ABSTRACT

Title of Document: DISK DESIGN-SPACE EXPLORATION IN TERMS OF SYSTEM-LEVEL PERFORMANCE, POWER, AND ENERGY CONSUMPTION

Nuengwong Tuaycharoen
Doctor of Philosophy, 2006

Directed By: Associate Professor Bruce L. Jacob
Department of Electrical and Computer Engineering
University of Maryland, College Park

To make the common case fast, most studies focus on the computation phase of applications in which most instructions are executed. However, many programs spend significant time in the I/O intensive phase due to the I/O latency. To obtain a system with more balanced phases, we require greater insight into the effects of the I/O configurations to the entire system in both performance and power dissipation domains.

Due to lack of public tools with the complete picture of the entire memory hierarchy, we developed SYSim. SYSim is a complete-system simulator aiming at complete memory hierarchy studies in both performance and power consumption domains.

In this dissertation, we used SYSim to investigate the system-level impacts of several disk enhancements and technology improvements to the detailed interaction in

memory hierarchy during the I/O-intensive phase. The experimental results are reported in terms of both total system performance and power/energy consumption. With SYSim, we conducted the complete-system experiments and revealed intriguing behaviors including, but not limited to, the following:

- During the I/O intensive phase which consists of both disk reads and writes, the average system CPI tracks only average disk read response time, and not overall average disk response time, which is the widely-accepted metric in disk drive research.
- In disk read-dominating applications, Disk Prefetching is more important than increasing the disk RPM. On the other hand, in applications with both disk reads and writes, the disk RPM matters.
- The execution time can be improved to an order of magnitude by applying some disk enhancements. Using disk caching and prefetching can improve the performance by the factor of 2, and write-buffering can improve the performance by the factor of 10. Moreover, using disk caching/prefetching and the write-buffering techniques in conjunction can improve the total system performance by at least an order of magnitude.
- Increasing the disk RPM and the number of disks in RAID disk system also have an impressive improvement over the total system performance. However, employing such techniques requires careful consideration for trade-offs in power/energy consumption.

DISK DESIGN-SPACE EXPLORATION IN TERMS OF SYSTEM-LEVEL PERFORMANCE, POWER, AND ENERGY CONSUMPTION

By

Nuengwong Tuaycharoen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Table of Contents

List of Tables

List of Figures

CHAPTER 1: INTRODUCTION

1.1. Problem Description

The 90/10 rule states that 90% of the execution time is spent in 10% of the code. Most studies, therefore, focus on the computation phase which contains the most repeated number of instructions--i.e. the main loops, are executed, believing that it is the most important part in the entire course of execution. The argument for this is to make the most repeated case fast. However, this dissertation takes a different path. We are not looking at the duration that the most repeated instructions are executed; we are looking at the duration that the most time spent in the execution.

Run #	User (s)	Kernel (s)	I/O stall (s)	Total (s)
1 (cold cache)	93.11	15.06	600.83	709
2 (warm cache)	92.7	16.3	397.00	506
3 (warm cache)	92.8	14.3	425.90	533
4 (warm cache)	93.3	14.3	460.40	568
5 (warm cache)	93.6	14.3	441.10	549

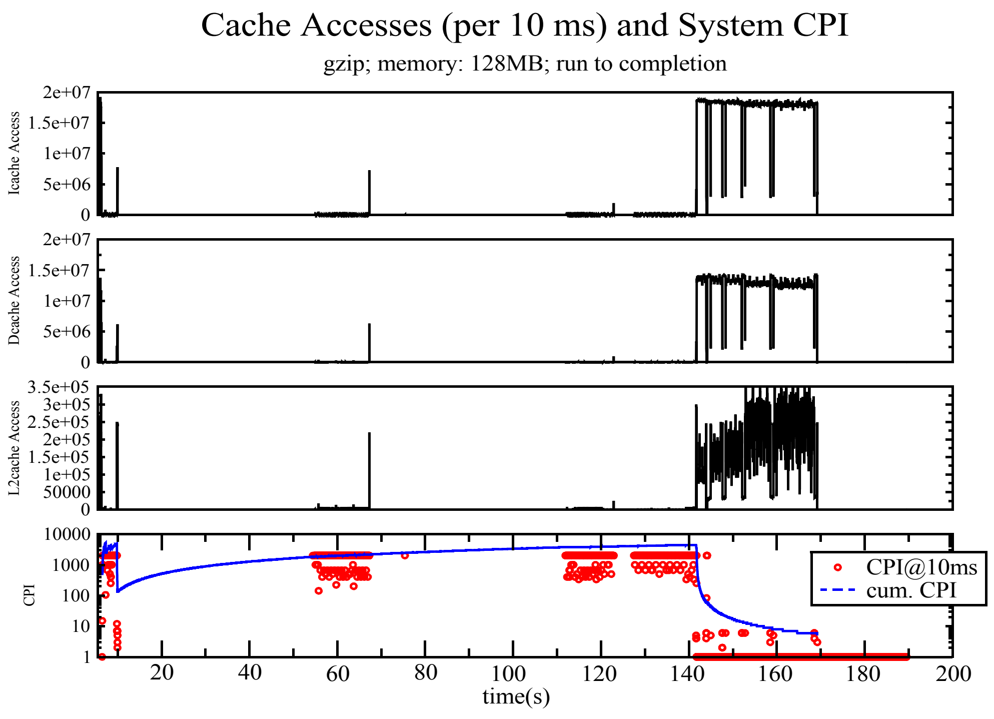
Table 1.1: Execution Time Breakdown for System #1: 750MHz CPU with 96MB memory

Run #	User (s)	Kernel (s)	I/O stall (s)	Total (s)
1 (cold cache)	90.4	6.4	164.20	261
2 (warm cache)	90.1	6	126.90	223
3 (warm cache)	89.8	5.7	129.50	225
4 (warm cache)	90.5	5.5	121.00	217
5 (warm cache)	90.3	6.1	168.60	265

Table 1.2: Execution Time Breakdown for System #2: 750MHz CPU with 128MB of memory

Table 1.1 shows the execution time breakdown for gzip in a real system. The system has a 750MHz CPU with 96MB of the system memory and runs Fedora Core 3. Table 1.2 shows the execution time breakdown for gzip in another real system. The second system is the same system as the first system, but the system memory is set to 128MB. One would expect that the memory in both systems should be large enough to run a SPEC benchmark Though the systems spent significant amount of time executing the user code, they also spent more time stalling for I/O.

Figure 1.1 shows the simulation results of an entire execution of gzip, a spec benchmark, on our complete-system simulator--SYSim, in a single-processing environment. The system configuration used in this example is a 2-GHz Pentium processor, 128MB of main memory, and a 12k-RPM disk drive with built-in disk cache. The figure shows the interaction between all components of the entire memory hierarchy, including the level-1 instruction cache, the level-1 data cache, the level-2 unified cache, the DRAM, and the disk drive. All graphs use the same x-axis, which represents the execution time in seconds. The x-axis does not start at zero since the system boot time is excluded. Each data point is collected for every 10 milliseconds epoch. The CPI graph shows 2 system CPI values: one is the average CPI for any 10 milliseconds epoch, the other is the accumulated average CPI. The duration with no data point displaying means no instructions are executed due to the I/O latency. The application is run in single-user mode to make accurate calculation of execution time. Otherwise, the kernel would swap to other processes on a system call to read from disk. Therefore, disk delay shows up as stall time. The course of execution when the accumulated average CPI is over 100 is the I/O intensive phase (i.e. before the 140th second), while the



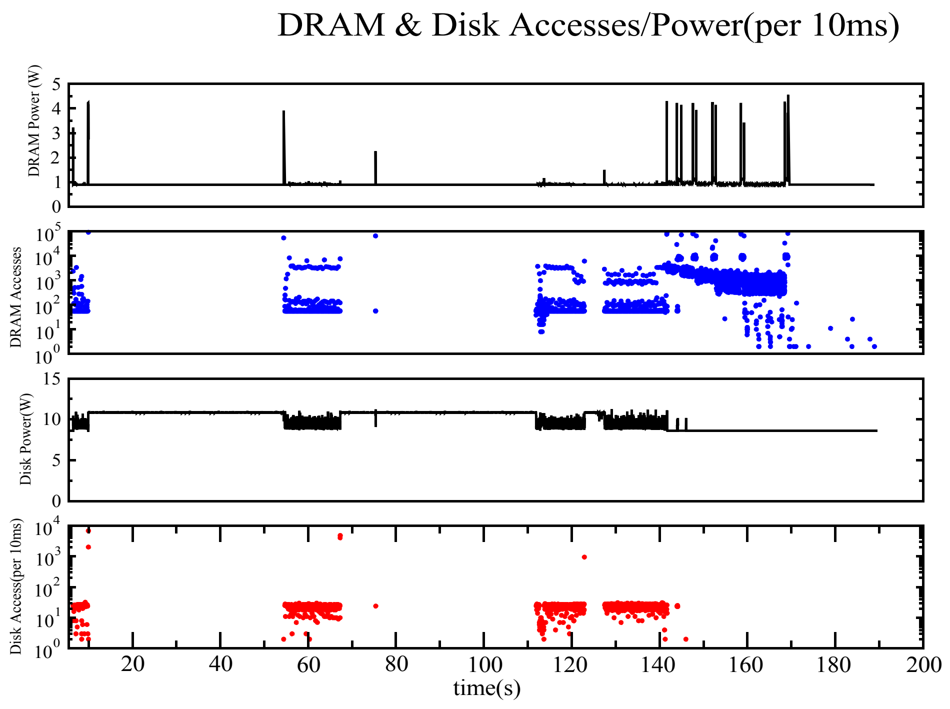


Figure 1.1: The System CPI. The figure shows the System CPI over the entire run of gzip. The system configuration is a 2-GHz processor with 128MB of memory and a 12k-RPM disk. The CPI graph shows 2 CPI values: one is the instant CPI for every 10ms, another is the accumulated average CPI. The duration having no data point means no instructions are executed due to the I/O latency. The course of execution when the accumulated CPI is over 100 is the I/O intensive phase, and the course of execution when the CPI is below 100 is the computation phase.

course of execution when the CPI is below 100 is the computation phase. Note that the CPI, the DRAM accesses, and the Disk accesses are in log scale.

During the course of the execution, there are I/O intensive phase and steady or main computation phase. The figure shows that the program spent a significant amount of the time, if not most, in the I/O intensive phase due to the I/O activities. Unlike the disk, the other components in the memory hierarchy cause very little activity during the I/O intensive phase. On the other hand, those components are accessed regularly during the computation phase which is where the most instructions are executed, and a phase during which the disk is mostly idle. Therefore, the I/O intensive phase has been exposed as a significant component due to the I/O latency with respect to the total execution time.

Most studies skip the I/O intensive phase due to the claim that it is unimportant as it is only executed once. However, despite being run only once, the I/O intensive phase takes far longer than other phases. The underlying reason for the lack of attention to this issue is that the I/O intensive phase is dealing with I/O activities which only a small number of tools implement due to the complexity and time-consuming experimentation. These tools can take years to develop, and a single data point on an experiment can take weeks or even months to execute. For these reasons, most publications conduct an experiment only on the computation phase and claim a single digit CPI value. Unfortunately, they entirely ignore the effects of the I/O intensive phase. As figure 1.1 shows, the CPI value can vary by many orders of magnitude due to the I/O activities during the long I/O intensive phase. CPI finally reduces to a single-digit number during the computation phase as portrayed by many studies. Therefore, the techniques that simply ignore the I/O intensive phase and claim 10% or even a factor of 2 improvement over only computation phase, as exhibited in most processor, cache, and DRAM enhancements, may have only minor impact for the entire course of execution.

The solution to this problem is to investigate the I/O. To obtain a system with more balanced phases, we require more understanding of the effects of the parameter configurations of the I/O to the entire system, especially during the I/O intensive phase. A variety of disk optimization techniques including caching, write buffering, prefetching, and parallel I/O have been invented to optimize I/O operations. These techniques have been in place in the server-classed disk drives for over 10 years. As the technology is getting cheaper with the time, there is a shift to using these techniques in PC disk drives as well. For example, in general, an optimizing technique would be first introduced in server drives. Then, a few years later, the technique will be implemented in desktop drives. After widely used in desktops for another few years, it would be applied in mobile drives. As a result, an optimization technique would take approximately 10 years to shift from server class disks to mobile drives. In addition, with better technology, the disk physical characteristics are also improved, including the RPM (rotational speed in term of round-per-minute), the seek time, and the disk drive interface.

The effectiveness of these techniques in term of total system performance, however, is not clear because they have been studied in isolation by different researchers using different methodologies. As the performance gap between the processor and disk-based storage continues to widen, increasingly aggressive optimization of the storage system is needed. This requires a good understanding of the real potential of the various I/O optimization techniques and how they work together. Therefore, we are required to study the effects in the full-system scale.

To our knowledge, this dissertation is the first to explore disk design space during the I/O intensive phase, and the results are reported in both total system performance and the power/energy consumption. We will show later that the overall system performance can be improved greatly by the enhancements in the disk drives, i.e. using disk caching and prefetching can improve the performance by the factor of 2, and write-buffering techniques can improve the performance by the factor of 10. Moreover, the combination of disk caching/prefetching and the write-buffering technique is the most important enhancement to focus on since the enhancement can improve the total system performance over an order of magnitude without increasing the energy consumption significantly. Increasing the disk RPM and the number of disks in RAID disk system also have an impressive improvement over the total system performance. However, since such techniques can consume significant energy, they have trade-off to be considered carefully. Our studies also revealed that as the capacity of the main memory decreases to the capacity which causes memory page swapping during the I/O intensive phase, the overall system performance decreases greatly. This type of behavior will also continue to show as the size of application's memory footprint grows, which is the trend for the future applications. Therefore, increasing the memory size will only solve the problem in the short term. The long-term solution is to improve the disk system performance.

Recently, the design trade-off of performance versus power consumption has received much attention because of the following [36]:

1. the ever-growing number of disk-based mobile systems that need to provide services with the energy supplied by a battery of limited weight and size;
2. the technical feasibility of high performance computation due to heat extraction; and

3. concerns about the operating costs of large systems caused by electric power consumption as well as the dependency of systems operating at high temperatures because of power dissipation. For example, a data warehouse of an Internet service provider with 8000 servers requires 2 MW [36].

Thus, the demand for low-power systems is increasing not only for mobile systems but also for general-purpose systems. Additionally, the energy consumption of the computer systems will scale up as they become more complex.

In general, optimization techniques aiming at performance do not necessarily optimize power consumption and can often lead to more power consumed. An et al. [33] demonstrate this fact with spatial database application. They evaluate three spatial indexing methods for memory resident database in embedded systems from both the energy and performance angles. Their experimental results show that one indexing method is superior to others in performance angle while aggravating the power dissipation. Other publications also exhibit that performance optimization techniques can aggravate the power dissipation. For example, in [34], the simulation results of an embedded system running an MPEG video show that using a 4-way set-associative, burst SDRAM L2 cache in addition to the L1 cache improves performance by approximately 10% but almost doubles the total energy consumption. Therefore, since there are trade-offs in the performance and energy consumption, care must be taken to apply the performance optimizations.

Energy-efficient design requires reducing power dissipation in all parts of the design. Design decisions in a part of a system (e.g., the micro-architecture of a computing element) can affect the energy consumption in another part (e.g., memory and/or memory-processor busses), or even affect the energy consumption in many parts. For example, reducing miss rates in L1 cache can reduce the access and power consumption in other lower-level memory. Another example is that a power reduction technique in the memory can be responsible for increasing the power consumption in the processor as demonstrated by Kandemir et al. [40]. They evaluate five state-of-the-art high-level compiler optimizations on energy consumption, considering both the processor core (datapath) and memory system. They found that, while most performance oriented optimizations reduce the overall energy consumption, they also increase the energy consumption in the datapath. As a result, energy-efficient system-level design must address the reduction and balance of power consumption in all constituents.

Nowadays, the cost per memory bit is extremely low, and sheer memory size is rarely the main issue in system design. In stead, memory performance and power are now the key challenges. Memory accesses become slower with respect to the processor and consume more power with increasing memory size. Many studies show that memory power and access time dominate over 50% of the total power and performance for computations with large storage requirements[37, 38, 1]. As a result, memory becomes the main bottleneck.

All advanced memory organizations rely on the concept of memory hierarchy. High hierarchy levels are made of small memories, close to computation units, and tightly coupled with them. Low hierarchy levels are made of increasingly large memories, far from computation units, and shared. Hierarchical organizations reduce memory power by exploiting non-uniformities in access frequencies. Most applications access a relatively small area in memory with high frequency, while most memory locations are accessed a small number of times. In a hierarchical memory, frequently-accessed locations should be placed in high hierarchy levels, closer to the processor, thereby minimizing average cost per access.

One technique which is implemented in many levels of memory hierarchy is caching. Caching temporarily holds data that is likely to be utilized in a faster memory, called cache. The term “cache” is used in every level in the memory hierarchy where the technique is applied. Therefore, terms, such as Memory Cache, Disk Cache, and File System Cache, confuse most people, even the people in Operating Systems and disk research. Sometimes they use these terms interchangeably. To clarify the terms used in this dissertation, the terms are defined as following. All Memory Cache, Disk Cache, and File System Cache serve the same purpose, which is to hide the disk latency by caching the data in the location closer to the processor. In this dissertation, Memory Cache indicates the level-1 data cache, level-1 instruction cache, level-2 cache, and so on. Disk Cache, sometimes called Disk Buffer, is a set of memory chips physically located in a disk drive. Disk Cache usually exists without the knowledge of the operating system, and it is controlled by the processing unit embedded in the disk drive. The File System Cache is a part of the system memory, managed by the operating system and reserved for files which are read from the disk system. Therefore, the File System Cache is physically located in the main memory which is usually in the DRAM, but it can be anywhere in the memory hierarchy.

Approaches to memory optimization considering both power and performance in the literature can be grouped into three classes [36]:

1. *Memory hierarchy design* assumes a given dynamic trace of memory access, obtained by profiling an application, and produces a customized memory hierarchy. Many publications present strategies for the optimal cache

configuration [43, 44, 45, 46, 47], and others partition each memory level into subbanks to be able to put into low-power mode when not used [47, 49, 50, 51]. Many recent publications insert specialized buffers in the hierarchy [52, 53, 54, 55, 56, 57] to improve data locality in each memory level and/or to reduce traffic between them. These specialized buffers are also used as instruction compression buffers [58, 59, 60, 60].

2. *Computation transformations for memory optimization* assumes a fixed memory hierarchy and tries to modify the storage requirements and access patterns of the target computation to optimally match the given hierarchy. For example, data structure selection [63, 64], register and memory allocation [65, 66], dynamically switching devices to low-power mode [67, 68], extending the low-power duration of a device by code transformation [69, 70], and reducing memory address bus transitions [71, 72, 73, 74].
3. *Synergic memory and computation optimization* tries to concurrently optimize memory access patterns and memory architecture [41, 42, 62].

In this dissertation, we explore the disk drive optimization techniques, which are physical improvements and enhancements with additional hardware. Since we only concentrate on customizing a level in memory hierarchy without attempts to modify the access pattern, our approach can be classified into the first group, *Memory hierarchy design*.

A magnetic disk has been considered a fundamental component in a computer system since 1965 [39]. It primarily serves as long-term, non-volatile storage for files, and as a level of the memory hierarchy below main memory. Disk is included in the virtual memory implemented in most popular operating systems as a slow memory during program execution. Though disk is an indispensable part of general-purpose computer systems, so far no literature addresses the complete picture of the memory hierarchy including disk, or how memory systems (caches and DRAM) interact with disk in both performance and power dissipation. As we will show in the next section, one reason is that there are no proper tools available in the public domain for such studies. Such a tool would have to demonstrate accurate interactions between the caches, the main memory, and the disk via I/O requests from the operating system. The components must be implemented in detail to capture the systemic interactions between them. Furthermore, as low-power consumption is another system requirement, the tool needs to estimate the instantaneous power dissipation in each component to reflect the efficiency in power consumption. Such tools are considered very complex to implement.

Therefore, SYSim was created to be a complete-system simulator aiming at complete memory hierarchy studies. SYSim focuses on demonstrating the detailed interaction in memory hierarchy in both performance and power domains. In this dissertation, we employed SYSim to explore several disk enhancements and the disk physical technology improvements during the I/O intensive phase. The experimental results were reported in both total system performance domain and the power/energy consumption domain.

1.2. Contribution and Significance

The contribution in this dissertation is two-fold. First, we explore several disk enhancements and disk physical technology improvements in both isolation and in combination, considering both total system performance and the power/energy consumption, focusing on the I/O intensive phase. Secondly, we create a complete-system simulator, SYSim, to demonstrate the detailed interaction in memory hierarchy in both performance and power domains.

With SYSim, we are able to conduct the complete-system experiments to evaluate the disk optimization techniques effect on actual total system performance and power/energy consumption. To our knowledge, this dissertation is the first to explore several disk enhancements and technology improvements both in isolation and in combination during the I/O intensive phase of applications. The disk enhancements we studied include disk caching, prefetching, write buffering, and parallel I/O. In addition, the disk technology improvements we explored include the disk seek time, rotational speed, and interface data rate. The results are reported in terms of both total system performance and the power/energy consumption. We captured the following intriguing behaviors:

- During the I/O intensive phase which consists of both disk reads and writes, average CPI tracks only average disk read response time not overall average disk response time, which includes both disk read/write response time. This is important because most disk studies report performance in terms of average disk response time.
- The effect of the size of the disk cache is limited to the presence of the cache with a particular size. That is increasing the size of the disk cache will not result in better performance if the disk cache is already large enough. This behavior is also agreed

with the file system cache size and disk cache size exploration by Zhu and Hu in [75].

- In the disk read-dominating benchmarks, Disk Prefetching is more important than increasing the disk RPM. That is rotational latency and bandwidth can be overcome by simple prefetching during disk read phase.
- In the benchmarks containing both disk reads and writes, the disk RPM matters. This is because the disk maintains the concepts of non-volatile storage, so disk write requests must be processed to the disk immediately. The experiment shows that using some techniques, such as, NVRAM to buffer the writes, may improve the performance significantly.
- Increasing the number of disks in a RAID system does not proportionally translate into better performance. For instance, increase number of disks form one to eight does not improve performance by the factor of 8. Worse, the power/energy consumption does increases proportionally by the number of disks: a system with 8 disks dissipates roughly 8 times the power of a single-disk system.
- The cost of writing in RAID system is significant as the RAID systems usually suffer from small writes [80]. If the cost of a write is reduced, such as by implementing write buffer mechanism, the overall system performance will be improved significantly.
- Individual DRAM chips dissipate little power, but a system must have a substantial amount of DRAM to keep the disk from dissipating significant power. Moreover, when there is sufficient DRAM capacity in the system, the total DRAM power can be significant.

- The energy consumption seems to have more significance than the power dissipation. The energy consumed can grow significantly with different disk parameters because the I/O latency substantially prolongs the program execution time.
- In the systems with fast disks, increasing the system bandwidth alone fails to improve performance directly. To significantly improve total system performance further, disk enhancement techniques are required.

1.3. Organization of Dissertation

The dissertation is organized as follows: Chapter 2 provides an overview of the memory hierarchy. Both Caches and DRAM-based memory systems from the system level down to the circuit level are discussed to provide the reader fundamental insights about the memory hierarchy. Chapter3 gives a background about disk drives in computer systems, and describes a drive’s physical components, data organization, and interfaces. Chapter 4 discusses related works in the literature for the dissertation. It consists of insight about system simulators in the research community, disk drive enhancements, and disk drive technology improvement that we considered in this dissertation. Methodology for proposing simulator, SYSim, is discussed in Chapter 5. It details the parameter settings in our experiments, the simulator implementation details, and sample output to provide more insight in our proposed complete-system simulator. Chapter 6 presents the experiments and results for the memory system studies, mainly about system-level behaviors focusing on the disk system configuration. Finally, we end the dissertation with the conclusions.

CHAPTER 2: MEMORY HIERARCHY

In this chapter, we will be providing background information regarding the memory hierarchy in a general-purpose computer system, primarily focusing on the cache and main memory. The chapter will consist of detailing the memory hierarchy in general. Other topics discussed will be the first level in the memory hierarchy and cache. A detailed background on cache design is given to serve as the foundation while discussing concepts used in cache tools. We also include the basics of cache design. The cache tools we used in our experiment, CACTI and Wattch, are also introduced, as well as a general explanation of how the selections have been made for the cache configuration and power dissipation calculation. After that, we discuss the main memory included in a general-purpose computer, based on DRAM (Dynamic Random Access Memory). Then, the basic structures of DRAM devices and memory system organizations are described in some detail. The section starts with the description of the smallest unit of the DRAM, a memory cell, moves upward to the DRAM device, and then to DRAM system organization. Next, the DRAM memory access protocol is discussed to provide an explanation of fundamental DRAM operations and interactions between them. Finally, the chapter concludes with abstracted concepts of DRAM memory controller design.

2.1. Memory Hierarchy

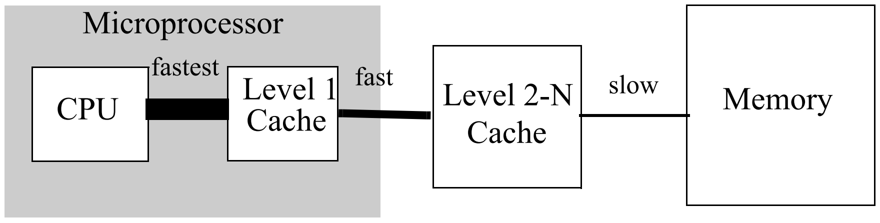


Figure 2.1: Memory Hierarchy in Typical Computer Systems.

A relatively unlimited amount of fast memory with low cost is always a requirement for future computer systems. Memory hierarchies have been invented to support this requirement. A memory hierarchy is defined as the hierarchical arrangement of storage in computer architecture. The hierarchy takes into consideration the advantage of both locality of accesses and the cost-performance ratio of memory technologies. The principle of locality states that most programs do not access all code or data in well-distributed spatial or temporal distributions. The programs have some forms of locality in their accesses. Another principle is the memory hierarchy organization. Each level of the hierarchy is organized in such a way that it can be accessed with higher speed and lower latency than lower levels. These two principles are the basis of the hierarchy, which is based on memories of different speeds and sizes. Since fast memory technology is expensive, a memory hierarchy organizes different speeds and sizes of memory into several levels, with the smaller, faster, and more expensive memory placed closer to the processor. The goal is to provide a memory system with cost comparable to the cheapest level of memory and speed comparable to the fastest level of memory. The levels of the hierarchy usually are inclusive, meaning that data located in the upper levels are also included in the levels below. The data in the memory hierarchy may also be exclusive, which means data is allocated in only one level at a time among multi-level caches. Note that each level maps addresses from a larger memory to a smaller but faster memory that is placed closer to the processor in the hierarchy. As part of address mapping, the memory hierarchy also provides address checking and protection schemes preventing harmful address accesses.

2.2. Virtual Memory

Virtual memory allows programs to run in a memory address space whose size and addressing are independent from the computer's physical memory. If a program exceeds the physical memory capacity, virtual memory automatically loads or unloads pages without the user program knowing. A page is simply a chunk of memory that is loaded or unloaded as a single unit. Therefore, virtual memory reduces the startup time for a program since only the necessary pages are loaded at startup. On the other hand, in a multiprocessing environment, multiple processes can run simultaneously with their own independent address spaces. Virtual memory divides physical memory into pages and allocates them among different processes. It locates the unnecessary pages in some secondary storage and loads only pages necessary for multiple processes at a given moment. It automatically manages the two levels of the memory hierarchy, which are main memory and secondary storage. Virtual memory also protects the processes' address spaces by restricting the processes to only the address spaces to which they belong.

With virtual memory, the address referenced by the processor is called a virtual address. The virtual addresses are translated by a combination of hardware and software to physical addresses, which are used to access main memory. This process is called memory mapping or address translation. A data structure called a page table is used in address translation.

Each page table entry is indexed by the virtual page number and contains the physical page address. Two virtual page numbers can map to the same physical page frame. However, the size of the page table is quite large compared with the size of the system memory: the page table can occupy approximately 0.1 -1% of the system memory. Additionally, the address translation process would deteriorate the system performance if every memory access required another access for address translation.

The TLB (translation look-aside buffer) is introduced to mitigate the cost of the address translation process. Since the address translations for accesses have spatial locality, the TLB is used to improve address translation by caching recently accessed page table entries. The process of address translation via TLB can be placed before or after the cache depending on what scheme is used. Therefore, address translation is directly related to caching, and it will be discussed further in the next section regarding cache operations. The page table entries cached by TLB have been recently referred to by the processor and therefore their physical pages are located in main memory. When the processor refers to an address, the system looks it up in TLB first. If the entry is found in the TLB, the referring virtual address is translated to a physical address accordingly. This process eliminates the need to look up the address in the actual page table located in main memory. Therefore, accessing memory to look up a page table entry for address translation is not necessary in this case. However, when a TLB miss occurs, which is when the referring page table entry is not found in the TLB, the system unavoidably accesses the page table for the entry. The TLB misses may cause a series of activities to allocate a new page for the entry, including memory page allocation, page table entry creation, and TLB entry insertion, depending on the status of the page.

The TLB miss is handled by the operating system or hardware. If the memory page was previously allocated, the operating system would only require looking up of the page table entry in the page table and inserting it into the TLB. Then the access can be restarted. In the process of scanning through the page table, the operating system may find that the referring entry is actually *unmapped*. A *mapped* virtual page is defined as a virtual page previously allocated by the operating system, and whose mapping information is currently maintained under the operating system's awareness. In contrast, an *unmapped* page can occur in two situations which are (1) it has not been previously allocated by the operating system, or (2) it has been de-allocated. In either situation, its mapping information is discarded, and the system initiates the page allocation process mentioned above. Virtual memory systems can be categorized into two classes: those with fixed-size blocks, called pages, and those with variable-size blocks, called segments.

2.3. Caches

The term cache is applied to the techniques buffering reusable data since locality exists in the reference stream. Caching can improve the performance in many computer system levels. Examples include file caches, disk caches, name caches, web caches, etc. Generally, caches in computer architecture refer to the first levels of the memory hierarchy. Caches have been an integral and ever-growing part of modern microprocessors. Approximately half of a modern microprocessor's die area is allocated for caches and the number of transistors dedicated to caches is still growing. They are usually implemented with a number of arrays of SRAM cells because an SRAM cell is superior in terms of performance versus a DRAM cell. Caches also use the same fabrication process as the processor core.

2.3.1. Cache Memory Cell

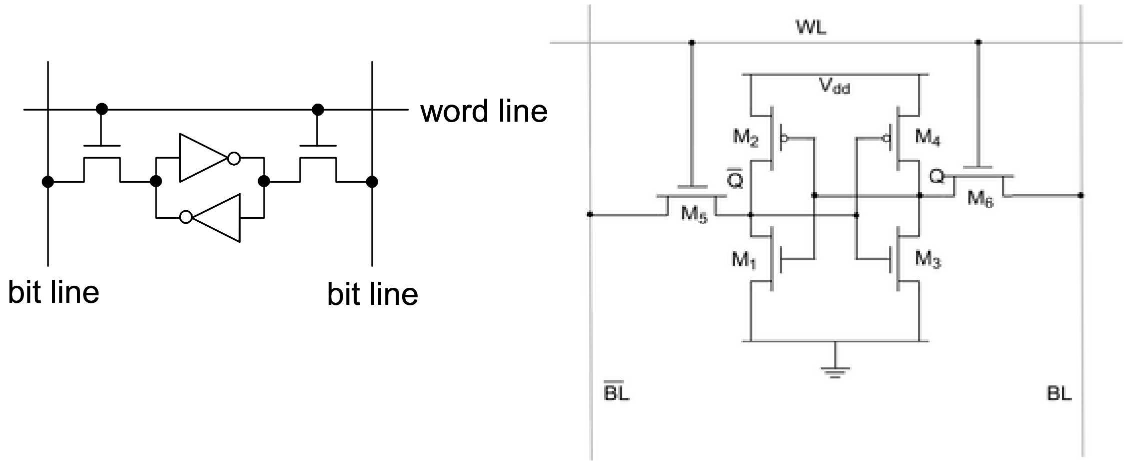


Figure 2.2: The basic SRAM cell--six-transistor memory cell(6T MC).

Figure 2.2 shows the basic SRAM cell, which is an implementation of a six-transistor memory cell (6T MC). This 6T MC cell is connected in a way that it restores the charge back into the memory cell, so the charge does not decrease and the need for refreshing the cell is eliminated. The 6T MC has only one port, which can be used to either read or write a values. When there is a read access, the wordline (WL) is asserted and the voltage difference between the bitline pair is detected. The pair of cross-coupled transistors and the access transistors pull the voltage all the way up to Vdd or down to the ground according to the bit data value. After that, the bitlines are precharged to the original voltage for the next access. In the case of a write, the bitline is driven with a differential voltage from an external source according to the new data to be written. The wordline WL is then asserted and the value that is to be stored is latched into the memory cell. Most conventional designs use this full-CMOS six-transistor memory cell with different variations including sizing, physical layout, and transistor threshold voltages for low power.

2.3.2. Cache Operations

A cache consists of multiple blocks of data. Data blocks are usually organized in a set-associative manner. For example, a cache with four sets and two blocks per set is called two-way set associative. A data block mapped onto a particular set can be freely placed in any of those two blocks in the set. The one-way set associative cache is called a direct-mapped cache and the cache with one set is called fully-associative. Intuitively, more blocks per set translate into more freedom to place the data block to any available blocks in the set, but, as a result, more time and/or power is taken to process the lookup. A cache consists of two arrays, which are the tag array and the data array. The tag array contains an address tag on each block frame that gives the address the data block contains. It also contains a valid bit to identify the validity of the tag entry.

Figure 2.3 shows the read operation to a cache. The virtual address from the processor is divided into virtual page number and page offset. The virtual page number is translated to a physical page address via a page table entry in the TLB. Then, the physical page address is combined with the page offset to produce a physical address. Next, the physical address is divided into tag, index, and block offset. The block offset field selects the desired bytes from the block, the index field selects the set in the cache, and the tag field is compared in parallel against the tags of a selected cache set for a hit. If the address tag is matched with any of the tags from the selected cache set, it is called a cache hit. If it is not a match, it is called a cache miss, and the physical address is forwarded to the next level of memory hierarchy. Note,

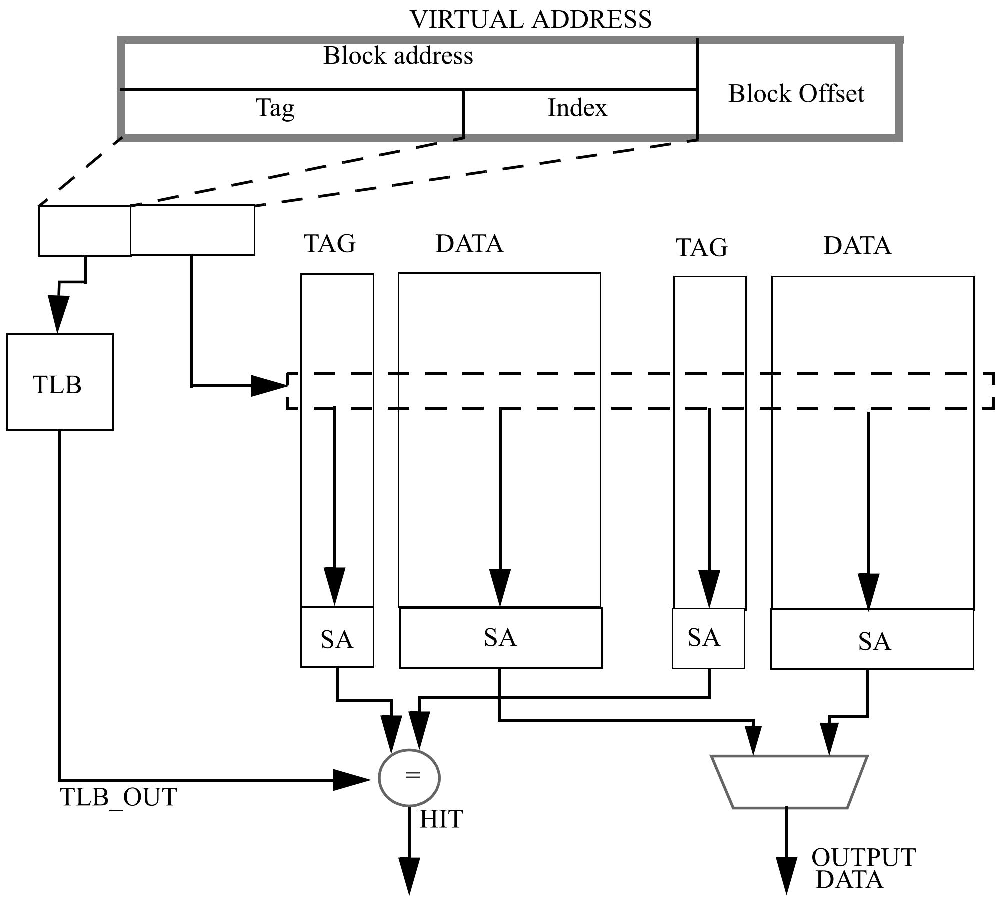


Figure 2.3: Block Diagram of a 2-way set associative cache organization,

accessing the cache and address translation at the TLB can be done simultaneously depending on how the virtual address is divided and translated.

The address translation process can be different than the process described above. It depends on whether the virtual address or the physical address is used to tag and index the cache. The virtual address referred to by the processor core is not the same as the physical address which is used to refer to the location physically in memory, so the system requires address translation. Modern processors handle the address translation with the cooperation between the TLB and the operating system. The differences in caches and TLB placement in the memory hierarchy create different address translation schemes. These different address translation schemes are described below.

The first scheme is physically indexed, physically tagged (PIPT). In PIPT caches, both the tag and index of the cache are in the form of physical addresses. Meaning that the tag and index are identified after the completion of the address translation. Therefore, the TLB lookup process has to be completed before the cache can be accessed, which can slow down the system. However, if the referring page table entry is not a valid entry in the TLB, the system has to access the memory for the entry, translate the address, and then access the cache sequentially. As a result, the benefit of using TLB can be overshadowed by sequential memory accesses for the TLB miss process followed by the actual memory access.

The second scheme is virtually indexed, virtually tagged (VIVT). In VIVT caches, unlike PIPT, both the index and tag are identified from the virtual address directly from the processor without the translation. This VIVT scheme improves the speed of cache access significantly by eliminating the address translation. However, the scheme suffers from several problems, including:

1. Care must be taken to changes in TLB entries and changes in address space since virtual address translations usually are changed as part of normal kernel operation. Cache lines must be flushed if the cache lines' translations have changed.
2. Cache line Aliasing Problem: multiple virtual addresses may exist for the same physical address, even in a single address space. Each of these virtual addresses should never be in the cache at the same time, even though they represent the same data.

To solve mentioned problems in VIVT, the virtually indexed, physically tagged (VIPT) scheme is introduced. The VIPT scheme can maintain the speed of cache accesses comparable to VIVT. The scheme is as described in the previous section. The index is identified in the virtual address, but the tag is identified in the physical address. VIPT can solve the aliasing problem because the tag refers to the physical address. Therefore, VIPT can detect aliasing when two identical tags exist in the cache. Depending on OS page mapping and shared memory protocol, a VIPT cache can be constructed in such a way that cache-line aliasing will never occur.

Since, in VIPT, the process of cache lookup and TLB lookup can be processed simultaneously, the cache access speed of VIPT is improved versus PIPT. However, the processor can not acknowledge a cache miss until the address translation is complete.

Finally, the last scheme, physically indexed, virtually tagged (PIVT), is basically not used and is not discussed further.

When a cache miss occurs, a block must be selected to be replaced with the referring data. A direct-mapped cache selects the block specified by the address, since there is only one location that the data can go. On the other hand, the set-associative and fully-associative caches have many blocks to select from. There are a number of strategies employed for selecting the block to be replaced, which are collectively called the cache replacement policy. The most common policy is called least-recently used (LRU), which selects the block that has not been accessed for the longest time.

There are two basic options when writing data to the cache, which are write back and write through. Write back writes the data to the block in the cache. The modified cache block is written to a lower level of the memory only when it is replaced. On the other hand, write through writes to both the block in the cache and to the block in the lower-level memory. Since the data are not needed on a write, there are two common options on a write miss, which are write allocate and no-write allocate. Write allocate loads the block to the cache on a write miss, and then restarts the write which causes a write hit. In contrast, no-write allocate does not load the data in the cache, but modifies the block in the lower level where the data is located.

To allow concurrent accesses, a cache can be given multiple ports. Multiported caches can be implemented using different methods. The most common methods are true multiporting and creating multiple independent banks. True multiported caches include additional access transistors for each port, which cause a significant increase in memory area and wire length within the cache. On the other hand, multiple independent banks divide the cache into small banks, where each bank is a simple single-ported cache. Multiple concurrent accesses can be satisfied if the accesses are to different banks. The disadvantage of multiple independent banks is that the cache controller requires additional complexity and intelligence to control each individual bank.

The cache controller functionality includes controlling cache operations and accesses to comply with cache strategies mentioned above. Its function also includes effectively managing other functions, such as multiporting with multiple banks which requires intelligence to control accesses and prevent bank conflicts. Lastly, the cache controller controls the interfacing mechanism to the lower level of memory when a miss occurs. Although caches can be implemented in many different ways, the simple cache implementation described in this section serves as a fundamental cache design.

One approach to improve cache performance is to reduce the cache miss rate. All misses can be sorted into three categories: compulsory, capacity, and conflict. Compulsory misses are misses caused by accessing the blocks for the first time. These misses occur to bring the blocks into the cache. Capacity misses are misses due to the limitation in the size of the cache because the cache can not hold all the blocks. Conflict misses are misses caused by multiple blocks mapped to the same set, but the set cannot hold all blocks mapped to it. To reduce the misses, there are three common cache organization strategies: increasing the cache size, increasing the cache associativity, and increasing the block size. While increasing the cache size is a costly fool-proof method, increasing the cache associativity and the block size have their optimal points. Increasing both cache associativity and the block size too aggressively can cause the miss rate to increase.

2.3.3. CACTI: An Integrated Cache Timing, Power, and Area Model

CACTI [12] is a widely-accepted analytical model for the access and cycle times of on-chip direct-mapped and set-associative caches. The inputs to the model are the cache size, block size, and associativity, as well as array organization and process parameters. CACTI was originally written by Wilton and Norm Jouppi at DEC WRL. It is available publicly for academic purposes.

Figure 2.4 shows the organization of the SRAM cache being considered in CACTI. First, the decoder decodes the address, then the appropriate row is selected according to the decoded address by driving one wordline in the data array and the corresponding wordline in the tag array. Only one wordline in each array can be asserted at a time. Along the selected

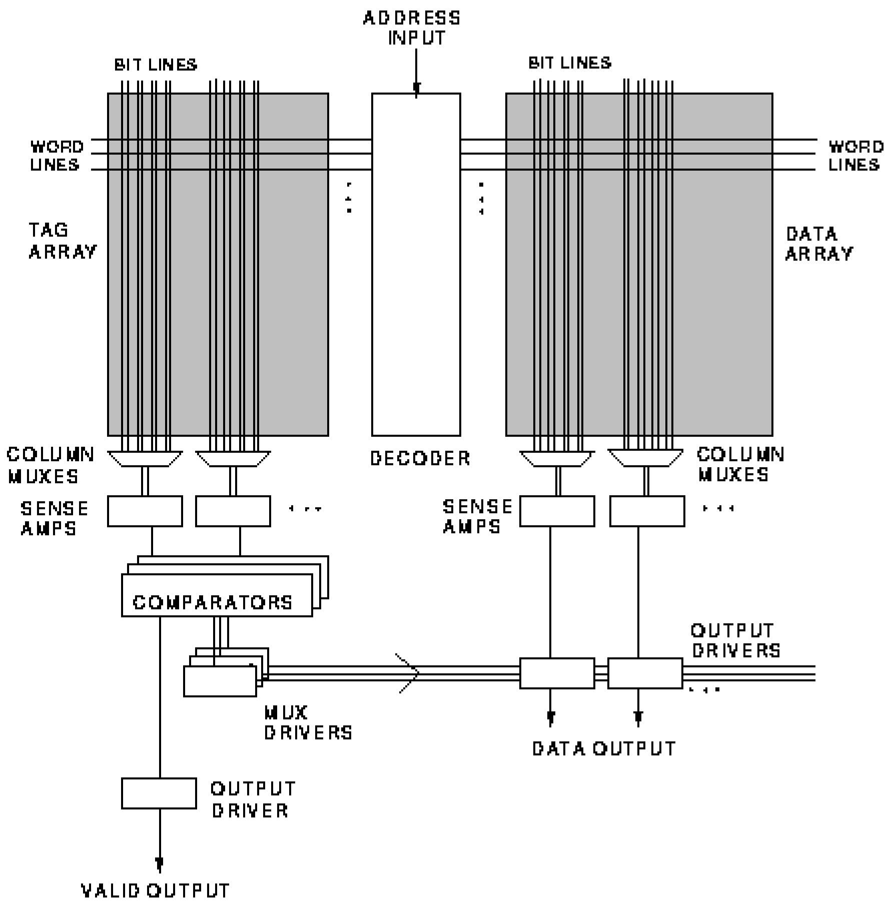


Figure 2.4: Cache structure.

wordline, each memory cell is associated with a pair of bitlines, which are initially precharged high. Then, each memory cell in that row pulls down one of its two bitlines according to the value stored in the memory cell.

Each sense amplifier detects the changes in multiple pairs of bitlines, whose number depends on the layout parameter. The sense amplifier determines the value of the memory cell by detecting which bitline in a pair is pulled down. Multiple pairs of bitlines can share one sense amplifier by inserting a multiplexor before the sense amps. To specify the pair of bitlines to be detected, the select signals from the decoder are fed to the multiplexor.

The data from the tag array is compared with the tag bits of the address. The number of comparators required depends on the number of associativity of the cache, for example an N-way set-associative cache requires N comparators. The comparison results, whether a hit or a miss, drive valid output to the output multiplexors. These output multiplexors select the appropriate data from the data array in the case of a set-associative cache or a cache in which the data array is wider than the output width. Additionally, the output multiplexors drive the selected data out of the cache.

Table 2.1 and Table 2.2 show the input and output parameters of CACTI.

Input Parameter	Use
C	Cache size in bytes
B	Block size in byes signifying the number of bytes in a single cache entry
A	Cache associativity
TECH	Technology node in micrometers
$N_{subbanks}$	Number of cache subbanks
b_0	Number of bits of output data
b_{addr}	Number of bits of system address

Table 2.1: CACTI input parameters

Output Parameter	Use
N_{dwl}	Number of segmentations of the wordline (Data)
N_{spd}	Aspect ratio control parameter (Data)
N_{dbl}	Number of segmentations of the bitline (Data)
N_{twl}	Number of segmentations of the wordline (Tag)
N_{tspd}	Aspect ratio control parameter (Tag)
N_{tbl}	Number of segmentations of the bitline (Tag)

Table 2.2: CACTI output implementation parameters

CACTI calculates the access and cycle times by estimating delays of the cache components, including:

- decoder
- wordlines (in both the data and tag array)
- bitlines (in both the data and tag array)
- sense amplifiers (in both the data and tag arrays)
- comparators
- multiplexor drivers
- output drivers (data output and valid signal output)

The delay of each of these components is estimated separately and the results combined to estimate the access and cycle time of the entire cache. The delay of each component is estimated by decomposing each component into several equivalent RC circuits, and using simple RC equations to estimate the delay of each stage.

There are two potential critical paths in a cache read access. One is the time to access the tag array and the other is the time to access the data array. The time to read the tag array, perform the comparison, and drive the multiplexor select signals is compared with the time to read the data array. If the former is larger, then the tag array is the critical path. Otherwise, the data array is the critical path. Despite the cache designer's attempts for faster tag path compared with the data path, it is not always possible to do this. Therefore, both sides must be modeled in detail to determine the critical path.

The cycle time is calculated by adding the access time and the precharge delay together. The precharge delay is assumed to be dominated by the wordline fall time and bitline rise time in the data array. The wordline fall time is approximately equal to the wordline rise time and a constant bitline rise time is assumed to be equal to four inverter delays (each with a fan-out of four).

To determine the optimal configuration, CACTI performs an exhaustive search over all combinations of the output parameters corresponding to the specified input parameters. The implementation with the best behavior among all criteria is considered the optimal based on the CACTI's optimization algorithm, which is different for each version of CACTI.

CACTI 2.0 is an extension of the first version of CACTI with support for fully-associative caches, multiported caches, feature size scaling, and power modeling. The inputs are the cache capacity, associativity, cacheline size, number of read/write ports, and feature size. Its analytical models compute the access time and the energy consumption of the cache for all combinations of possible configurations. The calculation for each configuration divides the data and tag array into smaller subarrays. Finally, CACTI returns the configuration that has the best access time and energy consumption as determined by its optimization function as mentioned above.

The CACTI 2.0 optimizing function takes into account only the access time of the cache and the energy consumption. CACTI 2.0 does not have a concept of the total area or the efficiency (percentage of area occupied by the bits alone) of each configuration. It approximates the wire capacitance and resistance that is associated with the wires in many parts of the cache because it does not have a detailed area model. Additionally, since the optimizing function considers only the access time and the energy consumption, the cache configuration output may not be efficient in area or aspect ratio.

