flash memories. Devices typically include 64 bytes for ECC support bits. When we consider that 4 bytes are used for the CRC32 code, 60 bytes are at our disposal. Support bits required for BCH are $n \times t$ where n is the degree of the finite field and t is the number of correctable errors. Because we limited the number of correctable errors to 12, a maximum of 23 bytes are used for parity, per page.

If the above techniques do not perform well and result in a substantial increase

in overall access latency, cell density can be reduced to improve reliability. One may not be able to reduce density at all if a page is already in SLC mode. In this case, the block would have to be removed permanently and never considered when looking for pages to allocate in a disk cache. Once a page transitions to single level cells, it is not feasible to go back to multi-level cells if the transition was in response to wear-out.

Once a page is reconfigured, this page is marked as invalid and an empty page is allocated while an access to the hard disk drive is scheduled. A descriptor for this empty page is sent to the Flash memory controller when writing data. If there is no empty page available, a Flash memory block is evicted and erased using the block-level data structures for LRU.

Density reduction does come at a price since it increases the Flash cache miss rate. By observing the access behavior for a fixed time interval, one can determine whether this page is frequently accessed or not. If this page is found to be frequently accessed, reducing the latency by allocating an SLC based empty page and migrating the frequently accessed page to this location improves overall latency. Furthermore, a small number of pages with reduced density can satisfy the majority of accesses with low latency, because many accesses to files are spatially and temporally a tailed distribution (zipf).

6.2.3 Function and location of Flash memory—Why it is a non-volatile disk cache

One may think that because Flash is such a dense device and consumes much less power than disk, it could be an excellent candidate to replace disk drives in servers. This section explains why a Flash should be treated as a disk cache for servers. This leads to discussing where the Flash should be integrated on a server considering a programable Flash controller described in the previous sections. We explain these issues by first explaining Flash based file systems that are currently available and then describe how these filesystems don't scale as Flash memory sizes increase. Finally, we show how a disk cache is much simpler and requires less performance and storage overhead.

Flash FileSystem

Due to the organization of a Flash and Flash endurance, conventional filesystems have had to either be augmented (emulated) or built from scratch to cover the characteristics of a Flash. Figure 6.15 shows how today's Flash filesystem are accessed

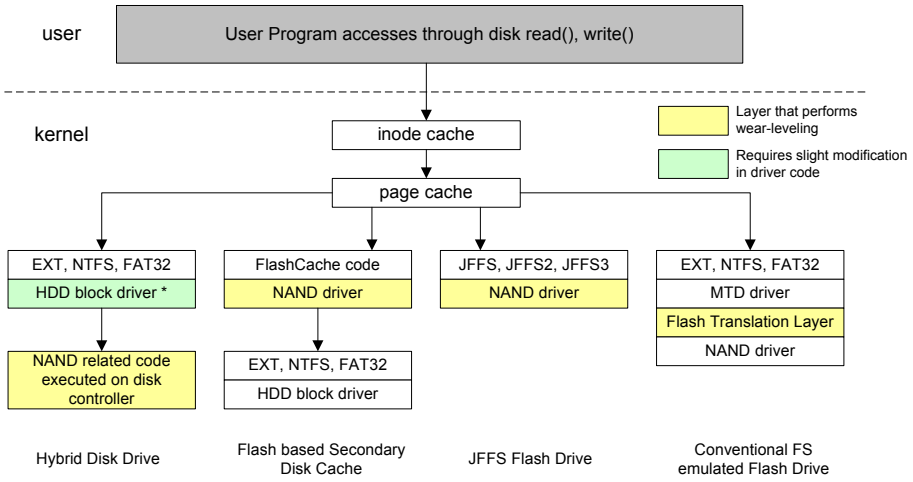


Figure 6.15: Block diagram of how Flash is accessed based on the filesystem mapped to Flash or role of Flash (FlashCache)

and where wear-leveling is performed. Each technique dedicates a specific software layer to do Flash block management.

Flash Translation Layer (FTL) based filesystem To make Flash backward compatible with conventional filesystems like FAT32 and EXT2, an additional Flash translation layer (FTL) has been built to remap conventional disk requests into Flash friendly requests. A gap exists due to conventional disk requests built to be block friendly. Flash has been regarded neither as a char nor block device proving why it requires a remapping of a disk request. FTL solutions are offered in typical USB Flash drives. However, recent trends suggest that the code overhead in the FTL is substantial and will likely require other techniques to reduce this overhead.

JFFS, JFFS2, JFFS3 filesystem In the open source community, Flash based filesystems were built ground up to enable commodity operating systems to access and utilize Flash. The continuous efforts through JFFS, JFFS2, JFFS3 have shown that Flash can be mounted and used transparently on a commodity system. JFFSx is a log structured file system optimized for Flash. It originally was built to support NOR Flash, but later included support for NAND Flash. Due to the out-of-place write updates, JFFSx is able to decouple writes and erases. Erases occur during Garbage Collection and reclaims dead Flash pages. Several issues in regard to scalability are mentioned and need to be addressed in the future. For Flash based solid state disks to exist these solutions must be addressed at the system level.

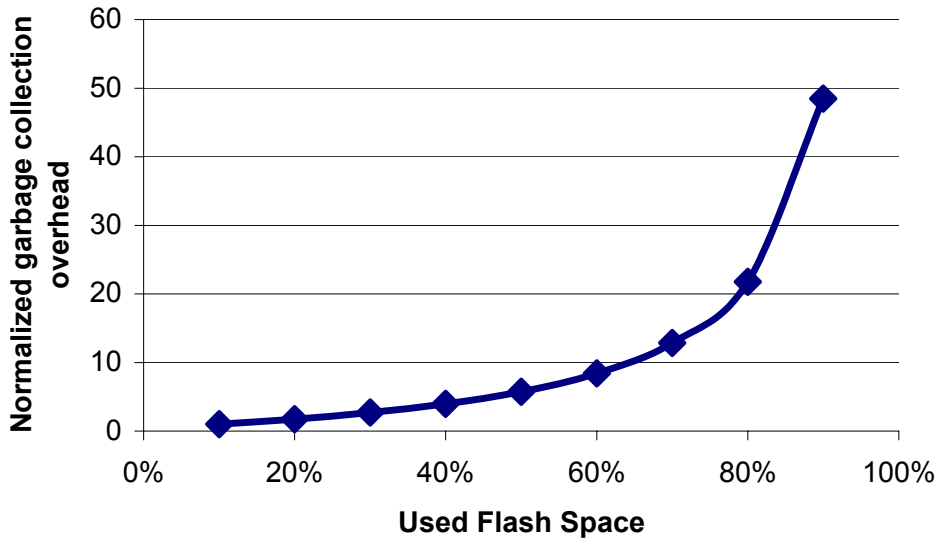


Figure 6.16: Garbage collection (GC) overhead in time versus occupied Flash space in a 2GB Flash, GC overhead in time is a product of GC frequency and GC latency. It is normalized to the overhead at 10%

How Flash based filesystems scale?

Filesystems store files along with headers and data structures that help find the file and store the file header. This implies additional storage overhead to store headers. Mount time for file systems is an important feature. One drawback we can clearly see in Flash based filesystems is that they spend much time on mounting the filesystem since it has to read the entire data structure used in the filesystem from Flash. This is a way to reduce direct updates to frequently updated data structures in the filesystem that may potentially wear-out a Flash quickly. The memory overhead in Flash based filesystems has been noted to linearly increase in Flash memory size, which implies it will exponentially increase annually because Flash sizes increase exponentially. To reduce mount time and memory overhead, new techniques that try to reduce the amount of filesystem updates and only stores a partial amount of the filesystem's data structure by caching it on system memory have been proposed [8]. This method is similar to what we find in conventional filesystems or demand based paging. However, it is still unclear that these methods are useful due to the performance overhead and even potentially hurt Flash.

In addition, another concern in managing Flash based filesystem's is garbage collection. The overhead in garbage collection increases as we fill up the Flash with

files. This becomes a big problem because garbage collection generates un-necessary writes and erases in Flash. Again, it hurts performance and endurance as the Flash is occupied. Figure 6.16 shows how the time spent garbage collecting increases as more Flash space is used. Because Flash is unlikely to be cheaper than disk drives, if used as a storage device in servers they are likely to fill up quickly.