CACTI 3.0 adds a detailed cache area model to CACTI 2.0. CACTI 3.0 calculates the area occupied by each component of the cache for each possible configuration. The model produces both the efficiency in performance and the aspect ratio of the entire cache for each configuration. To determine the best configuration, the optimizing function considers access time, power consumption, efficiency of the layout, and aspect ratio. The detailed area model accurately calculates the wire lengths and the associated capacitance and resistance of the address and data routing tracks. This results in more realistic power estimates. Finally, CACTI 3.0 also supports fully independent banking of caches.

2.3.4. Wattch

Wattch [19] is a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level. Wattch is claimed to be 1000 times or more faster than existing layout-level power tools, and yet maintains accuracy within 10% of their estimates as verified by using industry tools on leading-edge designs. It provides a power evaluation methodology within the popular SimpleScalar framework. Since, in this dissertation, Wattch is used as a power estimation tool for caches, this section will discuss Wattch only in the context of cache power estimation.

Wattch calculates the dynamic power consumption (P_d) as:

$$P_d = CV_{dd}^2af$$

where

C is the load capacitance,

V_{dd} is the supply voltage,

f is the clock frequency, and

a is the activity factor.

The activity factor *a* is a fraction between 0 and 1 that represents the average switching activity on each clock cycle. Wattch estimates *C* based on the circuit and the transistor sizing as described below. *Vdd* and *f* depend on the assumed process technology as defined in the header file power.h. The user can choose among 0.10, 0.18, 0.25, 0.35, 0.40, and 0.80 micron technology, and Wattch will automatically resize the transistor accordingly.

Wattch calculates the power consumption based on only the capacitance of each stage, rather than both R and C. Additionally, Wattch analyzes and sums the power consumption of all paths, not only the critical path. In Wattch, certain critical transistors are automatically sized based on the model parameters to achieve reasonable delays.

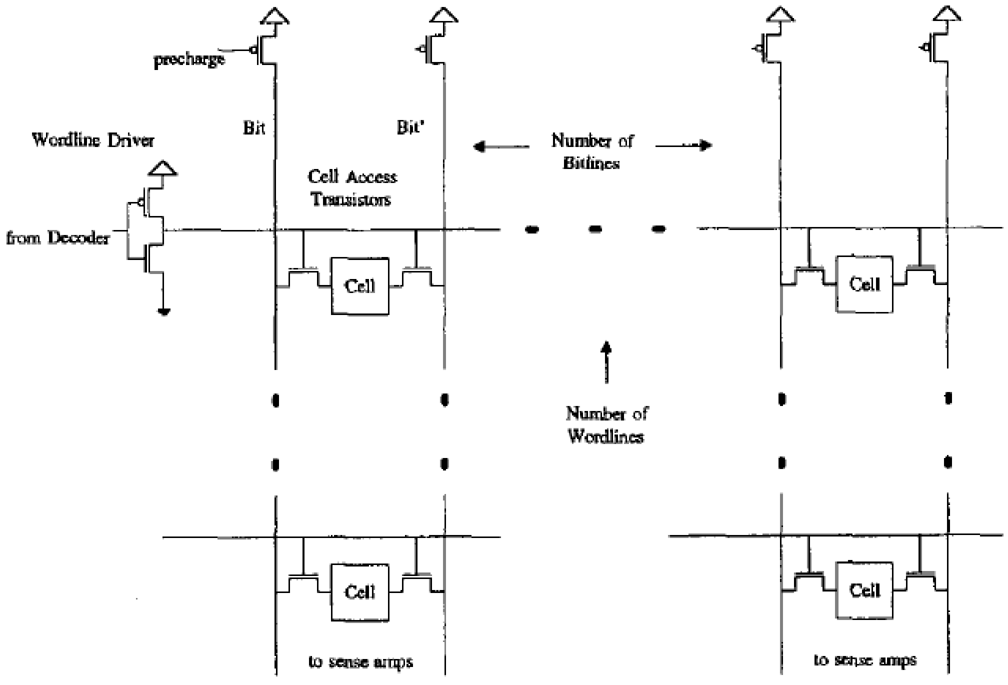


Figure 2.5: Schematic of wordlines and bitlines in Wattch array structure [19].

A cache in Wattch was implemented as an array structure. The power model of the cache is based on the number of rows, columns, and the number of read/write ports. These parameters affect the size and number of decoders, the number of wordlines, and the number of bitlines. In addition, these parameters are used to estimate the length of the pre-decode wires and the lengths of the array's wordlines and bitlines. The wordline and bitline capacitance are computed in a similar way. The wordline capacitance includes the capacitance of the wordline driver, the gate capacitance of the cell access transistor multiplied by the number of bitlines, and the capacitance of the wordline's metal wire. The bitline capacitance includes the diffusion capacitance of the pre-charge transistor, the diffusion capacitance of the cell access transistor multiplied by the number of word lines, and the metal capacitance of the bitline. The number of ports also affects the power consumption due to additional transistor connections on wordlines, two additional bitlines, and longer wires on both wordlines and bitlines.

Wattch authors estimate the physical implementations for cache structures using the help of the CACTI tools [12]. As described in the previous section, CACTI takes the cache size, block size and associativity as inputs, and chooses the organization that gives the smallest access time.

Wattch considers three different options for clock gating to disable unused resources in a multi-ported cache.

1. *All-or-nothing approach.* The full modeled power will be consumed if any accesses occur in a given cycle, and zero power consumption otherwise.
2. *Scaled linearly.* If only a portion of a cache's ports are accessed, the power is scaled linearly with the number of accessed port(s).

3. *Scaled linearly with 10 per cent.* It is the same as the second option except that unused units dissipate 10% of their maximum power, rather than drawing zero power.

To interface with SimpleScalar, the Wattch power model tracks which units are accessed on each cycle and how. The power model also varies the estimated power based on the number of ports used and which clock-gating scheme is used.

2.4. Main Memory: DRAM

The next level down in the memory hierarchy is the main memory. Main memory functions include servicing requests from the cache and interfacing with the I/O. Main memory is usually made up of a set of DRAM chips organized in a way that the memory requests can be sent in interleaving manners. It has been widely accepted that the computer system performance is mostly limited by the performance of DRAM-based memory systems. The reason is that the gap between the rate of DRAM memory system performance improvement and the rate of processor performance improvement has been continuously increasing during the past thirty years. This phenomenon is well-known as the memory gap. There are two main reasons in this phenomenon. First reason is the slow improvement in the interface between the processor and the DRAM due to off-chip location of DRAM. The second reason is the decision of implementing enhancements in DRAM chips relies heavily on the manufacturing costs. Therefore, only enhancements with significant performance improvements for minimal manufacturing cost are considered for standard DRAM devices.

This section discusses DRAM devices and the memory systems organizations. The content of this section is summarized from a part of David Wang's Ph.D. dissertation [17].

2.4.1. DRAM Memory Cell

Figure 2.6 shows the basic DRAM cell, which is implemented as a one transistor and

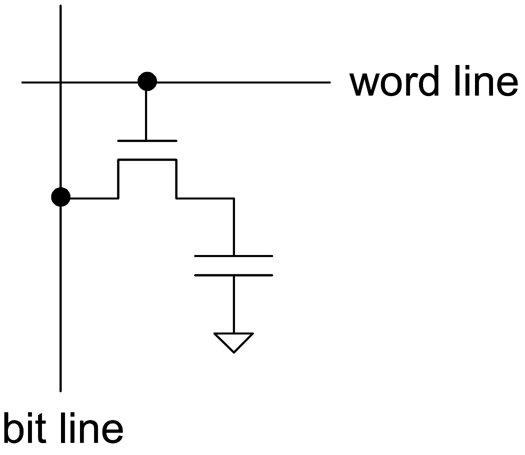


Figure 2.6: a DRAM memory cell--one transistor one capacity (1T1C).

one capacitor (1T1C) memory cell. This memory cell, widely used in modern DRAM devices, contains one data bit. The memory consists of one transistor as an access transistor and one capacitor to hold the charge according to the data of one bit. Before a memory cell can be read, the bitlines have to be precharged to $V_{dd}/2$. The memory cell is read by asserting the wordline to turn on the access transistor, and then the voltage representing the data value is placed on the bitline via the access transistor. The bitline voltage changes are only minimal, but can be detected by the sense amplifier. Then, the sense amplifier senses the value of the data and amplifies the signal up or down. Finally, the charges are restored into the capacitor and the access transistor is turned off by removing the voltage at the wordline. In the write process, only two steps are required. First, the bitline is driven with the new data value. Then, the row select is asserted on a wordline to turn on the access transistor and the data bit is latched into the memory cell. Therefore, the read process is essentially a read with a restoring write.

However, the charges stored in the memory cell capacitor leak through the access transistor. As a result, data stored in DRAM cells must be periodically read-out and written back to restore the charges representing the data value. Otherwise, the charges stored in the capacitor will no longer represent the originally stored data bit. This process is called *refresh*. The DRAM device is usually refreshed every 32 or 64 milliseconds to maintain the usability of the data.

In a DRAM array, the capacitance of a storage capacitor is much smaller than the capacitance of the bitline. Therefore, when the voltage representing the data value is placed on the bitline via the access transistor, the voltage on the bitline is changed minimally. This minimal voltage change on the bitline is difficult to measure in an absolute sense. Therefore, a differential sense amplifier is added in DRAM devices to detect the minimal voltage change by comparing the bitline voltage to a reference voltage.

2.4.2. Standard DRAM Device

Figure 2.7 shows a block diagram for a Fast-Page-Mode (FPM) DRAM device. The description of the standard DRAM device in this section will be based on this diagram. All DRAM devices consist of one or more arrays of DRAM cells, which are organized into a number of rows and columns. A column represents the smallest unit of addressable memory on that device, which can contain multiple bits of data. In this figure, the DRAM cell

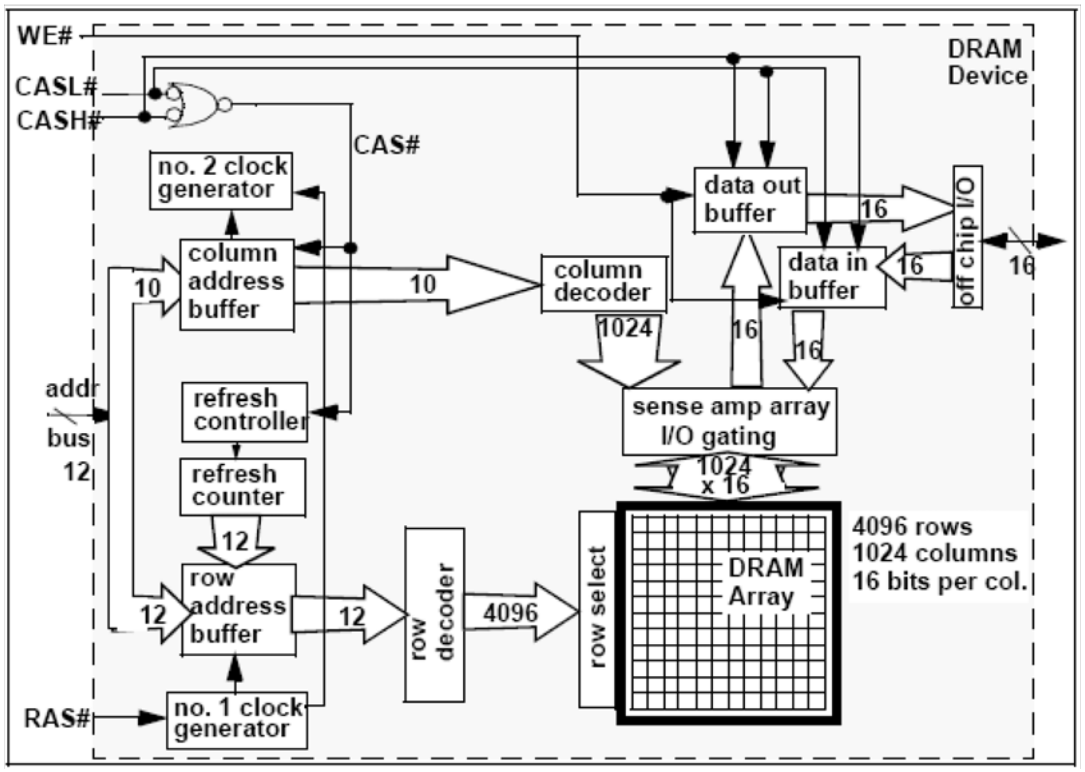


Figure 2.7: 64 Mbit Fast Page Mode DRAM Device (4096 x 1024 x 16) [17].

consists of 4096 rows, 1024 columns per row, and 16 bits of data per column. Modern DRAM devices can have multiple arrays in each device. These arrays are referred to as *banks*. A number of logic circuits are also included in a DRAM device to control the timing and sequence of the device operation, such as the clock generator and the refresh controller.

To access a row in a data array, the memory controller places the row address on the address bus and asserts the row address strobe (RAS) signal. The DRAM device buffers the address on the address bus in the row address buffer and then forwards the address to the row decoder. After that, the row address decoder asserts the specific row of the data arrays according to the accepted address. The data bits in memory cells along the selected row are placed on their corresponding bitlines and then they are sensed and remain in the array of sense amplifiers.

Generally, one or more column accesses follow a row access. After the row access is completed, a number of bits remaining in the sense amplifiers are accessed according to the column access command. Like the row access, the memory controller places a column address on the address bus. Meanwhile, the memory controller also asserts the appropriate column access strobe (CAS#) signals. The DRAM device then accepts the column address, decodes it and selects one column in the sense amplifiers according to the decoded column address. If the command is for a read, the data for that column is then placed onto the data bus and sent to the memory controller. On the other hand, if the command is for a write specified by the write enable (WE) signal, the data are overwritten with data from the data bus.

A DRAM device with a particular capacity can be manufactured in different configurations. For example, a 1 Gbit DRAM device can be configured into 8 banks x 16384 rows x 2048 columns x 4 bits/column, 8 banks x 16384 rows x 1024 columns x 8 bits/column or 8 banks x 8192 rows x 1024 columns x 16 bits/column. However, the larger row size means that the device with more bits per row consumes significantly more current per row activation than the configuration with less bits per row. The larger bitline also translates into more time to complete the refresh process for the entire array. Therefore, the different configurations cause differences in current consumption in DRAM devices. With the differences in current consumption, the DRAM devices require different timing parameters to limit peak power consumption of DRAM devices.

Finally, a set of DRAM devices can be organized to process a request together as a *rank*. A set of DRAM devices can be connected for a particular size of data bus. For example, 8 DRAM devices with 8-bit column can be connected together to create a rank for 64-bit wide data bus to form a single rank of memory.

In SDRAM and DDRx SDRAM devices, a column read command transports the data out from the DRAM devices in burst manner. The column read command moves a variable number of columns, called a burst, as specified on the programmable mode register. The burst mode is possible in these DRAM devices because each column of the device is uniquely identifiable. Given a column address of a multiple-column burst, the SDRAM-based device rearranges the data in the burst and places the data of the requested address first. This capability is known as critical-word forwarding.

In DDRx SDRAM burst mode, multiple columns are moved concurrently from the sense amplifiers to the read latch. Then, the data is pipelined through a multiplexor to the external data bus. This process is called prefetching. With the burst mode, the operating data rate of DDRx SDRAM devices can be improved significantly compared to SDRAM devices. However, the disadvantage of the prefetch architecture is that short-burst accesses are no longer available.

2.4.3. DRAM-Based Memory System Organization

This section discusses how to organize multiple DRAM devices to create a memory system. First, we would like to clarify the terms using in the DRAM research community and also in this dissertation. A *channel* is an interconnection that DRAM devices are connected to and these devices operate in lockstep with respect to each other. Usually, one channel is controlled by one DRAM memory controller. However, one DRAM controller can control more than one channel in concert to create a more efficient memory system. For example, the multiple channels in FPM DRAM were invented to sustain throughput required by high performance workstations and servers prior to SDRAM. The word *bank* is currently used by DRAM device manufacturers to describe the number of independent DRAM arrays within a DRAM device. Multiple banks in a DRAM device support more parallel accesses to the data in the different banks simultaneously. Read requests can be processed simultaneously if they access different banks. The word *rank* is now used to denote a set of DRAM devices that operate in lockstep fashion to commands in a memory system. In DRAM devices, a *row* is simply the group of storage cells that are activated in parallel in response to a row activation command. In general, DRAM devices are connected as ranks of DRAM devices operating in lockstep. This organization causes a specific row in all DRAM devices in a rank to be activated concurrently with a single row activation command. This means that a DRAM row actually spans multiple DRAM devices of a given rank of memory. A *column* of data is the smallest independently addressable unit of memory in a DRAM device.

Memory system organizations in many computer systems are typically non-uniform. The system can contain different sizes and organizations of DRAM devices. The reason for non-uniformity in a memory system is flexibility. Most computer systems are designed to allow end users to arbitrarily upgrade the capacity of the memory system by inserting and removing commodity memory modules. To support upgrades by the end user, DRAM controllers have to be flexible and handle different configurations of DRAM devices and modules that the end user could place into the computer system.

The memory module was created to alleviate the cumbersome memory upgrade process. Essentially, memory modules are made of small electronic boards that contain a number of DRAM devices so that multiple DRAM devices can be inserted and removed from the system board conveniently. Memory modules also provide a standard interface so that different manufacturers can conform to produce compatible memory upgrades for different computer systems. Memory modules have been developed progressively over decades to provide flexibility and compatibility between different systems. Modern memory modules also include an extra DRAM device to serve as an ECC check bit.

Different memory modules have different configurations and timing parameters. To provide the memory controller with necessary information, a small flash memory device is integrated onto the memory module. This small flash memory device is known as a Serial Presence Detect (SPD) device. SPD provides the configuration parameters and timing characteristics of the memory module to the memory controller at system initialization. With SPD, the memory controller can obtain the memory module information required to effectively access the DRAM devices on the module.

The 30 pin Single In-line Memory Module (SIMM) was first standardized in the late 1980’s. Then, the advent of 72 pin SIMMs made the 30-pin SIMMs obsolete. SIMMs are single-inline, which mean both sides of the module’s contacts represent the same electrical contacts. In the late 1990’s, 72 pin SIMMs were obsolete due to the arrival of dual in-line memory modules (DIMMs). DIMMs are larger in dimension than SIMMs and provide a 64 or 72 bit wide data bus interface. Unlike a SIMM, a DIMM has electrically different contacts on different sides of the DIMM.

Registered memory modules have been introduced to reduce electrical loads of a memory system with large numbers of DRAM devices. In a large memory system, the electrical loads are segmented through the use of registers that (1) separate the loads of the DRAM devices on the module from the system and (2) buffer the address and control signals at the interface of the memory module. The load segmentation limits the number of electrical loads and shortens the control signal paths of the memory system. However, registered memory modules introduce longer delays to memory access.

The topology and organization of the DRAM memory system are important because memory system topology determines the signal path lengths and electrical loading characteristics of the memory system. However, due to the sensitivity of memory systems to the manufacturing costs, the memory system topology has remained essentially unchanged since the Fast Page Mode DRAM (FPM) era. Synchronous DRAM (SDRAM) and Dual Data Rate SDRAM (DDR) also employ this memory system topology. The later memory systems also adopt the topology with the trend of fewer ranks.

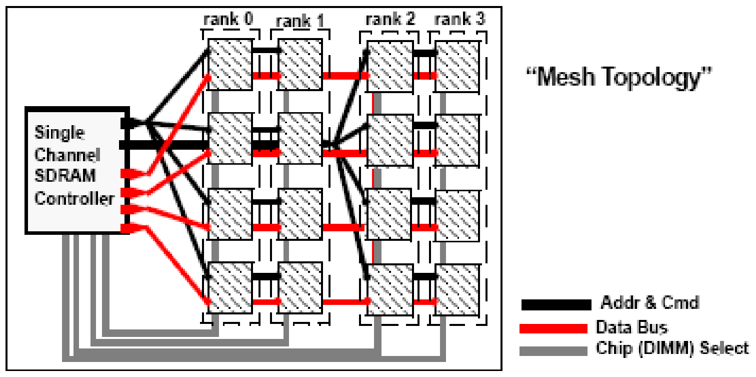


Figure 2.8: The Classic Memory Topology [17].

Figure 2.8 shows an example of the classic memory topology. The figure shows a memory system of 16 DRAM devices are organized into four separate ranks of memory, which are connected to a single DRAM controller. The bi-directional data bus is divided into the size of column in a DRAM device and connected to one device in each rank. The uni-directional address and command bus also connects to every DRAM device in the system. In this topology, a command is sent via the address and command busses to all DRAM devices in the memory system. Specified by the chip-select signal, one a selected rank is activated to process a read command or receive the data for a write command. Therefore, the rest of the DRAM devices in the system ignore the data and command being sent by the memory controller.

2.4.4. DRAM Commands

The DRAM memory access protocol is difficult to analyze in detail and is considered to be a complex task. The large number of combinations of commands in modern memory systems causes the analysis of the DRAM access protocol to be complex. This section provides an introduction to basic DRAM commands and their functions. A more detailed analysis of the DRAM memory access protocol interacting with various DRAM commands can be found in [17].

Figure 2.9 shows the data movement caused by different DRAM commands. The figure is used throughout this section as a generic DRAM device to define the basic memory access commands. The generic DRAM access protocol described by Wang [17] is based on a resource usage model. The resource usage model holds the condition that two different commands can be processed concurrently if they do not require the access to the same resource at the same moment. However, there are other parameters which must be satisfied, such as timing parameters to limit the power dissipation of the DRAM systems. Figure 2.9 illustrates four interleaving operational phases for a DRAM command.

1. the command is transported through the address and command busses and decoded by the DRAM device.
2. data are moved within a bank. The data can be moved in two directions: either from the cells to the sense amplifiers for a read request, or from the sense amplifiers back into the DRAM arrays for a write command.
3. the data is moved through the shared I/O gating, read latches and write drivers.
4. the DRAM device transports the data between the host's memory controller. The data is placed onto the data bus by the DRAM device in case of a read command or

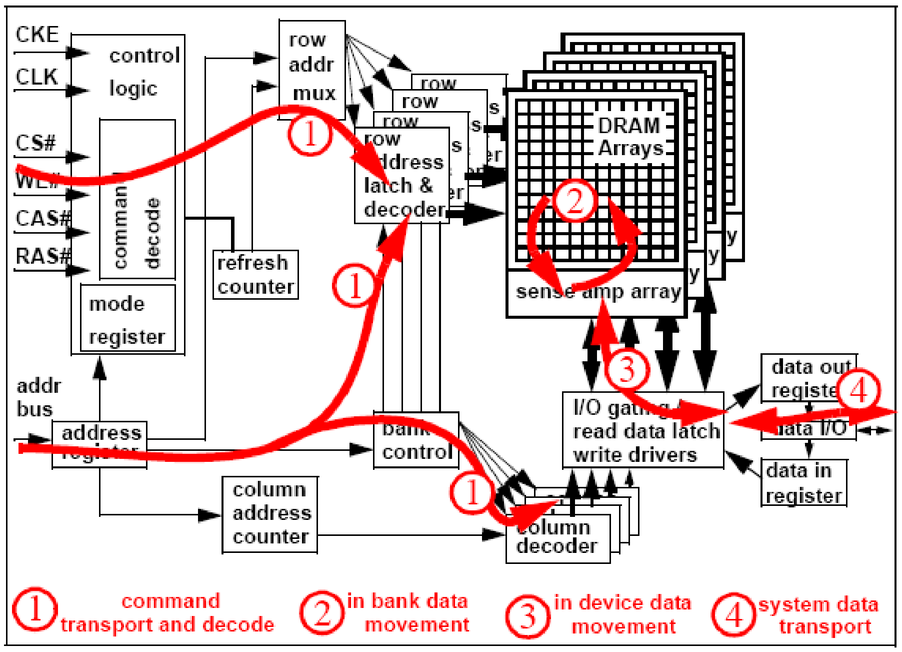


Figure 2.9: Command and data movement on generic SDRAM device. [17].

by the memory controller in case of a write command. Since the data bus may be connected to multiple ranks of memory, multiple commands to different ranks can cause conflicts on the data bus.

To operate the DRAM memory systems, there are five generic DRAM commands to be discussed. Each command associates with a number of timing parameters, which are usually defined in the DRAM device data sheet to describe the specific command behaviors. The five commands include row activation commands, column read commands, column write commands, precharge commands, and refresh commands. A row activation command moves data from the DRAM cells to the sense amplifiers. The data remain in the sense amplifiers for the following column read/write access commands to access multiple columns of data. A precharge command resets the array of sense amplifiers and the bitlines and prepares the sense amplifiers for the next row access command. Finally, a refresh command retains the electrical charges in the memory cells of a particular row.

Additionally, some modern DRAM devices also support commands involving complex actions, for example, a compound column read and precharge command, posted-CAS command in DDR2 SDRAM, and additional complex commands to manage specialized hardware. However, we introduce only generic commands in this section as a background for DRAM operations.

2.4.4.1. Row Activation Command

To access data from the DRAM arrays, the first step is to move the entire row of data to the sense amplifiers. A row activation command moves data from the DRAM arrays to the sense amplifiers. To access another row of data, the charges must be restored from the sense amplifiers. Therefore, the row activation command is associated with two timing parameters: t_{RCD} and t_{RAS}. The Row Column (Command) Delay or t_{RCD} is the time for the row activation command to move data from the DRAM cell arrays to the sense amplifiers. Then, a column read access command or a column write access commands can transport the data from the sense amplifiers to the memory controller via the data bus.

The second timing parameters t_{RAS} deals with the charge restoration process to the DRAM cells. Due to the data movement to the sense amplifier, a row activation command discharges the DRAM cells of the accessed row. As a result, To prepare the sense amplifiers for the subsequent access to a different row, the data charges must be restored from the sense amplifiers back into the DRAM cells. The Row Access Strobe latency or t_{RAS} is defined as the time that a row access command discharges and restores data from the row of DRAM cells. After t_{RAS}, the data restoration process is completed, the sense amplifiers are ready, and the DRAM array can be precharged for another row access to the same bank.

2.4.4.2. Column Read Command

The function of a column read command is to move particular columns of data from the array of sense amplifiers to the memory controller via the data buses. The column read command consists of four different but overlapping phases.

1. The column address and command are transported through the address and command bus, and then decoded by the DRAM device.
2. The specific data columns are accessed at the sense amplifier array of the specific bank and moved to the I/O gating.
3. The data are transported out to the data bus via the I/O gating.

4. The data are transported on the data bus for the time duration of t_{Burst} or the data burst duration.

A column read command is associated with two timing parameters: t_{CAS} and t_{Burst}. The Column Access Strobe Latency (t_{CAS} or t_{CL}) is the time for the DRAM device to place the first chunk of the requested data onto the data bus after the moment that memory controller sends the column read command. t_{CAS} includes step 1 through 3 until the first chunk of the requested data has finished the movement from the sense amplifiers onto the data bus. In modern memory systems, data are sent over the data bus in bursts, usually in terms of 2, 4 or 8 beats on the data bus. One beat in Dual Data Rate systems usually means a half clock cycle. The t_{Burst} or the data burst duration is conventionally defined in terms of a unit of time instead of a unit of clock cycles.

2.4.4.3. Column Write Command

The function of a column write command is to move data from the memory controller to sense amplifiers of a specific bank. The column write command is quite similar to the column read command with different direction in data movement. Therefore, the same set of operating phases are repeated here with reversed sequence.

1. The column address and column write command are transported through the address and command bus.
2. The memory controller places the data onto the data bus.
3. The data are transported from the data bus through the I/O gating,
4. The data arrive at the sense amplifiers of the appropriate bank.

A column write command is associated with only one timing parameter, t_{CWD}. Column write delay or t_{CWD} is the time the memory controller waits before placing the data onto the data bus, after it issues the write command. t_{CWD} is defined differently in different memory systems. In earlier DRAM and SDRAM, memory controllers place both the write data and the command at the same time; as a result, t_{CWD} is equal to zero. In DDR SDRAM, write data is delayed one full clock cycle, and in DDR2, the write delay is one cycle less than t_{CAS}. Another timing parameter to be introduced is the write recovery time. The write recovery time or t_{WR} is defined as the time between the moment the data burst ends and the moment the data complete their movement into the DRAM arrays.

2.4.4.4. Precharge Command

There are two steps in the process of accessing data on a DRAM device. First, data are moved from the DRAM cells to the sense amplifiers by a row access command. Second, a number of column access commands move the data in the sense amplifiers to/from the DRAM devices from/to the DRAM controller. Before the data from a new row can be accessed, a precharge command prepares the DRAM device. The precharge command resets the array of sense amplifiers and the bitlines to the original state. The precharge command also consists of two different phases.

1. The precharge command is transported to the DRAM device,
2. The selected bank is precharged.

The precharge command is associated with one timing parameter t_{RP}. The row cycle time t_{RC} is summation of two row-access related timing parameters, t_{RP} and t_{RAS}. Usually, the row cycle time of a DRAM device is an indicator for the speed of the DRAM device to access data, including,

1. Moving data from the DRAM cell arrays into the sense amplifiers,
2. Restoring the data to the DRAM cells, and
3. Precharging the bitlines to the reference voltage level and making ready for another row access command.

Therefore, the row cycle time restricts the data retrieval speed of the DRAM device when accessed to different rows in the same DRAM bank.

2.4.4.5. Refresh Command

A refresh command is used to periodically retain the data value in the DRAM cells because the electrical charges in the storage capacitor gradually dissipate through the access transistor. Therefore, to retain the data value, the data stored in DRAM memory cells must be periodically read out and written to the full value. This can be done by a refresh command. The time interval between refresh commands must be shorter than the time period in which data in storage cells deteriorate to indistinguishable values. The refresh command also has disadvantages of consuming bank bandwidth and power. Therefore, varieties of different refresh mechanisms are used by different systems to reduce controller complexity and/or bandwidth impact.

A refresh command reads the row address from an internal register, and then the DRAM device sends all banks the row address. After that, each bank refreshes that row concurrently. The refresh command is associated with a timing parameter t_{RFC}. The refresh cycle time or t_{RFC} is at least equal to or longer than the row cycle time t_{RC}. In modern

DRAM memory systems, the memory controller typically issues a refresh command once every 32 or 64 milliseconds for each row in a bank.

The sequence of DRAM commands can vary in different systems depending on the policy of the memory controller. For example, it is more beneficial to remain an active row of data at the sense amplifiers in case of applications with high locality in memory accesses. The reason is the subsequent memory accesses can retrieve data from the same row directly without accessing another row. This can save both latency and energy. On the other hand, applications with low locality of accesses would favor memory systems that immediately precharge the DRAM array and prepare the DRAM bank for another row access. The memory systems in the former case that keep rows active at the sense-amplifiers are called open-page memory systems, and the memory systems in the later case that precharge a bank right after a column access are called close-page memory systems.

2.4.5. Memory Controller

The system controller correctly and efficiently manages the flow of data among the processors, I/O devices, and the memory system. The DRAM memory controller is located inside the system controller. The function of the DRAM memory controller is a subset of the system controller: to manage the flow of data to and from DRAM devices. The interface protocol of a DRAM memory controller is characterized by the DRAM access protocol and timing parameters.

Most DRAM devices are manufactured without any intelligence. The devices only operate when the commands are sent to them. The data sheets are provided by DRAM manufacturers to specify the timing constraints for individual DRAM commands. To maintain the correctness of DRAM operations, the DRAM controller must operate within the timing parameters defined in the datasheet.

This section provides an overview for a number of important issues to the design and implementation of modern DRAM memory controllers. Specifically, following items are particularly important to the design and implementation of a DRAM memory controller:

- Row-buffer Management Policy
- Address Mapping Scheme
- Memory Transaction and DRAM Command Ordering Scheme

2.4.5.1. Row-buffer Management Policy

Row buffer management policies are the policies that manage the operation of sense amplifiers. In modern DRAM devices, arrays of sense amplifiers act as temporary buffers for a previously-accessed row of data. Modern memory controllers typically employ the following two policies to manage the operations of sense amplifiers in DRAM devices: the open-page policy and the close-page policy. Different row-buffer management policies also exist. The row-buffer management policy impacts the selection of the address mapping scheme, the memory command re-ordering mechanism and the transaction re-ordering mechanism for DRAM memory controllers.

The open-page row-buffer management policy is aimed at favoring memory access sequences directed at the same row of memory. The open-page row-buffer management policy keeps sense amplifiers open, meaning the sense amplifiers are not immediately precharged. So, each sense amplifier holds an entire row of data for subsequent access. This policy assumes that different columns of the previously accessed row may be accessed again in the near future. If the subsequent memory read access is to access the same row as the previous memory access, the read access could take the minimal latency of t_{CAS}. The reason is that only a column access command is needed to satisfy the access. However, if the access is directed to a different row of the same bank, the memory controller would perform a series of actions, including precharge the DRAM array, perform another row access, then perform the column access.

Unlike the open-page policy, the close-page row-buffer management policy is aimed at favoring random accesses, which tend to map to different rows of memory. It is adopted in memory systems designed for large-scale multiprocessor systems or some specialty embedded systems. Due to the combination of memory request sequences, the spatial locality of the resulting memory access sequence is greatly reduced. Additionally, with different timings due to the different command combinations, the resulting sequence of DRAM commands in an open-page system is very difficult to schedule efficiently, as compared to the same memory access sequence in a close-page memory system.

2.4.5.2. Address Mapping Scheme

The purpose of an address mapping scheme is to reduce bank conflicts and increase parallelism in the memory system. In a DRAM memory system with an open-page row-buffer management policy, a sequence of consecutive read requests to the same row of data can be performed in pipelined fashion, while a similar sequence of read requests with close-page row-buffer management policy causes longer latency. In a memory system that utilizes the close-page row-buffer management policy, the latency of each access remains relatively the same. With the difference in the access preferences, optimal address mapping schemes are different for open-page and close-page memory systems.

Open-page Baseline Address Mapping Scheme

In a system that utilizes the open-page row-buffer management policy, consecutive cacheline addresses should be placed into different channels, then adjacent cachelines should be mapped into the same row, same bank, and same rank. The baseline address ordering is as follows: row, rank, bank, cachelines per row, channel, and cacheline offset, respectively from most significant bit to least significant bit.

Close-page Baseline Address Mapping Scheme

The key assumption of the close-page row-buffer management policy is that there is little spatial locality in the sequence of memory accesses. In close-page memory systems, mapping in a similar manner as with open-page would result in a bank conflict, which greatly under-utilizes available memory bandwidth. To avoid bank conflicts, adjacent lines are mapped to different channels, then to different banks, then to different ranks. The baseline ordering is as follows: row, cachelines per row, rank, bank, channel and cacheline offset, respectively from most significant bit to least significant bit.

In addition to considering row-management policies and due to flexibility and scalability, memory system organization parameters that can be varied are typically assigned to the highest address range. For example, rank and channel, which can be changed by user upgrades. Therefore, the lower-order address assignment can remain unchanged while the memory modules are altered in the system. However, this mapping scheme allows an application to utilize only a subset of the memory address space and would limit the availability of the memory to fewer ranks. An address mapping scheme with more scalability would have less rank or channel parallelism to memory accesses.

2.4.5.3. Memory Transactions and DRAM Commands

A design engineer must consider the additional complexity of a high-performance DRAM memory controller. The DRAM controller design must take into account the specific DRAM memory system behaviors, application specific requirements, and the type and number of processing elements in the system. Fortunately, some basic strategies have been invented to aid in the design of a high performance DRAM memory controller. To name a few, the strategies are: bank-centric organization, write caching, and seniors first. These are common to many high-performance DRAM controllers. Also, specific adaptive arbitration algorithms are unique in specific DRAM controllers.

Write Caching

Write Caching has been used in many levels of memory hierarchy. The basic idea of write caching is that write requests are typically non-critical, but read requests may be critical in terms of performance. Additionally, DRAM devices perform poorly in cases of consecutive read and write requests. Therefore, caching write requests and allowing read requests to proceed ahead are beneficial to performance.

DRAM-Bank-Centric Request Queuing Organization

One approach that can benefit when multiple commands are processed in a DRAM memory controller is multiple queues arranged in a per bank basis. In this approach, DRAM commands that access the same bank are sent to the same queue. The per-bank queuing approach allows a memory controller to efficiently schedule requests to the same bank, either to the same row or different rows of the same bank. Additionally, bank-centric organization with a bank-rotation mechanism can process concurrent requests to different banks, which results in greater utilization of the memory system.

Feedback Directed Scheduling

Typically, the transaction requests do not contain priority information that allows a memory controller to schedule the transactions more effectively. With direct communication between a processor and an integrated DRAM memory controller, the DRAM memory controller can schedule DRAM commands based on the availability of resources and the DRAM command access history. To achieve the high performance, these integrated DRAM memory controllers have to be aware of state and access history of the processor contexts.

Chapter 3: Overview of Disks

The disk drive is a highly complex electro-mechanical system developed over decades of research and experimentation. To name a few, the disk drive system incorporates many disciplines , including physics for magnetic recording and the read/write heads, material science for various materials used in the disk platter and coating, mechanical engineering for the actuator and the slider carrying the recording head, electrical engineering for the spindle motor, its control and the servo mechanism of the actuator, electronics for the read/write channel and the various control electronics and computer science for architecture of the drive controller and its cache, firmware and algorithms that controls the operation of the disk drive.

A magnetic disk has been considered a fundamental component in a computer system since 1965 [39]. Magnetic hard disk drives will continue to be the dominant form of secondary storage for the foreseeable future. These drives primarily serve as long-term, non-volatile storage for files and as a level of the memory hierarchy below main memory. The disk is included in the virtual memory system implemented in many popular operating systems as a slower form of memory during program execution.

The performance gap between the processor and the disk drive is greater than the processor and the DRAM gap. The processor performance has doubled approximately every two years, and DRAM device data rates are increasing at a rate of 100% every three years [17] with each new generation of DRAM devices. Unfortunately, the disk drive access time improves only 10-15% a year [78]. The solution to this ever-growing problem is to investigate the I/O.

A variety of disk optimization techniques including caching, write buffering, prefetching, and parallel I/O have been invented. These techniques were introduced in server-class disk drives over a decade ago. As technology becomes less expensive with time, these techniques are increasingly applied to workstation disk drives as well. In addition, with better technology, the characteristics of the physical disk are also improved, including the RPM (rotational speed in rotations-per-minute), the seek time and the disk drive interface. As the performance gap between the processor and disk-based storage continues to widen, increasingly aggressive optimization of the storage system is needed. This requires a profound understanding of the real potential of the various I/O optimization techniques and how they work together. Therefore, we must study the effects of an entire system.

This chapter presents a high-level discussion of disk drive technology. It also includes an explanation on how a disk drive works. This background will serve as the foundation for better understanding of the other parts of the dissertation. This fundamental understanding will help clarify a number of the design issues and trade-offs that can affect the performance and power consumption of a disk drive and the disk-based storage subsystems discussed later. The content of this chapter derives greatly from the Disk section in [102].

3.1. Classifications of Disk Drives

Disk Drives can be classified by a variety of methods. One way to classify disk drives is by the drive's form factor. Modern disk drives are usually manufactured in one of four form factors, namely 3.5", 2.5", 1.8" and 1". These numbers indicate the width of the sealed disk drive unit. However, form factors alone are not an effective manner to classify a disk drive. The reason is that form factors do not indicate the underlying technologies inside the disk drives. Disk drives with similar form factors can be equipped with very different functionality, performance and reliability.

A more conventional way to classify a disk drive is according to the application platforms, which traditionally are:

- The server class drives to be used in high-end or enterprise systems,
- The desktop class drives to be used in personal computers and low-end workstations
- The mobile class drives to be used in laptop or notebook computers.

The disk drive's characteristics and the requirements of each class are specified by the application environment in which the disk drive is being used. Server drives require high reliability and performance. Desktop drives require low cost due to the highly price-competitive personal computer market. Mobile drives require low power consumption. Today, the boundaries for these classifications have become unclear. The reason is that some features in a class start to drift to other classes. For example, reliability is required for all disk drives and there has been an ever-growing trend that some higher-end systems are beginning to use desktop drives in some applications to take advantage of their low cost.

Another disk drive classification is the type of interface the drive provides. Current interfaces in modern disk drives are: Fiber Optic Channel (FC), parallel SCSI (Small Computer System Interface), parallel ATA (Advanced Technology Attachment) and the emerging serial ATA (SATA) and serial attached SCSI (SAS). Server class drives are commonly available in either FC or SCSI interface. Desktop, mobile and consumer electronics drives invariably come with an ATA interface. Since server class drives which use a SCSI interface are more than twice as expensive as desktop drives, people often mistakenly think that SCSI interface is expensive. However, the cost of server class drives is mostly due to expensive technologies that are implemented in server class drives to give high reliability and performance, not the SCSI interface. Nevertheless, some high-end storage systems are starting to use ATA desktop drives in certain applications to achieve a lower system cost.

3.2. Areal Density Growth Trend

Areal density is measured in terms of the number of bits that can be recorded per square inch. It is one of the most important parameters a disk drive. This parameter determines the amount of data that can be stored on each platter for a given disk diameter. Areal density specifies the total storage capacity of a disk drive given the number of platters it contains. Even though there are many other contributing factors, ultimately this is the one single most important parameter that governs the cost per megabyte of a disk drive. The rapid growth rate of areal density over the past thirty years has driven the storage cost of disk drives down to the level that makes it the technology of choice for online data storage. Recently, areal density has reached the point where it has become economically feasible to miniaturize disk drives. This technology improvement opens the consumer electronics opportunity for such small disk drives. Areal density also has a profound influence on performance.

Areal density consists of two components, namely tpi and bpi. The recording density in the radial direction of a disk is measured in terms of the number of tracks per inch, or tpi. The recording density along a track is measured in terms of bits per inch, or bpi. However, there are many factors that cause the disk drive to be unable to utilize the maximum areal density it can provide. For a rotating storage device spinning at a constant angular speed, the highest bpi is at the innermost diameter, or ID, of the recording area, while the outer track may utilize less bpi due to the limitation in data organization.

Some of the technology improvements that have enabled areal density growth include:

- thinner magnetic coating - improved magnetic properties
- better head design
- fabrication for smaller heads/sliders
- flying height - spacing between head and magnetic material, resulting in higher linear bit density or bpi
- accuracy of head positioning servo, enabling narrower track pitch or tpi

Hsu and Smith [78] reported that, for IBM 3.5-inch server class disks, the linear density has been increasing by approximately 21% per year, while the track density has been going up by around 24% per year since 1988. In the last few years, areal density has increased especially sharply, so that with a least-squares estimate (no weighting), the compound growth rate is as high as 62%. With only the areal density increasing, the average disk response and service times are improving by about 9% per year. This performance improvement is due to areal density increasing. The data is packed more closely together and can be accessed with a smaller physical movement.

On the other hand, Hitachi GST's areal density growth rate is reported to be 60% per year since 1991, and the rate has further accelerated to an incredible 100% per year since 1997 [98]. This acceleration is the result of the introduction of MR read heads in 1991, GMR read heads in 1997, and AFC media in 2001. Since 1997, track densities have been increasing faster than linear densities, which is the principal factor for the continuing increase in areal density. The track density growth rate is reported at 50% per year while the linear density compound growth rate is around 30%. However, to achieve the areal density in the range of terabits per square inch in 2010, both tpi and bpi growth rate have to be scaled to 25% and 14%, respectively, and a compound growth rate of 46%.

3.3. Performance Metrics

The two widely used measurements of disk drive performance include response time and throughput. The response time is defined as the amount of time starting from the time a request is sent to the disk drive system until the moment the disk drive completes the data transfer. Requests to a disk drive system are usually referred to as I/Os, an abbreviation for input/output. Response time measures the speed of a drive to service a single request. On the other hand, throughput measures the disk drive's ability to serve an amount of data or a number of requests in a unit of time. Throughput is usually defined in terms of one of two units: number of I/Os per second (IOPS) or an amount of data transferred per second (MB/s). Both units are equivalent to each other as they can be converted to the other through the unit of each I/O request. Response time and throughput are closely related and are usually closely correlated. As a result, a drive with fast response time will generally also have high throughput.

For a given disk drive system, the characteristics of a workload can directly affect the performance of the disk drive system in terms of both response time and throughput. The characteristics of a workload that can influence performance include:

- Block size – a large block size takes longer time to transfer than a small block size.
- Access pattern – how much sequentiality or randomness does the sequence of

accesses have?

- Footprint - the size of the accessed disk area can affect the performance. Accessing only a small area of the disk translates into smaller seek distances between I/Os.
- Command type - performance of a read request and a write request can be different due to the disk enhancements equipped in the system.
- Command queue depth - the size of the queue can affect the performance. With a deeper queue, the request scheduler has more options to improve the disk performance by intelligent scheduling.
- Command arrival rate - since both read requests and write requests are sent to the disk system in the form of bursts, the longer the bursts, the longer the request wait time.

One fundamental element that determines both response time and throughput of a disk drive is the I/O completion time, which is defined as the time a disk drive requires to process and complete a user request. The I/O completion time consists of four major components: command overhead, seek time, rotational latency, and data transfer time,

3.3.1. Command overhead

Command overhead is defined as the time the disk drive's controller and electronics take to process an I/O request before the request is sent to the disk's mechanical parts. The command overhead at the disk drive's controller includes the time the controller interprets the command and allocates the necessary resources to service the request. This controller overhead is the major portion in command overhead. Another portion of the command overhead is spent at the end of the I/O request including sending a completion signal to the host and cleaning up unused resources. With better technology, command overhead has been steadily decreasing over the years. The main reason is that the hard drive's microcontroller and function-specific hardware have become faster.

3.3.2. Seek time

Seek time is defined as the duration of the disk drive to move the read-write head from its current track to the destination track to service the next request. Seek time consists of two components:

1. A travel time for the actuator to move from its current track to the destination track
2. A settle time for centering the decelerating head over the destination track and maintaining the center position of the track until data access process starts.

Since seek time deals with mechanical parts in the disk drive, it is one of the largest components of an I/O access. As a result, the disk research community usually considers seek time to be very significant. However, we will show later that, with single-user environment, seek time is unimportant in our experiments due to the application's behavior.Despite the fact that today's disk drives implement zone bit recording (ZBR), which makes the data organization different on the disk from zone to zone, treating ZBR disks as non-ZBR disks is in general an acceptably close approximation for seek time studies. The historical rate of increase in seek time is 8% per year [78].

3.3.3. Rotational latency

After seeking and settling the head, the disk rotates the platters to position the head at the destination sector. Rotational latency is defined as the time the disk rotates to position the head to the start of the destination sector. Disk manufacturers report the average rotational latency for a disk as half the amount of time of one revolution. A more conventional parameter to specify the disk rotational latency is by specifying rotational speed. The rotational speed is usually reported in terms of RPM or revolutions per minute. Therefore, the rotational speed is simply an inverse proportion to the rotation latency. The historical rate of rotational speed increase is 9% per year [78]. Like seek time, the rotational latency is one of the largest components of I/O because it is dealing with moving mechanical parts in the disk, which are relatively slow as compared with electronic components.

3.3.4. Data transfer time

Data transfer time is defined as the time the data is transferred to/from a disk drive from/to the host system. Data transfer time is proportional to transfer size and inversely proportional to data rate. The average transfer size is specified by the operating system and is determined by the application characteristics. The data rate of a disk drive can be generally be categorized into two types: (1) the media data rate and (2) the interface data rate. Media data rate or the internal datarate (IDR) is defined as the rate that data may be transferred in and out of the magnetic recording media. Generally, the media data rate depends mainly on bpi and RPM. The internal or maximum media data rate of Hitachi GST hard drives [98] increased at about 40 per cent per year. Today's server class hard disk drives have the internal data rates beyond 100 MBytes/s. The increase in internal data rate is mostly due to increasing bpi and disk RPM to 10,000-15,000 RPM. However, an increasing internal data rate comes with the cost of increasing power consumption. To mitigate the effects of this, disk drive designers have reduced the disk diameter.

On the other hand, the interface data rate is defined as the rate that data is transferred between the disk drive and the host. The transfer is via an interface. The interface is defined as the communication channel, which I/O requests and the data are sent from/to the host to the disk drive. There are several most common standard interfaces for disk drives today. For example, the parallel and serial versions of ATA in the personal computer realm, and parallel and serial versions of SCSI and the serial Fibre Channel in the server realm. Though ATA disks are usually used in personal computers, ATA disks are also deployed as RAID disk systems in some server applications to lower the overall cost.

3.4. The Physical Layer

3.4.1. Principles of Rotating Storage Devices

All rotating storage devices, with different recording methods and media, are based on common features and principles, including platter, read and write head concepts. Those rotating storage devices have a number of platters which hold recording material on their surfaces. The storage devices also have heads, which are transducers for detecting, extracting, and converting the signal on the recorded media into electrical signals and vice versa. The detailed functionality of the heads differs due to the different types of storage mechanisms and media. A read head is used to detect and retrieve the recorded data. Therefore, only the read head is required for read-only types of storage devices, such as a

DVD-ROM. On the other hand, recordable devices, such as magnetic disks and DVD-RW, require a write head in addition to record the data to the recording media.