The logical and physical location of Flash based disk caches

This section discusses where a Flash should be located both logically and physically in a server platform. Logically, for disk cache intensive applications which put more emphasis on processor and system memory power, it would make more sense to put Flash closer to the processor and treat it as part of the memory. As we showed in the previous sections, there are many data structures involved in making Flash memory reliable and programming the Flash controller. The size of these data structures amount to 10's of megabytes of to 100's of megabytes. These data structures are also tightly coupled to other data structures in the kernel.

With a reasonable memory requirement and tight correlation with the kernel, it would be more efficient to place the Flash and Flash controller at the hands of the processor instead of the disk controller. Another case for putting Flash closer to the processor is due to the assumptions written for interfacing with the hard disk drive. There are many assumptions in the disk interface code in the OS kernel, which is intended only for hard disk drives. As a result, some requests to disk maybe geared towards hard disk drives, making it less optimal for a Flash. We were able to measure that approximately 10% of additional execution time was spent in running hard disk drive friendly code. Qualitatively we believe for disk intensive applications, Flash should exist closer to the processor and be governed by the processor.

The Microsoft hybrid disk drive protocol revisions strengthen our argument. Microsoft has gradually augmented additional disk drive protocol that enable the OS to directly control the Flash than send messages to the disk drive controller and let it manage the Flash. In fact, recent revisions outline a protocol allowing some parts of the Flash to be pinned by the OS and disallowing the disk controller from evicting block in Flash.

Despite the case for logically associating the Flash device with the processor and the OS, Flash can physically exist virtually anywhere in a platform. As we had noted in section 6.2.2, the measured bandwidth to Flash is easily satisfied with today's I/O bandwidth standards available in servers. We also note that a clear roadmap is outlined for Flash to support higher bandwidths. One potential concern is the

relationship with stronger ECC code. As we had noted in 6.2.2, architectural support for accelerating ECC encode and decode time is critical in satisfying read and write throughputs to Flash.

6.2.4 Impact on storage device power

Integrating Flash memory as a part of system memory also affects the behavior of storage devices—disk drives. To understand the impact, we briefly describe the current trends in disk drive technology. According to [5], the price per bit and storage density appears to scale well beyond 2010 and is expected to be orders of magnitude more efficient than Flash. Internal data rates are also expected to increase at an exponential rate, despite some concern due to thermal management [46] which has been confirmed by [81]. However, what we do notice is the high power consumption for fast internal data rate disk drives that display lower seek and rotation time to deliver lower access latency. Integrating Flash memory as a secondary disk cache, definitely increases opportunities to put disk drives into sleep mode and reduce the required internal data rate. This translates into reducing the disk power consumption by adopting disk drives with mediocre internal data rate and spinning down disks when not accessed. Today's disk drives at 15,000 rpm consume more than 15W when active which is roughly equivalent to 8GB of DRAM and 100's of gigabyte of Flash in terms of power consumption.

6.3 Evaluation

We compare our results to conventional architectures that do not employ Flash and instead use only DRAM. We assume a conventional chip multiprocessor configuration (MP4/8/12) clocked at 1GHz with a 2MB L2 cache. To understand the endurance of Flash, we present results on expected lifetime of the Flash memory.

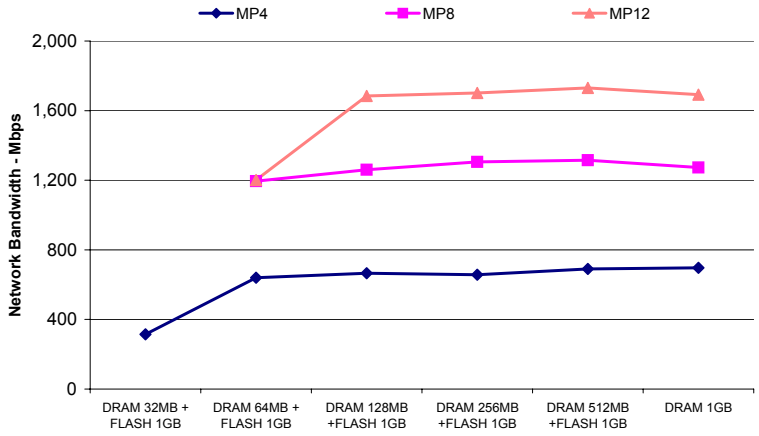
6.3.1 Server Throughput with Flash memory

Figures 6.17, 6.18, 6.19 depicts the overall throughput 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. Apart from the workloads generated from SPECWeb2005 (working set size is approximately 256MB), 1GB of DRAM does not capture the entire working set size. Our optimal multichip configuration requires less die area than our baseline configuration due to the compactness of Flash. From our early

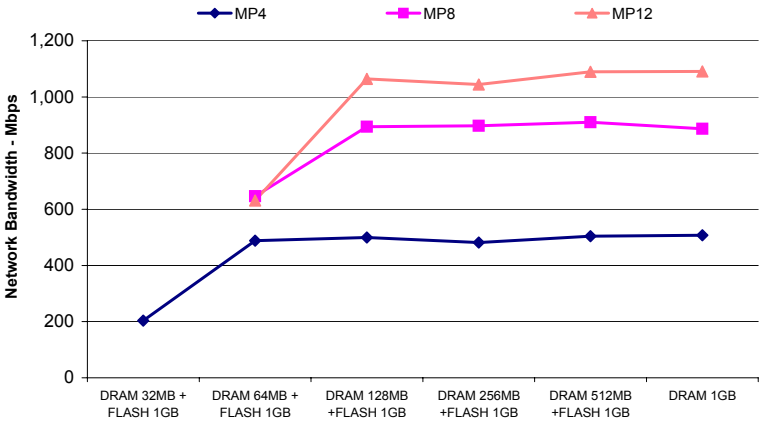
observations that large portions of the page cache can tolerate access latencies of tens to hundreds of microseconds, we find our optimal configuration to be 128MB of DRAM and 1GB of Flash for workloads that display short tailed behavior. In terms of area, this configuration requires 40% less die than our baseline of no Flash and 1GB of DRAM as shown in Figure 6.20. For Tier 3 workloads like TPC-C that display long tailed behavior, more DRAM memory in the primary page cache is required to see marginal degradation in throughput. Further, the amount of memory is related to the number of cores. As we increase the number of cores, Tier 3 workloads have a difficult time fully utilizing each core. This is due to the way locks are conservatively used in database servers (software). Hence, we see marginal improvement in throughput for TPC-C benchmarks as we increase from 8 cores to 12 cores. In summary, our results suggest using Flash memory as a secondary disk cache results in minimal degradation in throughput.

6.3.2 Overall system memory power

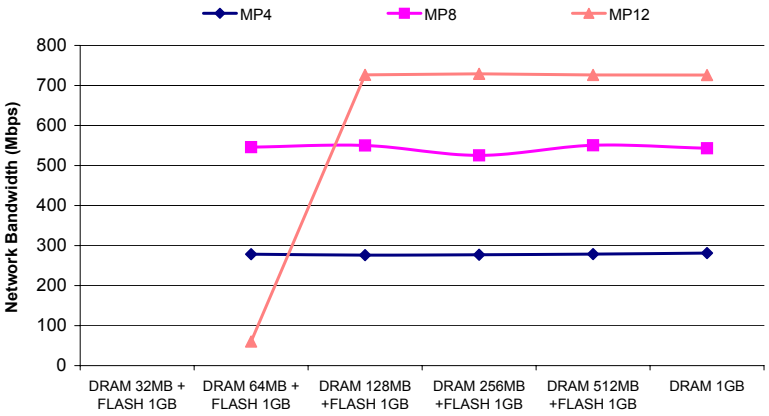
With respect to overall power consumed in system memory, our primary power savings come from the reduction in idle power from using Flash memory. It is several orders of magnitude. We compare our multichip configuration with DRAM configurations that have a) no power management—*active* mode and b) ideal 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 [51][61]. Figure 6.21, 6.22, 6.23 shows our results. As shown in Figure 6.21, 6.22, 6.23, the FlashCache architecture reduces overall system memory power by more than a factor of 2.5, even compared to an oracle power management policy implemented in software and hardware for DRAM. We assumed a hardware memory controller controlled through software puts DRAM to sleep whenever it is not accessed. 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) SURGE