The location of a specific block of data on a disk can be specified by radial and the angular coordinates. A disk drive with many platters also adds another dimension to the data coordinates, which is the surface number or the head number. To access data, the head must be positioned at the data location intended to be accessed. The mechanical parts on which the head is mounted move the head to the destination radial position. This process of positioning the head at a particular radial coordinate is called “seek”. The seek process ends when the head reaches the destination radial position and maintains position at the center of the track. Then, the disk is rotated to bring the destination data location under the head. The common approach is to rotate the disk with constant rotational speed rather than spin the disk from stationary state when needed. The reason for this is rotating the disk from a stationary state to a constant rotational speed takes a long time and much energy because the process involves moving mechanical parts. Therefore, rotating the disk with constant speed does not harm the performance as much as having the disk rotate starting from a stationary state. Additionally, since a disk request accesses many sequential bits at a time, rotating the disk at a constant speed will continuously bring many data bits under the head. Separate electronics participate in rotating storage devices; specific electronics control the rotation of the disks and the others control the servo mechanism of the head radial positioning. The servo directly affects the seek time, which is an important factor in performance.

Additionally, rotating storage devices have electronics to perform as the interface between the storage device and the host system. As discussed previously, the storage devices can have multiple platters, which are mounted on the same motor spindle to increase capacity. A disk drive with many platters also adds another dimension to the data coordinates, which is the surface number or the head number. Each surface usually has one dedicated head for it, so the positioning in this dimension is only a process of electronically switching to the right head on the destination surface.

3.4.2. Magnetic Recording

Magnetic recording is founded on materials that can be permanently magnetized, magnetic fields, and the interaction between them. Permanently magnetizable materials are called ferromagnetic materials and they are often used as the storage media for recording since they can provide non-volatility of magnetization. To record data onto the magnetized storage media, external magnetic fields are applied to induce magnetism in ferromagnetic materials in the specified direction. To detect and retrieve the recorded data, the magnetic fields of magnetized ferromagnetic material must be detected.

Ferromagnetic materials can be classified according to their magnetic behavior exhibited in hysteresis loops, into a hard magnetic material and a soft magnetic material. A hard or permanent magnetic material has high magnetic coercivity and high remanence, which is suitable for magnetic recording media. In contrast, a soft magnetic material is a material with low magnetic coercivity and low remanence, which is suitable for magnetic recording head.

The transitions in magnetic orientation between adjacent magnetized grains on the magnetic recording media are used to represent binary data. By convention, the presence of a magnetic field reversal represents a digital 1 and the absence of a field reversal represents a 0. This data representation has been used conventionally since the beginning of magnetic recording.

3.4.2.1. Writing

Writing is the process of recording magnetic transition patterns onto a recording medium. Theoretically, the process of writing requires applying sufficiently strong magnetic field to the designated magnetized grains to induce saturation magnetization onto the media. This process changes the transitions in magnetic orientation between adjacent magnetized grains. In other words, a magnetic recording of digital data storage employs saturated recording. Since the data is either a digital 1 or a 0, only the polarity of magnetization is required to determine the digital value. Therefore, to obtain the strongest signal possible, the recording medium is magnetized to saturation. As a result, saturation magnetization maximizes the signal-to-noise ratio in the magnetic recording media.

The write head and the write channel electronics perform signal conversion that encode the user’s data to the appropriate magnetic fields to be applied to the magnetic grains on the recording media.

3.4.2.2. Reading

Reading is the process of detecting and retrieving the data located on the media by determining the magnetic pattern recorded the in magnetic grains. The different transitions in adjacent magnetic grains represent a 1 or a 0. Therefore, the magnetic pattern represents a set of sequential data on the media. The magnetic pattern is detected by sensing the transition in adjacent magnetic grains.

The read head is the transducer that detects magnetic field transitions. The read head converts the magnetic field transitions to electrical signals that can be processed and interpreted by the drive's electronics. The details regarding the mechanism employed by the read head detection and conversion process is varied with the type of read head.

3.4.3. Mechanical and Magnetic Components

3.4.3.1. Disks

The recording medium for hard disk drives is basically a very thin layer of magnetically hard material on a rigid circular substrate. Some of the required characteristics of recording media include:

- A thin substrate to use less space
- A lightweight substrate use less power while spinning
- High rigidity for low mechanical resonance and distortion under high rotational speeds
- Flat and smooth surface to allow the head fly very low without making contact
- High coercivity (H_c) for the stable magnetic recording
- High remanence (M_r) for good signal-to-noise ratio
- A square hysteresis loop for sharp transitions between compartments

Magnetic material is composed of numerous grains of magnetic domains, whose size directly affects the material's magnetic properties and the media transition noise. Meaning that more grains are required at a grain transition boundary to reduce noise. This can be accomplished by reducing the grain size. However, decrease in grains size for higher TPI causes the grains become magnetically unstable.

3.4.3.2. Substrates

Recently, glass has been used as substrate materials in disk drives. Though brittle, glass has become widely used due to reduced disk diameters. Glass can be polished to produce a surface finish with high level of smoothness. This property makes glass more attractive than aluminum, even though the cost of glass is higher. Additionally, glass is more attractive because glass is hard and has better durability against head-contact damage. Therefore, it is a better solution for mobile applications. With higher tensile strength, glass can be manufactured in thinner and lighter forms.

3.4.3.3. Magnetic Layer

Originally, particulate media was used as the magnetic layer. However, thin film media is widely used currently, and the originally-used particulate media have become obsolete. Thin film media have a layer of magnetic metal deposited directly onto the substrate and bound there. Unlike particulate media, thin film media eliminate the need to use polymers to bind the magnetic layer to the substrate. Therefore, magnetic material in thin film media is not diluted by the nonmagnetic binder, i.e. polymers. This allows for a thinner layer of magnetic material and results in shorter magnetic transitions between adjacent magnetic grains. With narrower transitions, thin film media can provide higher areal density. Thin film media can be produced by a sputtering method. In a sputtering method, a low pressure gas, such as argon, is accelerated with a high voltage towards the surface of the target magnetic material. Then, the surface atoms of the target magnetic material are displaced by the energized argon ions accelerating toward them. As a result, the magnetic material atoms are ejected to bond with the substrate, resulting in the thin film of magnetic material on the substrate.

3.4.3.4. Disk Structure

Besides the substrate and magnetic layer, a magnetic disk is composed of multiple layers of a variety of materials. Starting from the substrate, there are layers of nickel-phosphorus, chromium, magnetic media, wear-resistant overcoat, and lubricant. The first layer above the substrate is nickel-phosphorus. Nickel-phosphorus provides a much harder surface protecting the disk from damage and can be polished to a very fine surface finish. Above that is a layer of chromium. Chromium provides a basic microstructure foundation for the magnetic layer material to be deposited on. Next is the magnetic layer, for which a cobalt alloy is normally used. A layer of hard-but-not-brittle overcoat is atop the magnetic layer to protect it from wear and tear and other damages. Hard carbon is usually used as the overcoat since it satisfies the requirements of being a chemically inert material while still having the ability to bond well with the magnetic layer. Hard carbon is sputtered onto the disk to produce the overcoat layer protecting the magnetic layer of the disk. Finally, a layer of lubricant is placed on the top to prevent possible damages from the contact between the head and the disk.

3.4.3.5. Spindle Motor

Modern disk drives use compact and efficient DC motors, such as the three-phase, eight-pole motor. The motor drives the spindle directly, whose stator is fixed to the bottom of the disk drive case. A part of the outer sleeve of the motor, the rotor, establishes the spindle. Disk platters are mounted on to the spindle. Speed of the spindle is electronically controlled by a servo system.

3.4.3.6. Bearings

To maintain smooth and quiet disk drive operations, disk drives are equipped with bearings to separate the rotating parts from the stationary parts. The function of bearings is to support and separate the spindle hub from the stator shaft. Originally, disk drives were equipped with metal ball bearings. Recently, fluid dynamic bearings (FDBs) have increasingly been used as a replacement for ball bearings. FDBs produce quiet disk drives by replacing the ball bearings with a thin layer of lubrication oil. The oil is high in viscosity, and it resides in a specifically-manufactured container. FBD contains no ball bearings to cause contact since it utilizes the liquid movement of a lubricant film. As a result, the FDB spindle motors can produce a quieter and smoother solution, compared to ball bearings, due to softer impacts between the parts, such as part contact, wobble, and shock. Recently, FDB motor costs have been decreasing due to mass production and improvement in manufacturing techniques. It will eventually cost less than a ball bearing motor due to relative scarcity of parts required.

3.4.3.7. Heads

Heads are the most important element of a disk drive. There are two types of heads in disk drives, which are read and write. The write head generates a magnetic field to change the magnetization direction on the magnetic media. On the other hand, the read head detects the magnetic recording pattern and retrieves the data from the media.

Write Heads

A basic inductive write head consists of a ring core and a coil of wire wrapped around the core. Ferrite is normally used as a magnetically soft material for a ring core. There is a short gap in the core to expose magnetic flux to the media. The head moves very closely above the magnetic recording media and the core gap is positioned just next to the media. Similar to electromagnets, which work by applying current through the coil, the current induces a magnetic field inside the core, whose direction depends on the direction of the applied current. Therefore, the head is called the inductive head. At the core gap, the two ends forming the gap establish two opposite magnetic poles. The magnetic flux moves outside the core from this gap. The magnetic flux from the gap magnetizes the media, which is a magnetically hard material. The result of magnetization is according to the material hysteresis loop characteristics or the media and the amount of magnetic flux applied from the core gap. The distance between the write head and the media has to be shorter than the write bubble, which is defined as the space which the magnetic field of the write head is strong enough for writing. After the write process, the resulting magnetization remains in the magnetically hard media, due to its non-volatile nature.

Digital magnetic recording uses saturation magnetization to record data. To change the orientation of the magnetic field in the media, controlled current is applied via the head. The direction and magnitude of the magnetic flux leaking to the media is the result of different directions and the magnitude of the applied current. Regardless of the previous orientation, the magnitude of the exposed magnetic flux has to be sufficient to completely change the orientation of the magnetic field of the grains next to the head to the desired orientation. For instance, to change the orientation of the media grain to the opposite, the current is applied to the head in the opposite direction of the grain's magnetic field orientation, and the magnitude of the magnetic flux from the core is sufficient to reverse the direction. As a result, saturated magnetization in the opposite orientation is established in the magnetic media at the designated grains. The process creates a transition in the media where magnetization changes to the opposite orientation. To create immediate transition in the process, the media must have a sharp hysteresis loop. This sharp transition in the media translates into closer grains and higher in linear recording density (bpi).

Another important factor to increase the areal density is the size of the head. With better technology, a head dimension can be precisely manufactured. Lithography is used to define the features of the head, and thin film process is used to construct all its components, including the core, the gap and the copper windings. While the dimension of the head is much more compact, the functionality of the head remains exactly the same as an inductive head.

Read Heads

The read head detects and retrieves the magnetic transitions that are recorded in the media in order to read the data located in magnetic recording. Unlike the inductive read head, magnetoresistive (MR) heads sense the flux directly from the media, not the changes in the flux. As a result, the MR sensor generates high peak differential voltage signals during the transition phase. The MR sensors are shielded at the front and back to prevent the MR sensor from detecting unwanted magnetic fields from adjacent transitions. Therefore, it detects the magnetic field from the transition right beneath it.

The the main reason for the rapid increase in areal density after the MR head was introduced in 1991 is the MR head can generate a signal many times larger than an inductive read head can do. An MR head has low inductance; therefore, MR head is applicable to high frequency systems. It is also independent from the rotational speed of the disk, since it detects magnetic flux rather than the rate of change of flux, which is how an inductive read head operates. The last feature is beneficial to small diameter drives with low RPM.

In the early 1990's, a sensor with giant magnetoresistance (GMR) was invented. GMR is a composite sensor with multiple thin layers of different ferromagnetic and anti-ferromagnetic materials. The GMR manufacturing process is a result of molecular beam epitaxy process.

Read/Write Heads

The head or read/write head is referring to the transducers for both reading and writing because the earlier disk drive models had only one inductive head used for both reading and writing. However, to improve the performance, modern disk drives have separated the read and write heads. The reason is that each type of head can be customized to suit its specific purpose better.

Currently, the head is a composition of an inductive write transducer and the MR (or GMR) read transducer placed on the same arm. The write head is placed behind the read head, but both move in tandem. Since the read head is not required to be located at the exact center of the track, a technique called "write wide, read narrow" is applied to today's disk heads. The technique is based on the narrower MR read sensor but wider write pole tip. Therefore, the write head writes a wider track than the width of the read head. This technique allows tracks to be packed together closer and results in increasing tpi.

3.4.4. Electronics

The physical components of a disk drive that directly relate to the total system performance are the electronics that control its operation. Most of the disk drive electronics are in the form of a set of IC chips on a small circuit board located outside the sealed HDA chassis. Only the arm electronics module is located inside.

3.4.4.1. Controller

The controller is composed of a variety of electronic components, including a processor, ROM, memory controller, host interface, data formatter, ECC & CRC encoder/decoder. It is consider the center intelligence of the disk drive. Its function is either to perform a disk drive task itself, or control other components to accomplish the task. The major functions of the controller include:

• Receive and schedule commands (I/O requests) from the user and report completion of a command back to the user.

• Manage the built-in disk cache.

• Control the HDA operations, including seek and data access.

• Manage policies including error recovery, fault, and power.

• Start up and shut down the disk drive.

3.4.4.2. Memory

The functions of memory in the disk drive include:

1. The controller uses a part of the memory as a scratch pad memory.
2. The interface uses a part of memory for speed-matching between the media data transfer rate and the data rate of the interface of the disk drive.
3. A part of memory is used to perform caching for fast data access.

3.4.4.3. Recording Channel

The recording channel receives control signals from the controller and then generates appropriate process for writing or reading the data from the media. It decides either read or write head and circuitry to activate. For write data, the recording channel applies the appropriate direction and magnitude of the voltage to the write head according to the control signal from the controller. Additionally, the recording channel interprets the data retrieved from the media by the read head.

- Write Channel

The write channel is a set of electronics, which is a part of the recording channel. It translates the user data from their digital format into the required magnitude and direction of currents to be sent to the write head. Then, the write head manipulates the magnetic transition in the media according to these currents.

- Read Channel

The read channel is a set of electronics, which is also another part of the recording channel. It translates the magnetic signals retrieved by the read head to the digital data originally recorded. The function of read channel circuit also includes error recovery by a modulation code decoder to recover the original user data and ECC/CRC information.

3.4.4.4. Motor Controls

Motor Controls have the function to control two disk drive motors: the spindle motor and the actuator voice coil motor (VCM). These motors are different in both the form of motion and the mechanical functions. The spindle motor spins the disk platters with a constant rotational speed. On the other hand, the actuator VCM accurately moves the actuator to position the head at the destination radial coordinate. Both motors can be controlled by changing either the amplitude or direction of the current to the motor.

The actuator servo control system requires special servo patterns, which are written on the disks at manufacturing time. These servo patterns provide accurate positioning information on the radial location of a surface. The actuator servo control system uses close-loop feedback control. The servo patterns are detected by the read channel and then they are forwarded to the decoding circuit. The decoding circuit decodes the signals into positioning information, which is fed to the servo control logic. The servo control logic compares the positioning information with the request's destination track position, sent by the disk drive's controller. The difference between the positioning information from the servo patterns and the request's track position is calculated and is used to generate a corrective action signal fed back to the VCM driver to adjust the position of the head. In today's disk drives, the actuator servo control includes a digital signal processor (DSP), an analog to digital converter (ADC), and a digital to analog converter (DAC). A DSP is used to perform the main functions of the control. An ADC is used for converting the analog servo signals to digital form, while a DAC is used for converting the digital correction signal into analog input for the power amplifier of the VCM driver.

For the spindle motor, the EMF voltage from the motor coil, generated by the spinning motor, is monitored to measure the motor's rotational speed in today's disk drives. Then, a close-loop control adjusts the rotational speed accordingly to keep it spinning at a constant rate.

3.5. The Data Layer

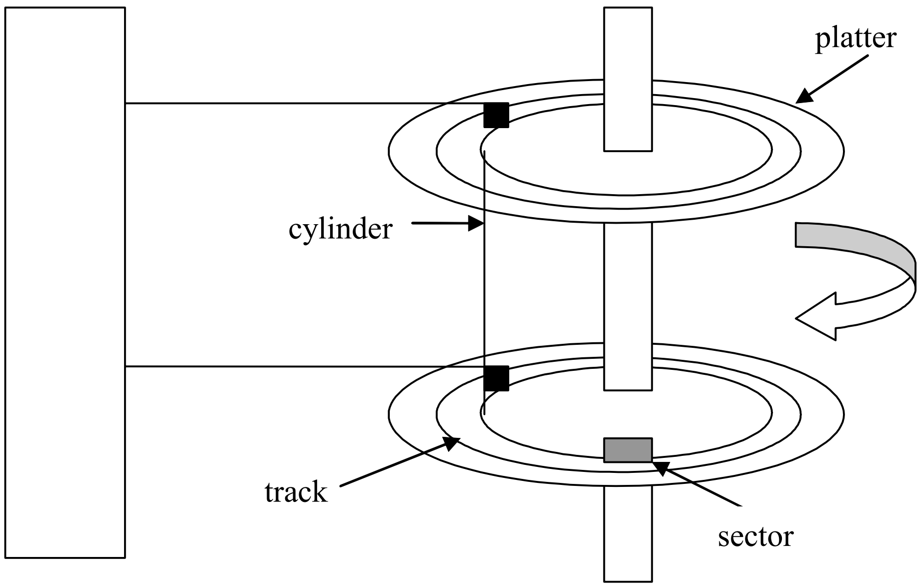


Figure 3.1: Basic Data Organization of a disk drive.

3.5.1. Disk Blocks or Sectors

Today, all disk drives use a fixed-size block formatting. The blocks are called sectors. Most disk drives, if not all, specify a sector size of 512 bytes for data. Each sector is separated from adjacent sectors by physical gaps, which have no recording data. The presence of a physical gap is to provide the read/write head with extra time and buffering while accessing an individual sector. Figure 3.2 illustrates the basic fields in a sector. A sector is composed of a preamble field, data address mark field, data field, ECC, CRC and a flush pad field. The first field of a sector is a preamble field or the sync field. The preamble field is approximately 10 bytes long. It defines the frequency and amplitude used to write the sector. The read channel uses this information to adjust its phase locked loop (PLL) and its automatic gain control (AGC) circuits. Located next to the preamble field is the data sync or data address mark. The data address mark contains a few bytes which is used to separate the preamble and the data. The third field is the data field. Currently, the size of data field is usually set to 512 bytes. However, people in the disk drive community are attempting to push the data field to 4KBytes for compatibility and performance reasons. For data recovery, error correcting codes (ECC) are attached to a sector. Finally, a cyclic redundancy checksum (CRC) is also included as a part of a sector to further ensure data integrity. The size of CRC is product dependent.

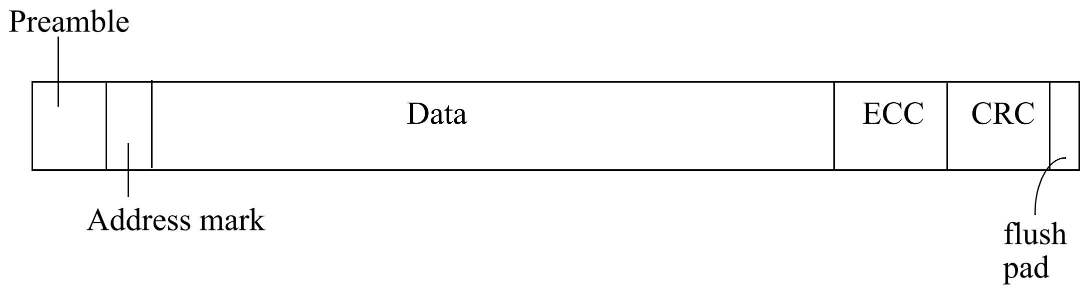


Figure 3.2: Components of a sector.

ECC and CRC are included to increase reliability. The difference between them is that the CRC of a sector is calculated over the sector's data while the ECC is computed over a set of individual sector information that includes the sector's data, the CRC, and the implied logical block address (LBA) of the sector. The hard error rate in a sector with ECC is reduced to 1 in 10^{14} for typical desktop drives and 1 in 10^{15} for typical server class drives. The last field in a sector is padding. The padding contains a few bytes to facilitate the process of flushing data through the read channel and other circuits. The padding field helps maintain the clock while the data is being flushed.

As areal density increases, data becomes more sensitive to errors due to a lower signal-to-noise ratio and the fact that the same size physical damage can affect more bits. More redundancy bits per sector in ECC are required to maintain the data integrity. Therefore, the disk drive industry is under pressure to increase the sector size.

3.5.2. Tracks

A platter of magnetic hard disk drives is composed of a group of concentric circles. This characteristic has been defined as a standard feature for magnetic hard disk drives from the start. Each circle is commonly known as a track. A location on each track is assigned as the beginning of the track. Since a track is a circle the beginning of the track is also the end of the track. A track contains many sectors, which are evenly spaced along each track and numbered accordingly, starting with one. Each track is also numbered with the outermost diameter (OD) being the first track (track 0) and the innermost diameter (ID) being the last track.

3.5.3. Cylinders

A disk drive has multiple surfaces with multiple tracks on each surface. Tracks with the same track number, one on each surface, form a cylinder. The disk drive generally reserves a number of outermost cylinders for internal uses, such as disk physical address mapping information. This information is not accessible by end-users. Those reserved cylinders are sometimes called the negative cylinders since cylinder 0 is the first user data cylinder.

3.5.4. Address Mapping

To be accessed by the host system, an individual sector in a disk drive is uniquely identified by an addressing scheme. The host system informs the disk drive which sector to be accessed by specifying the host sector address. Inside the disk drive, the disk drive maps the host sector address to disk physical address and then the correct destination sector is accessed. Therefore, there are two addressing schemes involved in disk drive address mapping: internal addressing and external addressing.

3.5.5. Internal Addressing

There are two addressing schemes generally used internally in a disk drive to map a physical address to a physical sector on a disk drive. First, a disk drive identifies a physical sector by using a physical block address (PBA) or absolute block address (ABA). The PBA or ABA is ranged between 0 and N-1 where N is the total number of sectors in the disk drive. Another scheme commonly used in disk drives is CHS or cylinder-head-sector addressing. CHS identifies a physical sector with 3 numbers, which include a cylinder number, a head number or surface number, and a sector number on the track.

3.5.6. External Addressing

The host commonly uses an addressing scheme called LBA or Logical Block Address to identify a sector in the host aspect. Unlike CHS consisting of three numbers, LBA simply specifies the sector with only one number. Even though the previously-introduced logical addressing is limited by the number of sector, track, and head, LBA is not. Additionally, LBA allows the ATA disk drive to recognize the address for a capacity over 8.4 GB.

3.5.7. Logical Address to Physical Location Mapping

When a request is sent from the host to the disk drive, the address of the request is converted from logical address in terms of LBA to physical address in terms of PBA. Then, the request PBA is mapped to the physical CHS address to identify the destination head, track, and sector. When the disk head reaches the end of the currently accessed track, it moves to the next track. Two approaches are generally used to decide which track the head should move to. One is to move to the next track on the same platter, and the other is to move to the next track located on different platters but the same cylinder.

3.5.7.1. Cylinder Mode

Cylinder mode moves the head to the next track on different platters but on the same cylinder. The benefit of this approach is it eliminates the need to reposition the actuator causing the head to seek. However, as the tpi rapidly grows, the actuator repositioning process to move to the next track on the same surface may not take as long as the process to move to the next surface. Therefore, with today's ever-increasing track density, it is likely that the actuator can reposition to the next track on the same surface more easily than if it switch to the next surface. Therefore, the other approach, Serpentine Format is more commonly used than Cylinder Mode in modern disk drives.

3.5.7.2. Serpentine Format

As we discussed earlier, today's disk drive have the ability to move the head to the next track on the same surface more easily than to move the head to the next surface. This is due to the rapid increase in tpi in the disk drive. Therefore, the next track on the same surface is physically located closer than the next track on the next surface. Additionally, the exact location of the destination sector is also unknown after switching the head to the next surface. The head needs to read the servo information to locate the current position before the destination sector is located. To prevent the problems, serpentine format is used to move the head to the next track on the same surface.

Compared with cylinder mode, serpentine formatting performs better in random disk access streams. However, cylinder mode outperforms serpentine formatting in the disk access streams with high locality. In those access streams, the average seek time in serpentine formatting is increased. To assure the performance in both types of access streams, a combination approach, called banded serpentine formatting, is introduced. The approach limits the serpentine formatting within a group of tracks, called a band, and the cylinder mode is used in the head movement between the bands.

3.5.7.3. Skewing

If all tracks on the same surface have their beginning position at the same angular position, when the head moves from the end of one track to the beginning of the next track, the beginning of the next track would pass the head already. Therefore, the head has to wait for a full rotation to access the beginning of the track. To prevent the performance loss due to the full rotation wait for the beginning of the track, skewing is introducing. In skewing, the beginning of each track is placed in different locations, depending on the head switching time. Therefore, when the head switches to the next track, the beginning of the next track comes after the head finishes switching.

3.5.8. Zoned Bit Recording

If all tracks on a disk surface have the same number of sectors, the surface doesn’t utilize all the capacity it can provide. The reason is only the innermost track would utilize the highest bpi. On the other hand, the other tracks would contain the same amount of data as the innermost track, even though they occupy more area on the surface. Therefore, most of the disk surface area is under-utilized. To utilize the surface area as much as possible, Zoned Bit Recording (ZBR) is introduced. ZBR divides tracks into groups, called zones. In each zone, the tracks have the same number of sectors. The tracks in the outer zone have more sectors than the tracks in the inner zone. Today's drives tend to have 3 or more zones. Therefore, the original data formatting can be considered as a special case of ZBR where the number of zones is one.

In ZBR, since the number of sectors per track is changed from zone to zone, the disk drive has to deal with variable data rates.

3.5.8.1. Variable Data Rate

Variable Data Rate occurs when the disk applies ZBR, and it is also called CAV (constant angular velocity) recording. With constant disk rotational speed, ZBR varies the number of sectors per zone; therefore, the data rate of the disk varies from zone to zone. The read/write electronics take care of variable data rate. This approach has advantages, including:

1. More data is stored at outer diameter (OD) tracks compared with the amount of data stored at inner diameter (ID) tracks.
2. The performance of the OD tracks is better than the performance on the ID tracks because the data rate on the OD tracks is higher. Therefore, most accesses occur at the OD with higher data rate. As a result, the overall performance of the disk is improved.

3.5.9. Servo

The function of the servo system in a disk drive is to control the movement of the read/write head. It has to maintain accuracy in the head movement in both the movement between the track and along the track. The servo system's function includes:

1. Controlling the movement of the head actuator from the current track to the destination track including the movement on the same surface and the movement to another surface (switching head). These movements are collectively known as seek. One parameter that affects the performance of the seek operation is the seek distance. The seek distance is the distance in terms of number of tracks that the head has to move from the current track to the destination track.

2. Maintaining the correct position of the read/write head along the track at the center of the track. When the head is accessing the data on a track, the servo system continues making corrective adjustments to maintain the position of the head on the track. This is to prevent the head from drifting off the center of the track. This process is called track following.

With ever-increasing tpi in modern disk drives, high accuracy in servo system is required. The servo system in modern disk drives is generally implemented with closed-loop control to accomplish the high accuracy in both of the previously mentioned functions. To provide the accurate position information of the sector on a disk to the read head, the servo information is written onto the disk surface directly. There are two approaches to place the servo information, which are dedicated servo and embedded servo.

3.5.9.1. Dedicated Servo

In the dedicated servo approach, the servo information is written on one of the surfaces in a multiple-platter disk drive. The servo information is usually written onto the middle surface in the disk stack at the manufacturing time. The head on that surface is only a read head and is called a dedicated servo head. Since all the heads in a multiple-platter disk drive are connected to the same actuator and move together, the dedicated servo head plays the master role for all heads. However, since modern disk drives tend to have only a few platters due to power consumption, dedicating one surface for servo information is considered too costly. Additionally, temperature can cause different physical changes in arms and disks on different platters. Therefore, the servo information on the dedicated surface may be displaced with respect to other surfaces. This problem can be solved by periodically calibrate the servo information with respect to other surfaces due to the temperature changes. This calibration process causes significant performance degradation, and the process is more complicated as the tpi increases. As a result, the dedicated servo has been obsolete since the mid 1990's.

3.5.9.2. Embedded Servo

Unlike the dedicated servo, the servo information in embedded servo is written with the data on the surface. Therefore, there is no dedicated surface or dedicated head in the embedded servo approach. The head on each surface performs both read/write data on the surface and reads the servo information on the surface. The servo information in the embedded servo is in the form of wedges on a disk surface. The wedge area is specifically reserved for servo information, written at disk manufacturing time. The wedges are evenly spaced around the disk. The embedded servo also provides two types of servo information, which are referred to as servo bursts and track id. The servo bursts are provided to prevent the head drifting off the center of the track. On the other hand, the track id is the servo information for seek operations. Since there are both a read head and a write head on every surface, care should be taken not to allow the write head to overwrite the servo information on the surface. Some problems should be mentioned in using embedded servo, including:

- The servo wedges should be placed closely to prevent the head drifting off the center of the track.
- The head requires servo information to determine the current track number and the destination track number. When seeking, the head must read the servo information as soon as possible to determine the track number. Therefore, there should be enough servo wedges on the surface, so the head does not have to wait for a long time to read the servo information to determine the track number.

Both problems suggest that there should be as many servo wedges as possible on a surface. However, the area containing user data is reduced when increasing the number of servo wedges. Additionally, the benefit of having many servo wedges is dependent on the disk access pattern; meaning, having many servo wedges improves the performance in case of a random access pattern, but degrades the performance in case of sequential access pattern. Modern disk drives typically have approximately 100 to 200 servos per track, and the servo information takes up space of approximately 8% to 12% of the total disk capacity.

3.5.9.3. Servo ID and Seek

In each servo sector, there are two servo coordinates: the radial coordinate and the angular coordinate. However, to seek, the servo system only requires the radial information to move the head to the destination track. The radial information is recorded in the form of the cylinder number, and it is repeated for all servo sectors on the same cylinder. The cylinder number is implemented as a Gray code, in which any two adjacent cylinder numbers will differ by only one bit. This is to guarantee the cylinder number value to be one of the two valid adjacent values rather than a totally different, unrelated value, which would be harder to distinguish from an erroneous value

3.5.9.4. Servo Burst and Track Following

As discussed earlier, the servo bursts are provided and are used to prevent the head drifting off from the center of the track. The process is conducted in two situations: at the end of seek operation and while the head is moved along the track. At the end of seek operation, the process is called settle and the time to settle is called settling time. On the other hand, the process in the second situation is called track following. Both settling and track following require servo bursts to maintain the head position at the center of the track.

There are four special magnetic patterns encircling each track on a surface at a servo wedge. They are called A, B, C, and D bursts. The A burst and the B burst are placed adjacent to a track but on different sides of the track. The A burst generates a signal called VA and B burst generates a signal called VB. If the read/write head is at the center of the track, VA and VB signal amplitude are equal. On the other hand, if the head is off the center of the track, either VA or VB signal amplitude is higher than the other to indicate which direction the head is off the center and how far. Then the servo control can adjust the head position accordingly to maintain the head at the center of the track. However, C and D bursts are required for the case that the head is located between two adjacent track, which causes both the VA and VB signal to go flat. Likewise, the signal generated from the C bursts is called VC and one generated from D bursts is called VD. The C and D bursts are placed immediately after the A and B bursts, but the C and D bursts' alignments are shifted from A and B's by the half of the servo burst width. Therefore, if the head is at the flat part of the servo bursts, the amplitude of VC and VD are used to identify the location of the head. As a result, the servo control can use these signals to adjust the head position accordingly to maintain the head at the center of the track.

3.5.9.5. Components of a Servo

Each servo sector is composed of a preamble field, an address mark field, a servo index field, the track id, and the four servo bursts. Servo sectors are also separated from user data by a gap. The first field in a servo sector is the preamble. Like the preamble in a data sector, the preamble of a servo sector is used to synchronize the read channel's PLL with servo's clock. Next is the servo address mark or the servo sync mark. The address mark informs the head that the servo information is next. The third field, the servo index field, provides the coordinates of the servo. The fourth field, the track id field, provides the track number in the form of a Gray code. Finally, the four servo bursts provide the signals to maintain the head at the center of the track as explained in the previous section.

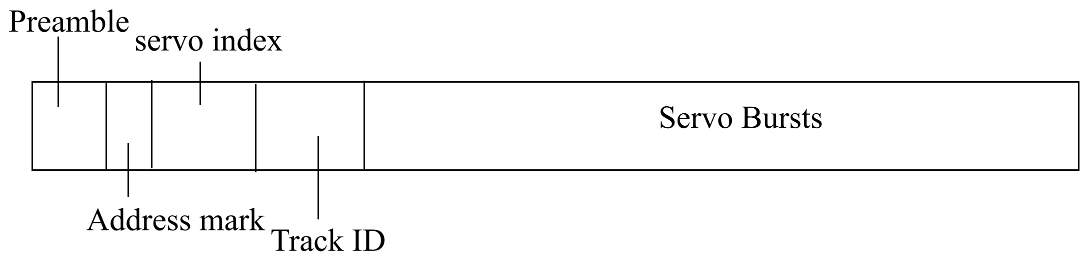


Figure 3.3: Components of a servo sector.

3.5.9.6. ZBR and Embedded Servo

Embedded servo causes more difficulty to the sector placement on a disk with zoned bit recording (ZBR). Without ZBR, the embedded servo wedges can simply be placed between sectors split spaced evenly around the surface. However, ZBR causes the sectors with the same sector number but on different tracks not to align on the same angular position. Therefore, to place the embedded servo wedges on a disk surface, sectors at the wedges are split. Due to splitting data sectors, two additional overheads are introduced for embedded servo, which are extra fields and extra gaps. First, each split sector requires its own preamble field and address mark field. Second, extra gaps are also required both before and after the split part. Those gaps are required for the process of switching heads from write head to read head when the head is entering the servo sector during write (writing the first split part of the sector and then reading the adjacent servo sector). Also, the gaps are needed to switch from read head to write head when the head finishes reading the servo sector and then continues to write the sector. These extra gaps and fields can be significant depending on the number of split data sectors.

3.5.10.Sector ID and No-ID Formatting

The information on an individual data sector was originally stored at a field referred to as a header or sector id. The header was placed just before the corresponding data sector. It contained (1) the sector's physical address (in CHS format), (2) a split flag whether the sector is split by the servo sector, (3) defective flag, and (4) the location whose data is moved to if the sector is defective. However, modern disk drives generally adopt the no-ID format. The no-ID format or headerless format eliminates the header physically placed along with the data. It utilizes the servo information to provide the sector's physical address, and stores other information at a no-ID table. The no-ID table is stored at the protected area of the disk, and it is loaded into the disk controller's memory at startup.

The no-ID format advantages include:

- Each track can contain more user data since the ID fields and their corresponding overheads are eliminated.
- tpi is increased because tracks can be placed closer. The ID-fields require wider tracks.
- Reliability is improved because there are no ID-fields which may get corrupted.
- Performance is improved as the total results because of the increase in tpi and track capacity.

3.5.11.Defect Management

Defects can occur due to multiple causes at the time of manufacturing or during daily usages. Different schemes are used to relocate the data due to defects depending on when the defects occur.

Relocation Schemes utilize logical block address (LBA) and allocate a different sector to that defective LBA. Mapping is done by the disk drive controller. The host does not have the knowledge of the LBA to physical disk address mapping or the defected LBA. As a result, the host believes that all of a disk drive's LBAs are usable for data storage. There are basically two common methods for re-allocating a new sector to replace one that is defective:

3.5.11.1. Relocation Schemes

Sector Slipping

Sector Slipping makes use of the very next good sector; if the sector after a defective sector is not defective, then the LBA of the defective sector is assigned to the following sector. In general, sector slipping does not directly degrade the performance. The reason is there is no interruption to the flow of sequential accessing. Only marginal extra time is required to skip the defective sector and access the immediate subsequent sector sequentially. Therefore, sector slipping is a preferable method for relocating defective sectors in case of minimal data storing on the disk. Otherwise, sector slipping has a disadvantage if data has already been stored on a disk. It must slip all data sectors after the defective sector, which requires the disk to read and re-write all sectors to their new assigned locations.

Sector Sparing

Instead of slipping all data to the next good location, sector sparing allocates a number of spare sectors in the disk drive for defective sectors. When a defect occurs, a defective sector can be allocated to a spare sector. For better performance, it is preferable to scatter these spare sectors around the disk drive. Therefore, a defective sector can be relocated to the closest spare sector. However, the performance might be degraded if the spare sector location causes the drive to move the mechanical parts resulting in significant delay.

A number of schemes are also used in modern disk drives. They are mostly based on these two basic schemes.

3.5.11.2. Types of Defects

There are two types of defects as mentioned earlier:

1. the defects that occur at the manufacturing time, called primary defects, and
2. the defects that occur later after leaving the factory, during daily usages, which is called grown defects.

These two types of defects are handled differently.

Primary Defects.

Before a disk drive leaves the factory, every sector of the disk drive has to be scanned for defective sectors. If found, those defects are called primary defects. The process of scanning sectors for defects is accomplished by reading and writing each sector in the disk drive. Usually, since there is no data stored on the disk drive to be slipped, the defective sectors are relocated by the sector slipping method. Finally, a list of defective sectors in the form of ABAs (absolute block addresses) are generated and stored in the P-List or Primary List. These ABAs in P-List are skipped during the disk drive maps LBA to ABA.

Grown Defects.

In contrast, grown defects or non-recoverable errors are defined as the defects that develop after a disk drive has left the factory. Usually, the defective sectors are discovered during the disk drive’s daily usages. The grown defects, unlike primary defects, are handled by the sector sparing method because user data has already been stored in the disk drive. A list of grown defects is called the G-List. Each entry in the G-List is a tuple of the defective sector address and the corresponding relocated sector address.

3.6. File System Caching

In the operating system point of view, the most important factor in I/O performance is not the speed of the disk, or how efficiently it is used, but whether it is used. To hide the I/O latency, caching, which is widely used in many levels of memory hierarchy to hide the latency of the lower memory level, is also applied in UNIX-based operating systems. This type of caching under the control of the operating system is called file system caching. Originally, the technique used to be called file caching and disk caching, depending on whether the logical disk address or physical disk address is used. File caching uses logical address, while disk caching uses physical address. However, researchers simply use the term file system cache in general for such techniques, and usually use the term disk cache to refer to a memory physically built into a disk drive.

In the operating system, file system caching uses main memory as a cache for disk data to improve I/O performance. Since main memory is much faster than disks, file caches significantly improve performance. Therefore, a file system cache is generally implemented in every UNIX-based operating system. However, different systems have very different I/O policies, and the performance of some I/O policies can differ by many orders of magnitude. These policies can be even more important than the underlying hardware.

I/O performance is limited to the interaction between the disk and the operating system. The hardware may determine the potential performance of the I/O, but the operating system determines how much potential is delivered. In particular, the file cache is critical to I/O performance for UNIX systems. File caching policy is important since file caches in high-end computers with better hardware can perform comparably to file caches in workstations with better file caching policies. Even though optimizations in memory systems can improve disk read performance, the operating system policy on writes can improve the file cache performance by many orders of magnitude.

Like processor caches, write back and write through are applied to file system caches on writes as well. The operating systems community uses the term asynchronous writes to refer to writes that allow the processor to continue after transferring data to a write buffer. This approach improves the performance if writes occur infrequently. Like writer-buffering in other levels of the memory, if writes are too frequent, then the processor may eventually stall until the write buffer is flushed. This situation limits the system speed to the speed of the I/O, which is the slowest level of the memory hierarchy. Note that a write buffer does not directly reduce the number of writes to the next level. Writes can be merged or overwritten to reduce the number of writes when they are waiting in the queue. However, a write buffer only allows the processor to continue while I/O is in progress if the write buffer is not full.

The effectiveness of caches for writes also depends on the policy of flushing dirty data to the disk, i.e. how often to flush and which data to be flushed. To protect against losing information in case of failures, applications will occasionally flush dirty data out of the cache in the main memory to the disk. Most UNIX operating systems have a policy of periodically writing dirty data to disk. By default, a safety window is typically set for all applications to 30 seconds. We will see this behavior first hand in our experiments.

CHAPTER 4: RELATED WORK

The contribution made in this dissertation is in two parts. First, we created a complete-system simulator, SYSim, to demonstrate the detailed interaction of a memory hierarchy in both the performance and power domains. Secondly, to study the I/O behavior during the I/O intensive phase of applications, we explored several disk enhancements and physical disk technology improvements in both isolation and combination. We studied the systems in terms of total system performance and the power/energy consumption. Therefore, this chapter consists of two parts: one is for Complete-System Simulations, and the other is the Disk Enhancements and Physical Technology Improvements.

4.1. Complete-System Simulations

Simulation is the most widely accepted approach in computer architecture, as information is obtained in simulations that is impossible to obtain in a real system. Unlike an experiment in a real system, the system in a simulation is not perturbed during an experiment by an attempt to measure statistical information.

Simulation can occur at various levels of abstraction. Cain et al. [3] compares various approaches of simulations as shown in Table 4.1. The Analytical models and CPI equations are fast, but they lack detail, which costs them precision. The precision of Trace-driven Simulation mainly depends on how the traces are collected. The software trace collection schemes pollute the traces due to software overhead. The hardware schemes require expensive investment in proprietary hardware and are probably not feasible for multi gigahertz processors. Trace collection also records only committed instructions, which do not reflect the inaccuracies created by speculative instructions. Cain et al. conclude that the complete-system (or full-system in their words) execution-driven simulation is the most precise and accurate.

Modeling Techniques	**Inputs**	**Benefits**	**Drawbacks**
Analytical models	Cache miss rates; I/O rates	Flexible, fast, convenient, provide intuition	Cannot model concurrency; lack of precision
CPI Equations	Core CPI, cache miss rates	Simple, intuitive, reasonably accurate	Cannot model concurrency; lack of precision
Trace-driven Simulation	Hardware traces; software traces	Detailed, precise	Trace collection challenges; lack of speculative effects; implementation complexity
Execution-driven Simulation	Programs, input sets	Detailed, precise, speculative paths	Implementation complexity; simulation time overhead; correctness requirement; lack of OS and system effects
Full-system, execution-driven simulation	Operating system, programs, input sets, disk images	Detailed, precise, accurate	Implementation complexity, simulation time overhead, correctness requirement

Table 4.1: Attributes of various performance modeling techniques [3].

Complete-system simulation means a system simulation that includes I/O, especially disk and OS effects in the simulation. Gurumurthi et al. [1] pointed out that the features of a low-power disk can also influence operating system routines such as the idle process running on the processor core. Hence, a model which includes the disk helps to characterize the processor power more accurately. During I/O operations, energy is consumed by the disk. Further, as the process requesting the I/O is blocked, the operating system schedules the idle process to execute. Therefore, energy is also consumed in both the processor and the memory subsystem. The cycles due to disk activity are accounted for as idle-cycles in the execution profile, which can take up to 10% of the execution cycles and up to 7% of the power in the processor and memory. Cain et al. [3] demonstrate that correct simulation of I/O behavior can significantly affect simulator accuracy. They demonstrate that the majority (50.9%) of the executed instructions in some benchmarks are the operating system's. The operating system causes the IPC to be different from the user-level simulation as much as 20%, which translates into over 100% difference in energy consumption. The authors in both [2] and [3] agree that omitting operating system activities can introduce errors that can exceed 100%. Chen et al. [2] reason that:

1. The operating system code has distinctive behavior, different from user code.
2. The state of the micro-architecture after the OS call is different from the state of the microarchitecture before the call was made.
3. The timing of activities scheduled in an OS call may affect the behavior of the microarchitecture.

Therefore, they emphasize that all architecture researchers should seriously consider adoption of full-system simulation, despite the up-front cost of doing so.

Many simulators that can estimate the power/energy numbers are in the area of embedded systems or specific environments [22, 23, 24, 25, 26, 27, 29, 30, 31], as the power-consumption is obviously more important to such systems. Most of them have an application to mobile systems for which the energy source is very limited or less likely to be replaced. However, none of them includes a disk model since a disk is not a standard component for such systems.

The development of complete system simulators in general-purpose systems has been directly motivated by the inability of user-level simulators to target complex workloads, i.e. database or network workloads. Their benefits are diverse and significant; including evaluation of hardware design, development of operating systems, and performance tuning of workloads. There are also many complete-system simulators, e.g. SimOS [4], Simics [5], g88 [6], gsim [7], Talisman [8], Pharmsim [3], and TFsim [9]. g88, gsim, and Talisman are in-order functional simulators for the 88000 processor, which can simulate a modified version of UNIX. SimOS is a dynamic full-system simulator that supports out-of-order processor models for the MIPS and Alpha instruction sets. Pharmsim is a dynamic full-system simulator based on SimOS and SimpleMP with an out-of-order processor model for the PowerPC instruction set. Simics is a commercial simulator that supports system-level simulation of five target architectures: Alpha, IA-32 (x86), PowerPC, SPARC, and x86-64. Simics can boot unmodified operating systems and it can be extended for cache timing simulations, but it only models simple (scalar, in-order) instruction execution. TFsim is the timing-first simulation which its timing simulator executes each dynamic instruction ahead of the functional simulator. It uses Simics as its functional component. However, no power consumption is reported by these simulators.

A handful of publications are focused on architecture-level power estimation tools for processors, i.e. Wattch [19], SimplePower [20], Architecture-level power estimation in [32], and Architectural Power Evaluation [21]. These simulators focus only on CPU power consumption; neither memory nor disk is included in their power estimation systems.

Additionally, they are user-level simulators which do not include the operating system effects.

Only a small number of publications actually implement complete-system simulators with power estimation. To our knowledge, the execution-driven, power estimation, complete-system simulators, which can run operating system on top and include the concept of disk, are SimWattch [2], SoftWatt [1], and Mambo [10].

SimWattch is a complete-system simulator that estimates performance and power consumption of an out-of-order issue superscalar microprocessors. It is based on Simics for performance simulation and on Wattch for power estimation. SimWattch takes the advantage of fast simulation in Simics by letting Simics fill an instruction trace at full speed into a FIFO queue and letting the instructions be consumed by Wattch at a slower pace. Wattch is employed as an architectural simulator that estimates CPU power consumption. One of the important steps is to convert the Simics instructions (SPARC-V9 instructions) to corresponding Wattch instructions (SimpleScalar's PISA instructions). Even though SimWattch can run an unmodified OS and it includes multiple I/O devices, the power consumption reported includes only the processor.

SoftWatt is based on SimOS. SimOS has three CPU models, namely, Embra, Mipsy, and MXS. Embra employs dynamic binary translation and provides a rough characterization of the workload. Mipsy provides emulation of a MIPS R4000-like architecture. It consists of a simple pipeline with blocking caches. MXS emulates a MIPS R10000-like superscalar architecture. SoftWatt use MXS to obtained detailed information about only the processor and disk, and use Mipsy simulator to run the operating system and obtain cache and memory system profiles. After the simulation, the simulation log files are fed into the analytical power models to generate power values. Therefore, SoftWatt does not reflect the actual interaction between memory and disk since the memory system statistics and the disk statistics are the product derived from disjoint simulations. Also, there is a per-cycle power information loss due to post-processing approach.

IBM's Mambo is a complete-system simulator modeling PowerPC based systems. It provides building blocks for creating simulators that range from purely functional to timing-accurate. Functional versions support fast emulation of individual PowerPC instructions and the devices necessary for executing operating systems. Timing-accurate versions include the ability to account for device timing delays, and support the modeling of the PowerPC processor microarchitecture. While it has been used widely in IBM, it is not an open-source software available to the public.

All of these simulators dedicate more details to the processor side than the memory side. For simplicity, some implement the memory as a constant time and constant energy per access. Some implement the memory as banks, but are not very detailed. Some, using two disjoint processor simulators produce disjoint levels of memory accesses, may cause discrepancy in the memory. SYSim is proposed to fill this gap in the memory system research community. It is intended to be an open-source, complete-system simulator that can demonstrate the systemic behaviors of entire memory hierarchy. It also includes both performance and energy models. We hope that SYSim will be a better option for the research community.

4.2. Magnetic Disk Drive Enhancements and Physical Improvements

The performance of magnetic disk storage systems has improved only 10-15% per year, while the performance of processors has roughly doubled every two years. Over time, the performance gap between those two components continues to grow. To narrow the performance gap, more aggressive optimization of the storage system is required. The main reason for the slow growth in disk drive performance is the slow mechanical parts of storage devices. Since these slow mechanical parts can degrade the total system performance significantly, the importance of I/O optimization techniques has been widely acknowledged. As a result, many disk drive enhancements have been invented, including caching, write buffering, prefetching, request scheduling, and parallel I/O.

Many efforts have also been made to improve the underlying technology of the disk drive's physical characteristics. These physical characteristics, continuously improved over the years, include the tracks or bits per inch, average seek time, and rotational speed. These improvements are usually defined by using physical metrics, which is difficult to relate directly to the total system performance of real workloads. Therefore, using physical metrics, it is complicated to compare different physical improvements, since they are not directly related to the total system performance.