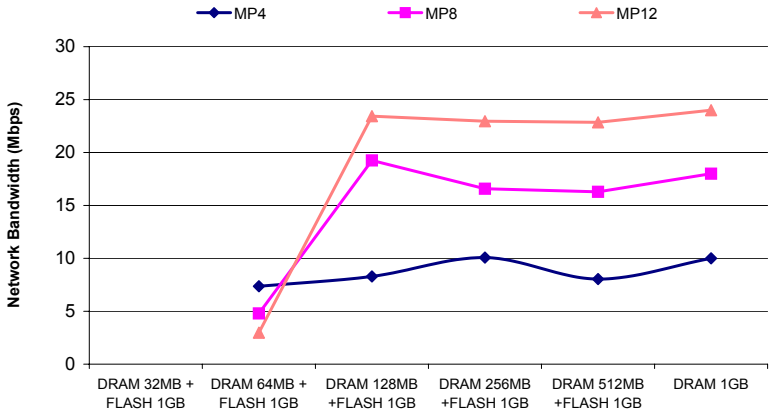


(b) SPECWeb99

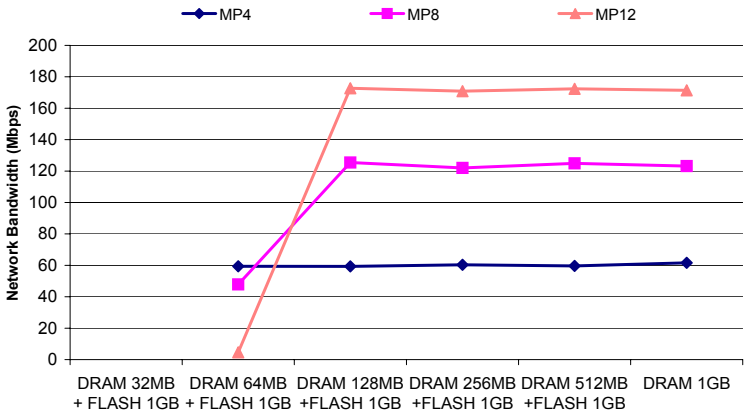


(c) Fenice

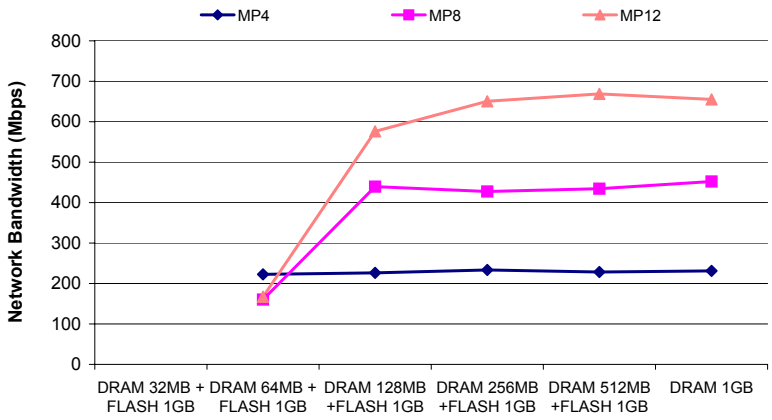
Figure 6.17: A throughput comparison for various system memory configurations using the FlashCache. The rightmost points are for a DRAM-only system.(SURGE, SPECWeb99, Fenice)



(a) SPECWeb2005-bank

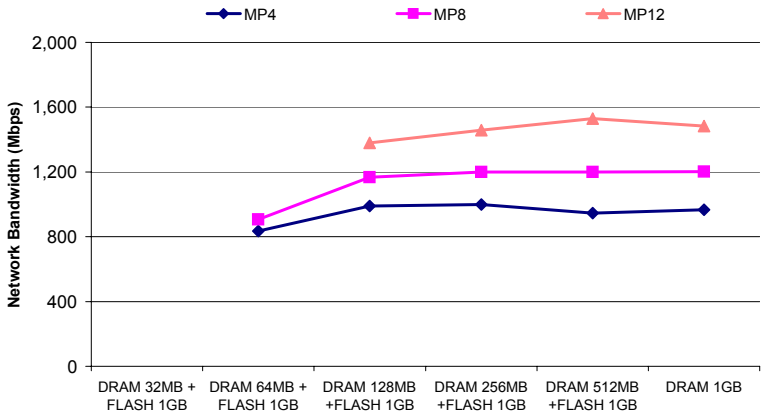


(b) SPECWeb2005-ecommerce

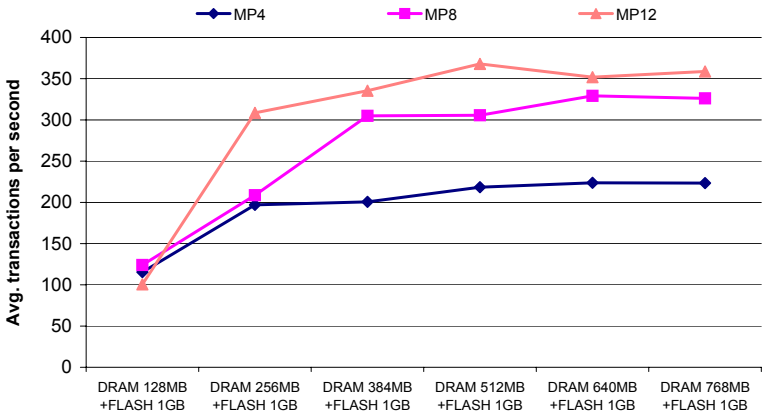


(c) SPECWeb2005-support

Figure 6.18: A throughput comparison for various system memory configurations using the FlashCache.(SPECWeb2005-bank, ecommerce, support)



(a) dbench



(b) TPC-C

Figure 6.19: A throughput comparison for various System memory configurations using the FlashCache. (dbench, TPC-C)

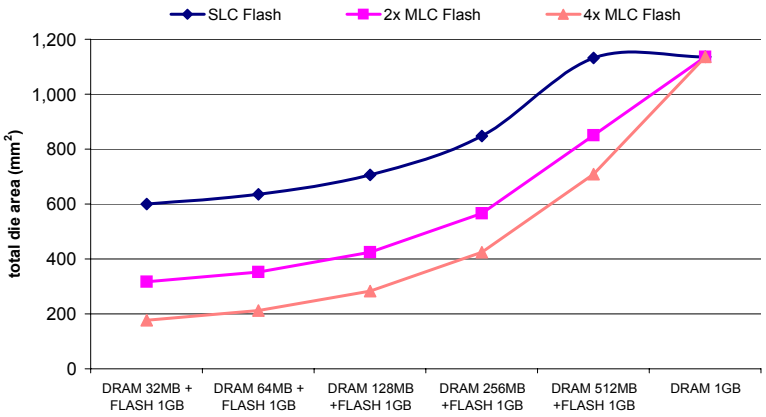
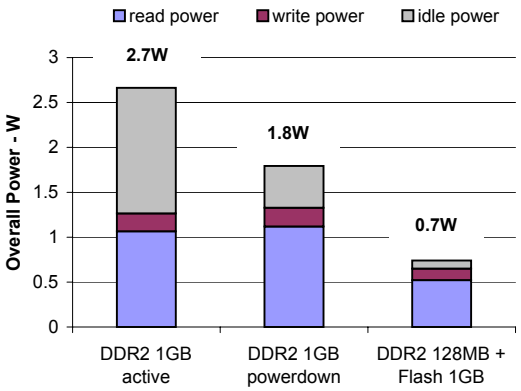


Figure 6.20: Die area

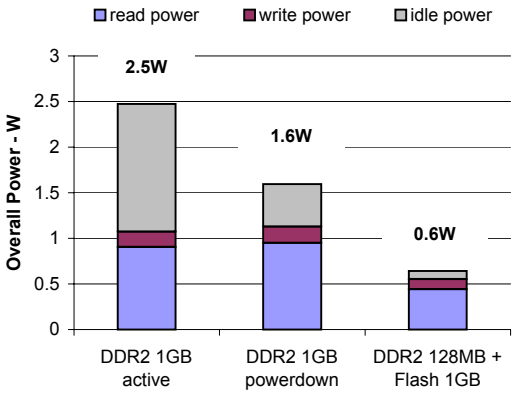
6.3.3 Behavior of Programmable Flash memory controller