Unfortunately, the relative effectiveness of these techniques and technology improvements is ambiguous since they have been investigated in isolation by different researchers using different methodologies. Some use discrete event simulation and analytical modeling. The others use trace-driven simulations. In some cases, the simulations are based on traces of real workloads and in others, randomly generated synthetic workloads. For example, Zhu and Hu [75] evaluate a built-in disk cache using both real and synthetic workloads, and report the results in term of response time. Smith [81] evaluates a disk cache mechanism with real traces collected in real IBM mainframes on a disk cache simulator. He reports the results in terms of miss rate ratio. Huh and Chang [77] evaluate their RAID controller cache organization with a synthetic trace, and Varma and Jacobson [94] and Solworth and Orji [95] evaluate destage algorithms and write cache, respectively, with synthetic workloads. SPEC2000 is also used, for instance, to evaluate a energy-aware, compiler-controlled disk prefetching in [91]. The most widely-used benchmarks in disk drive research would be hplajw, cello, snake [99], and TPC [100]. These are workloads which are used to evaluate many aspects of large-scale disk systems. Since many of the techniques have not been evaluated with real workloads on the same basis, their actual effect is not known. To effectively optimize the system performance, the actual advantage of the disk enhancements and physical improvements must be analyzed both in isolation and in combination especially in system-level points of view.

In this dissertation, we investigate how different techniques affect total system-level performance and power/energy consumption by using a complete-system simulation. The most similar work to ours is by Hsu and Smith [78]. They use a variety of workloads, including server and personal computer workloads, to systematically analyze the actual performance impact of various I/O optimization techniques by using trace-driven simulation. However, they evaluated each technique in isolation and neither in terms of total system performance nor in terms of power/energy consumption. Additionally, we focus on the I/O intensive phase on a single processing environment, widely used in personal computer systems. Therefore, we selectively studied the techniques aiming at improving the disk systems for future personal computer workloads. For instance, unlike large-scale database systems with hundreds of RAID disks, we investigate the RAID system with only small number of disks, i.e. 4 and 8 disks. Also, we investigate the disk cache located inside individual disk drives, rather than the cache in RAID controllers or the cache in the file system. The reason is that we would like to make an impartial comparison between the disk systems equipped with selected disk enhancements and a single disk system, which is widely used in personal computers. Therefore, it would be unreasonable to make comparisons against sophisticated and complex techniques, which exist only to improve large-scale server applications.

The selected disk enhancement techniques and the physical technology improvements we studied are described in detail as follows.

4.2.1. Disk Drive Enhancements

4.2.1.1. Disk Caching

Caching is a general technique for improving performance in many levels of computer systems. It temporarily holds data that is likely to be utilized in faster memory. The faster memory is called the cache. In this section, the data refers to disk blocks requested from the storage system, and the faster memory refers to dynamic random access memory (DRAM) built into the disk drive. We refer to this as disk cache, which is different from file system cache and processor cache as described in Chapter 1. Disk cache is a general term introduced in [81] as a buffer used to hold portions of the disk address space contents, which can be placed at many levels along the data path. The disk cache as a small piece of memory specifically built into a disk drive was first referred to as disk buffer in [83]. Today, the term disk buffer and disk cache are used to refer to the same entity.

The disk cache's specific function is to support I/O operations, which occur only through I/O requests via the operating system. Unlike the processor's cache, disk cache does not connect to the processor directly. Therefore, it is not considered as a part of the memory hierarchy directly, but it is added as a special feature in a disk drive for performance optimization. When an I/O request is sent to the disk system, the request either finds the requested data in the disk cache or is sent to the physical disk mechanism. Like in the processor cache, the hit ratio is defined as the ratio of the number of requests satisfied by the cache to the number of total requests, and the miss ratio is the ratio of the number of requests sent to the physical disk mechanism to the number of total requests. The data can be brought to the cache in two ways, which are (1) they are fetched by reference, referred to as caching, or (2) they are anticipated to be referenced in the near future, referred to as prefetching. In "Memory Systems" [102], Jacob et al. notes that disk caching is claimed to be a key factor in a disk drive's performance whose importance is comparable to other basic drive attributes such as seek time, rotational speed, and data rate.

Originally, disk cache was used as a speed matching buffer between the disk drive and the interface. The buffer is useful in two situations, one is because the disk drive and the interface operate at different speeds, and the other is when the host or the interface is not ready to receive the data. DRAM is usually used as this buffer memory. Since the data is already stored in the disk cache due to buffering, the function of disk cache extends to support caching and prefetching as well. When a request is sent to the disk system, the cache checks whether it holds the requested data. The data may remain in the cache since they have been referred to by the previous requests. Instead of sending the request to the physical disk, the request can be satisfied by the cache. Therefore, the need to move the mechanical parts in the disk drive is eliminated and results in performance improvement. Therefore, the performance improvement depends on the amount of data to be reused in the cache. Today's commercially available disk drives are generally equipped with a built-in cache as part of the drive controller electronics. The cache size ranges from 512KB for micro-drives to 16MB for the largest server drives, and is still growing. With ever-growing disk cache size, only a small fraction of the disk cache is used for speed matching buffer, while most of the disk cache is occupied by the caching data.

In "Memory Systems"[102], Jacob et al. pointed out that the effectiveness of a disk cache depends on two aspects: the disk cache organization and the disk cache algorithm. The first aspect, the disk cache organization, defines how the disk cache and its data are structured, including how the disk cache is organized and allocated, and how data are stored and identified in the disk cache memory. The second aspect of disk cache is the algorithms used, which defines the policies of how the cache is being utilized. The algorithm includes but is not limited to:

- destage policy--determine which piece of old data to destage/replace (destage is a process to write data from the cache to the disk.),
- prefetching policy--what data and how much data to prefetch
- scheduling policy--which requests should be processed first.

Jacob et al. also suggested that these two aspects are related but orthogonal in operation. One aspect's operation neither depends upon nor involves the other. However, some particular algorithms may be more suitable to be implemented with a specific type of structure. Since, in this dissertation, we are focusing on the I/O intensive phase of the execution consisting of mostly sequential requests served well with FCFS scheduling and LRU replacement policy, the second aspect will not be discussed in this dissertation.

Zhu and Hu [75] have suggested that large disk built-in caches will not significantly benefit the overall system performance because all modern operating systems already use large file system caches to cache reads and writes. In our experiments, we investigated the disk cache including the effects of file system caching. As suggested by Przybylski [101], the reference stream missed by the L1 cache has low locality. Like the reference stream in the processor caches, the reference stream to the disk system missed the file system cache. As a result, the locality in the stream is low. To reflect a realistic stream to the disk system, it is important to include the file system cache effect. Therefore, the reference stream including file system caching effect in our experiment is more realistic than the stream in the experiments for trace-driven simulations. This method is hard to implement, but extremely valuable, so we ensure that the resulting reference stream represents realistic behavior of real systems.

4.2.1.2. Prefetching

Prefetching is generally referred to the technique of acquiring the data before they are actually used in the system. Like caching, prefetching is implemented in many levels in memory hierarchy, including the disk systems. It involves two steps, which are predicting which data are likely to be used in the near future and fetching them before they are actually needed. The purpose of prefetching is to hide the stall time that the system has to wait if the data are fetched on demand. Hsu and Smith [78] pointed out that the overall effectiveness of prefetching at improving performance depends on the following:

1. the accuracy of the prediction
2. the amount of extra resources (memory use, disk and data-path busy time, etc.) that is consumed by the prefetching process
3. the timeliness of the prefetch, i.e., whether the prefetch is completed before the blocks are actually needed

There are several prediction schemes for prefetching on the host side to improve I/O performance in server applications, i.e. Database Management Systems in multiple parallel disk systems, or implemented in the operating systems. The host can prefetch in these systems more intelligently than multi device controllers or disk caches can, since applications can often better predict read requests. As examples given by Hsu and Smith [78], the prediction of the host-side prefetching is usually based on past access patterns [84, 85], system-generated plans [87, 88], user-disclosed hints [86], and even guidance from speculative execution [89]. Depending on the implementation, this information may be available to help with the prediction. However, only a few works explore the prefetching in disk cache [90, 81]. The disk drive prefetch generation is usually simply implemented as sequential prefetching. The reason for the disk drive to initiate prefetching is that the drive is aware of its own status, so it can avoid prefetching, which can be interfered from host requests. However, the drive does not know what data it stores, so it requires additional hints from the host, i.e. which data it should prefetch for complex prefetching schemes. Therefore, complex prefetching in the disk drive side is not a common practice and only prefetching schemes based upon the principle of locality of access are implemented. Recently, since energy consumption has become major concern, a number of publications are proposing Energy-Aware data prefetching that is orchestrated by the compiler [91] or implemented with Flash Drives [92].

Since prefetching is based on speculation, the prefetched data may not be used. However, prefetching comes with cost. The cost of prefetching includes the use of the disk drive's resources, i.e. the drive's electronics and mechanical parts, to bring the prefetched data into the cache. Therefore, prefetching can occupy the disk drive's resources and prevent the drive from doing other useful work. It may interfere with the user requests and may cause a user request to be delayed and to observe an increase in request response time. Therefore, a common strategy is to disable prefetch if there are I/O requests waiting to be serviced in the queue, and preempt any ongoing prefetch when a new I/O request arrives if the request requires the disk resources being used in the prefetch. If the new request can be serviced without requiring the other disk resources used in prefetching, such as a request hitting in the cache, the drive can continue prefetching. Besides the disk drive's electronics and mechanical parts, prefetched data also consume space in the disk cache. If the prediction of prefetch is not accurate, prefetching may cause cache pollution just like in a processor cache. Cache pollution occurs when prefetched data are not used in the near future, but they cause other useful data to be thrown out from the cache since prefetched data contends for this space. This may result in degradation of total performance. Therefore, care should be taken when applying prefetch to the disk system.

Most workloads generate disk request stream with high sequentiality. Therefore, simply sequential prefetching, especially on a cache miss, works well on all three criteria mentioned above, which are prediction accuracy, cost, and timeliness. In addition, a request for a large chunk of sequential data can be served more effectively with sequential prefetching in many storage systems. To improve prefetching performance further, small sequential requests are merged into one large request, which can be served more efficiently by the storage system. Therefore, it is a common practice for modern storage systems to be equipped with sequential prefetching after a cache miss. We focus on such a prefetch in this dissertation due to the sequential nature of the disk requests in the I/O intensive phase. The prefetched data are managed as if they were fetched by reference in this dissertation. Unlike our prefetch scheme, the prefetched data in other systems can be allocated in a separate buffer or can be managed in the cache differently from data fetched by reference. The interested reader is referred to [85] for an evaluation of such alternatives.

4.2.1.3. Write Buffering

The I/O subsystem is becoming a bottle-neck in the computer system due to the rapid growth in the processor speed and technology. Increasing the memory size for the file system cache and increasing built-in disk cache size will improve the caches' effectiveness to satisfy disk read requests. As a result, disk traffic will consist of mostly write traffic. There are many proposed solutions to this problem, including Log-Structured File Systems [93] in the operating system side and write-buffering in the disk drive side [95].

The term write buffering refers to the technique of temporarily holding written data in fast memory before the data are written into the physical storage permanently. Right after the data are accepted by the buffer, a write operation can be reported as completed. Unlike read caching, write commands to the write buffer do not require data in the cache. Therefore, the write commands are processed like a cache hit if there is available space in the cache memory. The write command latency is composed of only the drive controller overhead and data transfer time. No mechanical time is involved. However, an amount of the dirty data is “destaged”, written out to the disk media to free up space when the cache is mostly occupied, depending on the write-buffering policy. To prevent conflicts with user requests, destaging should be performed while the drive is idle. This avoids the interference with user requests which causes noticeable delay. In reality, the interference may be inevitable because the drive may be under a high usage with only minimal idle time. The situation is most likely to occur because disk writes usually come in bursts, so they easily fill up the buffer. In this case, destaging must take place while the drive is busy. Therefore, destaging adds more load to the drive in this case, which causes longer delays for user requests. As a result, write buffering may be only a technique to delay the disk writes instead of hiding them completely.

Even though write buffering may only delay the disk writes, delaying those writes may be beneficial. The write requests in the write buffers can be scheduled, merged, or overwritten for better performance, i.e. fewer requests must be sent to the physical disk. Most systems implement a write buffer by allocating a part of DRAM memory under the control of the disk controller for it. Therefore, in server systems, write buffering is usually disabled for reliability reasons. To prevent data loss of data, the write buffer can be implemented with some form of nonvolatile storage (NVS), such as non-volatile RAM (NVRAM) [95, 79, 77], a disk cache disk (DCD) [76], NAND Flash memory [82], or MEMS-based storage [96]. In some environments, (e.g., UNIX file system, PC disks), periodically flushing the buffer contents to the physical disk, i.e. every 30 seconds, is considered sufficient.

In summary, there are three key benefits when utilizing write buffering. First, write buffering is used to hide the latency of writes by delaying the writes to until an appropriate time, i.e. without interfering with user requests. Second, by merging and overwriting the write data in the buffer, the write buffering can improve the performance by reducing the number of writes to the physical disk. Finally, by scheduling the writes to the physical disk, the write buffering helps the physical disk to perform more efficiently.

While the write buffering technique performance can be affected by several destage parameters, such as high water mark, low water mark, and the size of write cache, in this dissertation we exclude those effects. We conducted a limit study of write buffering by assuming that the destage algorithm is perfect, so write buffering can completely hide the write latency. More details about destage algorithms can be found in [94] and [79].

4.2.1.4. Parallel I/O

Another widely used technique to improve I/O performance is to use parallel I/O. Parallel I/O simultaneously distributes multiple small requests to be serviced by several disks. A large request can also be serviced by multiple disks at the same time. Therefore, parallel I/O can improve both response time for individual requests and improve the throughput in for multiple requests. The two most common approaches are to distribute data among multiple disks are organizing the disks in a volume manner and organizing the disks into a stripe manner. The first approach, a volume manner, fills data into one disk until it is filled then moves to the next disk. On the other hand, the second approach, a stripe manner, divides data into small units called stripe units, and distributes the stripe units across the disks in a round-robin manner. Therefore, the volume manner can be considered as a stripe manner with an entire disk is one stripe unit. Even though the stripe manner has an advantage over the volume manner for parallelism, striping data across multiple disks introduces low reliability to the disk system. As a result, redundant information is added to the striped disk system to improve reliability. This striped disk systems with redundant information are collectively called Redundant Arrays of Inexpensive Disks is RAID [80].

The following is a quick summary of the most commonly used RAID levels defined in [80]:

- RAID 0: Striped Set

RAID 0 stripes data evenly across multiple disks without parity information for redundancy. RAID 0 was not included in the originally defined RAID levels because its reliability is not enhanced with redundancy. It was invented to improve parallelism in the disk systems for performance gains and to create a large logical disk space with multiple small disk drives.

- RAID 1: Mirrored Disks

A traditional approach for improving reliability of magnetic disks is mirroring. As suggested with the name, mirrored disks duplicate all data to the mirrored disks. Therefore, a write actually is two writes to the data disk and the mirrored disk. Mirroring disks is considered the most expensive approach to improving reliability since all disks are duplicated. An optimized version of mirrored disks doubles the number of controllers for fault tolerance, so it allows reads occur in parallel to improve performance.

- RAID 2: Hamming Code for ECC

RAID 2 imitates the DRAM bit-interleaving behaviors by bitwise striping the data across multiple disks with redundancy. The redundancy is accomplished by additional check disks to detect and correct a single error. Multiple disks are required to detect and correct a single error. Multiple disks are used to identify the erroneous disk. Therefore, RAID 2 suffers from low usable space in the disk system because multiple disks are required for check disks. For a group size of 10 data disks, we need 4 check disks in total.

- RAID 3: Single Check Disk Per Group

RAID 3 stripes data in the unit of byte with one dedicated parity disk. RAID 3 replaces multiple check disks in RAID 2 with a single parity disk. The principle is the disk controllers can detect that a disk failed, so multiple check disks in RAID 2 unnecessarily duplicates the function. If a disk has failed, the data on the failed disk can be reconstructed by the data on the remaining disks and the parity information. If the disk is the parity disk, the parity information can always be recalculated by the original data, and be stored easily in the replacement disk. This mechanism results in lowest reliability cost in RAID 3. So, the last two levels, RAID 4 and RAID 5, consider only improving the performance of small accesses, but the cost of reliability remains the same as RAID 3.

- RAID 4: Independent Reads/Writes

RAID 4 stripes data in block-sized granularity across multiple disks with one dedicated parity disk. The only difference between RAID 3 and RAID 4 is RAID 4 stripes per block, rather than per byte. Consequently, the performance of RAID 4 is improved over RAID 3 because all reads/writes can now be serviced independently. The reason is a write uses 2 disks to perform 4 accesses—2 reads and 2 writes—while a small read involves only one read on one disk.

- RAID 5: No Single Check Disk

While level 4 RAID achieved high performance for reads due to parallelism, writes are still limited to one per group since every write must read and write the check disk. The RAID 5 distributes the data and checks information across all the disks—including the check disks. Therefore, RAID 5 supports multiple individual writes per group. These changes make RAID 5 satisfy the best performance for both reads and writes. RAID 5 can perform small read-modify-writes close to the speed per disk of RAID 1 while it performs large transfers per disk and retains the high ratio of usable storage capacity of RAID 3 and RAID 4. Spreading the data and parity across all disks even improves the performance of small reads, since there is one more disk per group that can perform read concurrently.

In this dissertation, we implemented only RAID 5 since it is the most widely used RAID level. It also fits well with the concept of disk performance enhancements which we would like to compare against other techniques.

Chen et al. [97] studied the striping effects in a RAID 5 disk array. As mentioned earlier, a striping unit is defined as the maximum amount of logically contiguous data that is stored on a single disk. A large striping unit makes a file span only on only a few disks. A small striping unit spans a file across more disks. They found that the optimal striping unit for the write-intensive workloads to be four times smaller than in the case of read-intensive workloads. The reason is the overhead of maintaining parity causes full-stripe writes (writes that span the entire error-correction group) to be more efficient than read-modify writes of reconstruct writes. The optimal striping unit for reads in RAID 5 varies inversely to the number of disks, but the optimal striping unit for writes varies with the number of disks. In conclusion, they derived general design rules for striping data in RAID 5 systems to be one half of average positioning time times disk transfer rate.

However, the application behavior characterizations (read/write intensive) vary with the system memory size. That is, a read-intensive application can become a write-intensive one by decreasing the memory and vice versa. We chose a fixed striping unit of 16KB as it performed well in both read and write intensive applications reported in [97].

4.2.2. Disk Drive Physical Improvements

The storage of a disk drive consists of a set of multiple rotating platters, on whose surfaces data is recorded. Each surface has its own head to perform read/write operations, but all heads are attached to a single set of mechanical arms moving as one. Therefore, there are multiple dimensions to the performance of a physical disk, for instance, what is the rate at which the platters rotate (RPM), how fast does the arm move (seek time), and how closely packed is the data (areal density). All dimensions affect the overall performance differently in terms of response time and throughput. Additionally, the effectiveness of a disk depends on the disk access order. As a result, only providing the numbers in the terms of those physical features is not an obvious indicator to reflect the effect of the disk technology improvement to real-world performance. In this dissertation, we also evaluate the improvement in underlying physical technology in terms of the actual system performance/power of real workloads. Hsu and Smith [78] stated that it is complicated to isolate and quantify the performance impact of the disk technology improvement in the different dimensions. Their reason is that disk technology improvements can affect the performance metrics, i.e. access time, in many dimensions. They gave the examples that the often-quoted 10% yearly improvement in the access time of disks is actually a result of a multi-dimensional combination including higher rotational speed, faster seek time due to improvement in the disk arm mechanism, and smaller diameter disks with higher track density, which also reduce seek distance. In practice, the areal density reduces the seek time because the head is moved less to reach the destination track. Also, the areal density improves the internal data rate since, with the same rotational speed, the head can process more data when the data are packed more closely together. Higher areal density also improves the storage capacity per surface, and results in fewer disks for the data mapping mechanism. Observe that for various workloads in [78], the average response time and service time are projected to improve by approximately 15% per year because it takes into account the dramatic improvement in areal density and assumes that the workload and the number of disks used remain constant. In fact, our metric, the total system performance metric (CPI), has never been studied against the disk drive physical improvements. In this dissertation, we break down the improvements in disk technology into three major basic effects:

- Seek time reduction due to actuator improvement.
- Increase in rotational speed.
- Interface data rate

4.2.2.1. Seek Time

Seek time is the time to move the read-write head from its current track to the destination track to service the next request. Seek time is one of the largest components of access time because it is dealing with moving mechanical parts. Therefore, seek time plays a significant role of a disk access time. With the recent rapid increase in areal density, the mass production of smaller and lighter drive disk platters is the major cause in seek time improvement. Higher areal density translates into less distance for the head to seek, and the smaller disk drive platters with smaller and lighter actuators and arms can be easily moved in less time. Seek time is composed of two components, which are the travel time and the settle time. The settle time is the time the head requires to maintain to correct position on the track after arriving at the destination track. It includes the identification and confirmation of the correct destination track and the head is ready for data transfer. With rapid growth in tpi (tracks per inch), the travel time component in seek time decreases drastically, so settle time grows its significance. Typical average seek time for today's server-class disk drives is about 3.5 milliseconds while for desktop drives is about 8 milliseconds. Mobile drives' seek time is typically slower in order to reduce power consumption. Historically, an 8% improvement in the average seek time translates roughly into only 3% improvement in the average response time [78].

4.2.2.2. Rotational Speed

After the head has reached the destination track and settled at the center of the track, the disk rotates to bring the destination sector to the head. Rotational latency is defined as the time the disk rotates to bring the destination sector to the position under the head. Since magnetic disk drive rotational speed is constant, the average rotational latency on the datasheet is simply calculated as one-half the time it takes the disk to finish one complete revolution. The rotational speed is inversely proportional to the rotational latency. The rotational speed is another great component of access time because it deals with moving mechanical parts. The rotational speed or RPM (revolution-per-minute) improves in discrete steps. The rotational speed has historically increased at 9% per year, which corresponds to about a 5% improvement in average response time [78]. As portrayed by Jacob et al. [102], higher RPM is first introduced in the high-end server drives, and takes approximately 10 years to travel down to mobile drives. The new speed takes a few years to be adopted by the majority of server drives and then another few more years to be commonly used among desktop drives. Finally, the RPM speed takes another few more years to be introduced in mobile drives. Jacob et al. gives an example of the 7200-RPM disk drive, which was first introduced in server drives back in 1994. It was appeared in desktop drives in late 1990s, and the first 7200 RPM mobile drive was not available until 2003 - almost ten years after the first 7200 RPM drive was introduced in server systems. Today's high-end server drives run at 15K RPM, with 10K RPM being the most common, while desktop drives are mostly 7200 RPM [78].

4.2.2.3. Interface Data Rate

Interface data rate is the rate that the data can be transferred between the disk drive and the host over the interface. The most recent disk drive interfaces, data rates, and the bus latency per sector are shown in Table 4.2 below. All these interfaces are substantially faster than the disk internal data rate. In our experiments, we varied the interface transfer time per sector from 0.64 microseconds to 1 millisecond.

Interface Type	Data Rate (MB/s)	Bus latency (us) per 512B-sector
ATA-7	133	3.85
SCSI Ultra 320	320	1.6
SATA and SAS	300	1.7
future SATA and SAS	600	0.85
FC	200	2.56
future FC	400	1.28

Table 4.2: Latest Disk Interfaces and Their Data Rate

Most interface data transfers can be overlapped with the internal data transfer, excluding the very last block case. The reason is that every sector of data must be in the drive buffer entirely for error checking and possible error correcting code (ECC) correction before it can be sent to the host. For writes, all of the interface data transfer from the host can be overlapped with the seek process, disk rotation, and the disk drive internal data transfer if necessary.

CHAPTER 5: METHODOLOGY

SYSim is a model of an entire memory hierarchy that includes both performance models and energy models for cache, DRAM, and disk. The SYSim project is incorporated with several simulators for each component of the system. Figure 5.1 shows SYSim architecture and its components.

Bochs [11], a Pentium emulator, is used as the CPU model to generate the memory accesses and I/O interrupts. The cache model comes from Wattch [19]. The authors of Wattch integrate Cacti [12] to obtain the cache configuration with the best timing behavior for the cache.

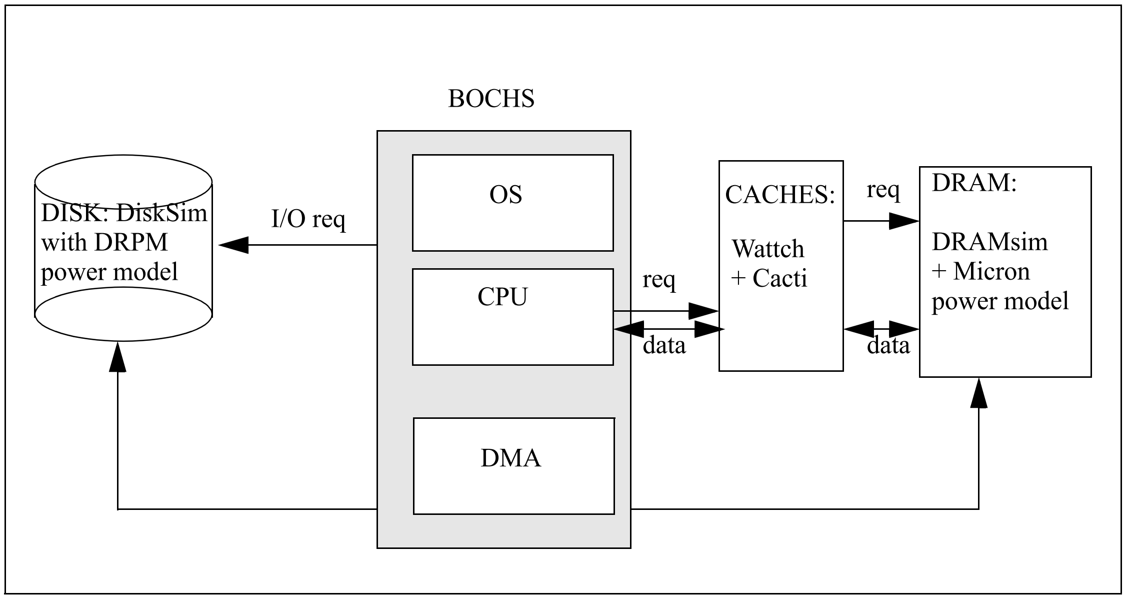

BOCHS
OS
CPU
DMA
DISK: DiskSim with DRPM power model
I/O req
req
data
CACHES:
Wattch + Cacti
req
data
DRAM:
DRAMsim + Micron power model

Figure 5.1: SYSim architecture. including CPU with OS from Bochs [11], caches with power model from Wattch [19] and Cacti [1 DRAMsim from University of Maryland [17] with power model from Micron [18], Disk with power model from DRPM [15] and DiskSim [14], a all interfaces between them

After a miss from the last level of the caches, the SYSim accesses the DRAM. The DRAM simulator from the University of Maryland [17] is integrated into the system to provide the timing behavior and also the power consumption. The simulator is implemented in a very detailed way since it has the concept of channels, ranks, and banks. As part of this dissertation, a power model [18] was incorporated into the DRAM simulator in order to generate the instantaneous power consumption.

Bochs has the basic model of a Disk, but neither timing nor power statistics is considered. The Disk model in Bochs takes the responsibility only to read and write data from/to the disk image. Therefore, we integrate a modified DiskSim [14] simulator as used in the DRPM paper [15] into the system. The DRPM version of DiskSim is used for only timing and power consumption statistics collection. We obtained a disk image from Bochs website which has Redhat Linux 6.0 installed. Finally, a set of programs from SPECINT2000 benchmark [13] are complied and installed in an OS-ready disk image.

5.1. The Processor Simulator: Bochs

Bochs is a highly portable open-source PC emulator written in Object-Oriented C++, which runs on most popular platforms. It includes emulation of the Intel x86 CPU, common I/O devices, and a customized BIOS. The typical use of Bochs is to provide complete x86 PC emulation, including the x86 processor, hardware devices, and memory. This allows running OS's and software within the emulator on a workstation. Currently, Bochs can be compiled to emulate a 386, 486, Pentium, Pentium Pro or AMD64 CPU, including optional MMX, SSE, SSE2 and 3DNow! instructions. In addition, Bochs is able to run most Operating Systems inside the emulation including Linux, Windows95, DOS, and Windows NT 4.

For the SYSim project, we compiled Bochs for Pentium since it is the most current released version of Bochs reported to be stable. We also obtained a disk image of Redhat Linux version 6.0 from the Bochs website. As the objective of the project is to construct a simulator exclusively for memory system, the CPU simulator is viewed as a black box that generates memory access requests to the L1 cache and I/O interrupts to the disk. It can be considered as a trace generator that can reflect the changes in memory organizations and the memory latency in the hierarchy, but it is more valuable than a trace generator because the requests respond to the timing in the memory hierarchy, and it reflects the effects of the unmodified operating system.

5.2. The Cache Simulator: Wattch

Wattch [19] is an architecture-level microprocessor power estimation tool. The power models are integrated into the SimpleScalar architectural simulator [28]. The cache model in Wattch is implemented as an array structure. The power model of the cache is based on the number of rows, columns, and the number of read/write ports. These parameters affect the size and number of decoders, the number of wordlines, and the number of bitlines. In addition, these parameters are used to estimate the length of the pre-decode wires as well as the lengths of the array's wordlines and bitlines. The wordline and bitline capacitance are computed in the same way. The wordline capacitance includes the capacitance of the wordline driver, the gate capacitance of the cell access transistor multiplied with the number of bitlines, and the capacitance of the wordline's metal wire. The bitline capacitance includes the diffusion capacitance of the pre-charge transistor, the diffusion capacitance of the cell access transistor multiplied by the number of word lines, and the metal capacitance of the bitline. The number of ports also affects the power consumption due to additional transistor connection on wordlines, two additional bitlines, and longer wire on both wordlines and bitlines.

Wattch authors estimate the physical implementations for cache structures using the help of the Cacti tools [12]. Cacti takes the cache size, block size and associativity as inputs, and chooses the organization that gives the smallest access time. Cacti models each component of the cache in transistor level considering the technology dependence parameters. Cacti authors compare the results with Hspice model.

Wattch consider three different options for clock gating to disable unused resources in multi-ported cache.

1. *All-or-nothing approach.* The full modeled power will be consumed if any accesses occur in a given cycle, and zero power consumption otherwise.
2. *Scaled linearly.* If only a portion of a cache's ports are accessed, the power is scaled linearly with the number of accessed port(s).
3. *Scaled linearly with 10 per cent.* It is the same as the second option except that unused units dissipate 10% of their maximum power, rather than drawing zero power.

Since the amount of clock gating in current processors falls somewhere between these styles, SYSim calculates all three clock gating styles on the fly. Then, it output all three power consumption numbers corresponding to the clock gating styles into the output file. So, the user has the freedom to choose which clock gating style he prefers.

5.3. The DRAM Simulator and Its Power Model

The DRAM simulator from University of Maryland [17] is integrated into SYSim. The DRAM simulator is considered an extremely detailed DRAM simulator, and it is extremely configurable. Every parameter can be set, for examples, type of the DRAM, the DRAM configuration (i.e., number of channels, ranks, banks, rows, and columns), the operating frequency, refresh policy, address mapping policy, close/open page policy, all timing parameters, etc. It includes the bus interface unit, the memory controller unit, the DRAM DIMMs, and DRAM devices. It has the concept of individual state of each channel, rank, and bank. Each memory access request is transformed to a transaction consisting of the combination of row activating, column read/write, and precharge commands. In order to generate such commands, the DRAM simulator considers the current state of each bank. It also takes the timing specification from the manufacturer's datasheet into account, and generates the timing diagram for command bus and data bus. It can simulate a wide variety of the DRAM types, i.e. SDRAM, DDR SDRAM, DDR2 SDRAM, DRDRAM, and fully-buffered DIMMS.

The DRAM simulator has been carefully validated against real hardware and three different detailed DRAM simulators, using in published DRAM studies. The accuracy demonstrated exceeds that of any other simulators.

To generate the power consumption, the power model by Micron [18] has been added to the DRAM simulator. The modified DRAM simulator can generate the power number and its related statistics to an output file on every epoch. Currently, the DRAM simulator is incorporated with a power model for DDR and DDR2 SDRAM. Basically, to calculate the power is to calculate the average power in one activation-to-activation cycle. That is, we calculated the power in each DRAM state and then multiplied it with the fraction of time the device spends in each state with respect to one activation-to-activation cycle. For simplicity, we consider the power model for DDR SDRAM first, and then make some extensions to cover the DDR2 case. The power consumption in DDR SDRAM is calculated as follows:

Parameter/Condition	Symbol	Units
OPERATING CURRENT: One bank; Active Precharge; t_{RC} = t_{RC} MIN; t_{CK} = t_{CK} MIN	$I_{DD}0$	mA
PRECHARGE POWER-DOWN STANDBY CURRENT: All banks idle; Power-down mode; t_{CK} = t_{CK} MIN; CKE = LOW	$I_{DD}2P$	mA
IDLE STANDBY CURRENT: CS_ = HIGH; All banks idle; t_{CK} = t_{CK} MIN; CKE = HIGH	$I_{DD}2F$	mA
ACTIVE POWER-DOWN STANDBY CURRENT: One bank; Power-down mode; t_{CK} = t_{CK} MIN; CKE = LOW	$I_{DD}3P$	mA
ACTIVE STANDBY CURRENT: CS_ = HIGH; One bank; t_{CK} = t_{CK} MIN; CKE = HIGH	$I_{DD}3N$	mA
OPERATING CURRENT: Burst = 2; READs; Continuous burst; One bank active t_{CK} = t_{CK} MIN; I_{OUT} = 0mA	$I_{DD}4R$	mA
OPERATING CURRENT: Burst = 2; WRITEs; Continuous burst; One bank active t_{CK} = t_{CK} MIN	$I_{DD}4W$	mA
AUTO REFRESH CURRENT; tRC = 15.625ms	$I_{DD}5$	mA

Table 5.1: The definitions of symbols in the DRAM datasheet[a]

a. Data Sheet Assumptions
IDD is dependent on output loading and cycle rates. Specified values are obtained with minimum cycle time at CL = 2 for -75Z, -8 and CL = 2.5 for -75 with the outputs open.
$0°C \le T_A \le 70°C$
$(V_{DD}Q)/V_{DD} = 2.5V \mp 0.2V$
CKE must be active (HIGH) during the entire time a REFRESH command is executed. That is, from the time the AUTO REFRESH command is registered, CKE must be active at each rising clock edge, until t_{REF} later.

There are parameters extracted from a DDR SDRAM data sheet involved in the calculation. Table 5.1 shows the definition of the I_{DD} values from a data sheet. In order to calculate the power, two states are defined. When data is held in any of the sense amplifiers, the DRAM is said to be in the “active state”. And after all banks of the DDR SDRAM have been restored to the memory array, it is said to be in the “precharge state”. Additionally, CKE, the device clock enable signal, is considered. In order to send commands, read, or write data to the DDR SDRAM, CKE must be HIGH. If CKE is LOW, the DDR SDRAM clock and input buffers are turned off, and the device is in the power-down mode.

From the definition of active/precharge states and CKE above, a DRAM device can be in four states:

1. Active Standby Power: $p(ACTstby) = IDD3N \times VDD \times (1 \angle BNKpre) \times (1 \angle CKEloACT)$
2. Active Power-down Power: $p(ACTpdn) = IDD3P \times VDD \times (1 \angle BNKpre) \times CKEloACT$
3. Precharge Standby Power: $p(PREstby) = IDD2F \times VDD \times BNKpre \times (1 \angle CKEloPRE)$
4. Precharge Power-down Power: $p(PREpdn) = IDD2P \times VDD \times BNKpre \times CKEloPRE$

where

- IDD values are defined in the data sheet and VDD is the maximum voltage supply of the device.
- BNKpre is the fraction of time the DRAM device is in precharge state (all banks of the DRAM are in precharge state) compared with the actual activation-to-activation cycle time.
- CKEloPRE is the fraction of time the DRAM stays in precharge state and CKE is low compared with the time it stays in precharge state.
- CKEloACT is the fraction of time the DRAM stays in active state and CKE is low

compared with the time it stays in active state.

Figure 5.2 shows the four states of a DRAM device during an activation-to-activation period with a read burst [18]. We assume that the CKE signal becomes low as soon as there is no activity in the DRAM device.

In addition, when the DRAM device is in Active Standby state, commands can be sent to the device. The activities corresponding to the commands cause an increase in the current sent to the DRAM device. For example, during the Active Standby state in figure 5.2, the current increases due to activation command, and then it drops to IDD3N. During the read process, the current is also pulled up. Finally, the DQ termination increases the current

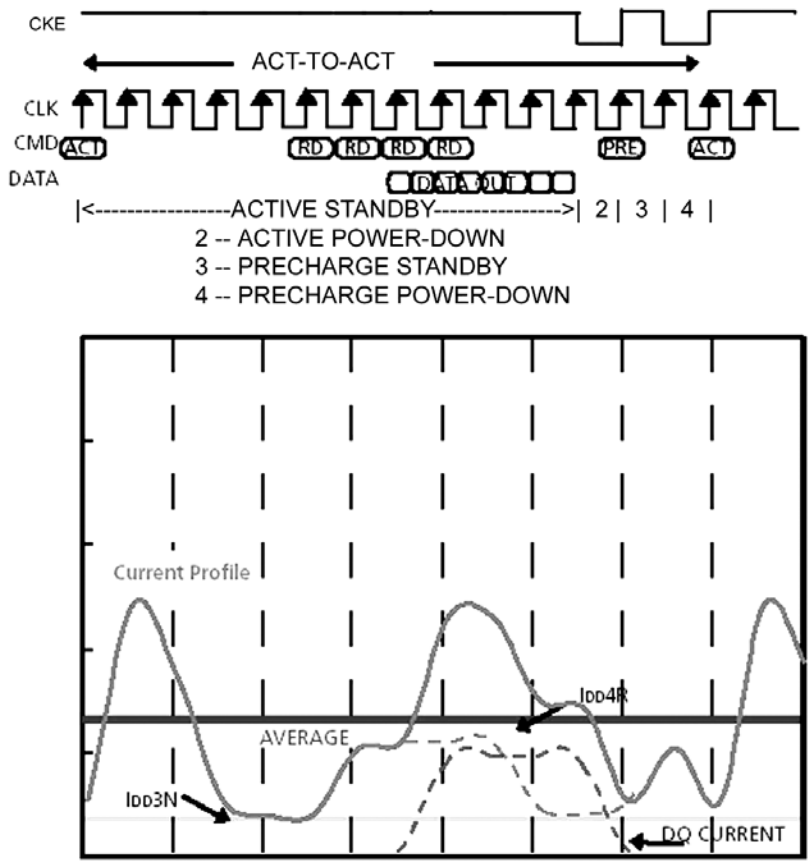


Figure 5.2: Read Current with I/O Power Included [18]. The four states of DRAM device are shown, Active Standby state, Active Power-Down state, Precharge Standby state, and Precharge Power-Down state. We assume other banks in the DRAM device are precharged. The DRAM device is in Active state when data is stored in any of the sense amplifiers, after an acitvation command. The device is in Precharge state if all banks are in precharged. The device is in Power-Down mode if CKE signal is low, and it is in Standby mode, otherwise.

during data transfer out from the device. Therefore, we have 4 more states in the Active Standby state.

1. Activate Power: $p(ACT) = (IDD0 \angle IDD3N) \times \frac{tRC}{tACT} \times VDD$
2. Write Power: $p(WR) = (IDD4W \angle IDD3N) \times WRpercent \times VDD$
3. Read Power: $p(RD) = (IDD4R \angle IDD3N) \times RDpercent \times VDD$
4. Termination Power: $p((DQ) = p(perDQ) \times (numDQ + numDQS) \times RDpercent)$

where

- tRC is the shortest activation-to-activation cycle time as specified in the data sheet.
- tACT is the actual activation-to-activation cycle time in the real system.
- WRpercent is the fraction of time the data, to be written, stays on the data pins compared with the actual activation-to-activation cycle time.
- RDpercent is the fraction of time the read data stays on the data pins compared with the actual activation-to-activation cycle time.
- p(perDQ) is the power of each DQ. It depends on the termination scheme. In this case, we use p(perDQ) = 6.88mW for DDR SDRAM.
- numDQ and numDQS are the number of DQ and DQS pins in the device, respectively.

And, Refresh Power: $p(REF) = (IDD5 \angle IDD2P) \times VDD$

Notice that IDD3N is deducted out from the calculation since we already include it in the p(ACTstdby). Also, in the current version of the DRAM simulator, we simulate a refresh command as a row activate command with a precharge command.

Then we scale the voltage and frequency to the ones we actually operate on. As a result, we obtain:

$$P(PREpdn) = p(PREpdn) \times \frac{useVDD^2}{maxVDD^2}$$

$$P(ACTpdn) = p(ACTpdn) \times \frac{useVDD^2}{maxVDD^2}$$

$$P(PREstby) = p(PREstby) \times \frac{usefreq}{specfreq} \times \frac{useVDD^2}{maxVDD^2}$$

$$P(ACTstby) = p(ACTstby) \times \frac{usefreq}{specfreq} \times \frac{useVDD^2}{maxVDD^2}$$

$$P(ACT) = p(ACT) \times \frac{useVDD^2}{maxVDD^2}$$

$$P(WR) = p(WR) \times \frac{usefreq}{specfreq} \times \frac{useVDD^2}{maxVDD^2}$$

$$P(RD) = p(RD) \times \frac{usefreq}{specfreq} \times \frac{useVDD^2}{maxVDD^2}$$

$$P(DQ) = p(DQ) \times \frac{usefreq}{specfreq}$$

$$P(REF) = p(REF) \times \frac{useVDD^2}{maxVDD^2}$$

Finally, sum everything up for the total power:

$$\begin{aligned} P(TOT) = P(PREpdn) + P(PREstby) + P(ACTpdn) + P(ACTstby) + P(ACT) + \\ P(WR) + P(RD) + P(DQ) + P(REF) \end{aligned}$$

In case of DDR2 SDRAM, most of the calculations remain the same except p(ACT), p(REF), and the I/O and termination power. For DDR2 SDRAM, p(ACT) before the voltage/frequency scaling is:

$$p(ACT) = \left[IDD0 \angle \frac{IDD3N \times tRAS + IDD2N \times (tRC \angle tRAS)}{tRC}\right] \times VDD$$

Then we scale it the same as in the DDR SDRAM case.

The refresh power p(REF) is:

$$p(REF) = (IDD5 \angle IDD3N) \times VDD \times \frac{tRFCmin}{tREFI}$$

In the power model of DDR2 SDRAM, the simulator supports two cases, 1) one-rank case, and 2) multiple rank case but at most four rank. For the one rank case, the termination powers are:

WriteTermination Power: $p(termW) = p(dqW) \times (numDQ + numDQS + 1) \times WRpercent$

Read Termination Power: $p(DQ) = p(dqR) \times (numDQ + numDQS) \times RDpercent$

Read Termination Power and Write Termination Power to other ranks are zero:

$$p(termRoth) = p(termWoth) = 0$$

where p(dqW) = 8.2 mW and p(dqR) = 1.1 mW.

In the case of multiple ranks, the read termination power and write termination power are the same with p(dqW) = 0 and p(dqR) = 1.5 mW. However, the DRAM needs to terminate from the other ranks. The termination powers from other ranks are:

$$p(termRoth) = p(dqRDoth) \times (numDQ + numDQS) \times termRDsch$$

$$p(termWoth) = p(dqWRoth) \times (numDQ + numDQS + 1) \times termWRsch$$

where

- p(dqRDoth) is the termination power when terminating a read from another DRAM, and is equal to 13.1 mW.
- p(dqWRoth) is the termination power when terminating write data to another DRAM, and is equal to 14.6 mW.
- termRDsch is the fraction of time that read terminated from another DRAM.
- termWRsch is the fraction of time that write terminated to another DRAM.

Finally, we sum it all to obtain the total power of the DDR2 SDRAM:

$$P(TOT) = P(PREpdn) + P(PREstby) + P(ACTpdn) + P(ACTstby) + P(ACT) + P(WR) + P(RD) + P(DQ) + P(REF) + p(termW) + p(termWoth) + ptermRot(h)$$

During the simulation, SYSim collects the statistics information in each epoch. At the end of the epoch, SYSim calculates the total power of each DRAM chip, and multiplies with the number of chips in a rank to generate the per-rank power. In an oracle fashion, SYSim switches the device to the power-down mode as soon as possible. During the time that the DRAM simulator is not called, SYSim also accounts this time as a power-down mode. The device is switched to either precharged or active power-down mode, depending on the state of the banks in the device.

5.4. The Disk Simulator: DiskSim

DiskSim [14] is an efficient, accurate, highly-configurable storage system simulator. It is written in C and requires no special system software. It includes modules for many secondary storage components of interest, including device drivers, buses, controllers, adapters and disk drives. Some of the component modules are highly detailed (e.g., the disk module), and the individual components can be configured and interconnected in a variety of ways. DiskSim can be driven by externally-provided I/O request traces or internally-generated synthetic workloads. DiskSim has been used in a variety of published studies to understand modern storage subsystem performance, to understand how storage performance relates to overall system performance, and to evaluate new storage subsystem architectures.

The disk module in DiskSim, which is extremely detailed, has been carefully validated against five different disk drives from three different manufacturers. The accuracy demonstrated exceeds that of any other disk simulators.

The DiskSim model that we used is taken from the DRPM paper [15]. The model includes power models to record the energy consumption of the disks when performing operations like data transfers, seeks, or when just idling as described as TPM model in DRPM paper. The model collects the latency of each state that the disk stays, and multiplies the latency with the power consumption number in Figure 5.3 to generate the total energy consumption. We modified the reported energy at the end of the simulation to report power in every epoch on-the-fly.

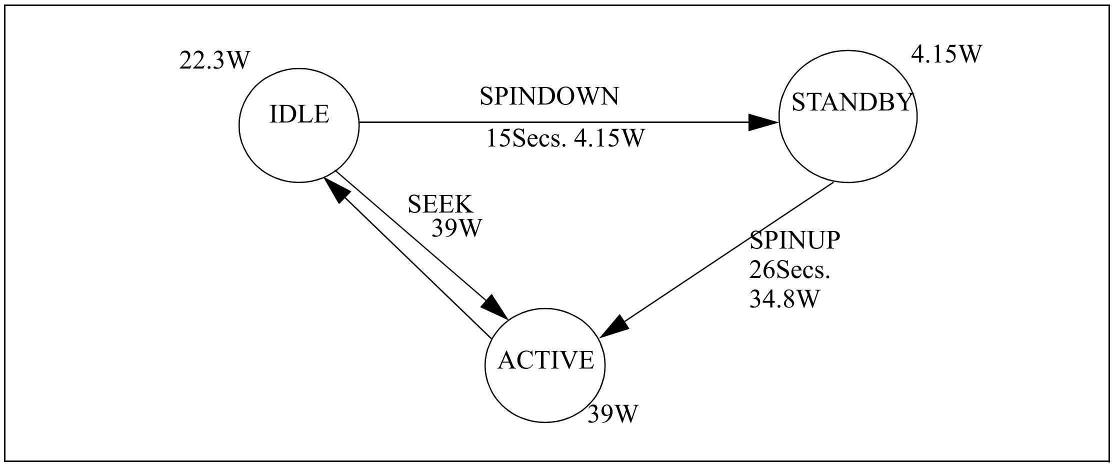


Figure 5.3: TPM Power Modes [15].

However, DiskSim models the performance behavior of disk systems, but does not actually save or restore data for each request. Therefore, we incorporate the DiskSim model on top of the Disk model of Bochs. We modified Bochs' Disk model to convert the I/O requests from the CPU to DiskSim requests and to place the requests in the DiskSim interrupt queue. After that, DiskSim is called to simulate the event and returns the latency. Then, after Bochs updates the timing, the Bochs' Disk model reads (or writes) the data from the disk image and return the control to the CPU.

5.5. The Benchmarks: SPEC2000

The SPEC CPU2000 benchmarks are intended to exercise the CPU, the memory hierarchy, and the compilers. Since we intended to study the behavior of the entire memory hierarchy, a set of benchmarks from SPEC CPU 2000 suite were used in our experiments. Seven benchmarks from SPEC2000 integer suite were selected, which are bzip2, gzip, gcc, mcf, parser, twolf, and vortex. And, a selection of two benchmarks from floating-point suite, ammp and mgrid, are also used in the experiments. They were compiled by gcc with static libraries on a Linux host system. Then, the binary files of the benchmarks and their input files are installed on a Redhat Linux disk image. Inputs for all benchmarks are reference inputs.

SPEC CPU2000 is the next-generation industry-standardized CPU-intensive benchmark suite. SPEC designed CPU2000 to provide a comparative measure of compute intensive performance across the widest practical range of hardware. The implementation resulted in source code benchmarks developed from real user applications. These benchmarks measure the performance of the processor, memory and compiler on the tested system. The data collected show that SPEC met its goals for memory footprint: most benchmarks are larger than common cache sizes, many are larger than 100MB, and none are larger than 200MB. Further details of the selected benchmarks can be found in the appendix of this dissertation.

5.6. Interactions

One of the main contributions of SYSim is to ensure the correct interactions between the components in the system at the epoch level. These interactions include:

- the interaction between the processor and L1 cache
- the interaction between multiple levels of the caches
- the interaction between the last level of the caches and DRAM
- the interaction between processor and disk via I/O requests
- the interaction between the disk and DRAM via DMA.