Sensitivity Analysis of Flash controller configuration

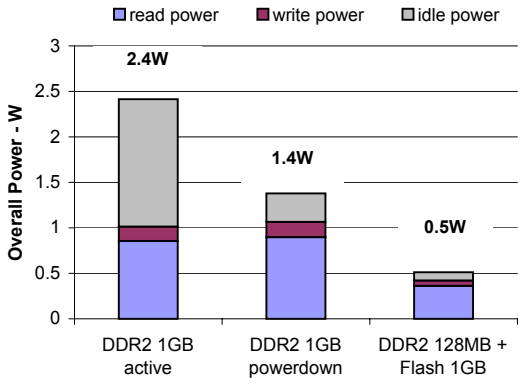
Figure 6.24 shows the breakdown of page reconfiguration events which occur when Flash blocks are read. This can either be a decision to increase ECC strength, program/erase time or switch the block from MLC to SLC mode. The objective is to minimize the latency cost function explained previously in section 6.2.2. The size of Flash memory was set to half the working set size of the application. These simulations were measured near the point where the Flash memory cells start to fail due to programs and erases. Wear out measurements are difficult to simulate on a full system simulator like M5. Thus, we used real disk traces and generated micro disk traces to observe configuration change on a Flash controller. The results confirm the benefits of a programmable Flash memory controller since the response to each benchmark is significantly different. The figure also suggests that as the tail length of a workload increases, we see less transitions from MLC to SLC since FlashCache capacity is more important for long tailed distributions. In fact, for a uniform distribution which is an extreme case of a long tailed distribution, we found 70% of descriptor updates are changes in code strength but no transitions from MLC to SLC. For exponential distributions, which are an extreme case of short tailed distributions, we see that MLC to SLC changes (density) dominate, because the increased miss rate due to a reduction in density is negligible. For the macro benchmarks, since they display tailed access behavior, we clearly see similar behavior to the micro benchmarks.



(a) SURGE

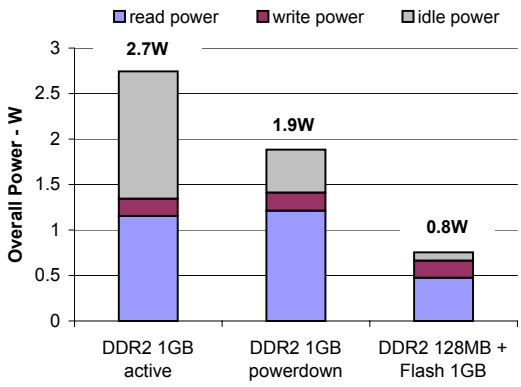


(b) SPECWeb99

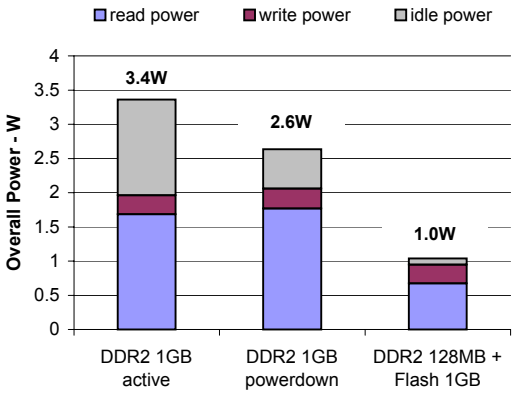


(c) Fenice

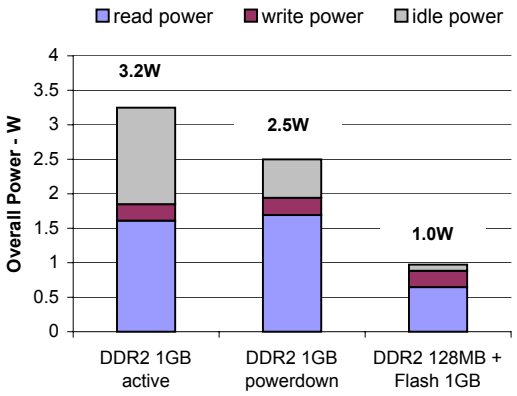
Figure 6.21: Overall memory power consumption breakdown. Our FlashCache architecture reduces idle power by several orders of magnitude. For powerdown mode in DRAM, we assume an oracle powerdown algorithm. (SURGE, SPECWeb99, Fenice)



(a) SPECWeb2005-bank

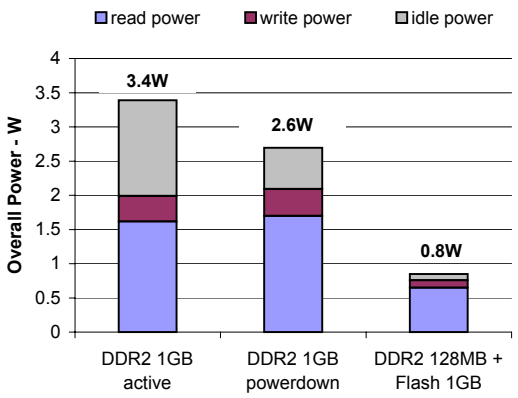


(b) SPECWeb2005-ecommerce

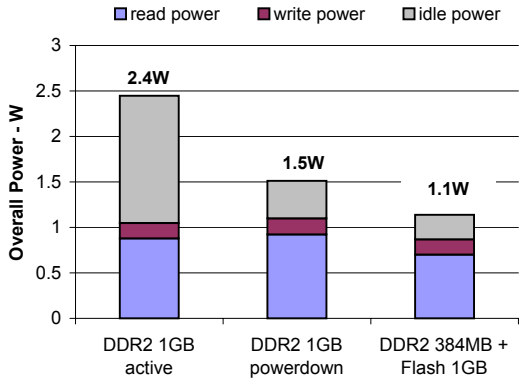


(c) SPECWeb2005-support

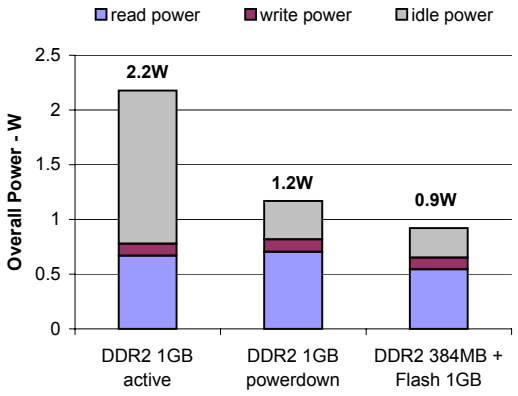
Figure 6.22: Overall memory power consumption breakdown. Identical assumptions from 6.21 applied. (SPECWeb2005-bank, ecommerce, support)



(a) dbench

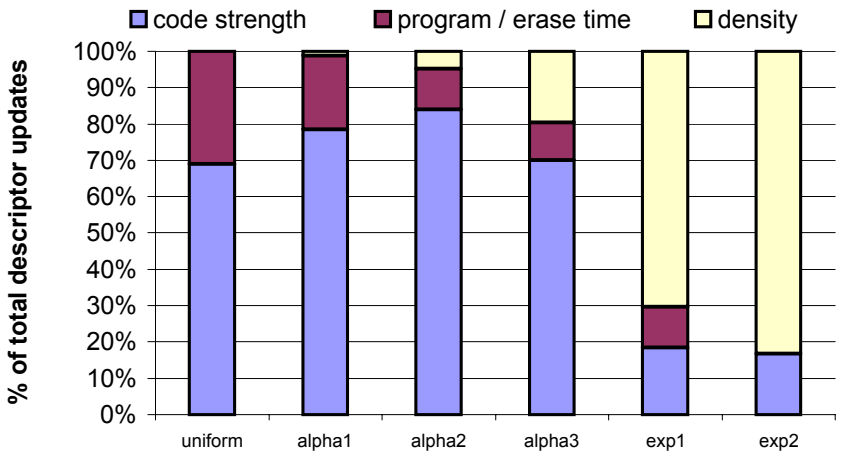


(b) TPC-C

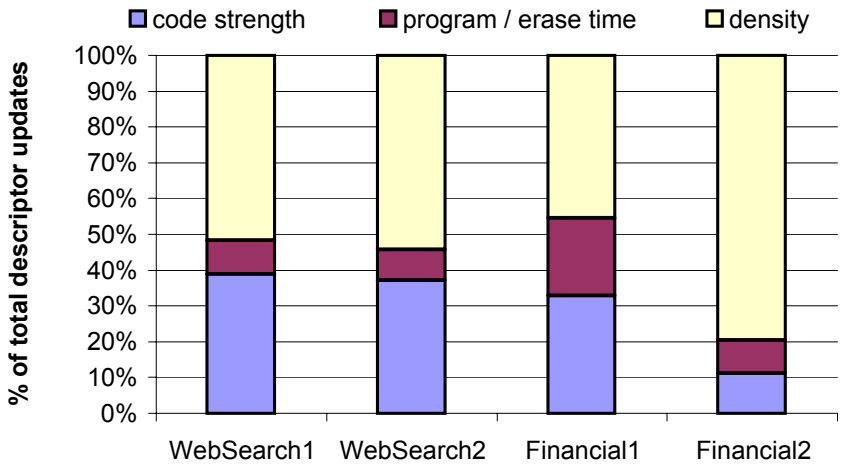


(c) TPC-H

Figure 6.23: Overall memory power consumption breakdown. Identical assumptions from 6.21 applied. (dbench, TPC-C, TPC-H)



(a) microbenchmark



(b) macrobenchmark

Figure 6.24: Breakdown of configuration changes due to wear-out

6.3.4 Overall Flash memory access latency

This section shows a comparison of Flash memory access latency for our programmable Flash memory controller versus two fixed Flash memory controllers. We compared our programmable implementation with a fixed controller supporting BCH 1 error correction and BCH 12 error correction at a point in time where we see marginal failures due to wear-out. Figure 6.25 shows our results. Due to the programmability and the ability to improve performance, our programmable memory controller does $10\% \sim 40\%$ better than BCH 1 error correction and more than $3.5\times$ better than BCH 12. This is primarily due to the fact that programmable controllers only increase code strength for regions that require this code strength and result in reduced hit latency. The ability to promote some pages from MLC to SLC also results in latency improvement which accounts for much of the difference between a programmable and BCH 1 only controller. A programmable controller does require more OS management, resulting in a performance overhead. However, the improvement in access latency is several microseconds and measuring the time spent in executing additional management code is less than the latency improvement. As the Flash memory ages, we found our programmable Flash memory controller greatly slowed down the increase in overall access latency due to aging.

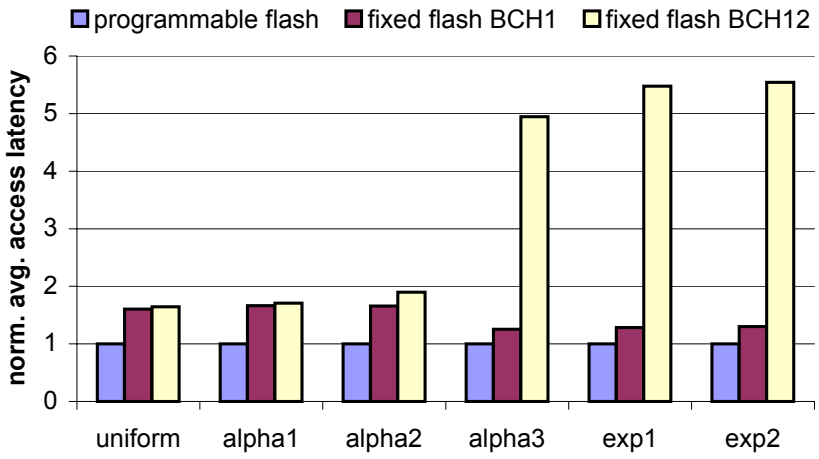
6.3.5 Wear level aware behavior

Flash memory lifetime without programmable Flash controller

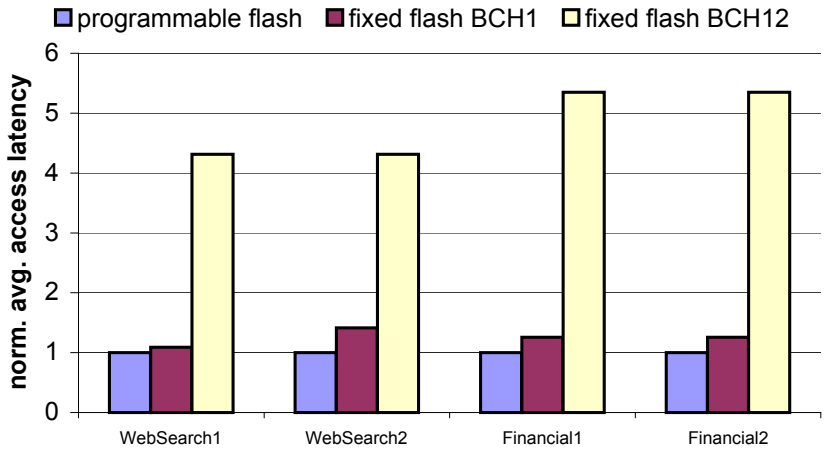
Figure 6.26 shows the predicted lifetime for varying Flash memory sizes assuming the lifecycle of a Flash memory to be a million cycles. These simulations assumed a conventional MP8 configuration (MP8 w/o 3D stacking) and varying FlashCache size that are a certain percent of the known working set size. For workloads that we were able to simulate, we found our FlashCache is accessed less than 4000 times a second. With ECC support for multiple error corrections, worst case lifetime can be extended even more.

Improved Flash memory lifetime with programmable Flash controller

Figure 6.27 shows a comparison of the normalized number of accesses required to reach the point of total Flash failure where none of the Flash memory pages can be recovered. We compare our programmable Flash memory controller with a BCH 1 error correcting controller. We expect a BCH 12 error correcting controller to have much better lifetime than a BCH 1 error controller, but because it requires a signif-



(a) microbenchmark



(b) macrobenchmark

Figure 6.25: Normalized comparison of overall average access latency to Flash

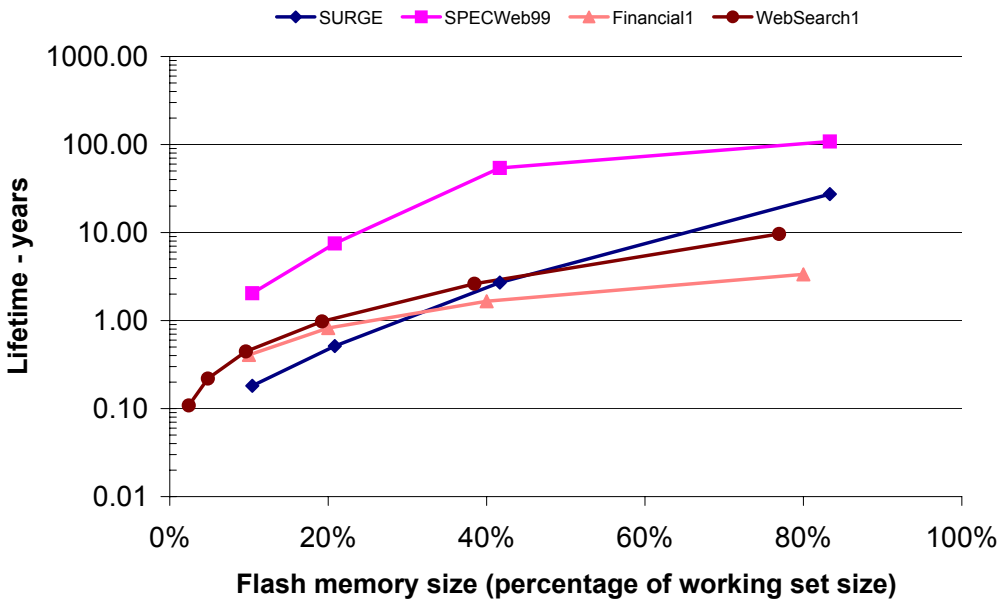
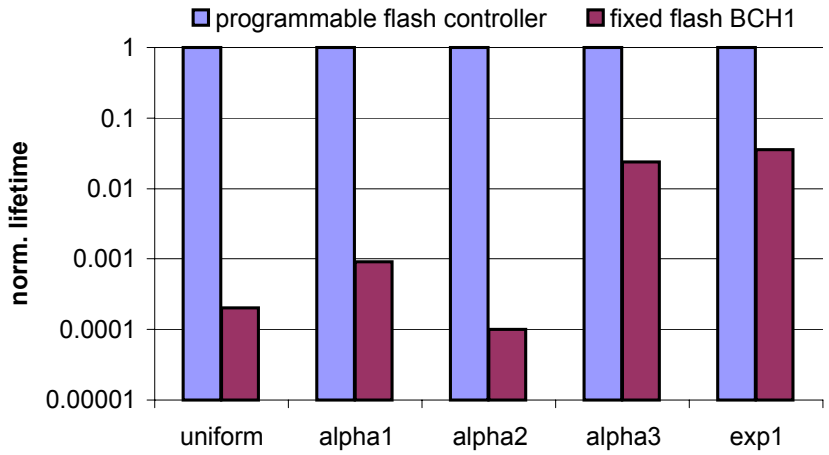
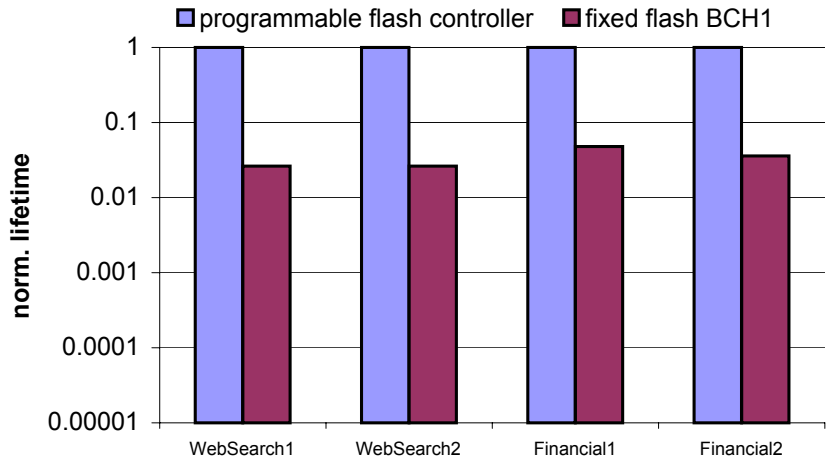