Figure 5.4 shows the interaction among the processor, caches, and memory for a Load instruction. For the interaction of the processor, L1 cache, multilevel of the caches, we implement them as modeled in SimpleScalar/Wattch. When the processor fetches an instruction, or executes a load or a store command, a memory request is generated. The page-frame portion of the requesting address is translated by the operating system via TLB, while the index portion from the page offset is sent to the L1 instruction cache. As we viewed the processor and the operating system as a black box that generates memory access requests and I/O interrupts, we assume that the TLB translates the page-frame address, except on a page fault. A page fault will be taken care of by the operating system. With the translated physical page-frame address and the page offset, the L1 cache decodes the translated physical address into set, tag, and offset. Then, the set is chosen, and the tag portion of the cache is accessed, and compared. If it is a hit, then the proper bytes of the

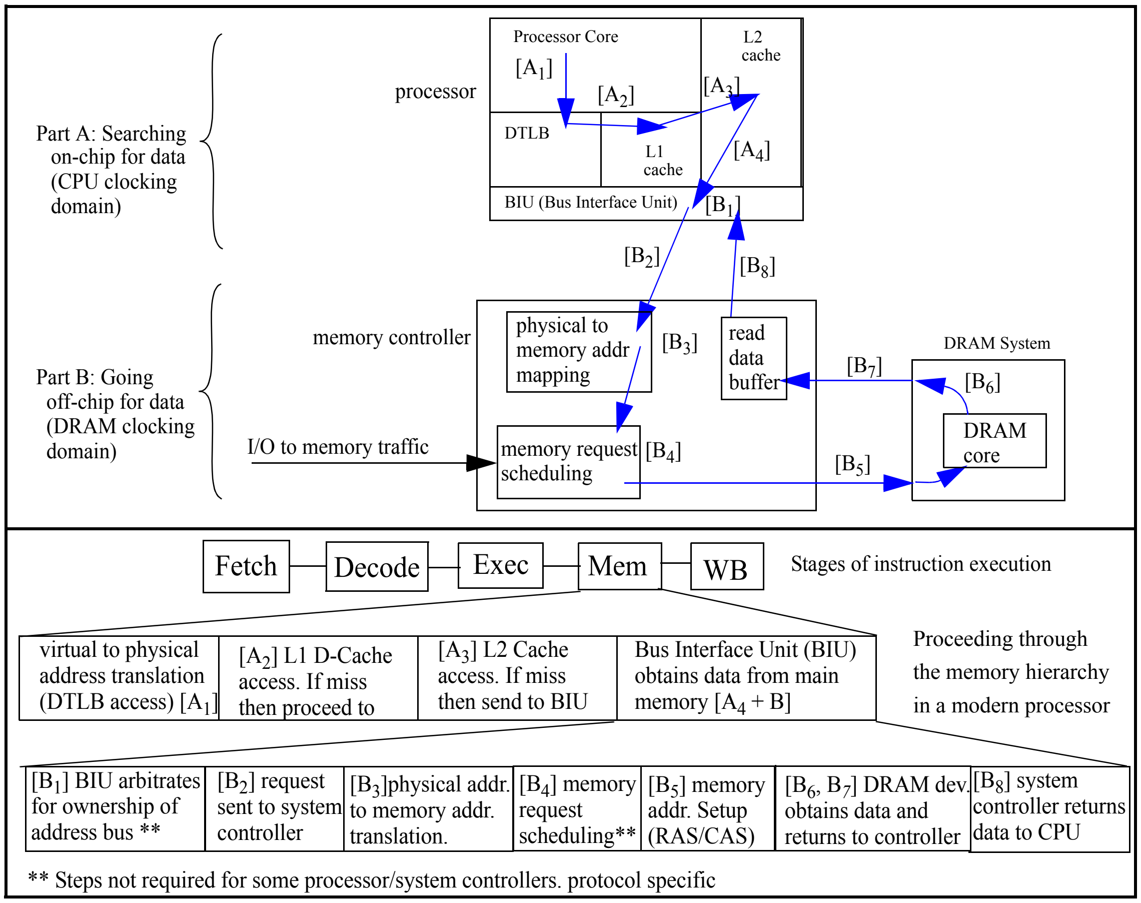

Part A: Searching on-chip for data (CPU clocking domain)
processor
Processor Core
L2 cache
[A1]
[A2]
[A3]
DTLB
L1 cache
[A4]
BIU (Bus Interface Unit)
[B1]
[B2]
[B8]
Part B: Going off-chip for data (DRAM clocking domain)
memory controller
physical to memory addr mapping
[B3]
read data buffer
[B7]
DRAM System
[B6]
DRAM core
I/O to memory traffic
memory request scheduling
[B4]
[B5]
Fetch
Decode
Exec
Mem
WB
Stages of instruction execution
virtual to physical address translation (DTLB access) [A1]
[A2] L1 D-Cache access. If miss then proceed to
[A3] L2 Cache access. If miss then send to BIU
Bus Interface Unit (BIU) obtains data from main memory [A4 + B]
Proceeding through the memory hierarchy in a modern processor
[B1] BIU arbitrates for ownership of address bus **
[B2] request sent to system controller
[B3]physical addr. to memory addr. translation.
[B4] memory request scheduling**
[B5] memory addr. Setup (RAS/CAS)
[B6, B7] DRAM dev. obtains data and returns to controller
[B8] system controller returns data to CPU
** Steps not required for some processor/system controllers. protocol specific

Figure 5.4: : Abstract Illustration of a Load Instruction in a Processor-Memory System [17].

block are furnished to the processor using the lower bits of the page offset, and the instruction stream access is done.

If it is a miss from L1 cache, the L2 cache is accessed. Like in L1 cache, the physical address is decoded to tag, set, and offset portion, and the L2 cache accesses the block. If it is a hit, the proper bytes (the size of L1 cache block) of the block are sent to the L1 cache. Then, the L1 cache chooses a block to be replaced, depending on the replacement policy, writes the replaced block to the appropriate block in L2 cache, and writes the missed block from L2 cache to the chosen block in L1 cache.

On the other hand, if a miss occurs in L2 cache, the request is sent either to L3 cache, if exists, or to the DRAM memory system. If the request must go to DRAM memory, the request will be put in the Bus Interface Unit (BIU) request queue inside the DRAM simulator. After the appropriate BIU entry has been selected, the status of the BIU entry is marked as SCHEDULED, then a memory transaction is created in the memory transaction queue. The transaction is broken into a series of appropriate row activation, column read/write, and precharge commands depending on the status of the accessing bank. After the commands are issued, the DRAM returns the most critical data (the size of memory bus) with respect to the DRAM timing specification, and the rest of the data is sent to the replaced block of the last-level cache until filled. Figure 5.4 shows how a processor and memory system, excluding disk effects, interact in a load instruction.

For the actual implementation in SYSim, the integrated cache and DRAM models do not contain any data; the data are obtained from the memory array in Bochs simulator. The integrated cache and DRAM only return the latency for each request and update the status of the components. The value of the returned latency includes only the time spent until the first chunk (critical word) of data returns, excluding the time returning the rest of the data.

On the other side of the operating system, when a page fault occurs, the operating system issues a command to transfer the new page from the disk to memory. As a disk request often involves block transfers, direct memory access (DMA) hardware is added to many computer systems to allow transfers of numbers of words without intervention by the CPU. DMA is a specialized processor that transfers data between memory and an I/O device while the CPU goes on with other tasks. Thus, it is external to the CPU and must act as a master on the bus. The CPU first sets up the DMA registers, which contain source and destination memory addresses and number of bytes to be transferred. Once the DMA transfer is complete, the controller interrupts the CPU. There may be multiple DMA devices in a computer system; for example, DMA is frequently part of the controller for an I/O device.

A newer protocol for the ATA/IDE interface is Ultra DMA. The key technological advance introduced to IDE/ATA in Ultra DMA was double transition clocking. Before Ultra DMA, one transfer of data occurred on each clock cycle, triggered by the rising edge of the interface clock (or strobe). With Ultra DMA, data is transferred on both the rising and falling edges of the clock. Double transition clocking, along with some other minor changes made to the signaling technique to improve efficiency, allowed the data throughput of the interface to be doubled for any given clock speed.

The actual implementation of DMA and Ultra DMA relies on the timing in Bochs. Bochs already implements DMA and the interaction with memory, but no timing is updated. By inspection from Bochs, a disk request causes the disk to read a sector from the disk to the disk buffer. Then, the data is sent to the memory 2 bytes at a time as the IDE/ATA interface is two bytes (16 bits) wide. We need to consider the transfer configuration, i.e. the frequency of the bus, what is the type of the interface, etc., to calculate the data transferring latency. For example, if we wish to transfer a 512-byte sector with DMA, the latency is equal to

$$latency = \frac{512B}{2BperClock \times BusFrequency},\text{ or}$$

$$latency = \frac{512B}{4BperClock \times BusFrequency}\text{ with Ultra DMA.}$$

After DMA completes transferring the data to the memory controller, the controller generates a transaction corresponding to the data from DMA. Then, the transaction is scheduled to the DRAM system when appropriate. Finally, the memory pages, corresponding to the data existing in any caches, are invalidated as the preparation for the data to be loaded to the caches later on.

In SYSim, we run all applications in single-user mode to make accurate calculation of execution time. Otherwise, the kernel would swap to other processes on a read() system call. Therefore, disk delay shows up as stall time. On the other hand, a write() system call returns to user code as soon as the data is transferred into kernel space. Instead of writing through to the disk system directly and waiting for the write to finish, the operating system buffers the writes in the memory and returns the control to the user application immediately. The operating system issues a long burst of buffered writes to the disk system periodically at later time. As a result, the disk read requests behave like blocking requests, and disk writes behave like non-blocking requests. Additionally, the way we implemented the interface is very straightforward as DiskSim allows us to specify the bus latency for transferring one sector. Therefore, in the experiment, we only varied the bus latency to see the differences between the different types of interfaces.

5.7. Parameter and Benchmark Selections

All the parameters using in the experiments are shown in the table below. We have chosen the parameters to suit the benchmarks which is SPEC2K, but still reflects the modern computer systems as the same parameters are still used in recent publications. The cache structure consists of a level-1 instruction cache, a level-1 data cache, and a level-2 unified cache. We use 0.10 micron-technology for the cache as defined in Wattch. The memory type is DDR SDRAM with one channel of an eight-byte wide bus. The DRAM parameters are set according to the datasheet of DDR SDRAM 128Mb chip from micron website. Table 5.2 shows the base configuration for the CPU, caches, DRAM, and Disk in our experiments. If not specified, the parameters are set according to the table.

For the benchmarks, since we focus on the entire memory hierarchy affected by the changed in parameter settings, we are using a subset of SPEC2K benchmarks. However, after a preliminary experiment showing in the next chapter, all benchmarks can be characterized into 2 categories; first is the benchmarks that show memory page swapping behavior due to not enough memory. This type of benchmark has both disk reads and writes. Second is the benchmarks that have no page swapping; therefore, only a series of consecutive disk reads are exhibited. As a result, we choose only bzip2 and ammp with different main memory sizes to represent both categories of behaviors. Since we focus on the I/O intensive phase of the execution, we run only the first 500 million instructions, which is disk-intensive.

CPU Parameters	
CPU speed	2GHz
L1-Icache	64kB; 64B linesize; 4-way with LRU repl.; lat = 1
L1-Dcache	64kB; 64B linesize; 4-way with LRU repl.; lat = 1
L2-cache	512kB; 128B linesize; 4-way with LRU repl.; lat = 6
cache technology sizing	0.10 micron
Memory Parameters	
memory type	DDR SDRAM
memory data rate	400 MHz
memory channel count	1
memory channel width	8 bytes
memory rank count	1
memory bank count	4
memory row count	4096 (for 128MB)
memory column	1024 (for 128MB)
DRAM chip density	128 Mb (for 128MB)
DRAM chip VDD	2.6 V
DRAM chip IDD0	155 mA
DRAM chip IDD2P	5 mA
DRAM chip IDD2F	55 mA
DRAM chip IDD3P	45 mA
DRAM chip IDD3N	60 mA
DRAM chip IDD4R	190 mA
DRAM chip IDD4W	195 mA
DRAM chip IDD5	11 mA

Table 5.2: Base Configuration for CPU, caches, and memory

DRAM chip's power per DQ	6.88 mW
Disk Drive Parameters	
Disk parameters	5400, 12000, 20000 with 4MB of disk cache
RAID stripe size	16KB
Disk sector size	512 bytes

Table 5.2: Base Configuration for CPU, caches, and memory

The disk parameter files are taken from DRPM paper. We choose 5400 RPM (or 5k) to represent yesterday's disk, 12000 RPM (or 12k) to represent today's disk, and 20000 RPM (or 20k) to represent tomorrow's disk. However, for the power consumption, after several disk drives currently available in the market have been surveyed, the power consumption for idle mode and active mode of 22Watts and 39Watts as specified in DRPM paper are no longer reasonable. Figure 5.5 below shows the plot of the RPM versus the idle power and the active power of 47 commercially available disk drives. The figure shows the idle power values are in the range of 5-16 Watts and the active power values are in the range of 7-23 Watts, while the RPM is between 7,200RPM and 15,000 RPM. No disk drive has the power consumption over 25 Watts. The reason might be the changes in disk drive technology which are increasingly moved toward low power venue in the past few years making the power consumption of a disk drive in real life diverge from the one described a few years back by DRPM paper. Therefore, a selection of more reasonable idle and active power consumption has to be chosen.

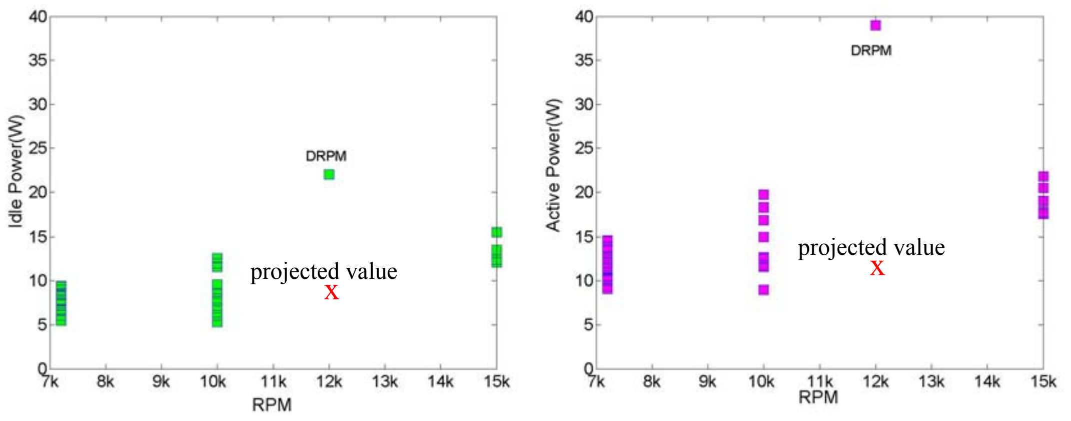


Figure 5.5: Idle and Active Power of 47 Commercially Available Disk drives. The figure on the left is for Disk Idle Power and RPM, and the figure on the right is for Disk Active Power. The data point marked as DRPM is the Idle and Active Power from DRPM paper. Obviously, the idle and active power from the DRPM paper are too high from the power numbers of the commercially available disks. The figure also shows our projected values used in the dissertation.

In this dissertation, since the disk speed that we study is either obsolete (5400 RPM) or not commercially available (12,000 RPM and 20,000RPM), we employ the following technique to project the power consumption. Son and Kandemir [91] suggested curve fitting method to estimate the idle and active power of a disk drive for a particular RPM. They collected several pairs of RPM and power consumption from commercially available disk drives, and projected them on a linear curve fitting. From the power consumption of the multi-speed disk drive shown in their paper, we use their linear curve fitting to project the idle power and active power for 5400, 12000, and 20000 RPM. The equation used in the curve fitting is below:

$$Pidle = (0.51 \times (RPM) / 1000) + 2.5$$

$$Pactive = (0.73 \times (RPM) / 1000) + 2.5$$

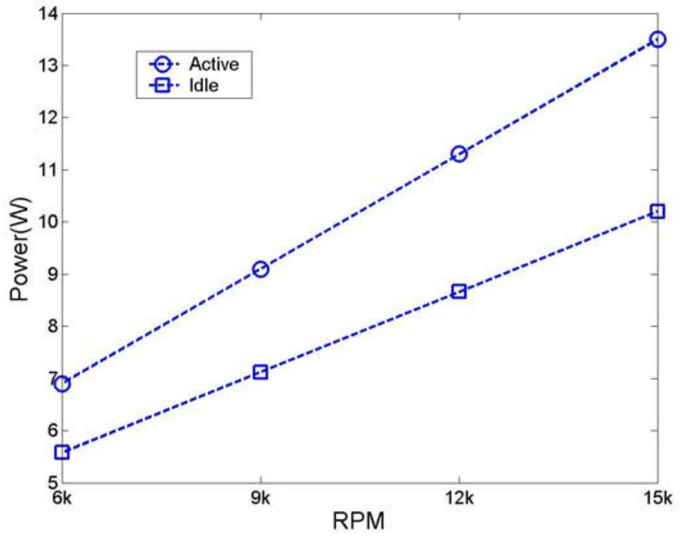


Figure 5.6: Son and Kandemir's Disk Power Projection for IBM Ultrastar 36Z15. The figure shows linear relationship between the power and the RPM as used in their experiments. This relationships reflect better representative values for the active and idle power for a currently available disk drive.

As a result, we use the following power consumption for 5400, 12000, and 20000-RPM disk drives in our experiment as show in the table 5.3:

RPM	Active Power (W)	Idle Power (W)
5,400	6.442	5.254
12,000	11.26	8.62
20,000	17.1	12.7

Table 5.3: Disk Active and Idle Power Values

For the RAID5 settings, we consider the configurations as shown in the figure 5.7 below. We conducted the experiments with 2 configurations of 4-disk RAID system which are (1) 2 controllers with 2 disks each or "2c x 4ds", and (2) 4 disks connected to only one controller or "4ds". For the 8-disk RAID system, we have 3 configurations, which are (1) 2 controllers with 4 disks each or "2c x 4ds", (2) 4 controllers with 2 disks each or "4c x 2ds", and (3) one controller with 8 disks or "8ds".

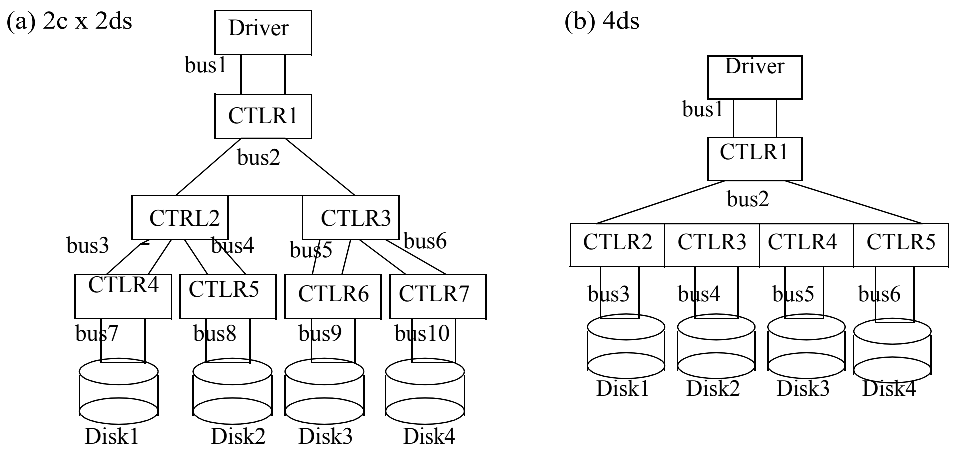


Figure 5.7: (1) RAID5 Configuration for an 4-disk system. The figure shows (a) 2 controllers with 2 disks each (2c x 2ds), and (b) one controller with 4 disks connected (4ds). Note that each disk also has its own controller.

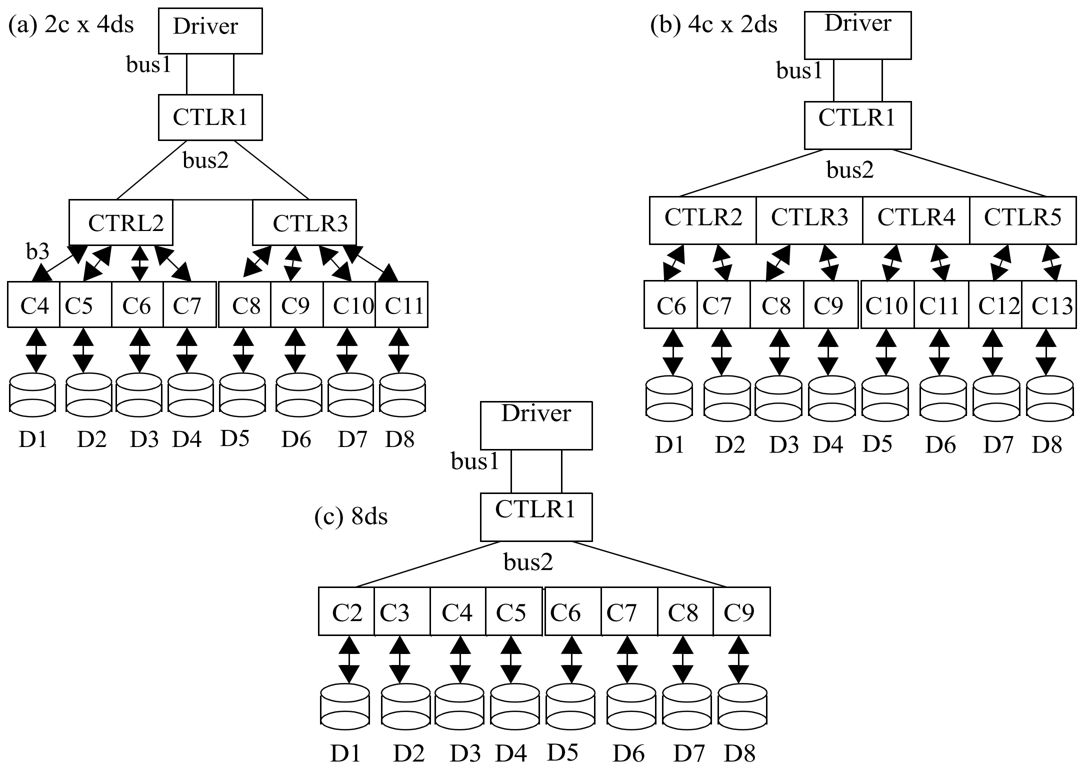


Figure 5.7: (2) RAID5 Configuration for an 8-disk system. The figure shows (a) 2 controllers with 4 disks each (2c x 4ds), (b) 4 controllers with 2 disks each (4c x 2ds), and (c) one controller with 8 disks (8ds). Note that each disk also has its own controller. Due to the limitation in space, a bus is reduced to a 2-head arrow, and the name of the bus is omitted.

5.8. SYSim and Real Systems Comparison

We compared the execution time breakdown results obtained from SYSim to the results obtained from a set of real systems. We ran SPEC's gzip on three different machines. The real system configurations are set to be comparable to SYSim configurations. Due to the limitation of the availability and compatibility in hardware, we compare SYSim with a set of available machines with comparable configuration, but not with the exact configuration. The first system has a 750MHz CPU with 96MB of the system memory and runs Fedora Core 3. The execution time breakdown for the first system is shown in Table 5.4.

Run #	User (s)	Kernel (s)	I/O stall (s)	Total (s)
1 (cold cache)	93.11	15.06	600.83	709
2 (warm cache)	92.7	16.3	397.00	506
3 (warm cache)	92.8	14.3	425.90	533
4 (warm cache)	93.3	14.3	460.40	568
5 (warm cache)	93.6	14.3	441.10	549

Table 5.4: Execution Time Breakdown for System #1: 750MHz CPU with 96MB memory

The second system is the same system as the first system, but the system memory is set to 128MB. The second system is comparable to a SYSim system configured with a 2GHz CPU and 128MB of the system memory. Even though the second system has a CPU of 750MHz, the CPU is an out-of-order core, which is approximately comparable to a 2 GHz in-order core in SYSim. Therefore, SYSim execution time statistics are also shown at the end of Table 5.5 for comparison.

Run #	User (s)	Kernel (s)	I/O stall (s)	Total (s)
1 (cold cache)	90.4	6.4	164.20	261
2 (warm cache)	90.1	6	126.90	223
3 (warm cache)	89.8	5.7	129.50	225
4 (warm cache)	90.5	5.5	121.00	217
5 (warm cache)	90.3	6.1	168.60	265
SYSim System Run	User and Kernel (s)		I/O stall (s)	Total (s)
run# 1	27.8		135.2	162.8

Table 5.5: Execution Time Breakdown for System #2: 750MHz CPU with 128MB of memory comparing with a SYSim system with 2GHz CPU with 128MB of memory

The first and the second real systems are the same system with only 32MB different in the memory size. However, the total execution time of both systems are as different by the factor of 3 and the I/O stall times are as different by the factor of 5. This result shows that the I/O effect exists in the real system. In Table 5.5, the execution time breakdown in both real system and SYSim are comparable.

The third system has a 2.4GHz CPU with 1GB of the system memory also running Fedora Core 3. The third system is to be compared with a SYSim system configured with a 2GHz CPU with 512MB of the system memory. Though the SYSim system has less in both processor frequency and memory size, our experiment results in the next chapter show that any systems running SPEC's gzip with any size of memory larger than 160MB will not cause any differences in the total system performance. Therefore, the memory size of 1GB or 512MB will perform similarly in this case. The execution time statistics in both actual system and SYSim system are shown in Table 5.6. Notice, the execution time breakdown in

Run #	User (s)	Kernel (s)	I/O stall (s)	Total (s)
1 (cold cache)	20	0.19	27.8	48
2 (warm cache)	20	0.19	19.8	40
3 (warm cache)	20	0.19	17.8	38
4 (warm cache)	20.1	0.20	18.7	39
5 (warm cache)	20	0.19	21.0	41.2
SYSim System Run	**User and Kernel (s)**		**I/O stall (s)**	**Total (s)**
run #1	27.8		33.1	60.9

Table 5.6: Execution Time Breakdown for System #3: 2.4GHz CPU with 1GB of memory comparing with a SYSim system with 2GHz CPU with 512MB of memory

both real system and SYSim system are also very similar in this case.

5.9. Sample Output

Figure 8-11 show the graphs which are generated from the sample output of SYSim. The system configuration is as described in Table 5.2, with 128MB of memory, a single 12k-RPM disk drive with disk cache. The system ran gzip to completion. Figure 5.8 shows the Sample Output of Cache Accesses and Total system CPI. The figure shows 4 graphs, which are (1) instruction cache accesses per 10 milliseconds, (2) data cache accesses per 10 ms (3) level-2 unified cache accesses per 10 ms, and (4) the total system CPI per 10 ms along with the accumulated system CPI. In the last graph, the duration with no data point means that there is no instruction executed.

Figure 5.9 shows the Sample Output of Cache miss rate and Disk Accesses. Figure 5.9 shows 4 graphs, which are (1) the miss rate of the instruction cache, (2) the miss rate of the data cache, (3) the miss rate of level-2 unified cache, and (4) the disk accesses per 10ms. Figure 5.10 shows the Sample Output of Cache power and Disk Power Dissipation. The figure shows 4 graphs, which are (1) the instruction cache power, (2) the data cache power, (3) the level-2 unified cache power, and (4) the disk power per 10 ms. All power dissipation values are in Watts.

Finally, Figure 5.11 shows the Sample Output of DRAM and Disk Accesses and Power Dissipation. The figure shows 4 graphs, which are (1) DRAM Power, (2) DRAM Accesses per 10 ms, (3) Disk Power, and (4) Disk Accesses per 10ms. The duration having no data point means that there are no accesses. All graphs share the same x-axis which is the execution in milliseconds, and each data point is the collection of average value over 10 milliseconds.

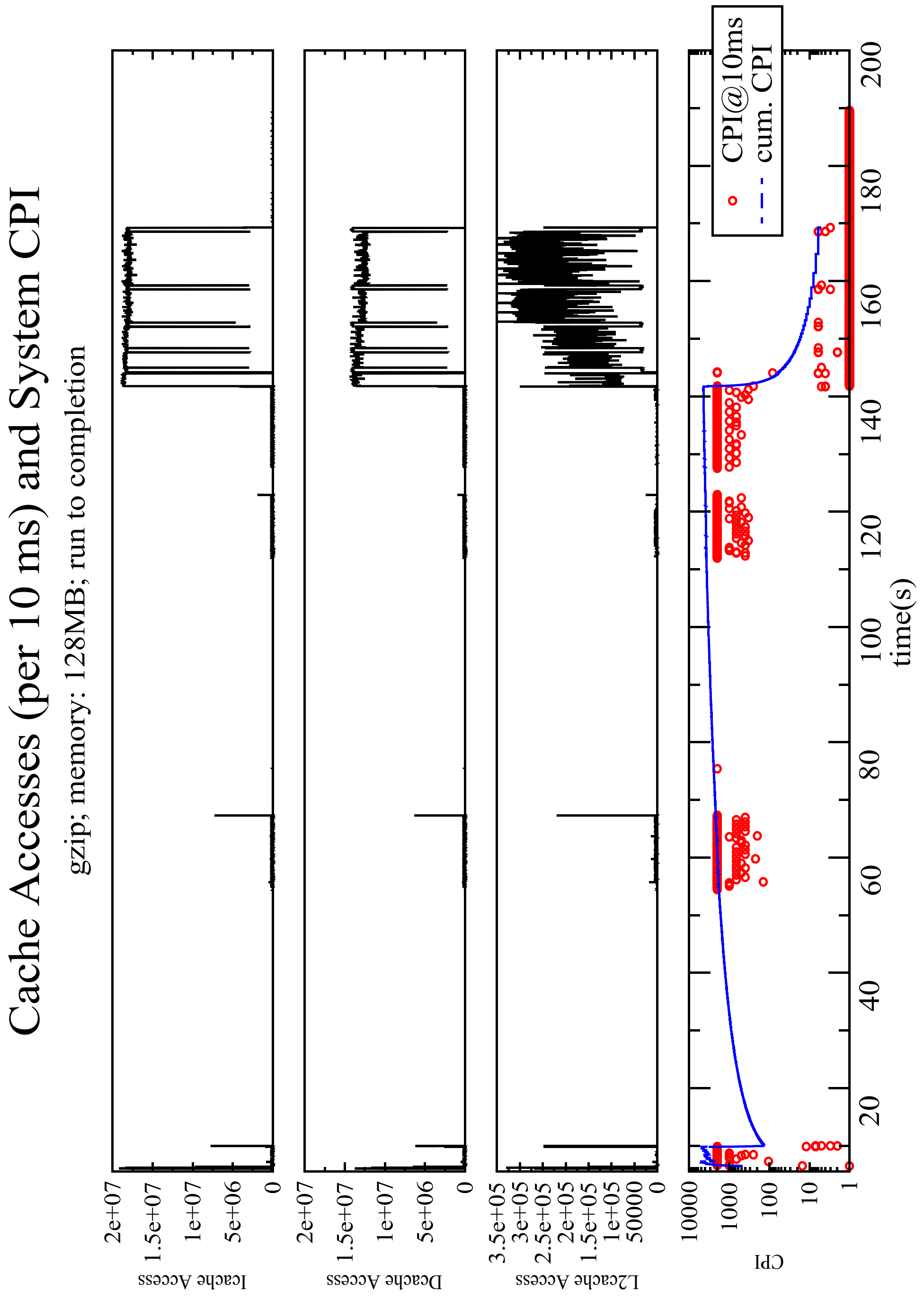


Figure 5.8: Sample Output of Cache Accesses and Total system CPI. The figure shows 4 graphs, which are (1) instruction cache accesses per 10 milliseconds, (2) data cache accesses per 10 ms (3) level-2 unified cache accesses per 10 ms, and (4) the total system CPI per 10 ms along with the accumulated system CPI. The duration having no data point mean there is no instruction executed.

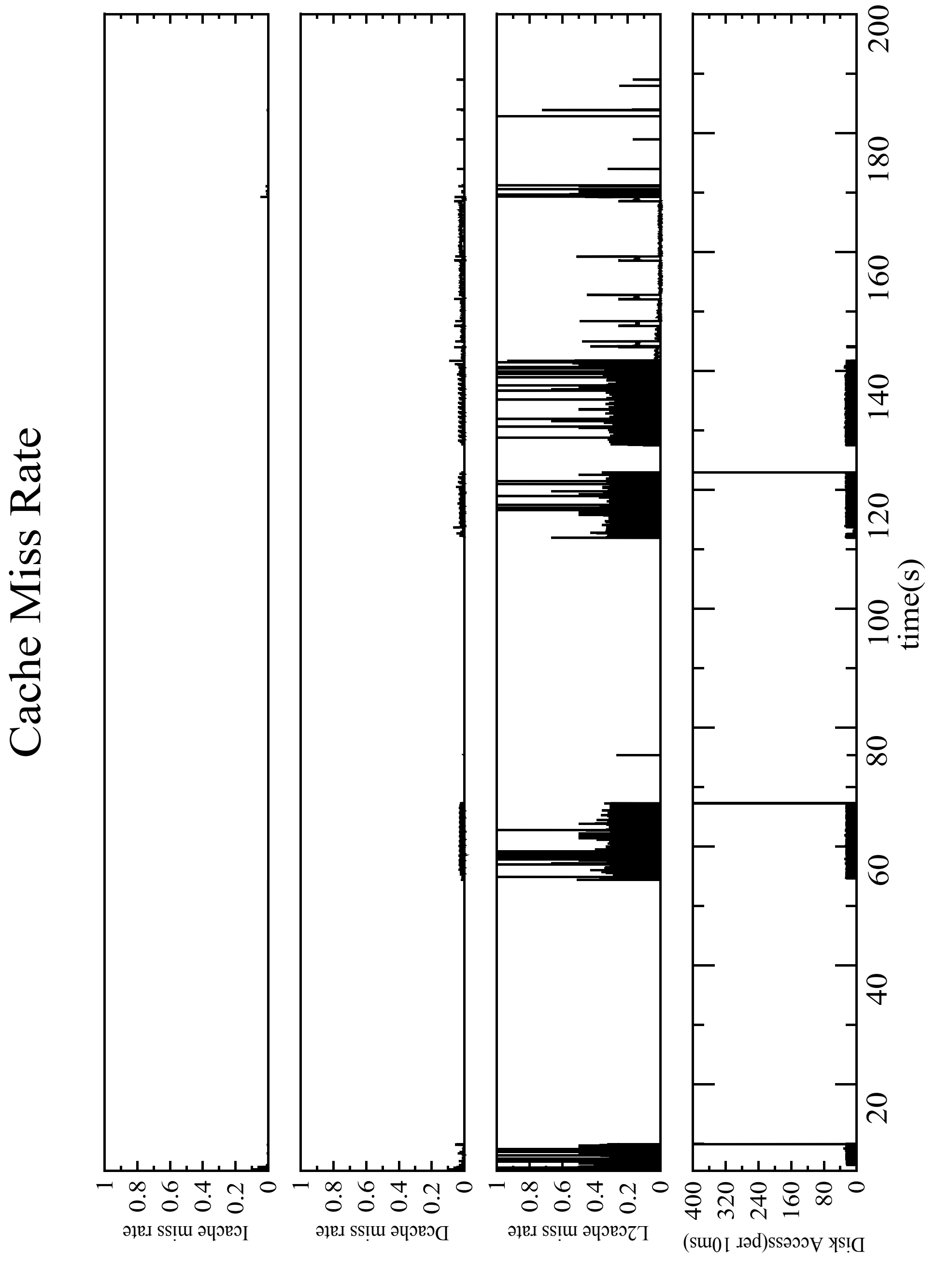


Figure 5.9: Sample Output of Cache miss rate and Disk Accesses. The figure shows 4 graphs, which are (1) the miss rate of the instruction cache, (2) the miss rate of the data cache, (3) the miss rate of level-2 unified cache, and (4) the disk accesses per 10ms.

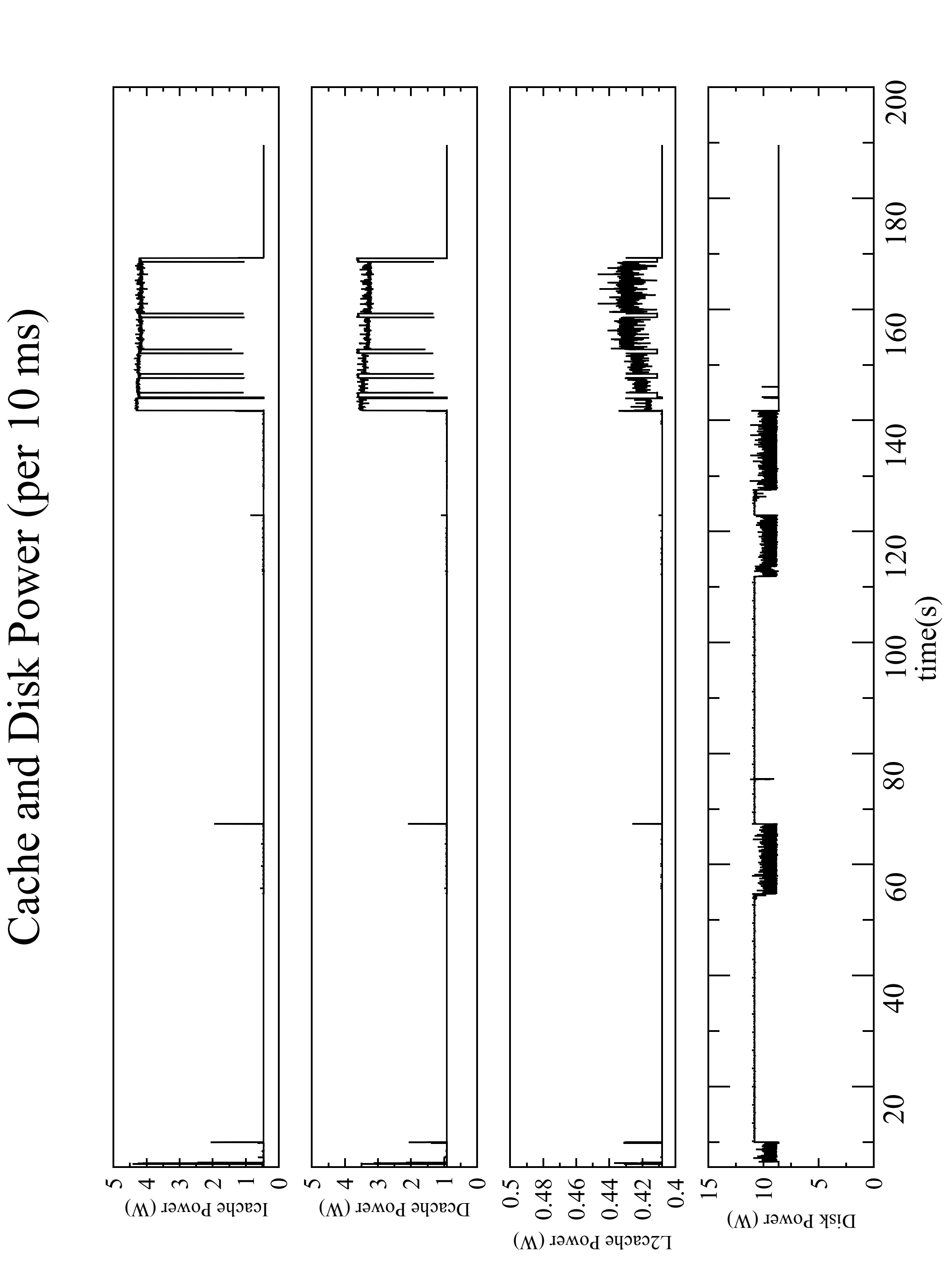


Figure 5.10: Sample Output of Cache power and Disk Power Dissipation. The figure shows 4 graphs, which are (1) the instruction cache power, (2) the data cache power, (3) the level-2 unified cache power, and (4) the disk power per 10 ms. All power dissipation are in Watts.

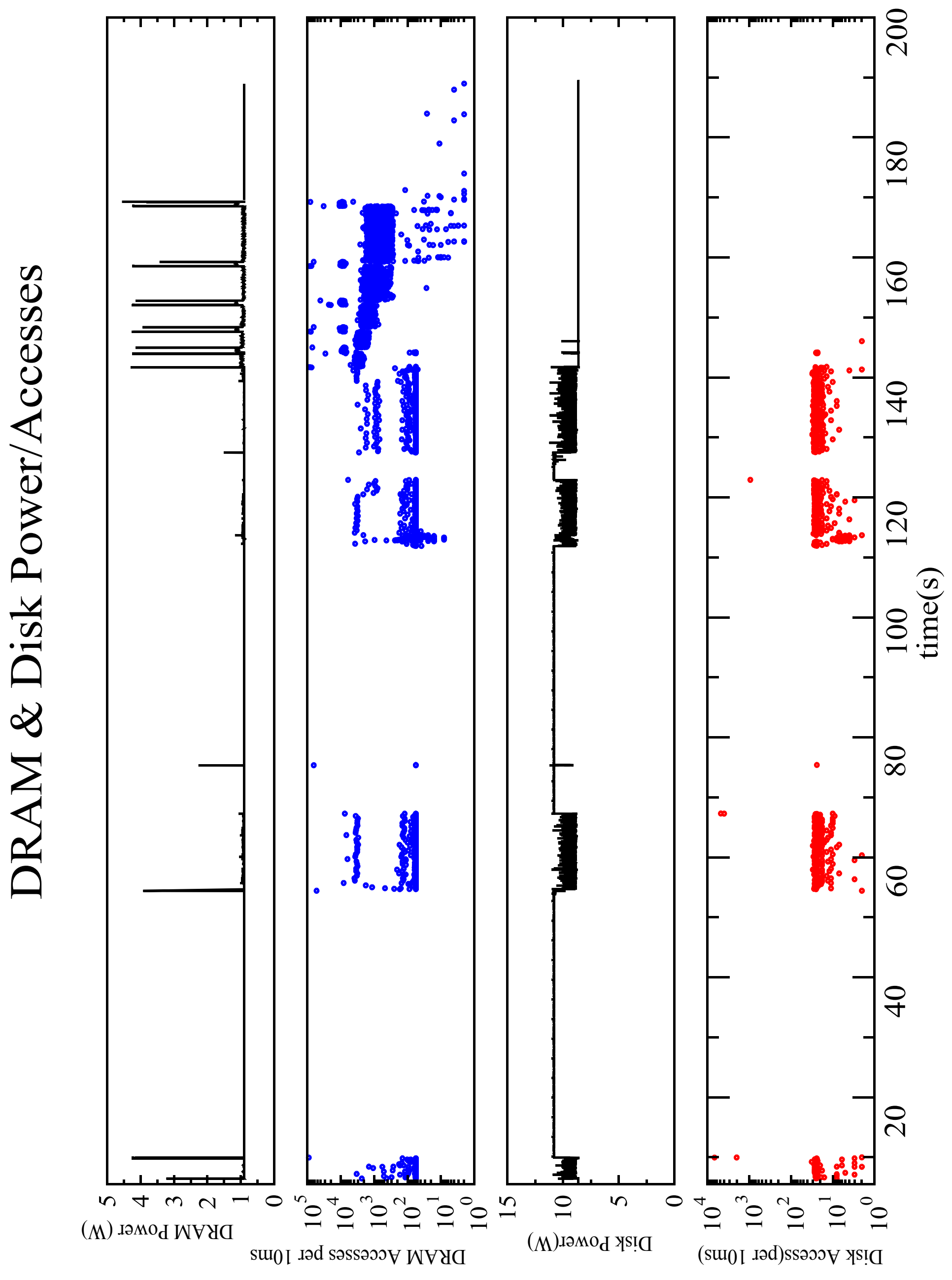


Figure 5.11: Sample Output of DRAM and Disk Accesses and Power. The figure shows 4 graphs, which are (1) DRAM Power, (2) DRAM Accesses, (3) Disk Power, and (4) Disk Accesses per 10ms. The duration having no data point mean there is no accesses.

CHAPTER 6: EXPERIMENTAL RESULTS

This chapter discusses the results from the experiments that utilized SYSim to investigate the system-level behaviors during the I/O intensive phase of an execution. Most applications spend a significant amount of the time, if not most, in the I/O intensive phase due to the I/O activities. During the I/O intensive phase, the other components in the memory hierarchy cause only very little activities. We conducted the experiments during the I/O-intensive phase, which tends to be within the first 500 million instructions of the execution. This chapter presents the impact of the variations in system memory size settings and disk design space having on total system performance and power. The experimental results are shown in terms of both total system performance and power/energy consumption.

6.1. I/O intensive phase

As we discussed in the introduction, an application tends to spend a significant amount of time during the I/O intensive phase. Again, Figure 6.1 shows the interaction of memory hierarchy components during the entire execution of gzip on our complete-system simulator--SYSim, while in a single user environment. The system configuration used in this example is a 2-GHz Pentium processor, 128MB of main memory, and a 12k-RPM disk drive with built-in disk cache. The other system configuration is as described in Table 5.2. Figure 6.1 includes graphs displaying comparisons between cache accesses and system CPI, all cache power, and DRAM and disk access/power. The system CPI is shown in both 10ms-

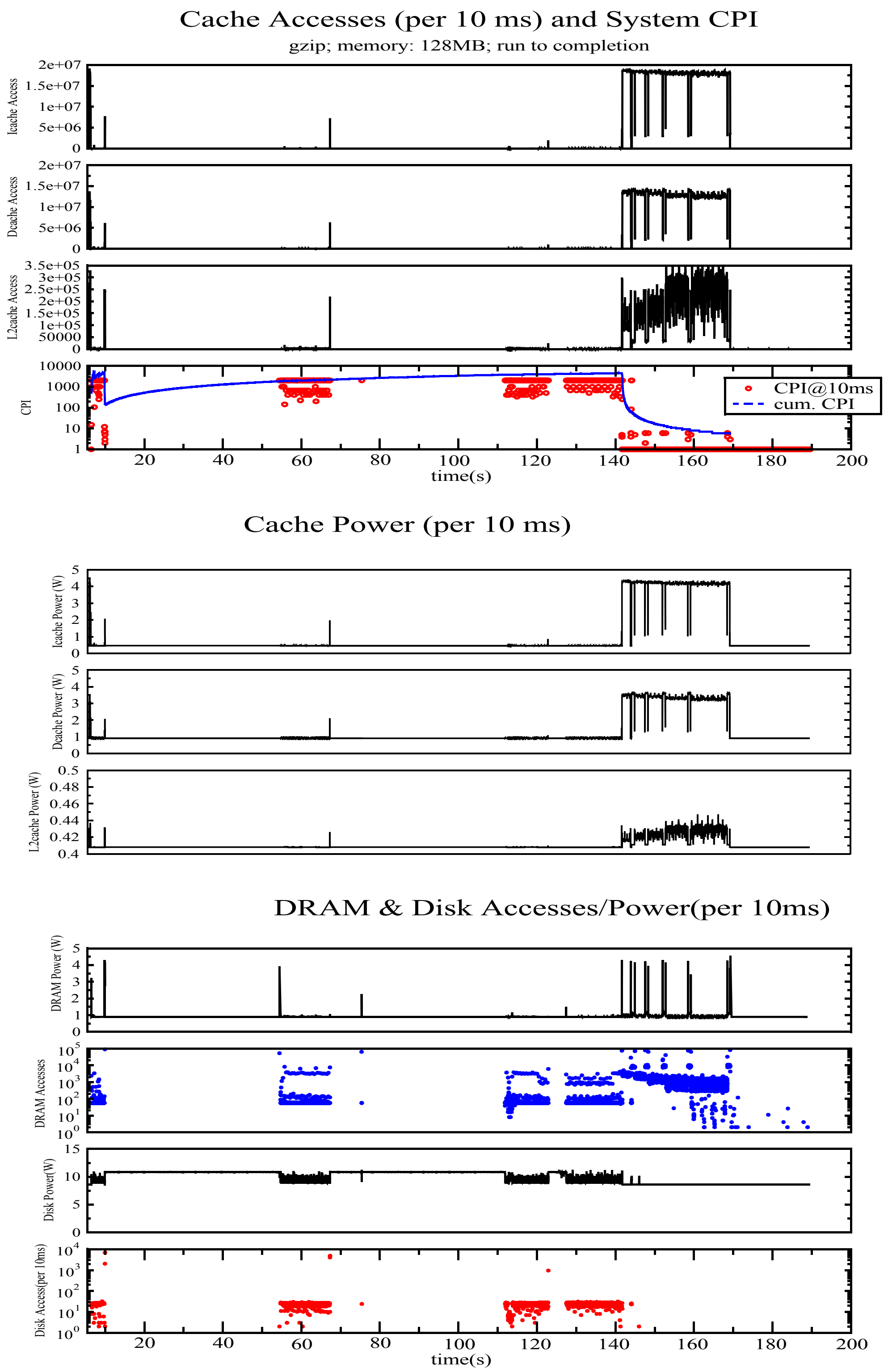


Figure 6.1: The System CPI. The figure shows the System CPI over the entire run of gzip. The system configuration is a 2-GHz processor with 128MB of memory and a 12k-RPM disk. The CPI graph shows 2 CPI values: one is the instant CPI for every 10ms, another is the accumulated average CPI. The duration having no data point means no instructions are executed due to the I/O latency. The course of execution when the accumulated CPI is over 100 is the I/O intensive phase, and the course of execution when the CPI is below 100 is the computation phase.

epoch average and total accumulated average. Accesses and power of level-1 instruction cache, level-1 data cache, and level-2 unified cache are all illustrated. All graphs use the same x-axis, which represents the execution time in seconds. The x-axis does not start at zero since the system boot time is excluded. Figure 6.1 is the same as Figure 5.8 to 5.11, the sample output in Chapter 5.