Figure 6.26: Flash memory endurance while varying the Flash memory size in the FlashCache architecture. Temporal endurance in years, assuming Flash memory endurance of 1,000,000 cycles

icantly higher access latency, it is not comparable to our high-speed programmable Flash memory controller. Our studies show that for typical workloads, our programmable Flash memory controller extends lifetime by a factor of 20 on average (see Figure 6.27). For a workload that would previously limit Flash memory lifetime to 6 months, we show it operates for more than 10 years using our programmable Flash memory controller.



(a) microbenchmark



(b) macrobenchmark

Figure 6.27: Normalized expected lifetime given the access rate and tolerable wear-out

CHAPTER VII

RELATED WORKS

Our work leverages emerging technology and applies it to server workloads executed on a chip multiprocessor architecture. We present previous work on chip multiprocessors, 3D stacking technology, non-uniform memory architectures and work investigating the impact of new memory devices.

7.1 Chip Multiprocessor Architectures

Olkuton et. al[77] initially presented the possibility of building a simple multiprocessor architecture that were friendly to heavily threaded applications. In doing so they looked at SPLASH benchmarks along with highly parallel media benchmarks. However, the working set of these benchmarks was not large and the overall platform level evaluation of a chip multiprocessor was not presented in this work. Li and Martinez[65] looked at the power reduction in parallel architectures. They presented an analytical power estimation model for multiprocessor architectures and evaluated this model on SPLASH2 benchmarks. An optimal power, thermal point was presented due to the diminishing return for increasing parallel processing width. As parallel processing width is increased, leakage power increases linearly. Barroso et. al[31] presented a chip multiprocessor architecture to support server applications. A simple 8 in-order multiprocessor architecture was presented with a shared L2 cache. However, this work did not investigate the impact of network bandwidth in the architecture. The estimated power consumption was not mentioned in this work either. Additionally, the benchmarks presented in this paper focussed on Tier 2, 3 server applications. Kozyrakis et. al[58] applied the embedded DRAM technology to DSP applications. Specialized vector engines which existed adjacent to the embedded DRAM, contributed to the speedup in computation for vector friendly applications. This work performed extremely well for multimedia and matrix-oriented vector applications. Compared to previous work on chip multiprocessors, our work extends

the benefits of chip multiprocessors by leveraging 3D stacking technology that allows us to add more cores and reduce core clock frequency in conventional simple in-order cores. We also modify the memory hierarchy exploiting the reduced access latency to large capacity on-chip DRAM, so the L1 caches in a chip multiprocessor architecture access on-chip DRAM directly and treat it as a part of main memory.

7.2 3D Stacking Technology

Black et. al[33] have presented circuit-level potential in this technology. They applied 3D stacking on an x86 core by implementing the floating point unit on the top layer and putting the rest of the execution units on the bottom layer. Their preliminary studies showed a 20% improvement in performance was achieved for SPEC2000 benchmarks and 3D stacking reduced the number of repeaters traditionally required for long global interconnects. With respect to work on 3D stacking technology alone, we have not seen any work that identifies a good application space for this technology and provides a thorough analysis.

7.3 Non-uniform Memory Architecture

In the case of non-uniform main memory, Ekman et. al[40] 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. Huang et. al[51] and Lebeck et. al[61] showed that a considerable amount of system memory power can be reduced with power aware page allocation. Our work extends the work from [40][6] and integrates flash memory as a secondary file buffer cache for web server applications. By adapting and simplifying the methods in [7] and incorporating them onto a cache replacement algorithm, we mitigate any wear-out problems.

7.4 Investigating the impact of emerging memory technology devices

Whenever a new memory technology device is introduced with potential, a study investigating the impact of adopting this new device has been presented. Unfortunately, most of the findings presented in papers are limited to the particular memory or storage device and cannot be universally applied to other memory devices.

Baker et. al[28] and Wu et. al[86] looked at integrating NOR Flash into the storage hierarchy. While their initial work is similar to our efforts in integrating NAND Flash onto a system, many breakthroughs in device technology and stronger understanding in erratic behavior in Flash cells suggest that NAND Flashes should be treated in a different manner. For example, when NOR Flash was first introduced it was believed to be less cost effective than DRAM and display similar access latencies. In addition, the storage density of NOR Flash was believed to be less than DRAM. In contrast, as we have shown in this dissertation, NAND Flash is much more denser and cost-effective while having a access latency of microseconds which is more than 3 orders of magnitude slower than DRAM.

In Park et. al[78], it was shown that Flash memory could be used directly for high performance code execution by adding an SRAM. In [6], 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 [7] to extend Flash memory endurance. [24] has shown error correction codes extend the lifespan of Flash memory with marginal overhead.

Other types of memory devices that use a different technique in storing information have been found to be non-applicable to server platforms that require a high storage density. Patterson et. al[80] and Desikan et. al[39] use a different method to store information but have displayed lower storage densities. The cell sizes for these memory cells are projected to be much larger than Flash and even conventional DRAM.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORKS

8.1 Thesis Summary

This dissertation presented the impact of technological advances in packaging technology (3D stacking) and memory devices (Flash) to server platforms that are an integral part of today's datacenters. The resulting systems have significantly improved energy efficiency potentially enabling ultra small form factor servers.

Specifically, we have found that due to 3D stacking technology a 12-way PicoServer running at 500MHz can deliver up to 1.4Gbps of network bandwidth within a 3.5W power budget in a 90nm process technology. These power results are $2 \sim 3\times$ better than a multicore architecture without 3D stacking technology and an order of magnitude better than what can be achieved using a general purpose processor. The ability to tightly couple large amounts of memory to the cores through wide and low-latency interconnect pays dividends by reducing system complexity and creates opportunities to implement system memory with non-uniform access latency. With the access latency of on-chip DRAM being comparable to the L2 cache, we found the L2 cache die area can be replaced with additional cores resulting in core clock frequency reduction while achieving higher throughput. For an area-equivalent PicoServer configuration using 12 processors at 500MHz without an L2 cache yields a substantial improvement in throughput while reducing power consumption by more than 50% compared to a conventional 8-way 1GHz chip multiprocessor with a 2MB L2 cache.

When integrating Flash memory onto a server platform, we found that our Flash-Cache architecture reduces idle power by several orders of magnitude while maintaining cost effectiveness over conventional DRAM-only architectures both in terms of cost and energy efficiency. We also found that due to the limitation in endurance for Flash, we found that a Flash based disk cache should be organized as a read-optimized and write optimized disk cache to better improve disk cache hit rate and extend the lifetime of a Flash based disk cache. A programmable Flash memory controller is re-

quired to deliver optimal Flash access latency and reliability which varies in workload behavior. From our observations, a typical server architecture can sustain a file access latency in the tens to hundreds of microseconds without noticeable loss in throughput. For Tier 3 workloads, we observe the write optimized Flash based disk cache is an excellent feature for quick database recovery and find that more DRAM and Flash may be required in system memory to appropriately cache the large databases.

8.2 Future Work

8.2.1 Managing datacenter energy efficiency at the rack level

There is still more work to do in terms of architecting an energy efficient datacenter at the rack level. Identifying the key elements in implementing a scalable and reliable management controller used to manage an entire farm of servers is an important aspect of research in this area. Prior work introduces control theory that adjusts the controller to the client load. However, prior work only tackles some aspects of this problem. It is still unclear how to build a scalable controller that satisfies many of these objectives while considering the many platform parameters one can take advantage of. These controllers must consider multiple objective functions like energy efficiency, heat and reliability. With the introduction of virtual machine consolidation and many core architectures, it will become an interesting problem to solve. Naive worst case provisioning or typical case provisioning will not always guarantee an optimal result.

8.2.2 Improving on-chip interconnect bandwidth with opto-electronic devices

Aside from 3D stacking technology, using on-chip optoelectronic devices may be another solution to delivering ultra high bandwidth at a low cost. Optoelectronic devices enable multiple transactions to be multiplexed simultaneously using wavelength division multiplexing (WDM). Interconnection networks built using this device consumes much less power than a interconnection network built with electrical wires. Early work on how it may be leveraged on-chip has been proposed in [55], but it is still unclear how it may be applied to server platforms. A detailed analysis of how these devices may be leveraged in the future to build scalable datacenters should be an interesting area of research we would like to pursue.

8.2.3 Delivering single threaded performance in server workloads

Some server workloads require a decent amount of single threaded performance. It is however difficult to deliver high single threaded performance while consuming low power which is necessary in today's computing platforms. In the short term, understanding the energy efficiency of conventional techniques like increasing the clock frequency, increasing the scalar issue width, out of order execution may be helpful in proposing incremental techniques that provide single threaded performance while consuming low power. In the long term, investigating a new microarchitecture that can provide both single threaded performance and multithreading will be a challenge.

8.2.4 Identifying a usage model for upcoming new memory devices

Associating how new memory devices could be introduced and integrated onto a server platform may be critical in building energy efficient servers. Like Flash, phase change memory (PCRAM) is emerging as a promising solution. It displays better write throughput than Flash and scales fairly well. Projecting the roadmap and behavior of phase change memory and understanding the potential value it may generate in the server space is definitely an interesting area of research. PCRAM is expected to replace NOR Flash in the near future and may even replace NAND Flash.

Understanding the potential usage of Floating Body DRAM (FBRAM) is another area of research. FBRAM is a promising solution for implementing L2, L3 caches on-die. It is also expected to directly compete with conventional DRAM since it can solve the scalability issues with DRAM.

8.2.5 Architectural support for future server workloads

It is highly likely that datacenters will support additional applications like image recognition to provide better quality of service and better user experience. For example, searching a image or video clip based on a given search phrase or image is a likely usage model. These workloads are known to display data level parallelism as well as thread level parallelism. To deliver high throughput and quality for these datamining, recognition workloads, more work should be done in understanding the commonly used kernels and how it may be realized in software and hardware. Moreover, it may be apparent to add instruction level support and acceleration engines for these workloads. In our video streaming workload Fenice, we observed a substantial portion of

time in processor spent on regular expression matching. In many instances, it may make sense to use the available silicon area in implementing acceleration engines for these new applications.

BIBLIOGRAPHY

[1] 82563EB/82564EB Gigabit Platform LAN Connect Networking Silicon Datasheet.

[2] ARM 11 MPcore. http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

[3] Evolution of network memory. http://www.jedex.org/images/pdf/jack_troung_samsung.pdf.

[4] FaStack 3D RISC super-8051 microcontroller. http://www.tachyonsemi.com/OtherICs/datasheets/TSCR8051Lx_1_5Web.pdf.

[5] HDD technology 2003. http://www.hitachigst.com/hdd/hddpdf/tech/hdd_technology2003.pdf.

[6] Hybrid Hard Drives with Non-Volatile Flash and Longhorn. http://www.samsung.com/Products/HardDiskDrive/news/HardDiskDrive_200504%25_0000117556.htm.

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

[8] JFFS3 (Journalling Flash File System Version 3). http://www.linux-mtd.infradead.org/doc/jffs3.html.

[9] Leon3 Processor. http://www.gaisler.com/cms4_5_3/index.php?option=com_content&task=view&%id=13&Itemid=53.

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

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

[12] MIPS32 34K family. http://www.mips.com/products/cores/32-bit_cores/MIPS32_34K_Family.php#f%eatures.

[13] National semiconductor DP83820 10 / 100 / 1000 Mb/s PCI ethernet network interface controller.

[14] OSDL DataBase Test Suite. http://www.osdl.net/lab_activities/kernel_testing/osdl_database_test_su%ite/.

[15] PowerPC 405 embedded core. http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050F%F778525699300651D97.

[16] Predictive technology model. http://www.eas.asu.edu/~ptm.

[17] (LS)^3-libre streaming, libre software, libre standards an open multimedia streaming project. http://streaming.polito.it/.

[18] RLDRAM memory. http://www.micron.com/products/dram/rldram/.

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

[20] Seagate Barracuda. http://www.seagate.com/products/personal/index.html.

[21] SPECweb2005 benchmark. http://www.spec.org/web2005/.

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

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

[24] TrueFFS. http://www.m-systems.com/site/en-US/Support/DeveloperZone/Software/Life%spanCalc.htm.

[25] University of Massachusetts Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[26] Zend platform. http://www.zend.com/store/products/zend-platform/zps.php.

[27] ITRS roadmap. Technical report, 2005.

[28] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proc. of Int'l Conf. on Arch. support for prog. languages and operating systems*, Oct. 1992.

[29] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat. 3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proc. of IEEE*, 89(5):602–533, May 2001.

[30] 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.

[31] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. In *Proc. Int'l Symp. on Computer Architecture*, June 2000.

[32] 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.

[33] B. Black, D. Nelson, C. Webb, and N. Samra. 3D processing technology and its impact on iA32 microprocessors. In *Proc. Int'l Conf. of Computer Design*, pages 316–318, 2004.

[34] T.-Y. Chiang, S. J. Souri, C. O. Chui, and K. C. Saraswat. Thermal analysis of heterogeneous 3-D ICs with various integration scenario. In *IEDM Technical Digest*, pages 681 – 684, Dec. 2001.

[35] T. Cho, Y. Lee, E. Kim, J. Lee, S. Choi, S. Lee, D. Kim, W. Han, Y. Lim, J. Lee, J. Choi, and K. Suh. A dual-mode NAND flash memory: 1-Gb multilevel and high-performance 512-mb single-level modes. *IEEE Journal of Solid State Circuits*, 36(11), Nov 2001.

[36] L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, Y. Liao, S. Strazdus, M. Morrow, K. E. Verlarde, and M. A. Yarch. An embedded 32-b microprocessor core for low-power and high-performance applications. *IEEE Journal of Solid State Circuits*, 36(11):1599–1608, Nov. 2001.

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

[38] W. R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A. M. Sule, M. Steer, and P. D. Franzon. Demystifying 3D ICs: The pros and cons of going vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[39] R. Desikan, C. Lefurgy, S. Keckler, and D. Burger. On-chip MRAM as a High-Bandwidth, Low-Latency Replacement for DRAM Physical Memories. In *IBM Austin Center for Advanced Studies Conference*, Feb. 2003.

[40] 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.

[41] P. Fazan, S. Okhonin, and M. Nagoga. A New Block Refresh Concept for SOI Floating Body Memories. In *IEEE Int'l SOI Conference*, Sept. 2003.

[42] M. J. Flynn and P. Hung. Computer architecture and technology: Some thoughts on the road ahead. In *Proc. Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms*, pages 3–16, 2004.

[43] B. Goplen and S. S. Sapatnekar. Thermal via placement in 3D ICs. In *Proc. Int'l Symp. on Physical Design*, pages 167–174, Apr. 2005.

[44] 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`.

[45] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: dynamic speed control for power management in server class disks. In *IEEE/ACM Proc. Int'l Symp. on Comp. Arch.*, June 2003.

[46] S. Gurumurthi, A. Sivasubramaniam, and V. K. Natarajan. Disk Drive Roadmap from the Thermal Perspective: A Case for Dynamic Thermal Management. In *Proc. Int'l Symp. on Computer Architecture*, June 2005.