The figure demonstrates different phases of execution: the I/O intensive phase and the computation phase. During the I/O intensive phase, the operating system reads the program and required data from the disk and allocates the memory pages for them. The caches are mostly idle, and the DRAM is sporadically written into. On the other hand, the disk is actively accessed. From the figure, the I/O intensive phase is from the start of the execution until the 140th second. Since the application was run in a single user mode, the disk access delay causes stall time in the execution. One notices long periods of disk activity, when the disk power is at its maximum followed by periods of disk bursts, i.e. from the 10th second to the 50th second and the 70th to the 110th second. These long disk activity periods are the result of write bursts caused due to write buffering performed by the file system management. Since perfect disk-side write buffering mechanism is not implemented in this system, the long write bursts have to be processed immediately to prevent the data discrepancy from any failures. The latency of this long period depends heavily on the memory page swapping algorithm used by the operating system. More page swapping means more write data to be buffered in the main memory due to pages swapped out, and longer write bursts to be scheduled to the disk. In this configuration, due to the large memory footprint of the application (as much as 180MB [13]), the operating system swaps out numerous memory pages. The write data buffered in the file system are periodically sent to the disk. If these write bursts are scheduled before the disk reads in a single user mode, the write bursts would prolong the execution time since the reads have to wait for the write bursts to be completed.

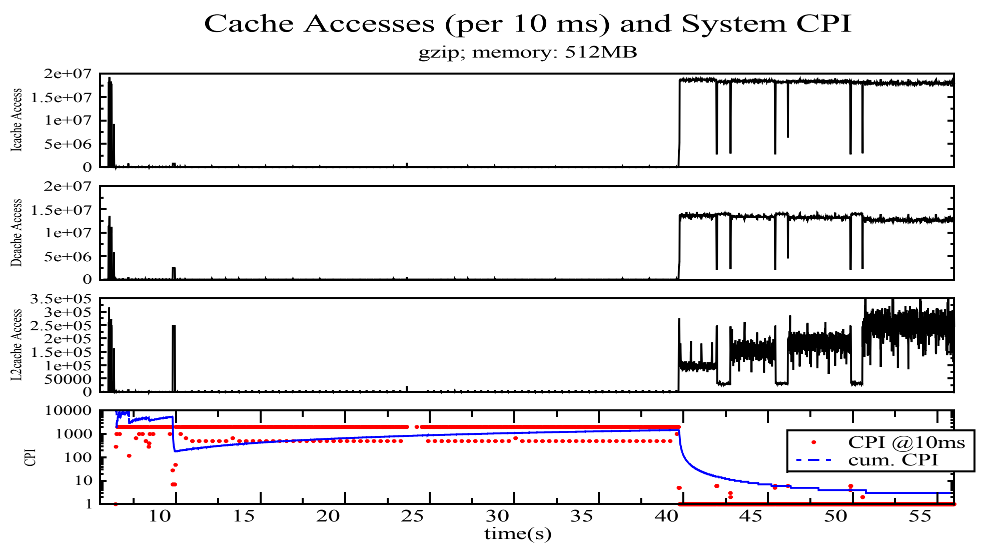
The second phase is the computation phase. During the computation phase, the caches and DRAM are accessed regularly while the processor read instructions and data from the caches and executes them. In this execution phase, the disk is rarely accessed since most of the required code and data are already loaded in the memory. In other applications, there might be disk accesses due to a larger memory footprint. The figure also illustrates a number of disk accesses during the computation phase, but these accesses have no performance impact. This is because these accesses are periodic disk write bursts, which are the results from write buffering under the file system. These write bursts would not lower system performance unless there are reads scheduled after the bursts.

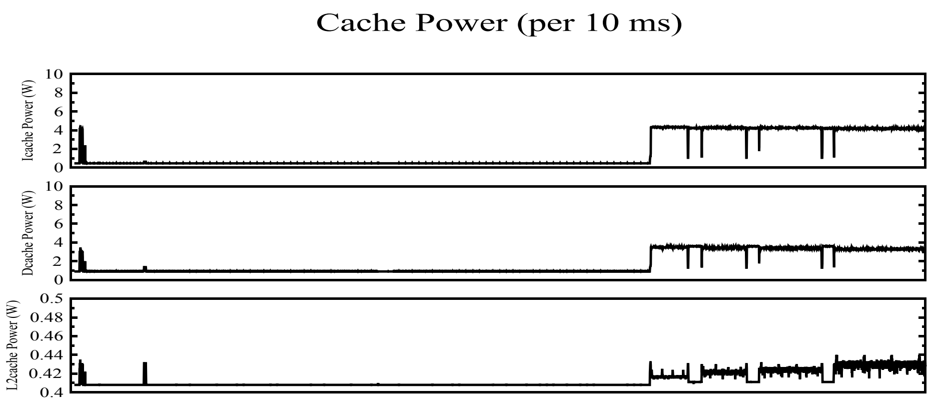
The last phase of execution, is often an I/O output phase. After the computation, an application would output the results to I/O, which can be the computer screen or a file. However, since SPEC's version of gzip performs only reads from I/O for input, but no file I/O for output, the figure does not demonstrate this I/O output phase.

As Figure 6.1 shows, the CPI value can vary dramatically, i.e. by many orders of magnitude, due to the I/O activities during long I/O intensive phase. CPI finally reduces to a single-digit number during the computation phase as observed in previous studies. If the CPI is calculated from the entire execution, the average CPI would be just under 10. However, as many researchers only concentrate on the computation phase, they claimed the final CPI number to be around 1. This misconception is mainly caused by excluding the I/O intensive phase. The final results would be an inaccurate average CPI and incorrect execution time estimation. Therefore, the I/O intensive phase is truly important to the entire execution of an application.

Let's have a look at what if we increase the memory size to the point where is no paging in the system. Figure 6.2 shows the results of the same system with 512MB running gzip. The I/O intensive phase is much shorter, but it remains a significant portion of the entire execution time. Though the I/O intensive phase is much shorter, write bursts remain, i.e. at the 25th and the 40th second. Therefore, even without the memory paging in the system, the disk request stream is composed of both read and write requests. As a result, the problem remains even in a system equipped with disk prefetching, so simple prefetching data from the disk is not a solution.

Figures 6.3 to 6.11 show the executions during the I/O intensive phases of all nine benchmarks used in the experiments: ammp, bzip2, gcc, gzip, mcf, mgrid, parser, twolf, and vortex. Again, all graphs are the results of the configurations of 128MB of memory with a single 12k-RPM disk equipped with disk cache. All results are shown for the first 500 million instructions. Obviously, all benchmarks demonstrate disk-intensive behavior during the execution: the disk is actively accessed and prolongs execution time due to the long latency. On the other hand, the number of cache and DRAM accesses are minimal, compared with the number of cache and DRAM accesses during the computation phase. Though some applications actively access the cache and DRAM, the numbers of accesses are not as high as during the computation phase, for example, ammp and parser. For mcf, system initialization ends after 10 seconds. During initialization only the disk is accessed. Following the I/O intensive phase, all memory components are actively accessed. Caches and DRAM accesses are scattered periodically because of long latency of the I/O between





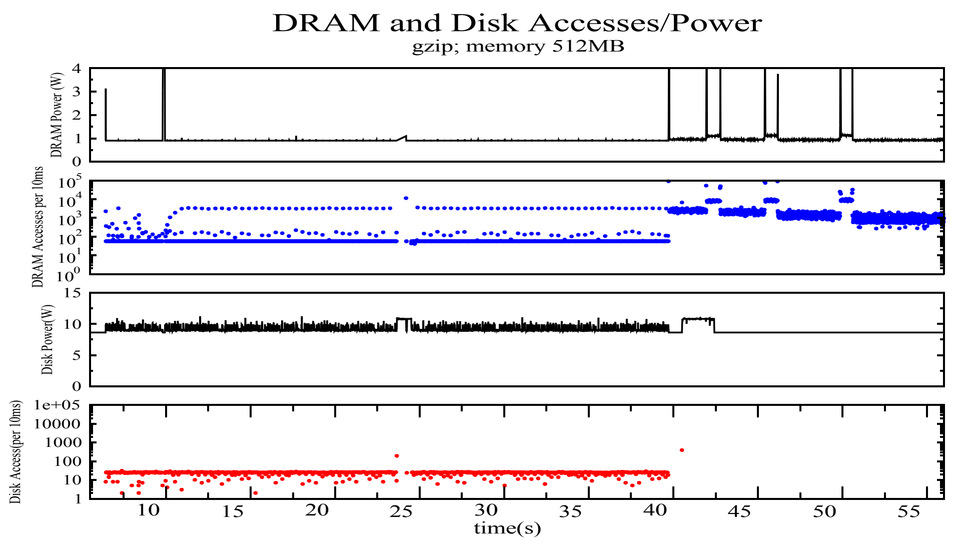


Figure 6.2: The interaction in memory hierarchy in a system with 512MB of memory. The figure shows the interaction between all components in the memory hierarchy including level-1 instruction cache, level-1 data cache, level-2 unified cache, DRAM, and a disk drive. Notice that initialization time reduces from 140 seconds in Figure 6.1 to 40 seconds in this figure.

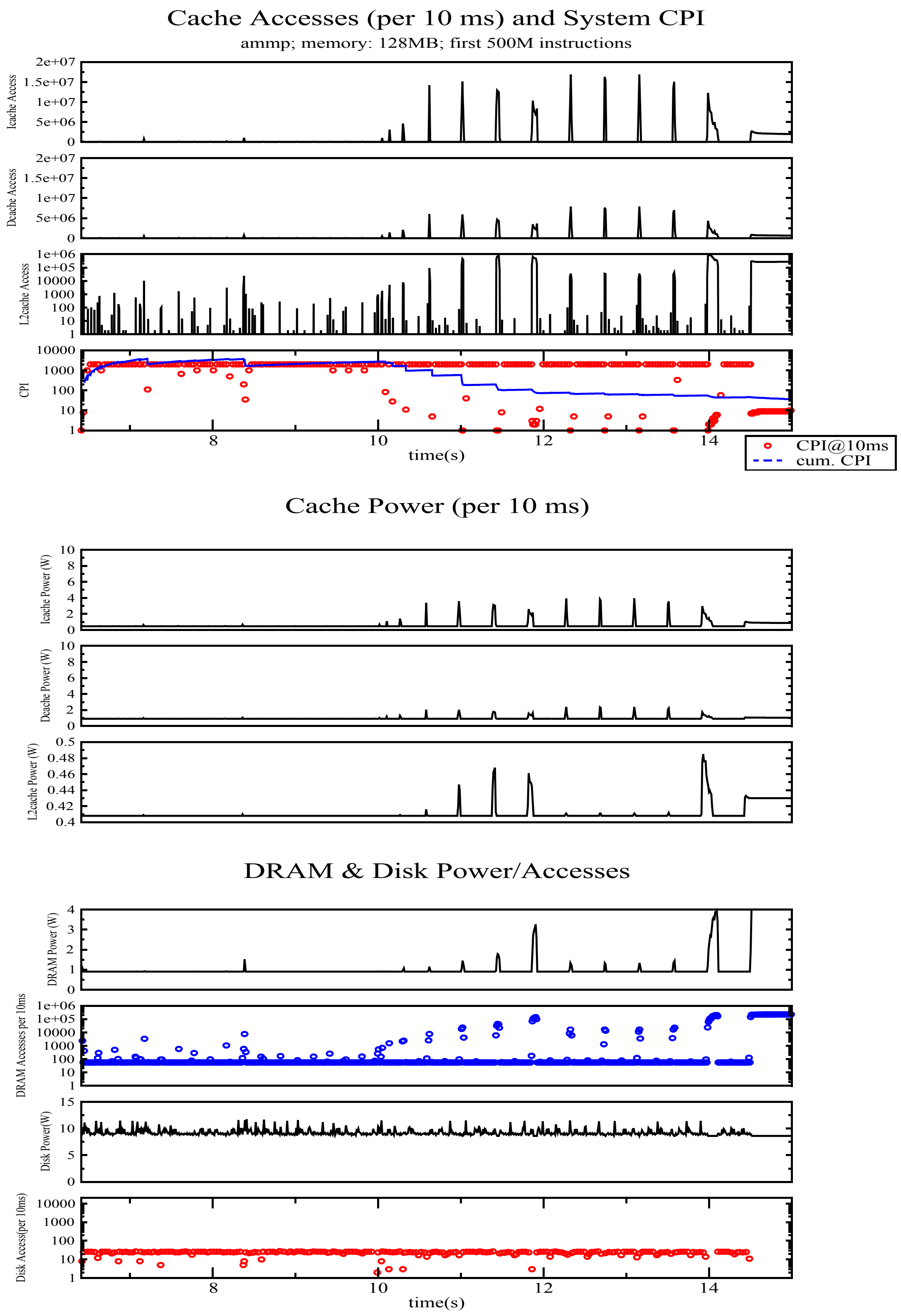


Figure 6.3: I/O intensive phase of ammp.

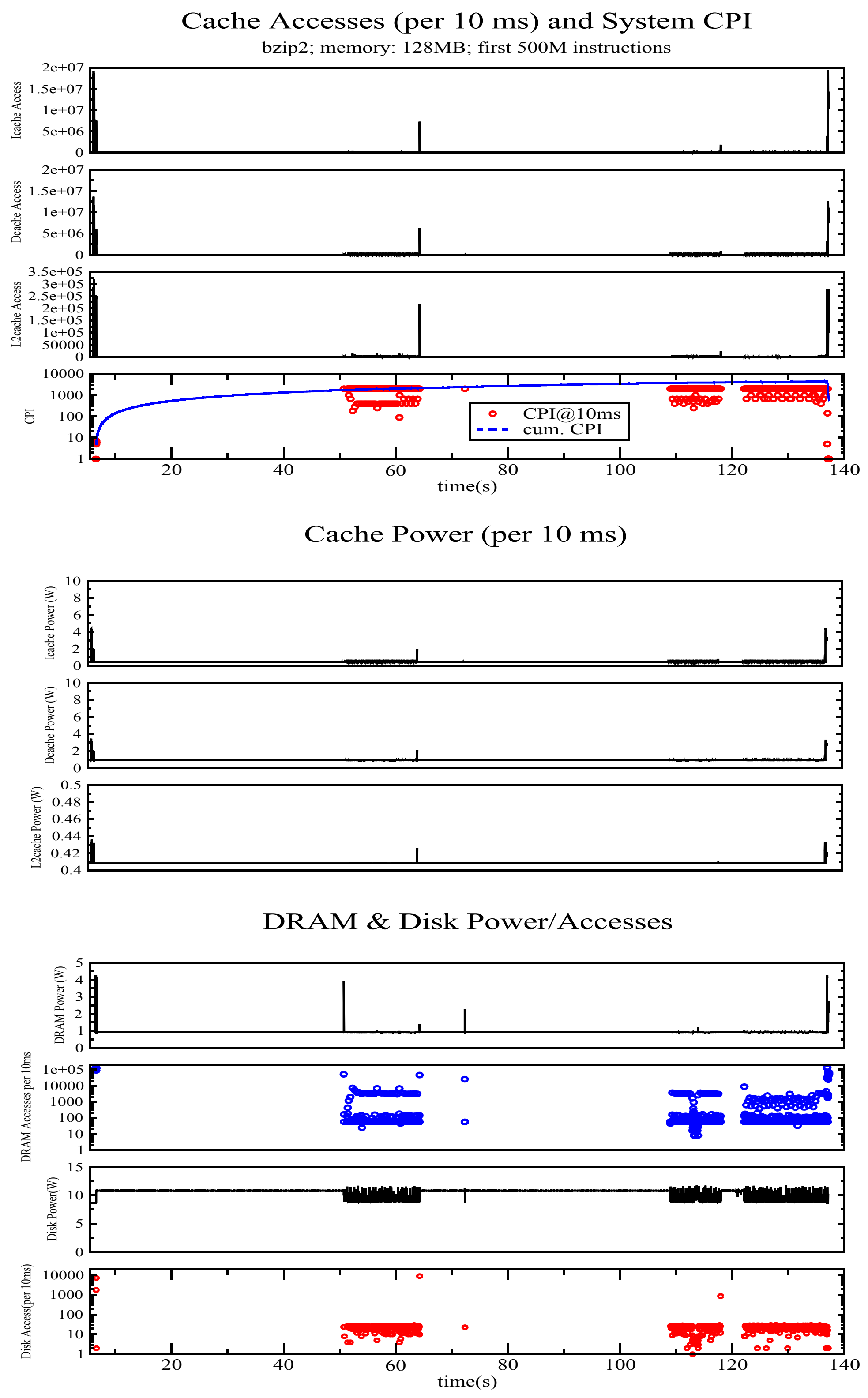


Figure 6.4: I/O intensive phase of bzip2.

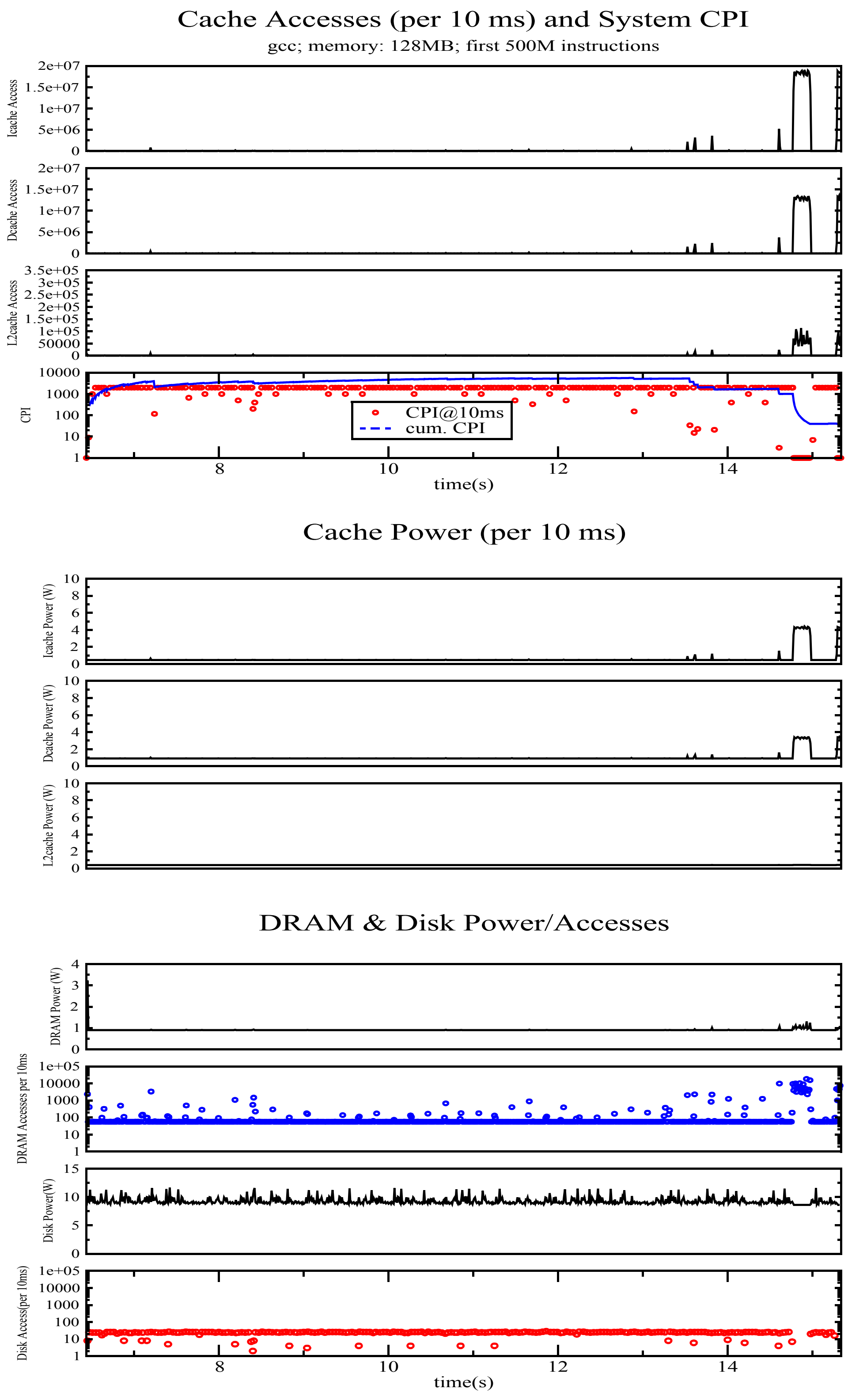


Figure 6.5: I/O intensive phase of gcc.

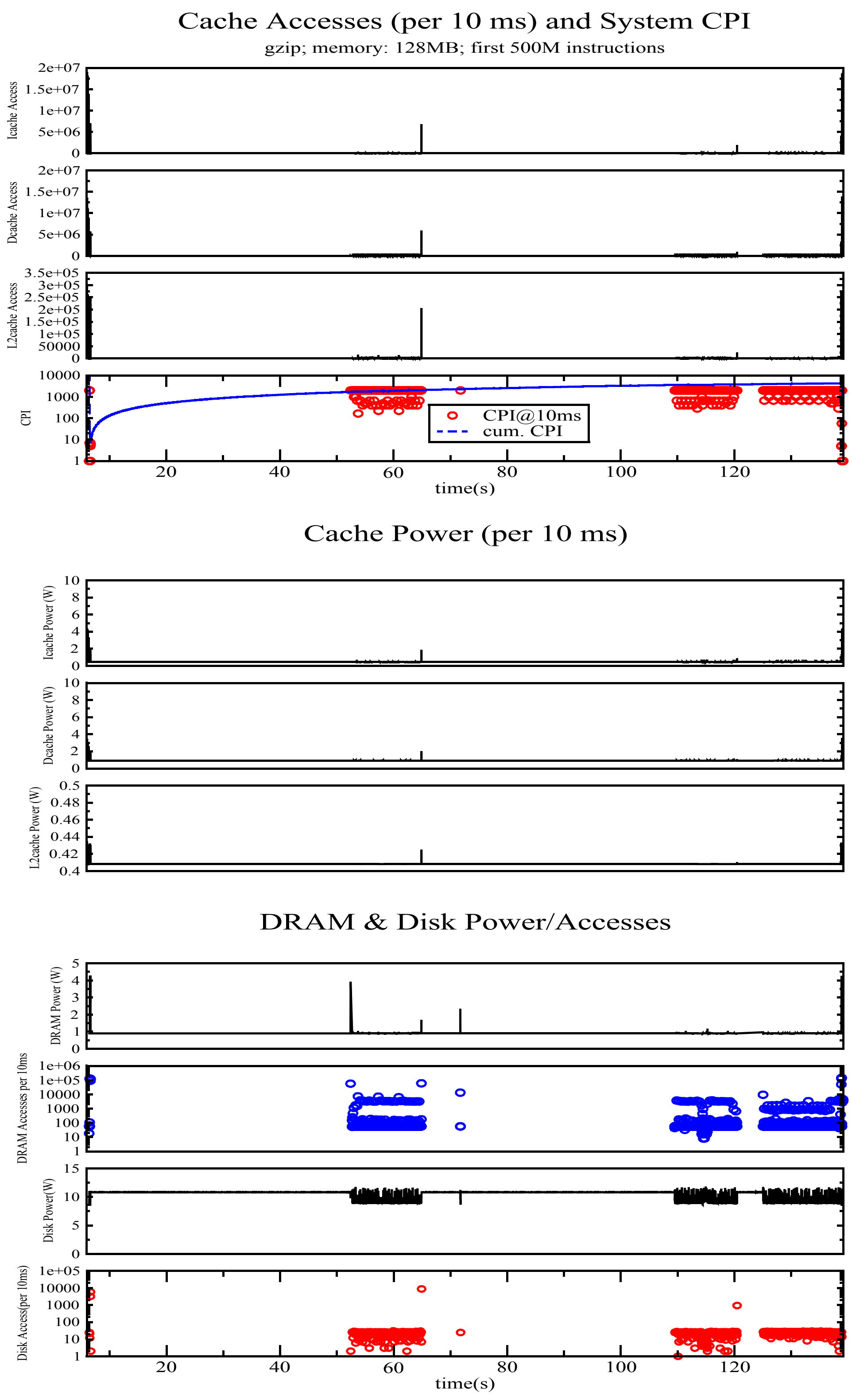


Figure 6.6: I/O intensive phase of gzip.

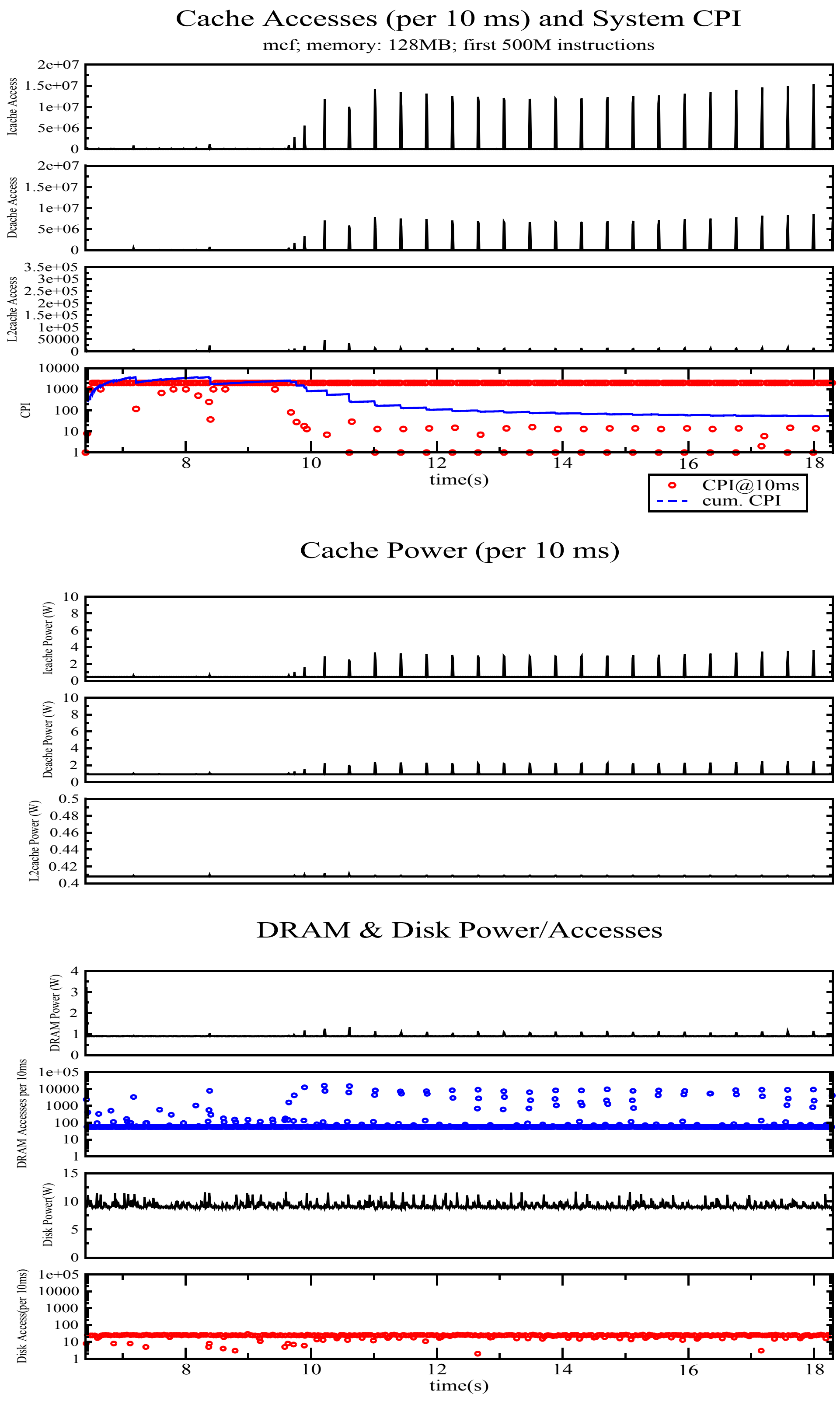


Figure 6.7: I/O intensive phase of mcf.

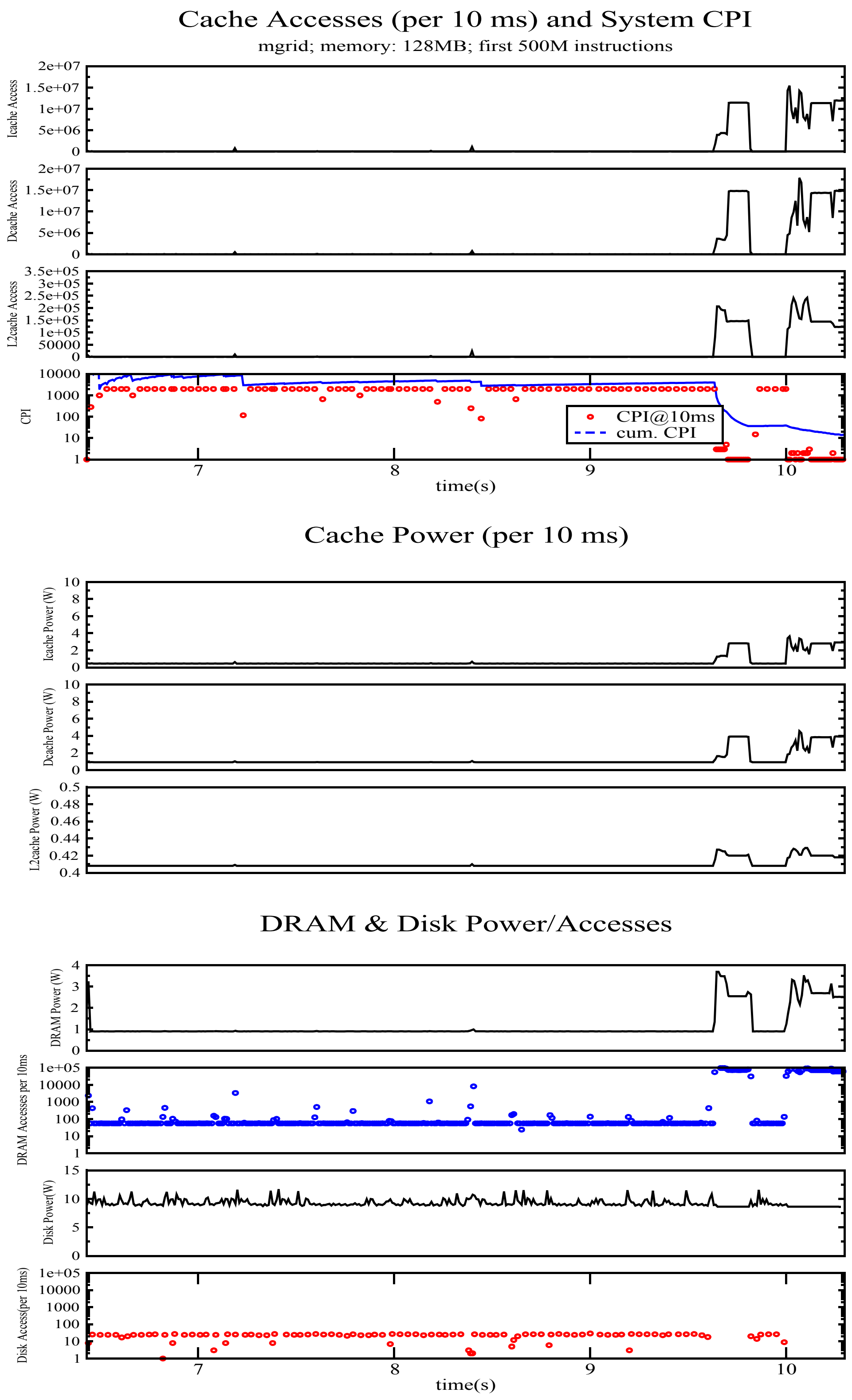


Figure 6.8: I/O intensive phase of mgrid.

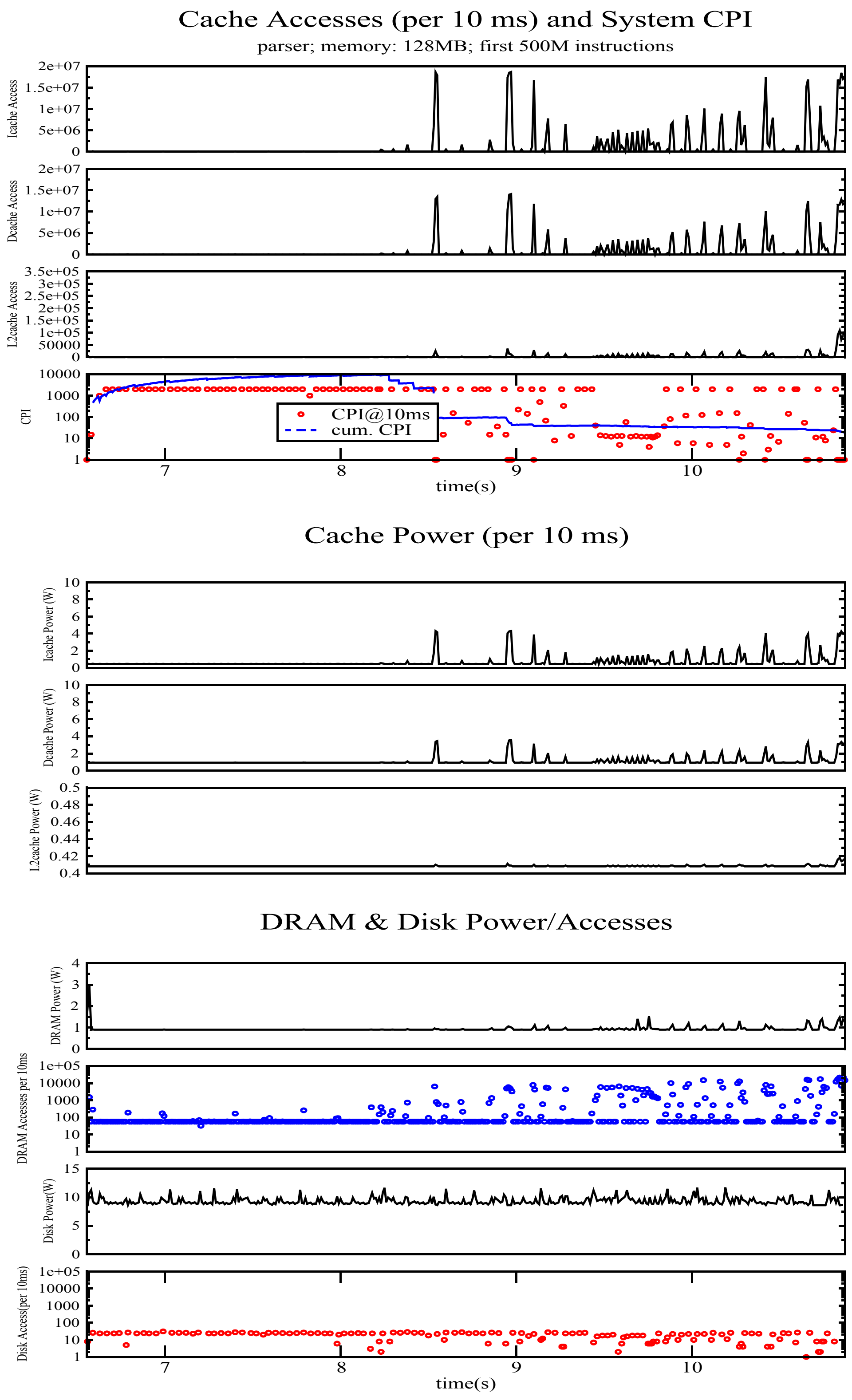


Figure 6.9: I/O intensive phase of parser.

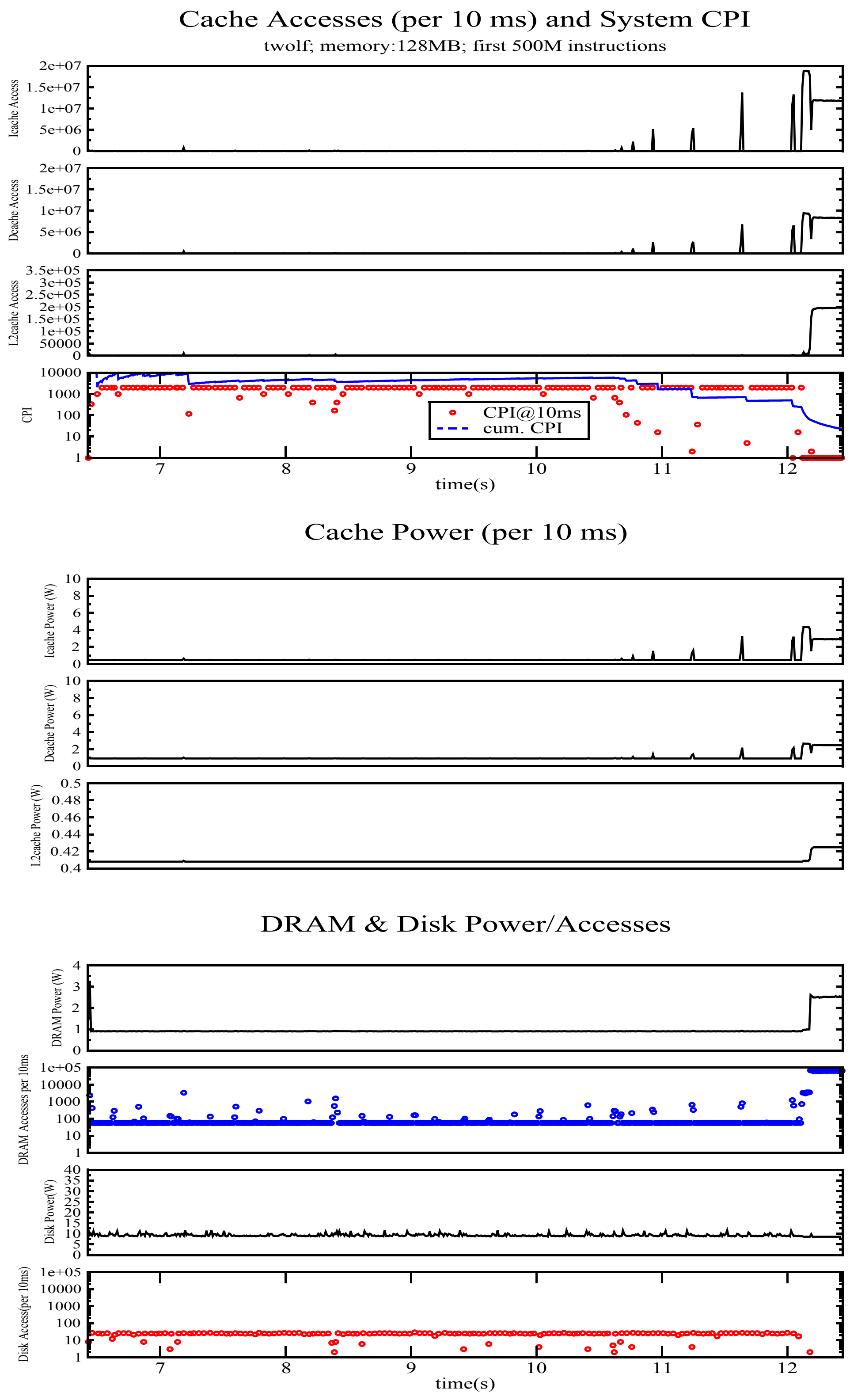


Figure 6.10: I/O intensive phase of twolf.

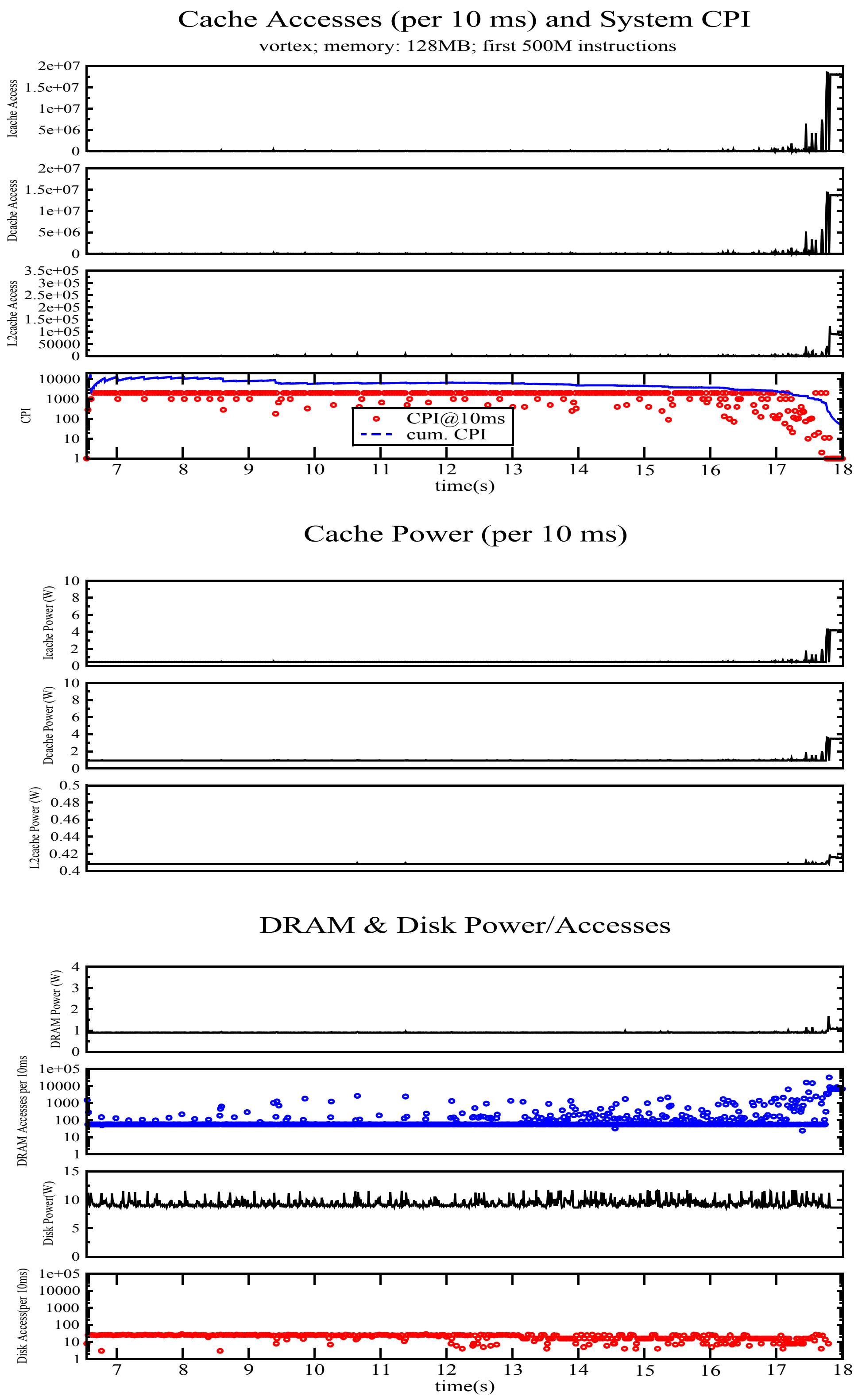


Figure 6.11: I/O intensive phase of vortex.

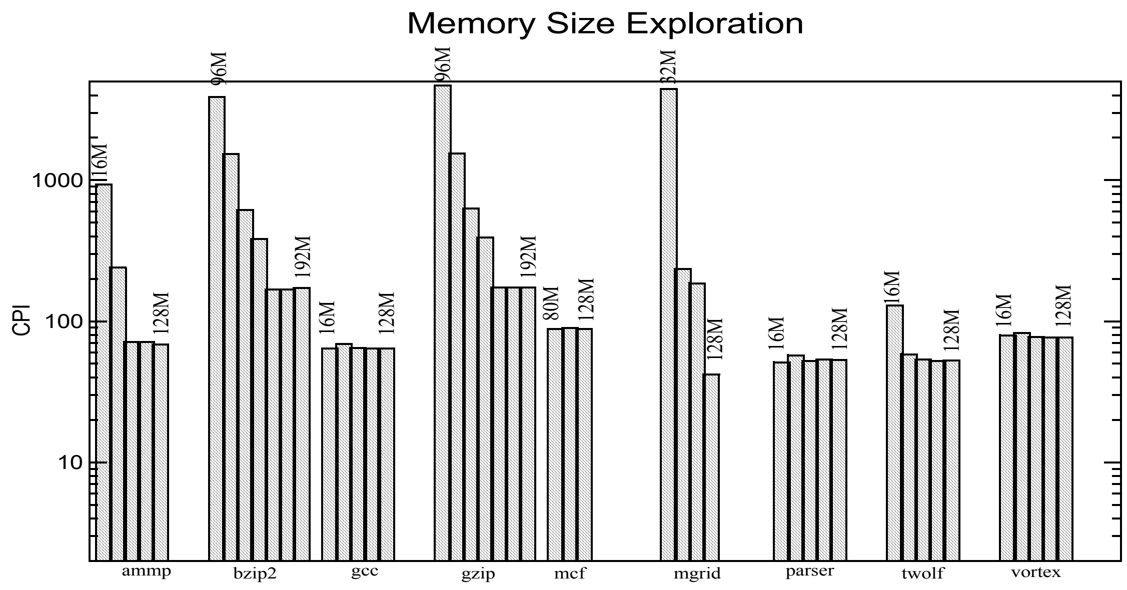
the accesses. Therefore, the disk latency has a significant impact on the total execution time for all applications.

The disk dissipates near the maximum power during the I/O intensive phase, while other memory components (level-1 caches, level-2 cache, and DRAM) dissipate marginal power. Even in the applications with regular accesses, other memory components still dissipate little power due to the long latency of the I/O spreading the accesses apart. The maximum instantaneous power dissipated for level-1 caches is approximately 4 Watts, but the average power for the entire phase is approximately 0.4 - 1.2 Watt. Despite its large size, the level-2 cache dissipates very little power (0.4 Watt) due to the clock gating style used in the cache. In level-2 cache, since it is accessed less frequently than the level-1 caches, only the accessed bank of level-2 cache is active during the access, and the rest are inactive. This may cause a performance penalty, but the mechanism saves significant power. The DRAM dissipates only roughly 4 Watts maximum at any instant and approximately 0.2 - 1 Watt on average. Since the DRAM has low activity during the I/O intensive phase, the DRAM power and energy mainly depends on the DRAM configuration: the power in the DRAM system is proportional to the number of the DRAM chips. In our experiment, the DRAM is set to only one rank with 8 chips, and its capacity is varied by changes in internal DRAM chip configuration. With no variation in the number of chips, the power dissipation of the DRAM would not be drastically affected.

6.2. Memory Size and I/O Behaviors

It has been widely accepted that the system memory capacity has a great impact to overall system performance. Different benchmarks require different memory, and different phases of the execution have different memory footprint. For our experiments, we executed nine different SPEC2000 benchmarks on different sizes of the main memory, and observed the total system performance in term of CPI and the number of disk requests generated. The results are shown in the figure 6.12. The minimum and the maximum size of the memory for each benchmark executed are as labeled. The minimum size of memory for each benchmark is the minimum size of memory that the system can run without a “not enough memory” error. The maximum size of the memory for each benchmark is the size of memory that does not have any different results than the system with smaller memory. Note, the y-axes in both graphs are in log scale.