[47] T. R. Halfhill. Z-RAM shrinks embedded memory. `www.innovativesilicon.com/en/pdf/z-ram.pdf`.

[48] T. Hara, K. Fukuda, K. Kanazawa, N. Shibata, K. Hosono, H. Maejima, M. Nakagawa, T. Abe, M. Kojima, M. Fujiu, Y. Takeuchi, K. Amemiya, M. Morooka, T. Kamei, H. Nasu, K. Kawano, C.-M. Wang, K. Sakurai, N. Tokiwa, H. Waki, T. Maruyama, S. Yoshikawa, M. Higashitani, T. D. Pham, and T. Watanabe. A 146mm^2 8Gb NAND Flash Memory with 70nm CMOS Technology. In *Proc. Int'l Solid-State Circuits Conference*, Feb. 2005.

[49] R. Ho and M. Horowitz. The future of wires. *Proc. of the IEEE*, 89(4), Apr. 2001.

[50] R. F. Hobson and K. L. Cheung. A High-Performance CMOS 32-Bit Parallel CRC Engine. *IEEE Journal of Solid State Circuits*, 34(2), Feb 1999.

[51] 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.

[52] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam. Compact thermal modeling for temperature-aware design. In *Proc. Design Automation Conf.*, June 2004.

[53] D. Ielminia, A. Spinelli, A. Lacaita, and M. van Duuren. A Comparative Study of Characterization Techniques for Oxide Reliability in Flash Memories. *IEEE Trans. on Device and Materials Reliability*, 4, Sep 2004.

[54] K. Kim and J. Choi. Future Outlook of NAND Flash Technology for 40nm Node and Beyond. In *Workshop on Non-Volatile Semiconductor Memory*, pages 9–11, Feb 2006.

[55] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martnez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi. Leveraging Optical Technology in Future Bus-based Chip Multiprocessors. In *International Symposium on Microarchitecture*, Dec. 2006.

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

[57] M. Koyanagi. Different approaches to 3D chips. `http://asia.stanford.edu/events/Spring05/slides/051205-Koyanagi.pdf`.

[58] C. Kozyrakis, J. Gebis, D. Martin, S. Williams, I. Mavroidis, S. Pope, D. Jones, D. Patterson, and K. Yelick. Vector IRAM: A media-oriented vector processor with embedded DRAM. In *Hotchips*, Aug. 2000.

[59] S. R. Kunkel, R. J. Eickemeyer, M. H. Lipasti, T. J. Mullins, B. O'Krafka, H. Rosenberg, S. P. VanderWiel, P. L. Vitale, and L. D. Whitley. A performance methodology for commercial servers. *IBM Journal of Research and Development*, 44(6), 2000.

[60] J. Laudon. Performance/watt: the new server focus. *SIGARCH Computer Architecture News*, 33(4):5–13, 2005.

[61] 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.

[62] 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. *IEEE Journal of solid-state circuits*, 38(11), Nov 2003.

[63] K. Lee, T. Nakamura, T. Ono, Y. Yamada, T. Mizukusa, H. Hashimoto, K. Park, H. Kurino, and M. Koyanagi. Three-dimensional shared memory fabricated using wafer stacking technology. In *IEDM Technical Digest.*, pages 165–168, Dec 2000.

[64] S. Lee, Y.-T. Lee, W.-K. Han, D.-H. Kim, M.-S. Kim, S.-H. Moon, H. C. Cho, J.-W. Lee, D.-S. Byeon, Y.-H. Lim, H.-S. Kim, S.-H. Hur, and K.-D. Suh. A 3.3V 4Gb Four-Level NAND Flash Memory with 90nm CMOS Technology. In *Proc. Int'l Solid-State Circuits Conference*, pages 52–53, 2004.

[65] J. Li and J. F.Martinez. Power-performance implications of thread-level parallelism in chip multiprocessors. In *Proc. Int'l Symp. on Performance Analysis of Systems and Software*, Mar. 2005.

[66] H.-K. Lim, D. K. Jeong, K. Kim, J. Park, and H. gyoo Kim. A single-chip storage LSI for home networks. *IEEE Communications Magazine*, 43(5):141–148, May 2005.

[67] S. Lin and J. Daniel J. Costello. *Error Control Coding, Second Edition*. 2004.

[68] J. Lu. Wafer-level 3D hyper-integration technology platform. `www.rpi.edu/~luj/RPI_3D_Research_0504.pdf`.

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

[70] D. A. Maltz and P. Bhagwat. TCP splicing for application layer proxy performance. Research Report RC 21139, IBM, Mar. 1998.

[71] R. E. Matick and S. E. Schuster. Logic-based eDRAM: origins and rationale for use. *IBM Journal of Research and Development*, 49(1), Jan. 2005.

[72] R. Micheloni, R. Ravasio, A. Marelli, E. Alice, V. Altieri, A. Bovino, L. Crippa, E. D. Martino, L. D. Onofrio, A. Gambardella, E. Grillea, G. Guerra, D. Kim, C. Missiroli, I. Motta, A. Prisco, G. Ragone, M. Romano, M. Sangalli, P. Sauro, M. Scotti, and S. Won. A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36MB/s System Read Throughput. In *Proc. Int'l Solid-State Circuits Conference*, pages 497–506, Feb 2006.

[73] A. Modelli, A. Visconti, and R. Bez. Advanced flash memory reliability. In *Int'l Conf. on Integrated Circuit Design and Technology*, pages 211–218, Oct 2004.

[74] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4), Apr. 2001.

[75] T. Ohsawa, K. Fujita, K. Hatsuda, T. Higashi, T. Shino, Y. Minami, H. Nakajima, M. Morikado, K. Inoh, T. Hamamoto, S. Watanabe, S. Fujii, and T. Furuyama. Design of a 128-Mb SOI DRAM Using the Floating Body Cell (FBC). *IEEE Journal of Solid State Circuits*, 41(1), Jan 2006.

[76] S. Okhonin, M. Nagoga, P. Fazan, L. Mathew, B. Nguen, H. Chen, and T. Stephens. FinFET based Zero-capacitor DRAM (Z-RAM) Cell for sub 45 nm Memory. In *IEEE Int'l Conference on Memory Technology and Design*, May 2005.

[77] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proc. Int'l Conf. on Arch. Support for Prog. Lang. and Oper. Sys.*, Oct. 1996.

[78] 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.

[79] J.-W. Park, Y.-G. Kim, I.-K. Kim, K.-C. Park, K.-C. Lee, and T.-S. Jung. Performance Characteristics of SOI DRAM for Low-Power Application. *IEEE Journal of Solid State Circuits*, 34(12), Dec 1999.

[80] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM: IRAM. *IEEE Micro*, 17(2), Apr 1997.

[81] E. Pinheiro, W.-D. Weber, and L. A. Barroson. Failure Trends in a Large Disk Drive Population. In *USENIX Conf. on File and Storage Technologies*, Feb. 2007.

[82] A. Rahman and R. Reif. System-level performance evaluation of three-dimensional integrated circuits. *IEEE Trans. on VLSI*, 8, Dec. 2000.

[83] F. Ricci, L. T. Clark, T. Beatty, W. Yu, A. Bashmakov, S. Demmons, E. Fox, J. Miller, M. Biyani, and J. Haigh. A 1.5GHz 90nm embedded microprocessor core. In *Proc. Symp. on VLSI Circuits*, June 2005.

[84] J. Schutz and C. Webb. A scalable X86 CPU design for 90 nm process. In *Proc. Int'l Solid-State Circuits Conference*, Feb. 2004.

[85] D. Wendell, J. Lin, P. Kaushik, S. Seshadri, A. Wang, V. Sundararaman, P. Wang, H. McIntyre, S. Kim, W. Hsu, H. Park, G. Levinsky, J. Lu, M. Chirania, R. Heald, and P. Lazar. A 4MB on-chip l2 cache for a 90nm 1.6GHz 64b SPARC microprocessor. In *Proc. Int'l Solid-State Circuits Conference*, Feb. 2004.

[86] M. Wu and W. Zwaenepoel. eNVy: a non-volatile, main memory storage system. In *Proc. of Int'l Conf. on Arch. support for prog. languages and operating systems*, Oct. 1994.

[87] L. Xue, C. C. Liu, H.-S. Kim, S. Kim, and S. Tiwari. Three-dimensional integration: Technology, use, and issues for mixed-signal applications. *IEEE Trans. on Electron Devices*, 50:601–609, May 2003.