We observed that all SPEC2000 benchmarks used can be characterized into 2 categories; first is the benchmarks that show memory page swapping behavior due to insufficient memory to hold the entire memory footprint in the I/O intensive phase, i.e. ammp, bzip2, gzip, and mgrid. This type of benchmark has both disk reads and writes. The numbers of disk reads and writes depend heavily on the size of the memory. The second category is the benchmarks that can fit into a small memory system, so the systems have no page swapping. Therefore, only series of sequential disk reads are exhibited. For example, gcc, mcf, parser, and vortex are categorized into this type of benchmark. This type of benchmark is not sensitive to the size of the memory as long as the system can provide enough memory to run without crashing. One would notice that in the latter type of applications, for example gcc,



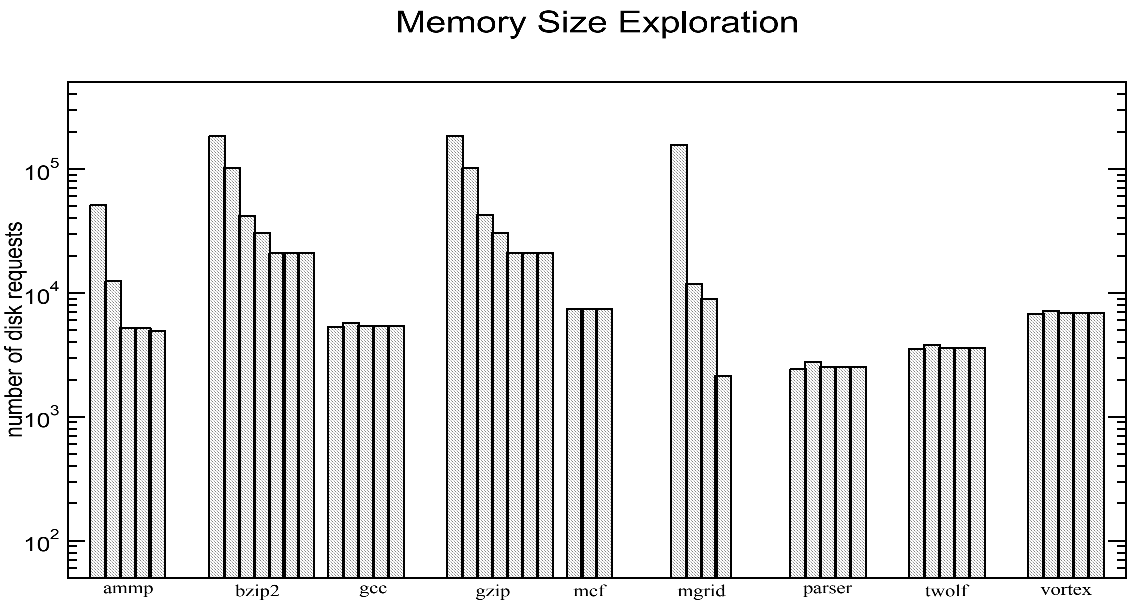
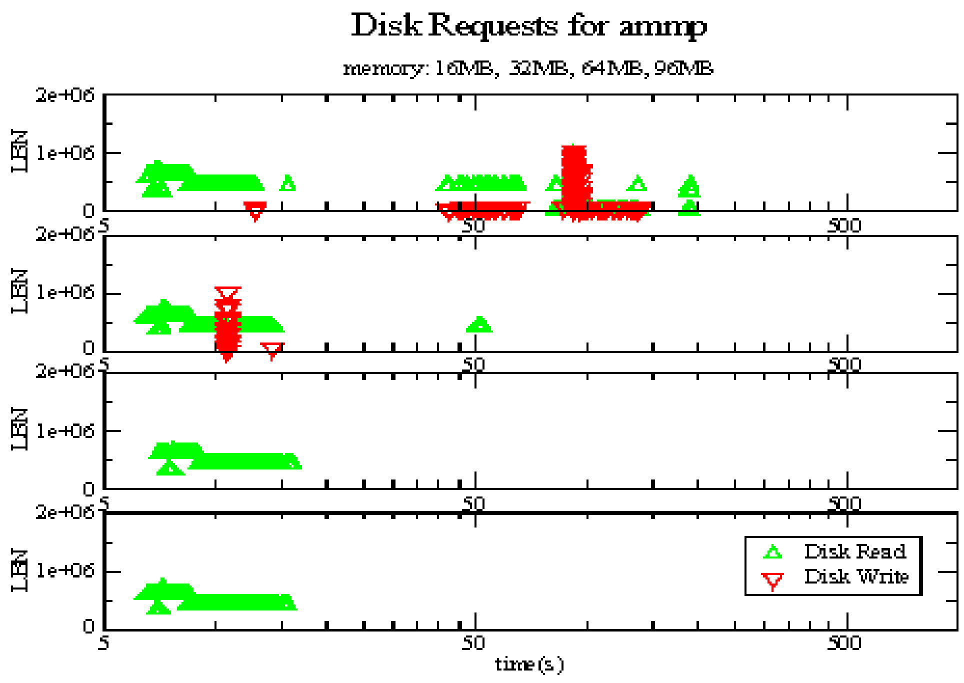


Figure 6.12: Memory Size Exploration. Changing in the system memory capacity has exponentially impact on the overall system performance (CPI). However, if the size of the memory is big enough to hold the memory footprint of the benchmark, the memory size has no effects. The figure shows not only the changes in CPI but also the changes in the number of the disk requests over 9 spec 2000 benchmarks. We run amp, gcc, parser, twolf, and vortex over the memory size of (16MB, 32MB, 64MB, 96MB, and 128MB), bzip2 and gzip over (96MB,112MB,128MB, 144MB, 160MB, 172MB, 192MB), mcf over (80MB, 96MB, 128MB), and mgrid over (32MB, 64MB, 96MB, 128MB). The smallest size of the memory for each benchmark is the smallest size of memory that the system can run without “not enough memory” error.

the memory size of 16MB can hold both the operating system and the application. Therefore, the operating system uses less than 16MB. As a result, in the applications demonstrating paging behavior, the systems exhibit paging even with the memory size larger than 100MB. This paging behavior is all due to the memory footprint of the application. Due to the paging behavior in those systems, in the subsequent experiments, we choose only bzip2 and ammp with different memory size to represent both categories of behaviors.

Figure 6.13 to 6.15 show the disk reads and writes over time in the first category of applications for ammp, mgrid, gzip, and bzip2. Each graph in the figure represents the data on the system with different DRAM capacity: less DRAM capacity on top and more at the bottom. There are both disk reads and writes in the request streams. The less DRAM capacity provided in the system, the more requests to disk. The majority of the increased disk requests are disk writes. On the other hand, Figure 6.16 shows the disk reads and writes over time in the second category applications for parser and gcc. Obviously, the applications in the first category exhibit increasing number of writes while the DRAM capacity decreases because the operating system swaps many pages out to prepare for new pages reading in. The applications in the second category applications do not write to disk, despite the size of the DRAM.



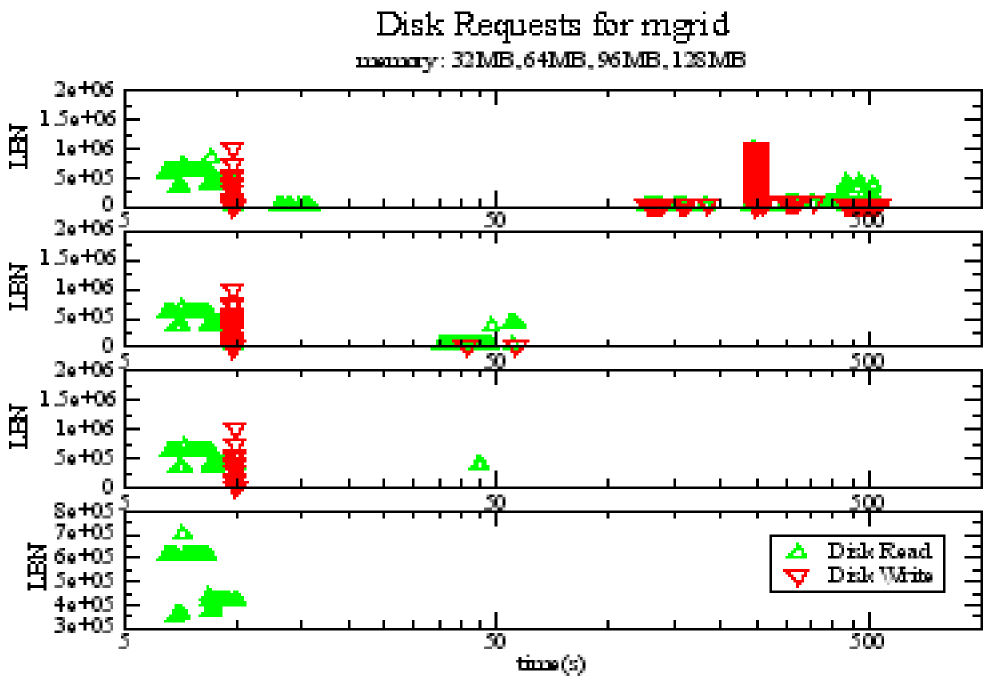
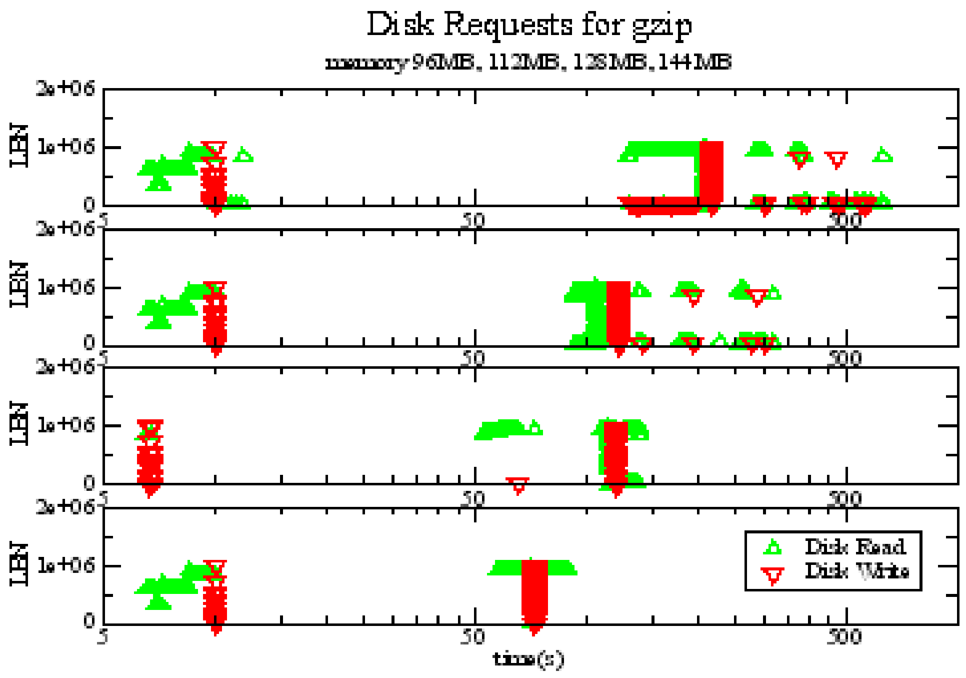


Figure 6.13: ammp and mgrid Disk Activities.



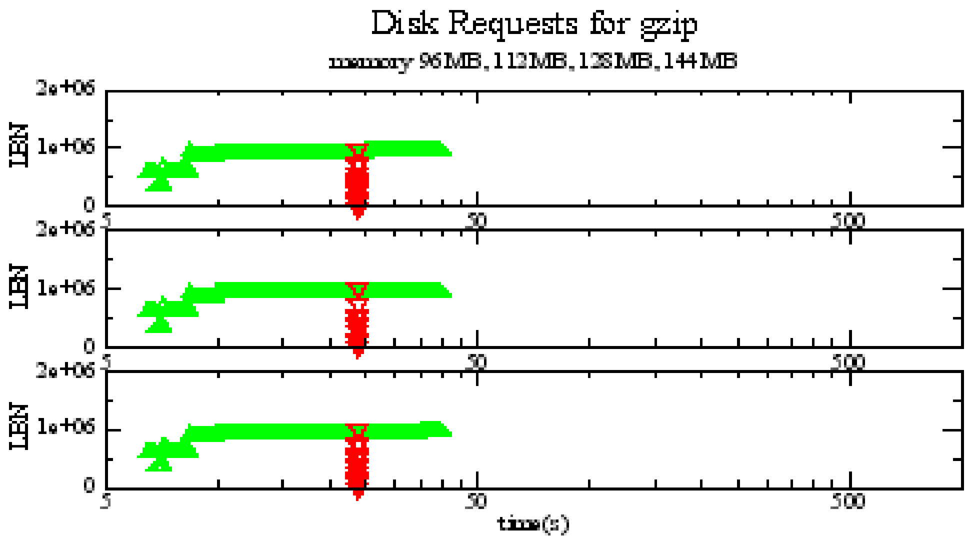
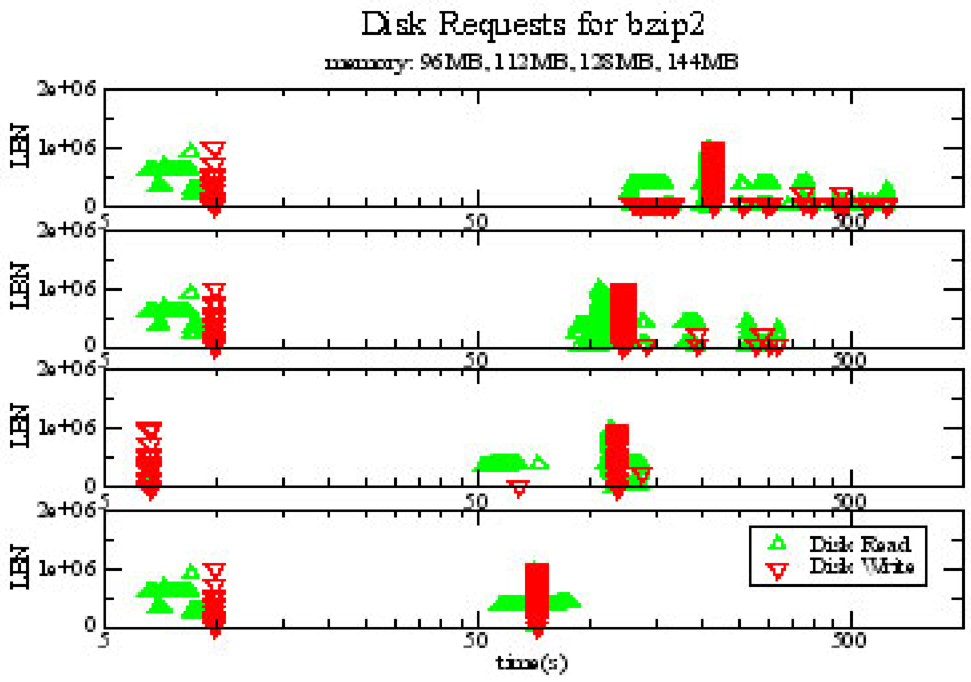


Figure 6.14: gzip Disk Activities.



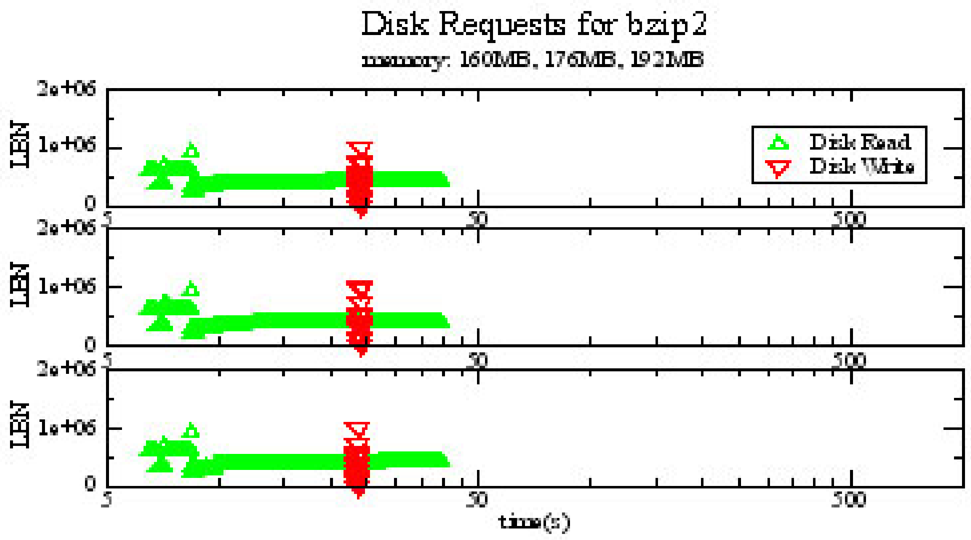
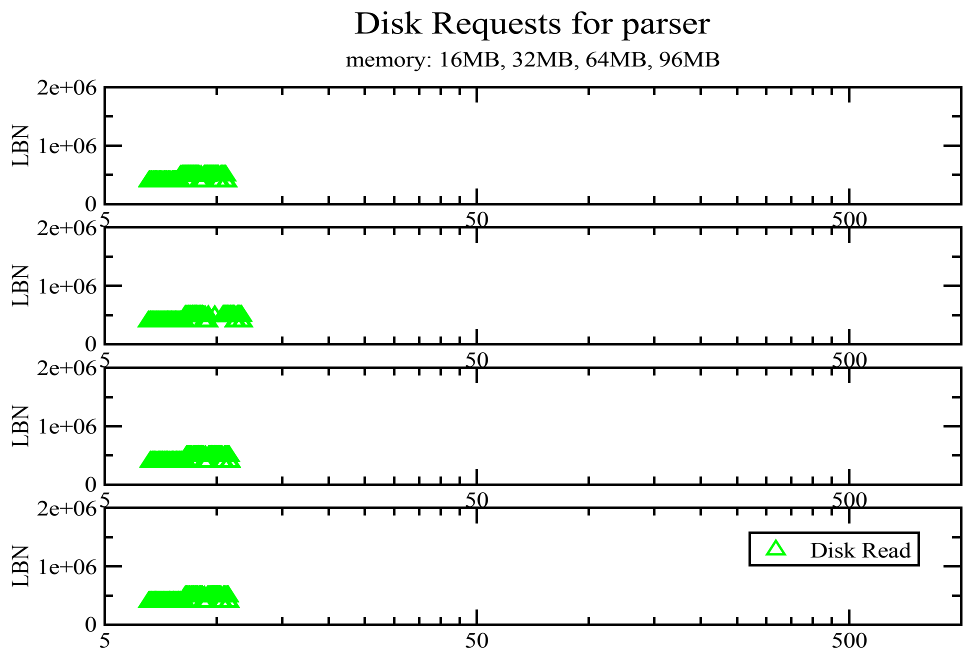


Figure 6.15: bzip2 Disk Activities.



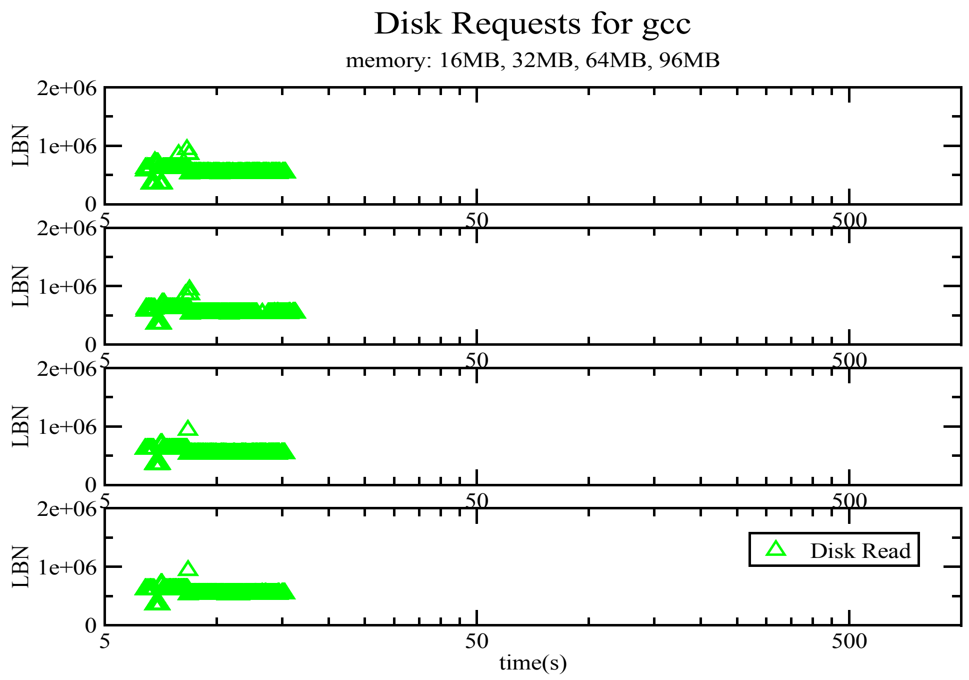


Figure 6.16: parser and gcc Disk Activities.

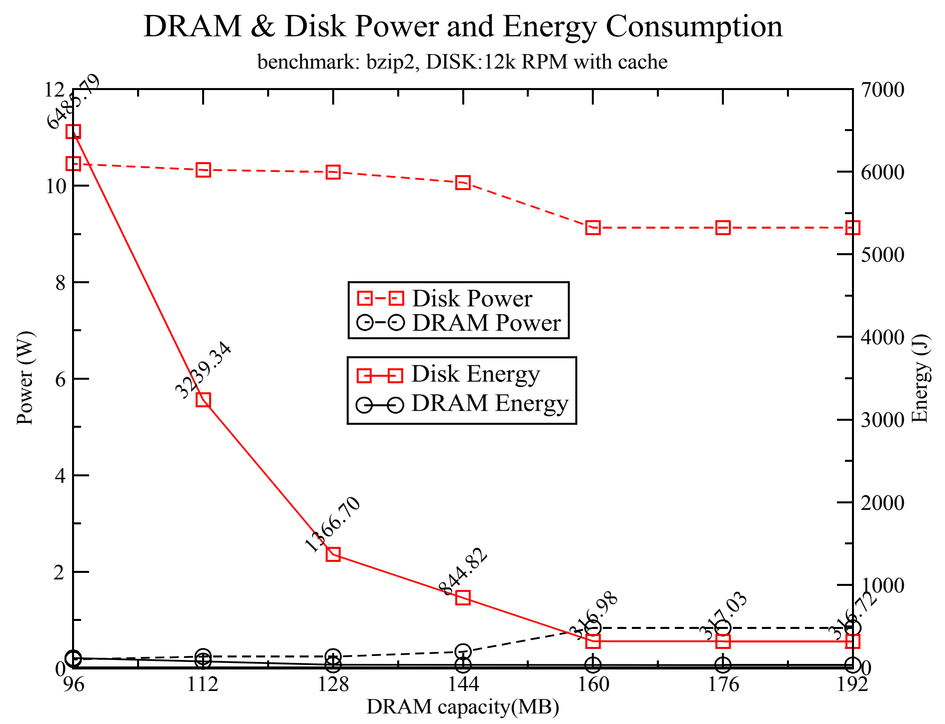
6.3. Power/Energy Consumption of the Disk due to Different Memory Size

In the previous section, we learned that the memory size has a substantial impact on the overall system performance. However, not only is overall performance important, but the power dissipation and energy consumption are also main concerns nowadays. We setup an experiment for the power dissipation and energy consumption. We varied the size of the memory and analyzed the power dissipation and energy consumption of the DRAM and the disk. The experiment focuses on only single disk systems. The disk configuration is a 12k-RPM disk with 4MB disk cache. Figure 6.17 shows the power dissipation and the energy consumption of such systems. Compared to the disk, DRAM dissipate a small amount of power. Indispensably, the system requires enough DRAM capacity to hold the application footprint, so the disk is accessed less and dissipates much less power.

On the other hand, the energy consumption is more important than the power dissipation in our experiments. For small size of the memory, the disk power dissipation remains relatively constant compared with the system with large memory, while the disk energy consumption is rapidly increasing. The reason is, though the power dissipation is limited by the maximum, the execution time is prolonged due to more pages swapping to the disk. The prolonged execution time causes the energy consumption to increase rapidly. Additionally, DRAM energy consumption becomes significant compared to the disk energy. The reason is the DRAM capacity is large enough to contain all needed memory pages, so the number of requests to the disk is reduced. Therefore, the disk consumes its minimum energy while the DRAM system is busiest. The ratio of the energy consumption of the disk over the energy of the DRAM reduces from 100:1 to 10:1 when enough DRAM capacity is added in the system. However, the ratio may reduce even more if more DRAM is added excessively and costs only more energy without any performance benefits.

Next, we conduct an experiment to characterize the power dissipation and the energy consumption of different RPM disks. Figure 6.18 shows the DRAM & Disk Power Dissipation and Energy Consumption with different RPM disks. The top graph shows the power dissipation of the DRAM and the disk in a single disk system. The bottom graph shows the energy consumption of the DRAM and the disk. We varied the memory size and study its impact on energy consumption/power dissipation. All results are show for bzip2.

The DRAM power and energy for each memory size are always the same for all disk RPMs. Interestingly, the power dissipation of the disk are varied with the disk RPM. On the other hand, the energy consumption is more intriguing. Lower RPM disk does not always mean lower energy. Unlike the power dissipation, the lowest RPM actually consumes the most energy when the memory size is small, and consumes the least energy when the memory size is large enough to hold the benchmark data. The reason is when the memory size is small, the disk with lower RPM spend more time to execute the same set of instructions since there are more disk requests to the disk due to memory page swapping. The disks with 12k-RPM and 20k-RPM consume relatively the same energy, but the CPI of the latter is much better than the former. All disks consume relatively the same energy at 144MB, but the different system performances vary as much as a factor of 2.5.



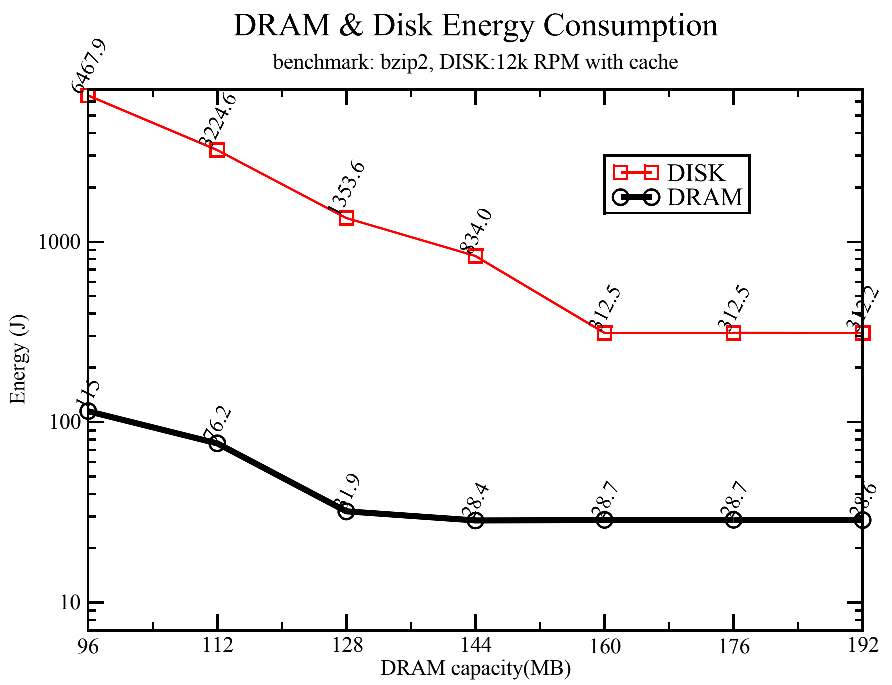
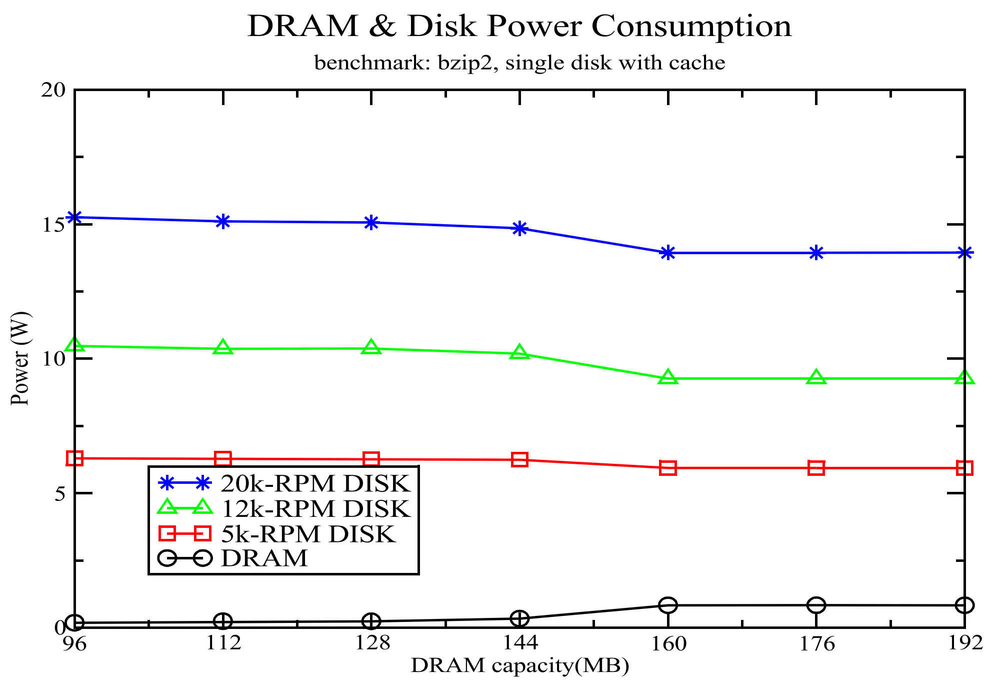


Figure 6.17: Power Dissipation and Energy Consumption of DRAM and a Disk. The configuration of the disk system is one 12k-RPM disk drive with 4MB of disk cache. We varied the memory size, but not the other memory parameters. The top graph shows both the power dissipation (dash line with left y-axis) and the energy consumption (solid line with right y-axis). The bottom graph shows only the energy consumption in log scale.



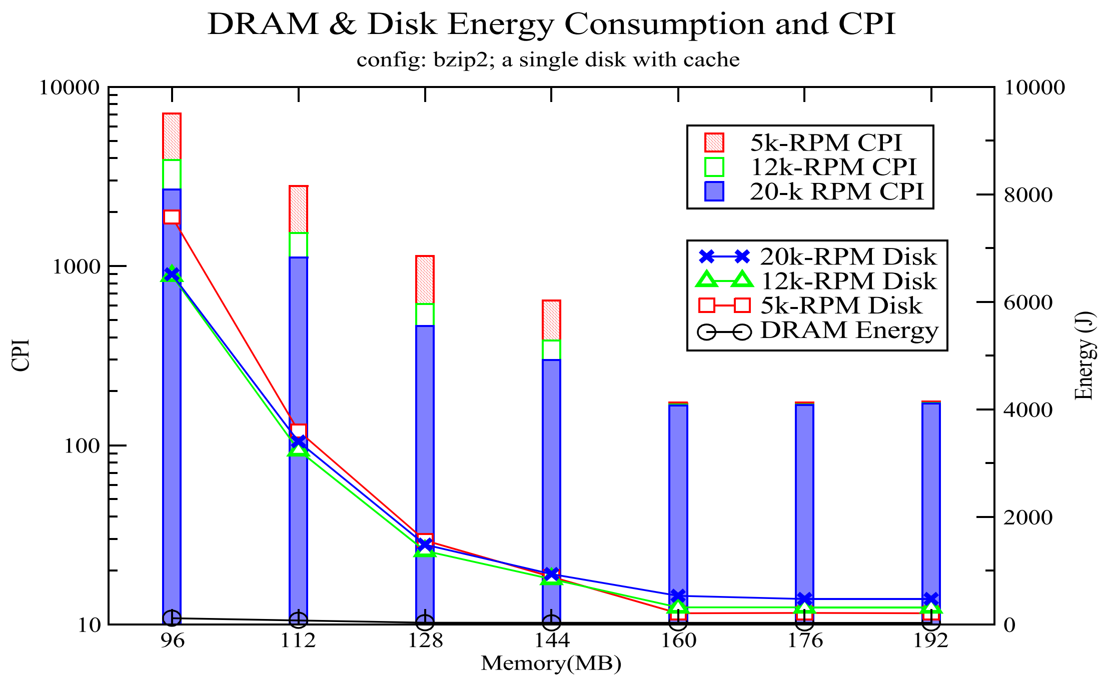


Figure 6.18: DRAM & Disk Power Dissipation and Energy Consumption. The top graph shows the power dissipation of the DRAM and the Disk in a single system. The bottom graph shows the energy consumption of the DRAM and the Disk and the total system CPI. We varied the memory size and ran bzip2. The DRAM power and energy for each memory size are always the same for all disk RPMs. Interestingly, the power dissipation of the disk are varied with the RPM while the lowest RPM means lowest power. On the other hand, the energy consumption is more intriguing. The lowest RPM actually consumes the most energy when the memory size is small, and consumes the less energy when the memory size is big enough to hold the benchmark memory footprint. All disks consume relatively the same energy at 144MB with different total system CPI.

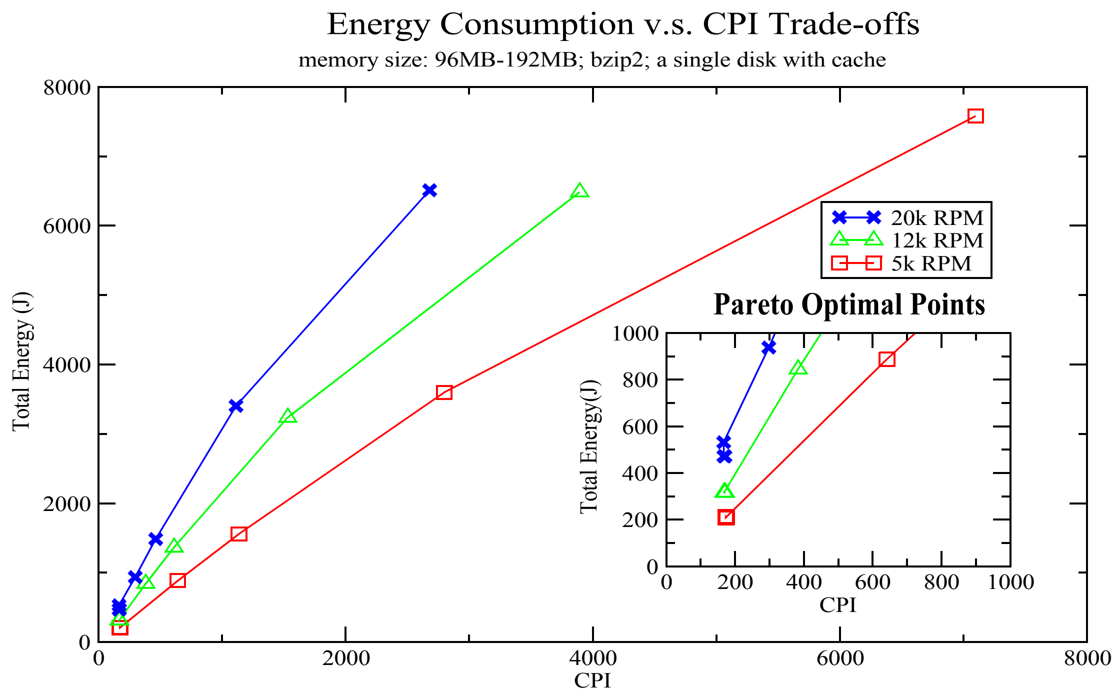


Figure 6.19: Energy Consumption v.s. CPI Trade-offs. The graph show the trade-offs plot combining total system energy consumption and the total system CPI. Each line represents a configuration with different RPM disk. There are 7 data points on each line, representing the size of memory from 96MB to192MB with the top-right data point is for 96MB. Largest DRAM capacity is to the left; obviously, more DRAM capacity translates into better in both energy and performance. The inset graph shows the pareto optimal points.

Lastly, Figure 6.19 shows the trade-offs graph between the Energy Consumption and the system CPI. Each line represents a configuration with different RPM disk. There are 7 data points on each line, representing the size of memory varied from 96MB to 192MB, corresponding to the memory size in the previous figure. On each line, the top-right data point is for the memory size of 96MB. The inset graph shows the data points of the configurations with the memory size of 144, 160, 176, and 192MB. As displayed, all lines are moving toward the origin as the memory is increasing. Among different RPMs, the higher RPM disks move toward the system optimal with higher rate. Hence performance is related to DRAM capacity more strongly than disk RPM. However, the lines stop at a certain CPI, even though the memory is increasing. This suggests that, with large system memory, higher disk RPM only increases system energy without performance impact. Using a fast RPM disk is a bad design point in this case. As a result, the slowest RPM disk is optimal.

6.4. Effects of Disk Physical Technology Improvement and Enhancements

6.4.1. Rotational Speed (RPM)

We conducted an experiment to explore the effects of the RPM on the disk on the overall system performance. For each benchmark, we set the disk system to single disk system and varied the RPM and the disk cache. The disk cache setting is either 4MB with prefetching or no cache at all. The memory size is set to a capacity of 128MB. Note, only bzip2 and gzip have memory page swapping at this stated memory size. Figure 6.20 shows the CPI due to the RPM and disk cache variation. The CPI is also in log scale.

First, we consider the benchmarks without page swapping, which have only read requests to the disk systems. Without disk caching and prefetching, the disk RPM benefit is very obvious; the faster, the better. However, when the disk RPM is fast enough, i.e. from 12k RPM to 20k RPM, improvements from a faster disk taper out. At this point, the latency from the other parts of the disk overshadow the benefit from higher RPM. With disk caching and prefetching, there is no significant difference in CPI for the benchmarks without page swapping because the disk caching and prefetching can hide the latency of the disk perfectly.

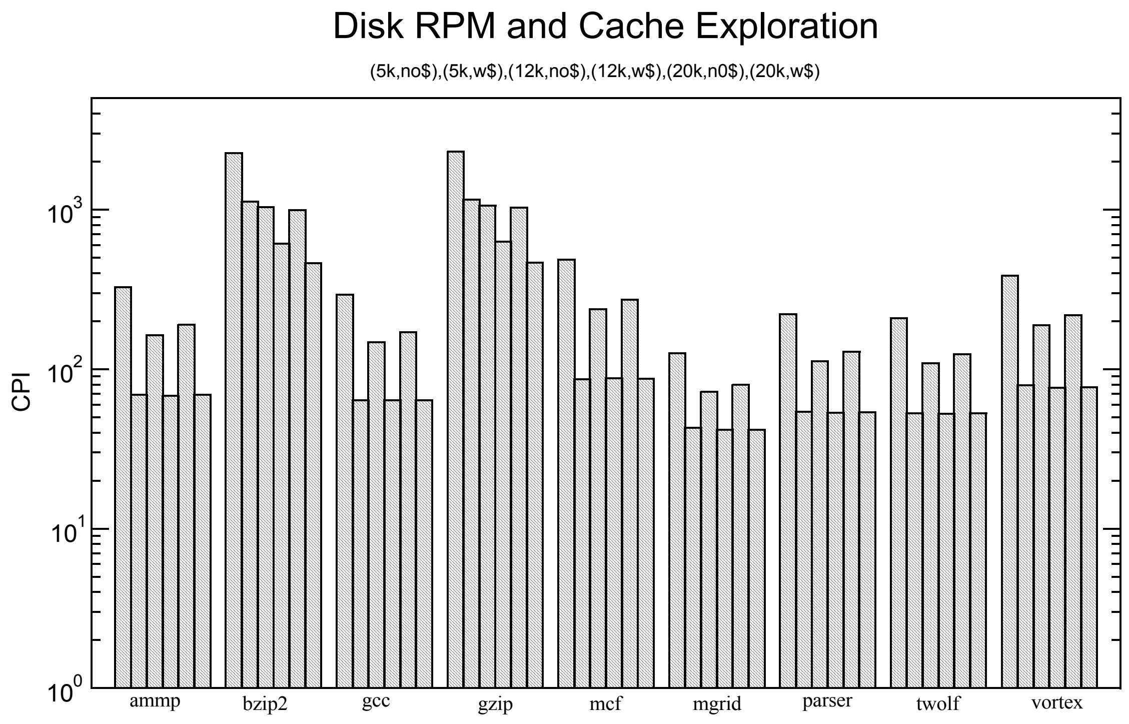
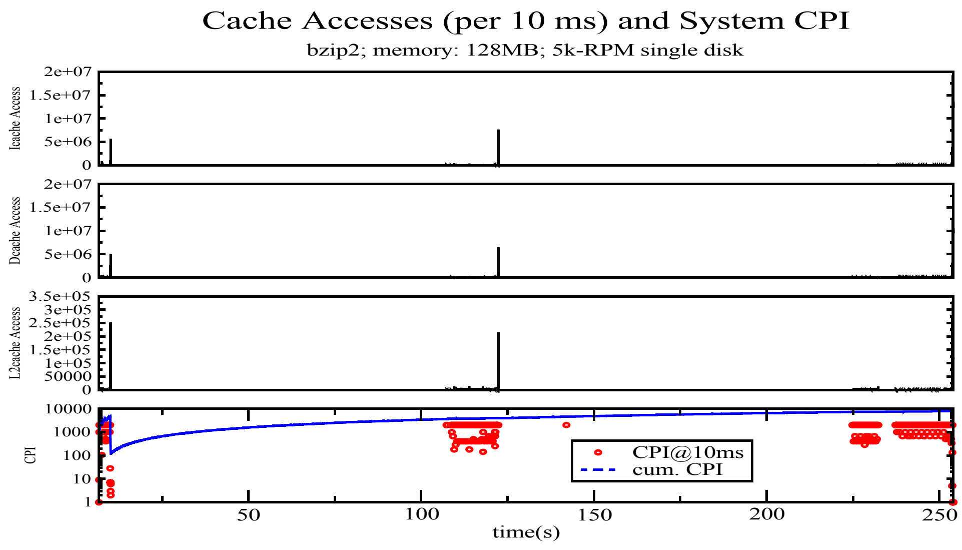
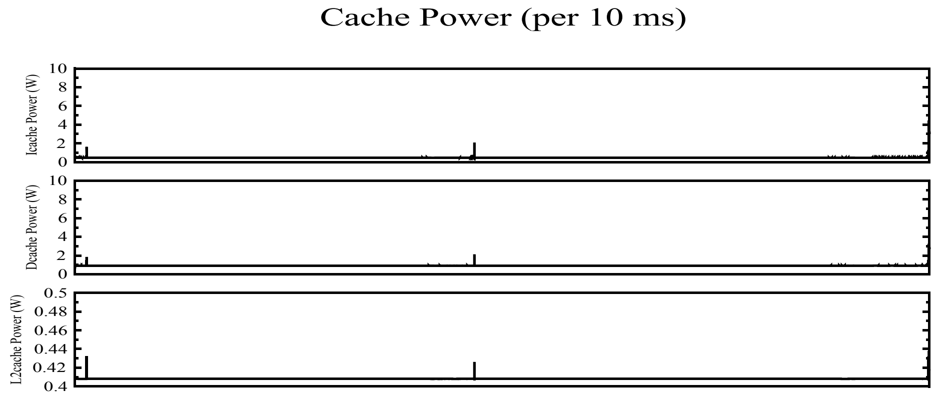


Figure 6.20: CPI due to the disk RPM and disk cache. The memory size is 128MB, and the disk system is a single disk system. We vary the RPM and the existence of the disk cache. The disk cache configuration is either 4MB disk cache with prefetching or no cache at all. For each benchmark, there are 6 bars which represent the CPI of (1) 5k-RPM disk with no cache,(2) 5k-RPM disk with cache, (3) 12k-RPM disk with no cache, (4) 12k-RPM disk with cache, (5) 20k-RPM disk with no cache, and (6) 20k-RPM disk with cache.Note that at the memory size of 128MB, only bzip2 and gzip exhibit the memory page swapping.

Second, in the benchmarks with disk reads and disk writes, as exhibited in bzip2 and gzip, the disk RPM does matter in both the systems with and without disk cache. The locality of the accesses decrease since the reads and writes access different areas of the disk. Even though either the reads or writes have high locality, the disk caching and prefetching cannot hide all their latency. The reason is the disk maintains the concepts of non-volatile storage, so reads issued after any writes have to wait until the writes are processed.

Finally, to be compared with our base case with a 12k-RPM single disk in Figure 6.4, Figure 6.21 and Figure 6.22 show the interaction of the memory hierarchy components in the system with a 5k-RPM single disk and with a 20k-RPM single disk, respectively. The system with 12k-RPM spends approximately 140 seconds while the 5k-RPM system spends





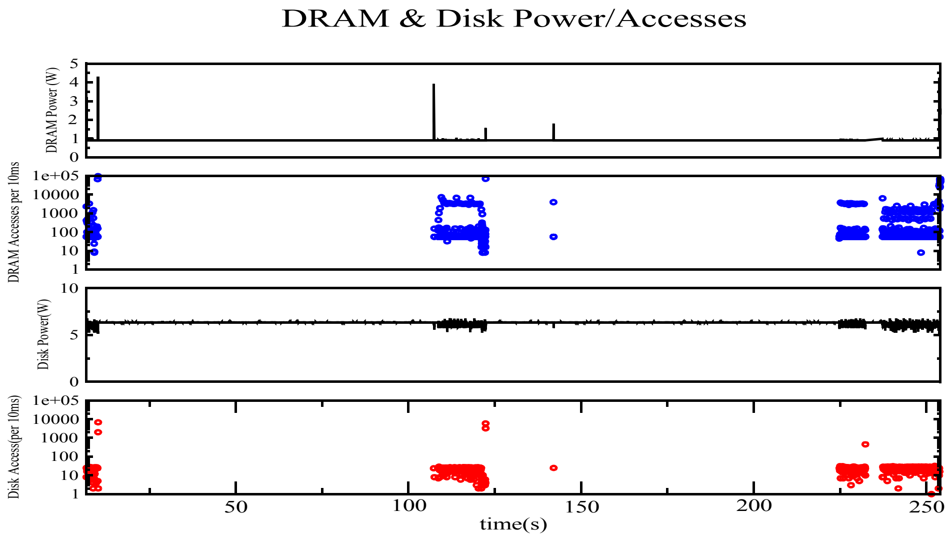


Figure 6.21: The interaction in the memory hierarchy for a system with a 5k-RPM disk drive.

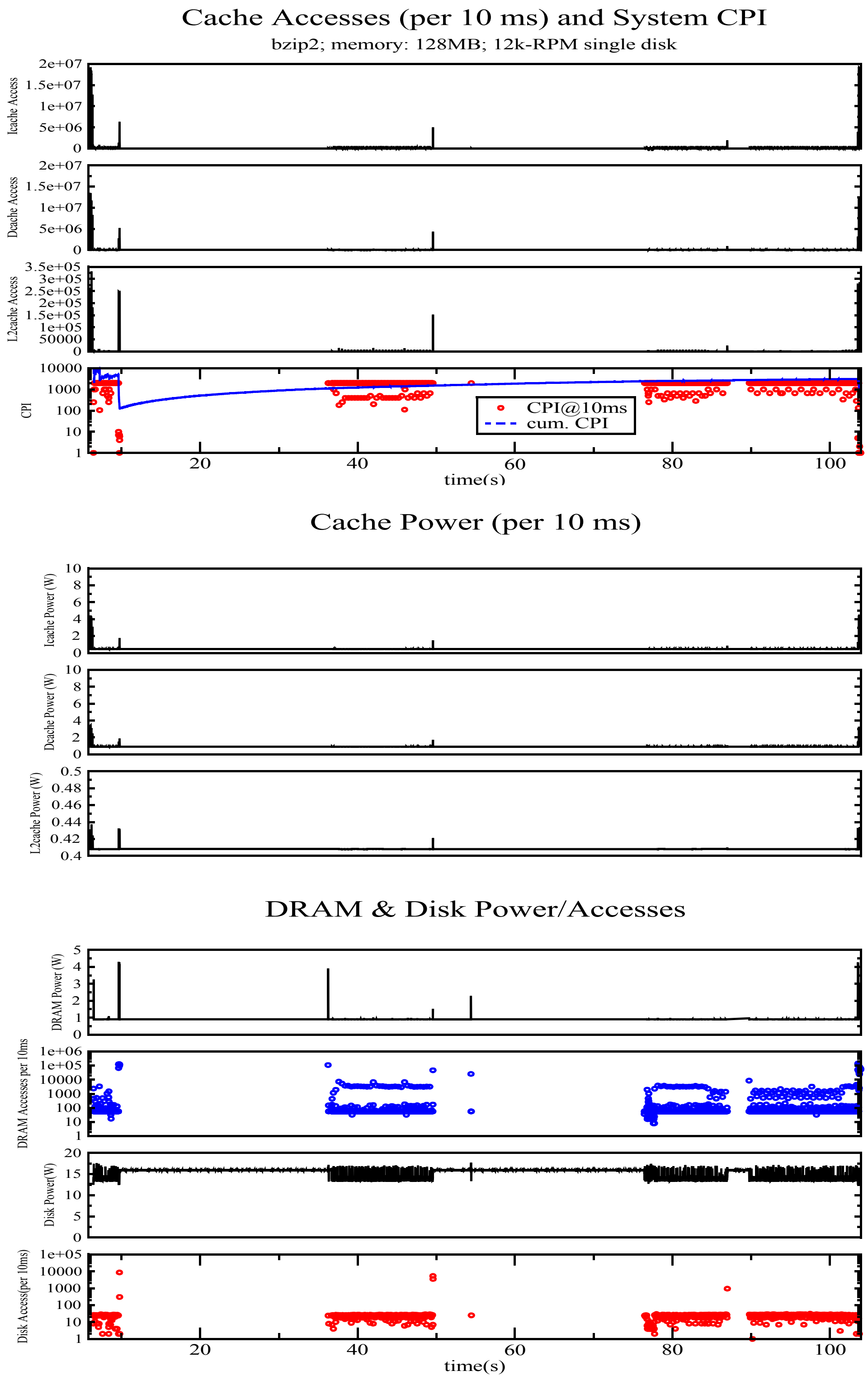


Figure 6.22: The interaction in the memory hierarchy for a system with a 20k-RPM disk drive.

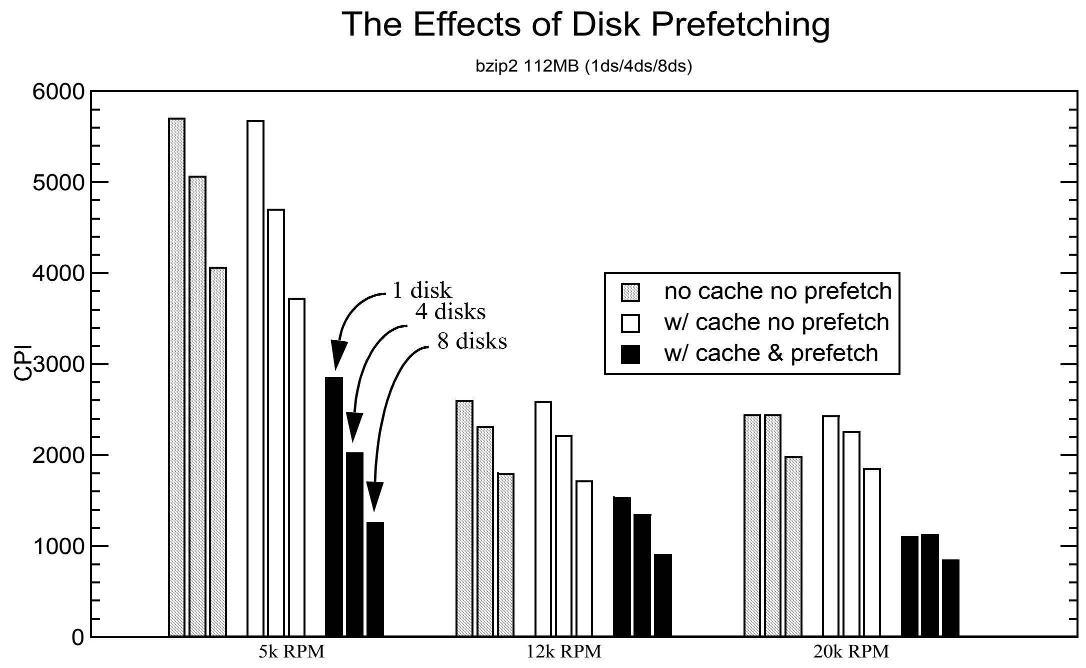
over 250 seconds on the same set of instructions. Using a 20k-RPM disk improves the execution to 110 seconds. The improvement in total execution time is mainly from the benefits over the write bursts. The write bursts can benefits from higher RPM disk directly because the faster disk would result in fast response for writes. As a result, the write bursts processed faster results in shorter write burst processing period. However, the benefit over the reads is not obvious. The reason is the disk subsystem is already equipped with disk cache, which absorbs the reads when the cache hits without disk mechanical parts involved.

6.4.2. Prefetching

As shown is Figure 6.20, disk caching and prefetching can gain significant performance, especially with both the benchmarks with only reads (no page swapping) and with reads/writes (with page swapping). Furthermore, disk caching and prefetching can completely hide the rotational latency in the benchmarks with only disk reads. In this section, we conducted the experiment to identify the importance of disk caching and prefetching separately.

In the experiment, the system configuration is set to 112MB of memory running bzip2. Figure 6.23 shows the CPI and disk average response time for the experiment. The three bars in each group represent (1) a single disk system, (2) a 4-disk RAID5 system, and (3) an 8-disk RAID5 system. The upper graph shows the CPI for each configuration, and the lower graph shows the average response time of the disk requests. Note, the CPI axis is in linear scale, but the disk average response time axis is in log scale. The height of each bar in the average response time graph is the absolute value, i.e. the value of the response time for each type is the exact value where the bar ends.

In the previous section, Figure 6.20 shows that, in the disk read-dominated benchmarks, disk prefetching is more important than increasing the disk RPM. That is, rotational latency and bandwidth can be overcome by simple prefetching in an application with only disk reads. From Figure 6.23, disk caching has only marginal effects to both the CPI and the disk average response time. However, disk caching with prefetching has significant benefits: up to the factor of 4 for the case of 5400 RPM with 8 RAID disks and the factor of 2 on average



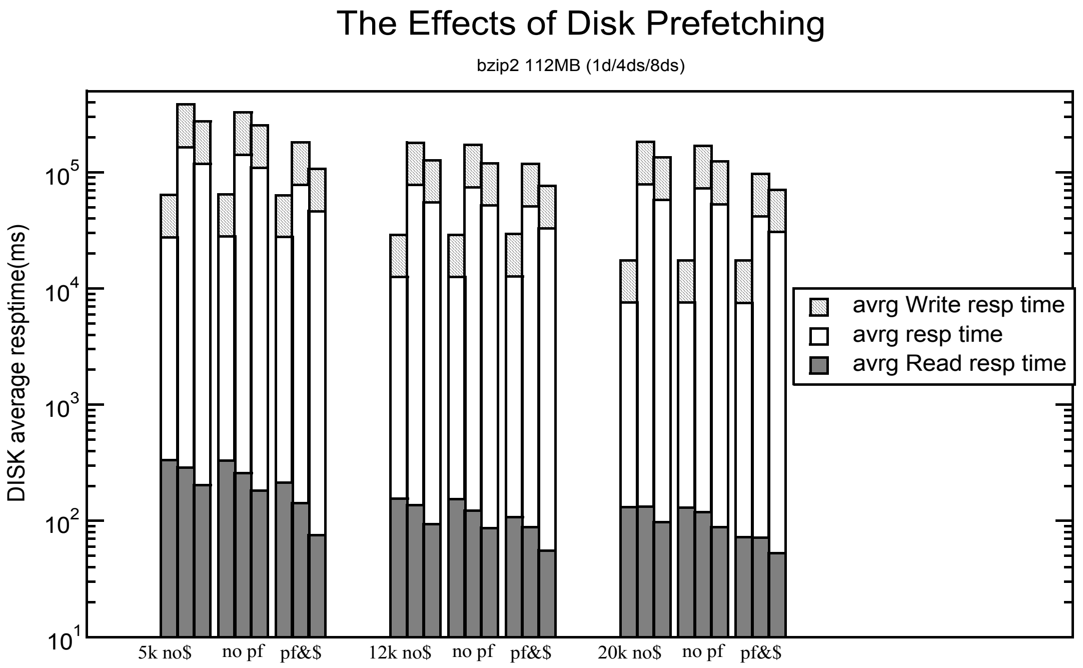


Figure 6.23: The Effects of Disk Prefetching. The experiment tries to identify the effects of prefetching and caching in the disk cache. The configuration is 112MB of memory running bzip2. The 3 bars in each group represent single disk system, 4-disk RAID5 system, and 8-disk RAID5 system. The figure above shows the CPI of each configuration, and the figure below shows the average response time of the disk requests. Note that the CPI axis is in linear scale, but the disk average response time axis is in log scale. The height of the each bar in the average response time graph is the absolute value.

over all configurations. Therefore, from this point on, we will only study disk cache with both caching and prefetching mechanism implemented. We refer to the Disk Cache that caches and prefetches as "Disk Cache" as referred to by the disk drive manufacturer.

As we focus on the RAID disk system in the next section, we will discuss another interesting behavior also demonstrated in Figure 6.23. The behavior is how the RAID disk system tends to have longer response time for disk writes due to parity calculations. This behavior will be discussed later. Despite longer response time for writes causing longer overall average response time, the overall performance is significantly improved.

To compare the behavior of the entire memory hierarchy in the system with disk prefetching, Figure 6.24 shows the interaction of the entire memory hierarchy in a system without disk cache, and Figure 6.25 shows the interaction of the components in a system with disk cache but no prefetching. Figure 6.26 illustrates the interaction of the memory hierarchy with disk caching and prefetching mechanism enabled. All system configurations have 112MB of memory running bzip2 with a single disk drive. All three figures demonstrate the power and accesses of caches, DRAM, disk, and the total system CPI. Both the systems without disk cache and with disk cache but no prefetching perform relatively the same as they complete the 500 million instructions in 500 seconds. With both disk caching and prefetching, the system can reduce the task execution time to 300 seconds. The contribution of disk caching and prefetching is mainly from its read absorbing behavior. The read bursts can be processed faster by the disk cache than by the mechanical parts in the disk, and this behavior results in shorter process time for read bursts. However, write bursts still take as much time as a system without disk cache, because they overwhelm the size of the disk cache.

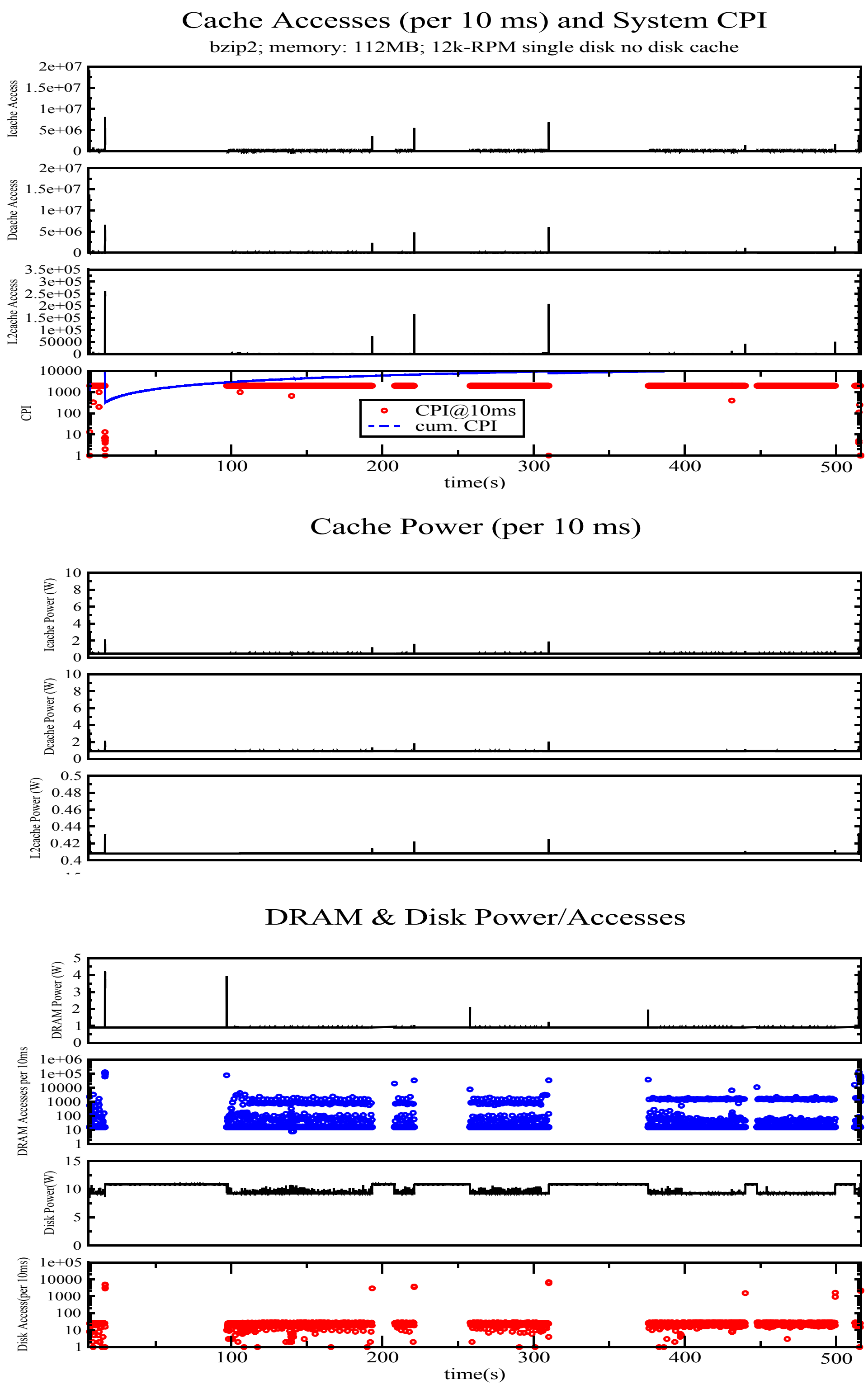


Figure 6.24: The interaction of the memory components in a system without disk cache.

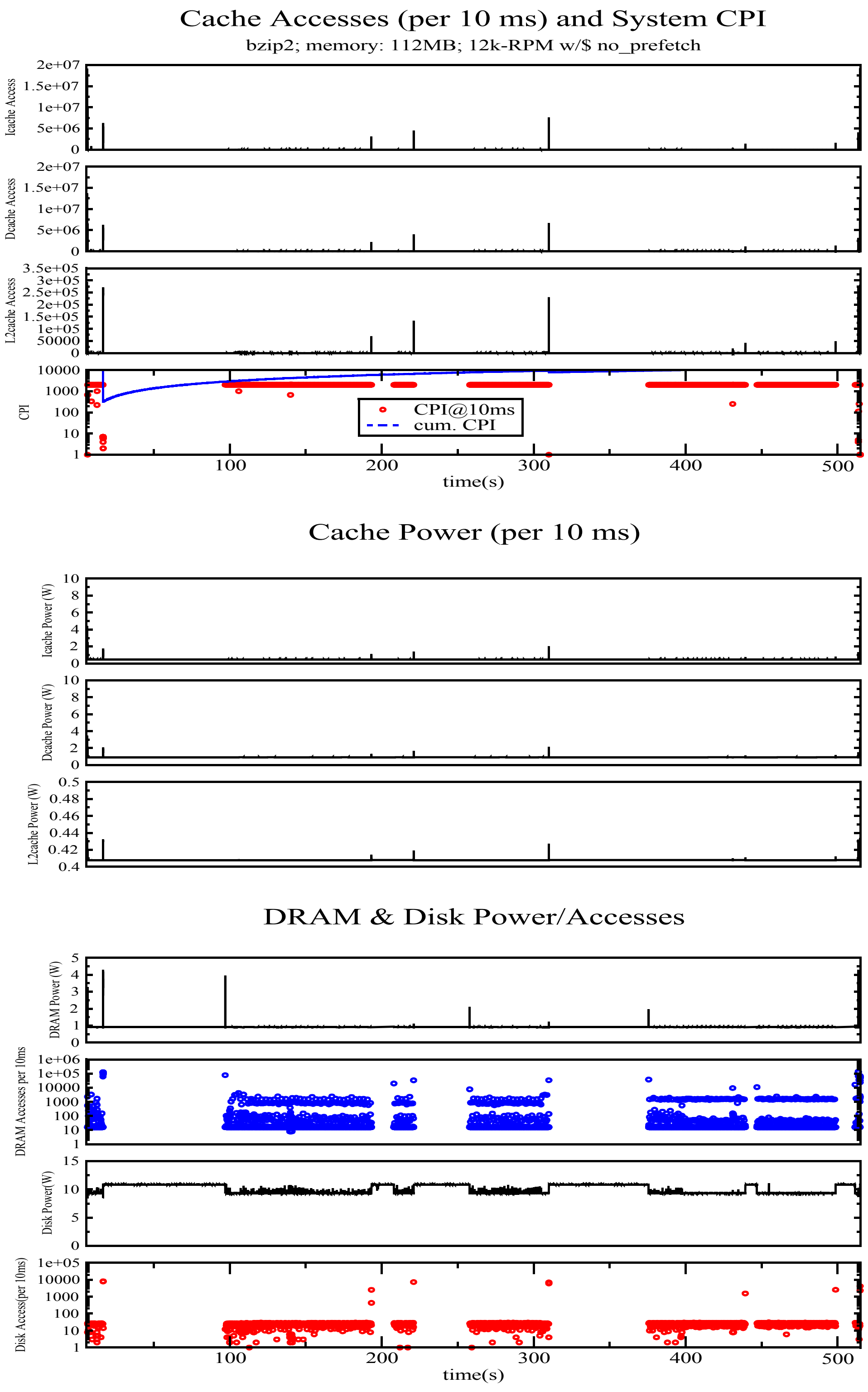


Figure 6.25: The interaction of the memory components in a system with disk cache but no prefetching.

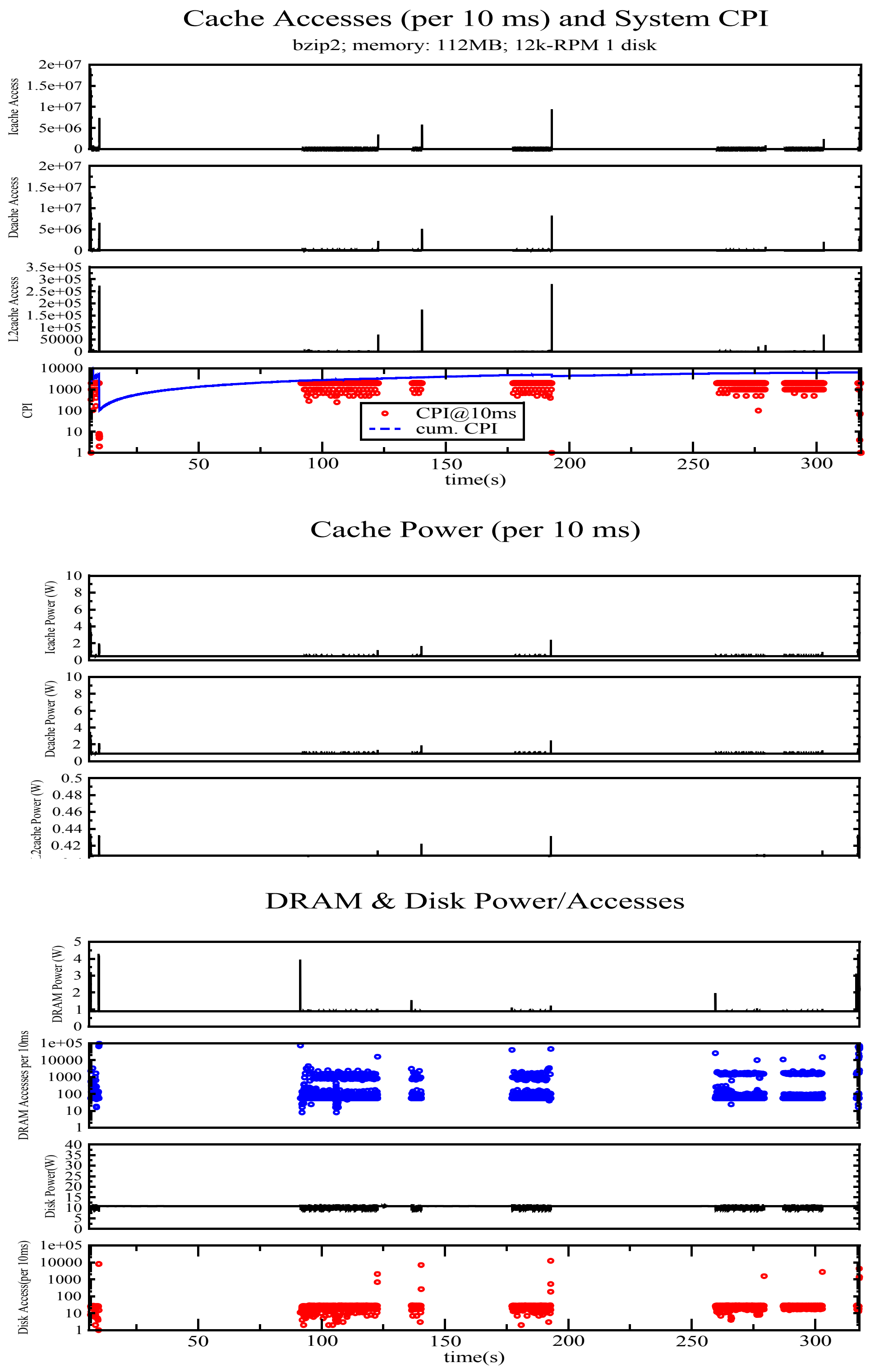


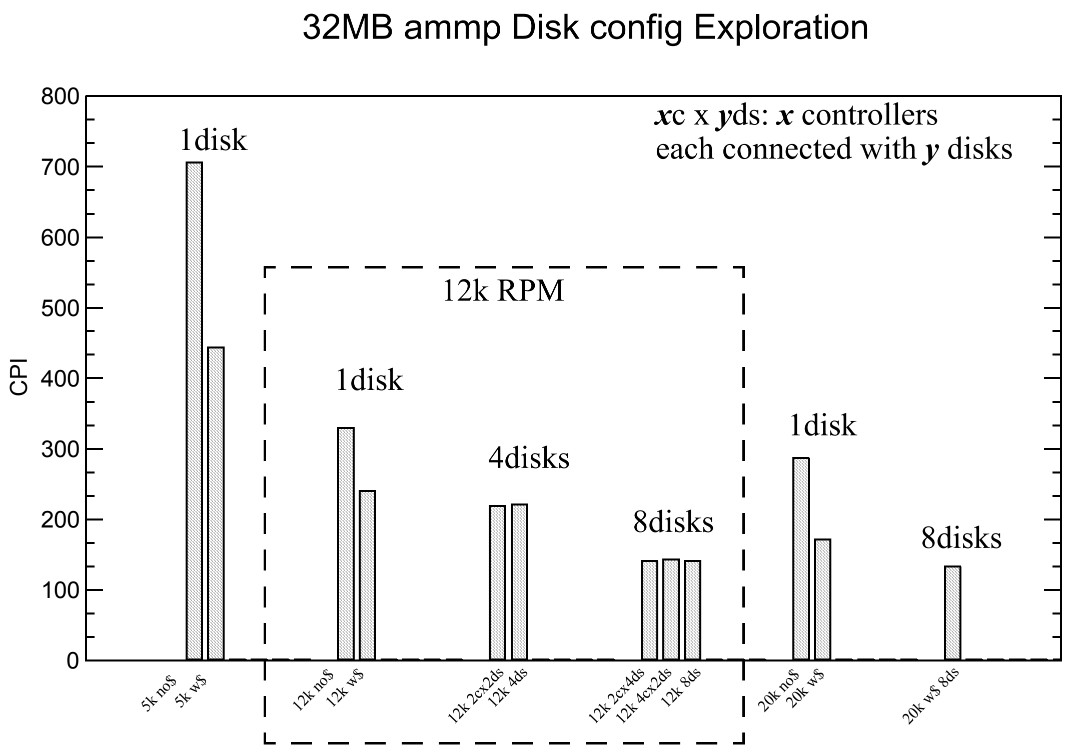
Figure 6.26: The interaction of the memory components in a system with disk caching and prefetching.

6.4.3. Parallel I/O: RAID5

In this section, we will discuss the RAID disk system and the RAID organization focusing only on RAID 5, which is the most popular RAID organization. We conducted the experiment with a single disk system, a 4-disk RAID system with 2 different organizations, and an 8-disk RAID system with 3 different organizations.

Figure 6.27 shows the CPI and the disk average response time for all RAID5 configurations described in the previous chapter. The system configuration used in this experiment is set to 32MB of memory running ammp with various RAID configurations made up of 12k-RPM disks with disk cache enabled. As illustrated in Figure 5.7, the label "***x***c x ***y***ds" refers to the RAID5 configuration with ***x*** controllers, each of which is connected with ***y*** disks. For comparison, the figure also shows the CPI and average response time of the system with 5400 RPM and 20k RPM. The first group was configured with a single 5400 RPM disk with and without disk cache. The last 2 groups were configured with 20k RPM disk(s). If the configuration is not labeled with "no$", each disk in each configuration has a 4MB disk cache. Like before, the graph above shows the CPI, and the graph below shows the disk average response time.

Let's consider the 12k-RPM disk system with the same number of disks. No matter how the disks are organized, the CPI and the average response time remains the same across the same number of disks. To explore the effects of increasing parallelism by increasing the number of disks, we move from 1 disk to 4 disks and from 4 disks to 8 disks. However, unlike Figure 6.23, increasing the number of disks from 1 to 4 seems not to have any



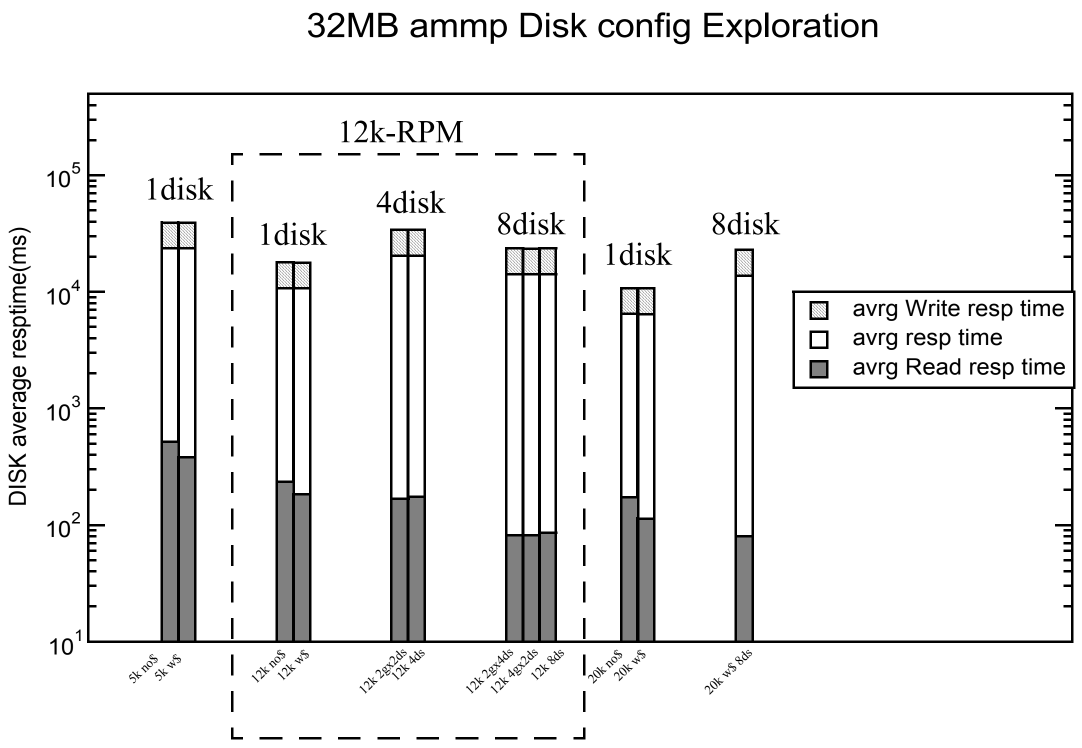
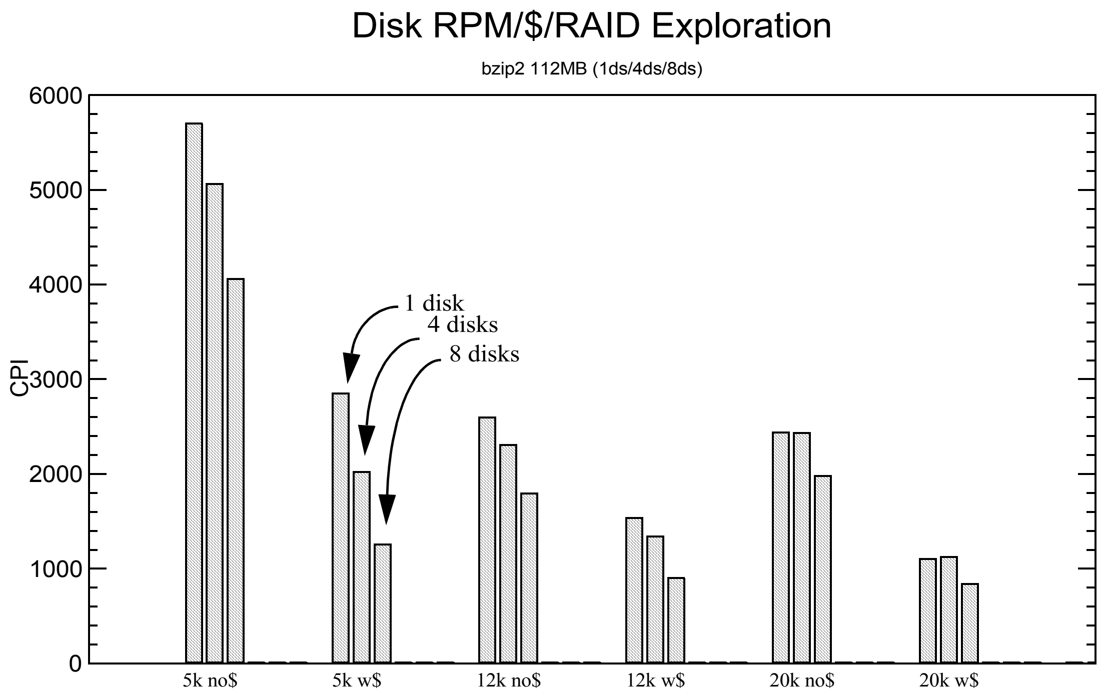


Figure 6.27: RAID5 configuration. The figure shows the effects of RAID5 configuration on the CPI and the disk average response time. The system configuration is 32MB of memory running ammp with 12k-RPM disk with disk cache. The figure also shows the results for different RPM disk for comparison.

obvious benefit compared to increasing from 4 to 8 in 12k-RPM disk systems. The possible explanation is that the complexity of scheduling and parity calculation overshadows the benefits of having multiple disks in the case of a 4-disk system in this application. As we move to 20k-RPM disk system, the benefits of having RAID system is less obvious even with 8 disks. Additionally, the performance of any 20k-RPM system is comparable to an 8-disk RAID system with 12k-RPM disks. Therefore, RAID in a high RPM disk system may have only marginal benefits over a system with a single high RPM disk. Care should be taken to choose the number of RAID disks in a uniprocessor system.

For the average response time, even though the write response time in a RAID system is much higher than the write response time in a single disk system, this trend does not translate directly into the overall performance. The write response time in a RAID system is higher due to parity calculations, especially the benchmarks with small writes. The cost of writing in a RAID system is significant [80]. If the cost of a write is reduced/eliminated, the overall system performance will be improved.

Figure 6.28 shows the CPI and the average response time of the disk systems with a different number of RAID disks and RPMs. The system configuration is set to 112MB of memory running bzip2. The 3 bars in each group represent a single disk system, a 4-disk system and an 8-disk system. Note, the CPI is in linear scale and the average response time is in log scale. Again, as the RPMs increase, the benefit of the RAID diminishes. As mentioned before, the benefits of having 4 RAID disks versus having only a single disk is not very obvious in a fast disk system. In a slow disk system (i.e., 5400 RPM), RAID has more tangible benefits over a non-RAID system. Nevertheless, the combination of using RAID, disk cache, and fast disks can improve the overall performance up to a factor of 10.



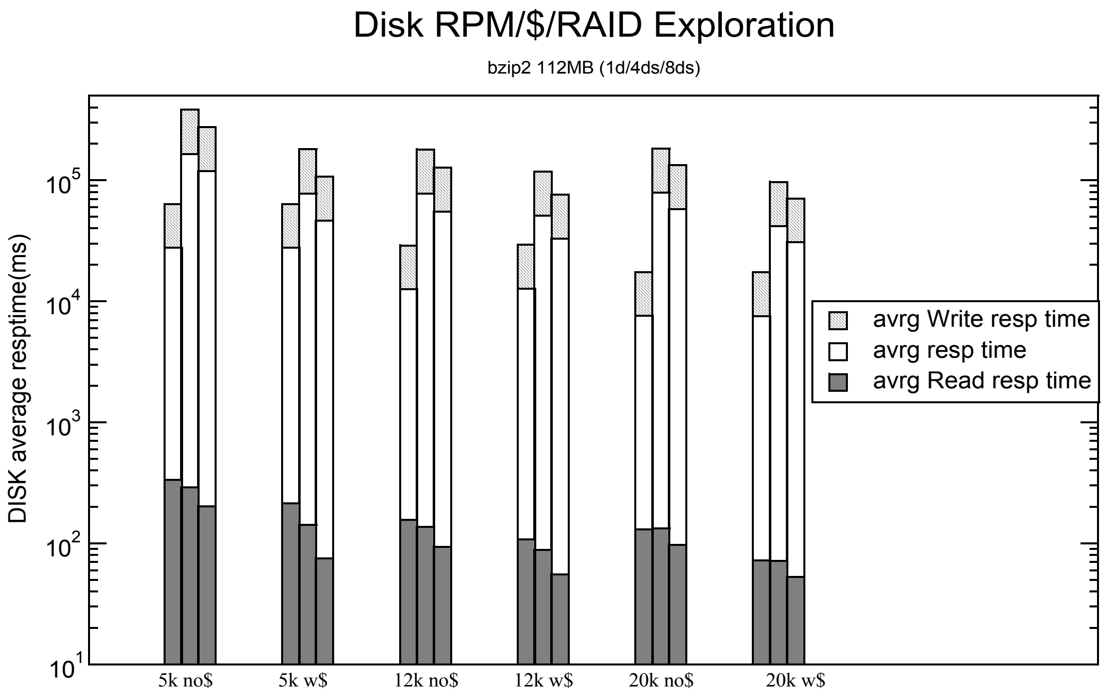


Figure 6.28: Disk RAID5 Configuration with different RPMs. The figure shows the CPI and the average response time of the disk systems. The system configuration is 112MB of memory running bzip2. The 3 bars in each group represent a single disk system, a 4-disk system and an 8-disk system.Note that the CPI is in linear scale and the average response time is in log scale.

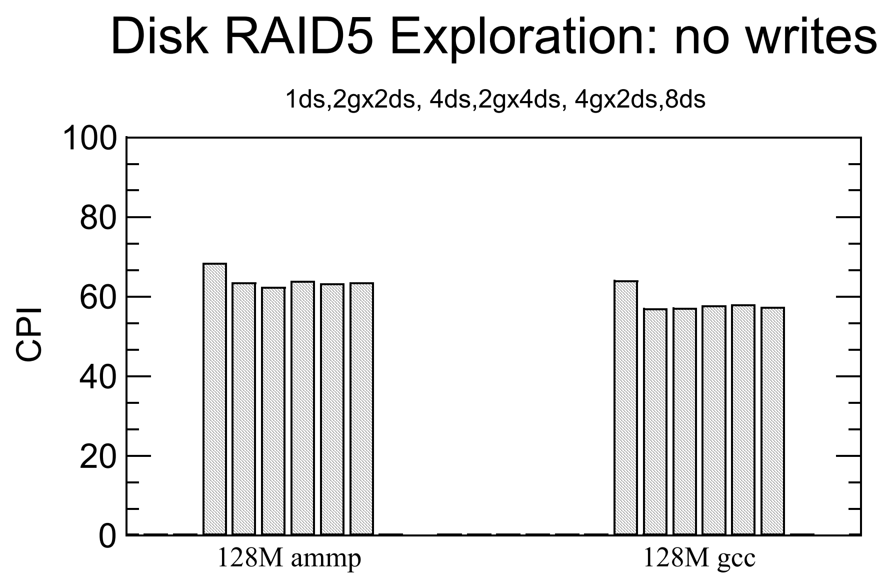


Figure 6.29: RAID5 with no writes. The figure show the effects of different RAID5 systems with benchmarks with only disk reads. The benchmarks are ammp and gcc with 128MB of memory. Obviously, RAID disk system has only minimal benefits over a single disk system.

Figure 6.29 shows the effects of RAID disk systems with different configurations on the total system CPI of the benchmarks with only disk reads. The benchmarks are ammp and gcc with 128MB of memory, and all disks in the system are equipped with a 4MB disk cache each. The RAID disk system has only minimal benefits over a single disk system since the disk requests are sequential. The reason is the caching and prefetching mechanism in the disk can hide most of the latency.

The interaction between the components in the memory hierarchy in the RAID disk systems are shown in Figure 6.30 and Figure 6.31. These two figures are to be compared against the system with a single 12k-RPM disk in Figure 6.26. Figure 6.30 shows the interaction results of a system with 4-disk RAID system, and Figure 6.31 shows the interaction results of a system with 8-disk RAID system over time. All system configurations are set to 112MB of memory running bzip2. Moving from a single disk

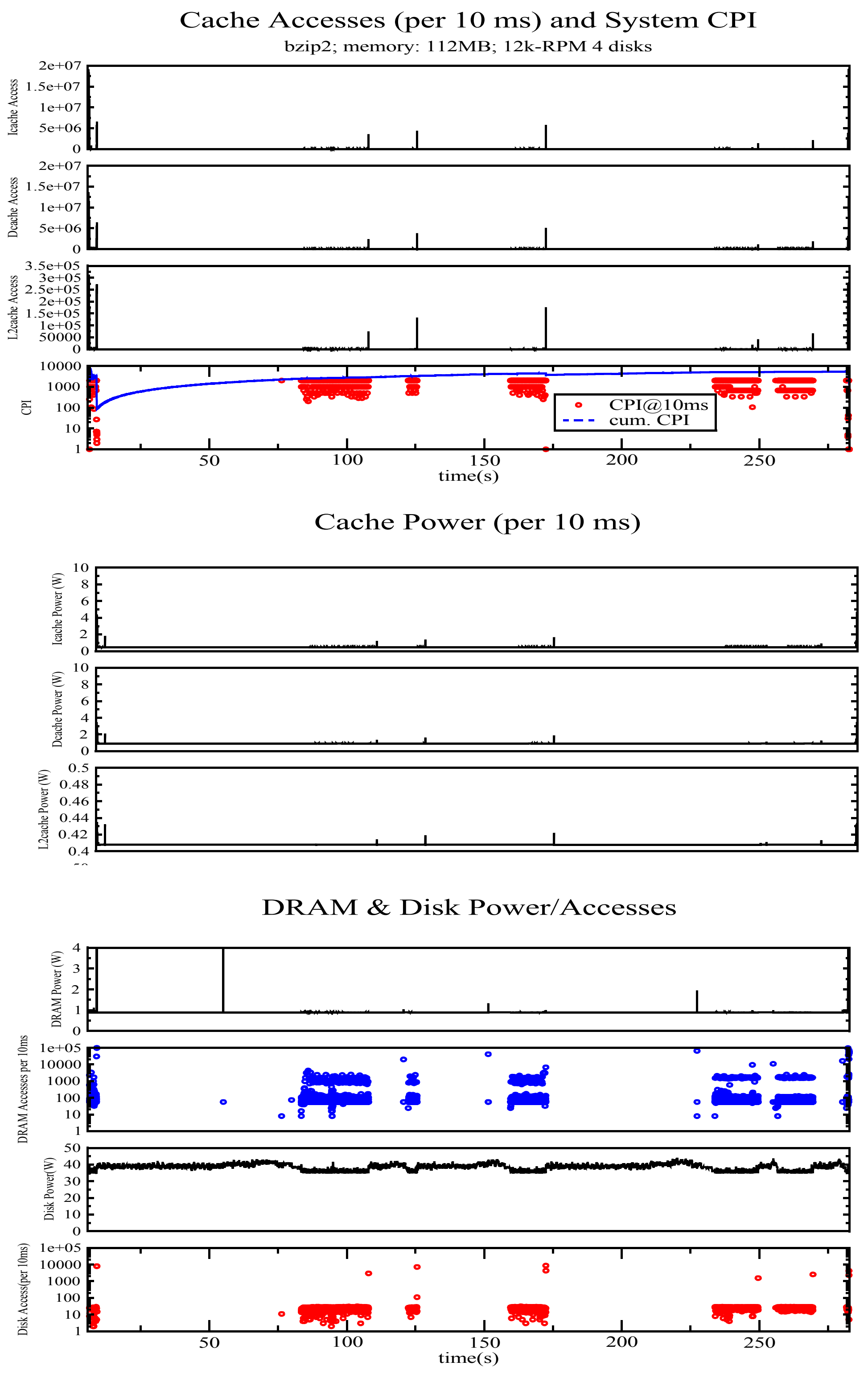
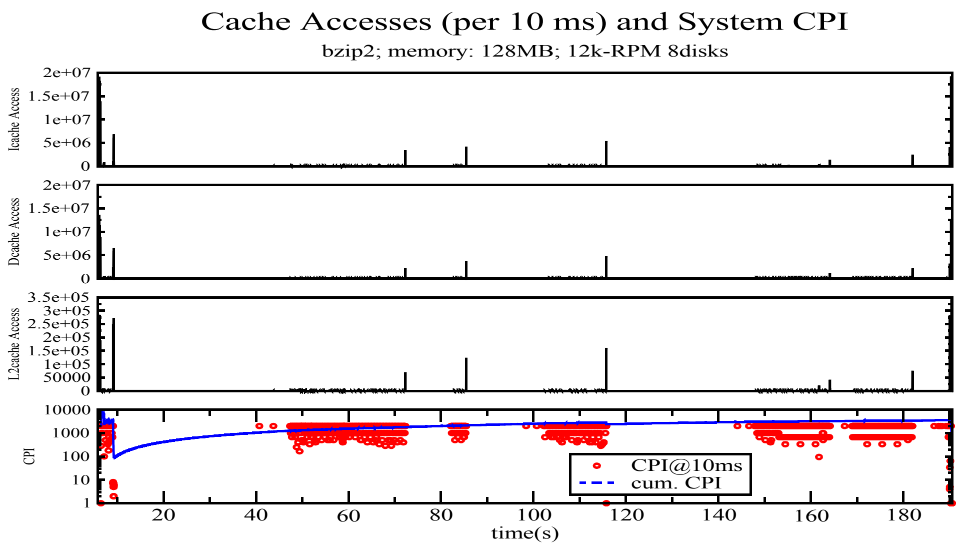
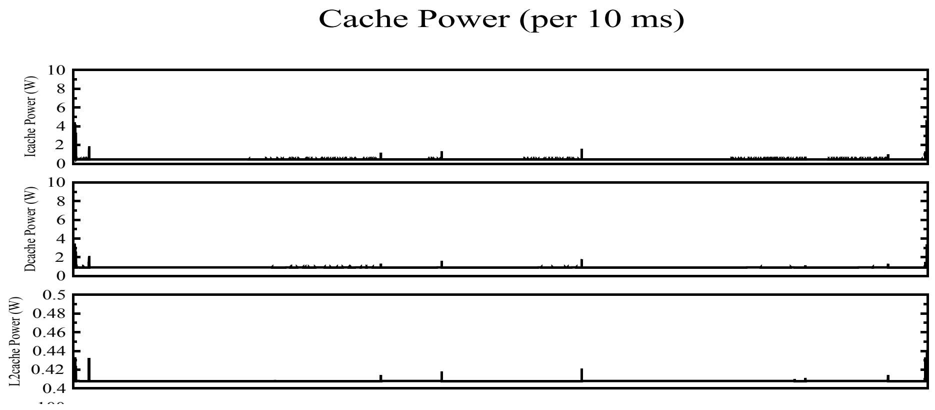


Figure 6.30: The interaction between the memory components in the hierarchy of a system with 4-disk RAID system.





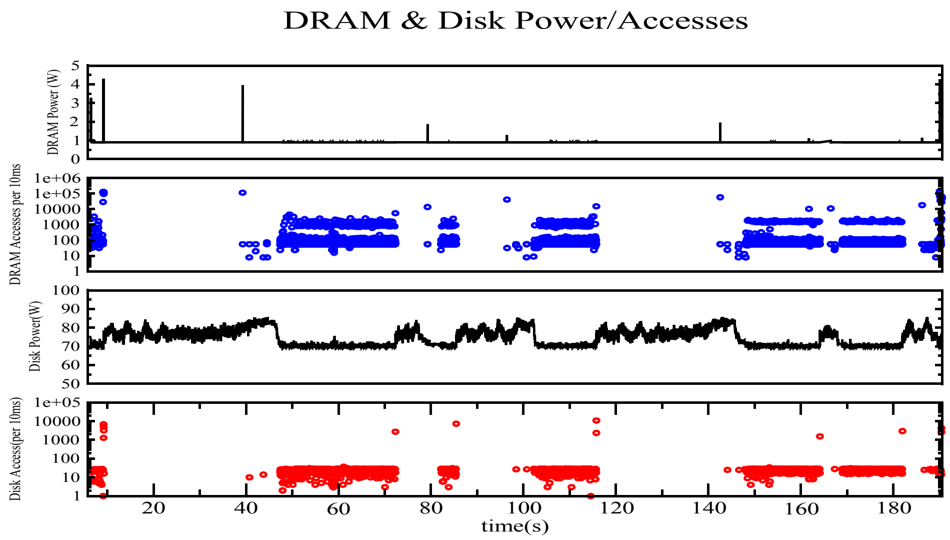


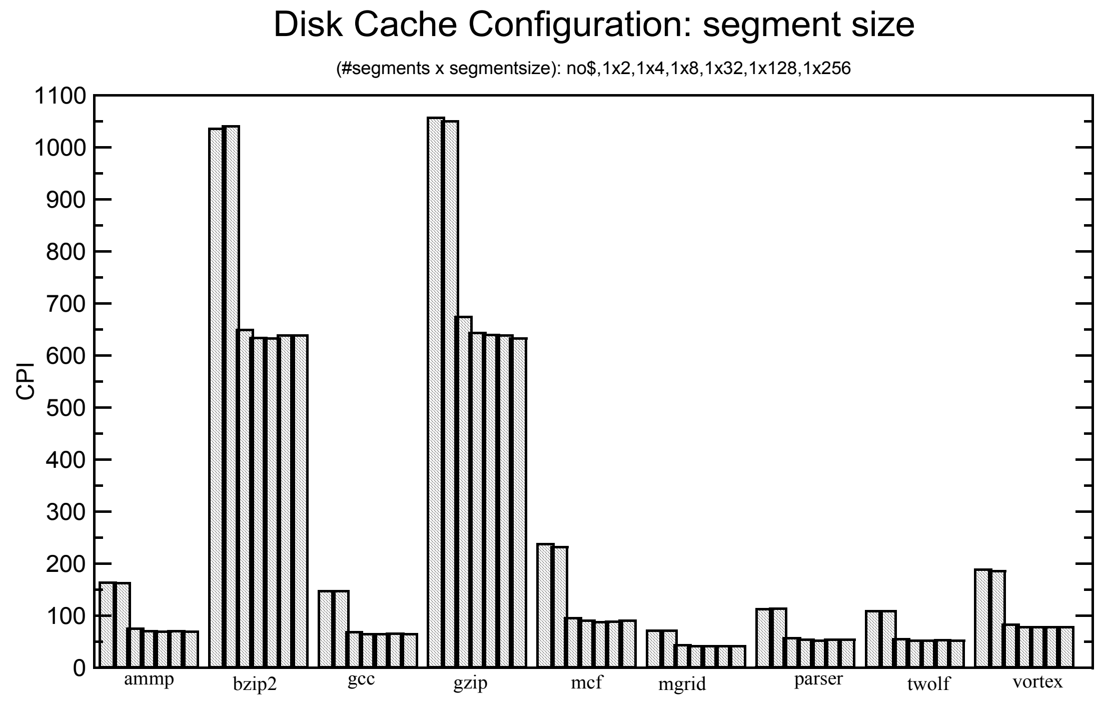
Figure 6.31: The interaction between the memory components in the hierarchy of a system with 8-disk RAID system.

system to a 4 disk system, the execution time improves from 325 seconds to 275 seconds approximately. Even better, an 8-disk system can improve the execution time to only roughly 190 seconds, approximately. Both read and write burst process times are improved since the bursts can be serviced faster. Despite the fact that the average response time for a single write is higher, the total performance including the benefit from parallelism in write bursts is also improved. However, the power consumption is also proportionally increased by the number of disks.

6.4.4. Size of the Disk Cache

We conducted a set of experiments to identify the effects of the disk cache size over the overall system performance. The size of the disk cache relates to the cost of the disk drive directly since the cost per GB for DRAM is currently about 50 times higher than for disk storage [78]. A disk cache is composed of a set of multiple segments. Each segment can vary in size in the unit of sector (512 bytes in general). Compared with processor cache, a segment can be seen as a cache line, and the segment size is therefore the same as a cache line size. As a result, to vary the disk cache size, we have to vary the number of segments and the segment sizes themselves.

Figure 6.32 shows the performance impacts of Segment Size Variation. The figure shows the impacts of different segment sizes with the same number of segments in the disk cache. In this case, there is only one segment. The system configuration is set to 128MB of memory with a 12k-RPM disk. There are 7 bars for each benchmark, which are (1) no cache, (2) 1 segment of 2 sectors each, (3) 1 segment of 4 sectors each, (4) 1 segment of 8 sectors each, (5) 1 segment of 32 sectors each, (6) 1 segment of 128 sectors each, and (7) 1 segment



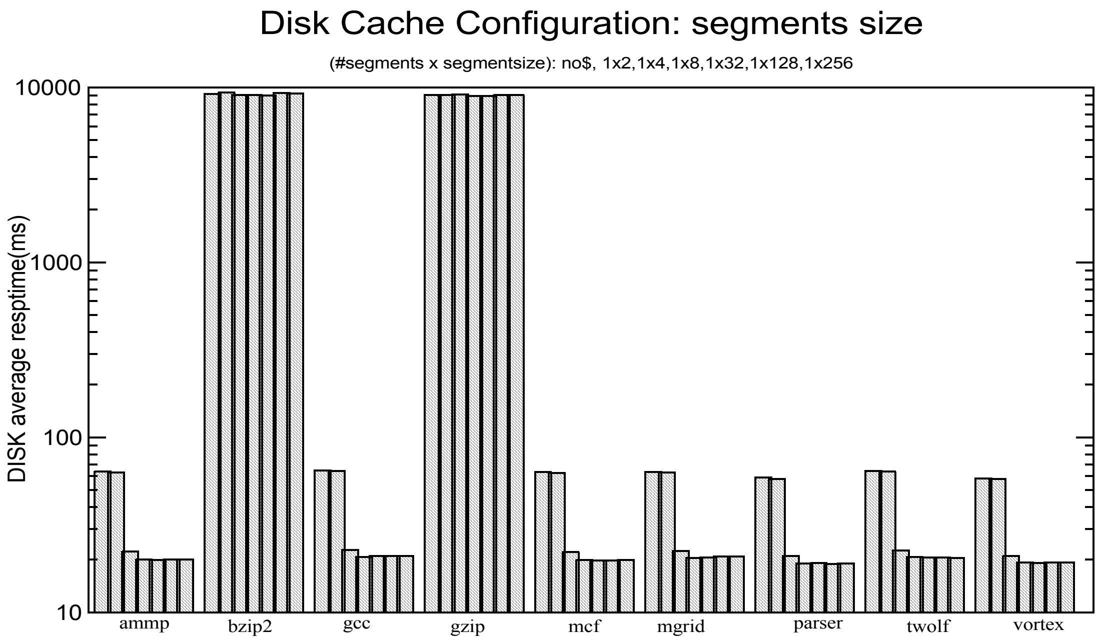


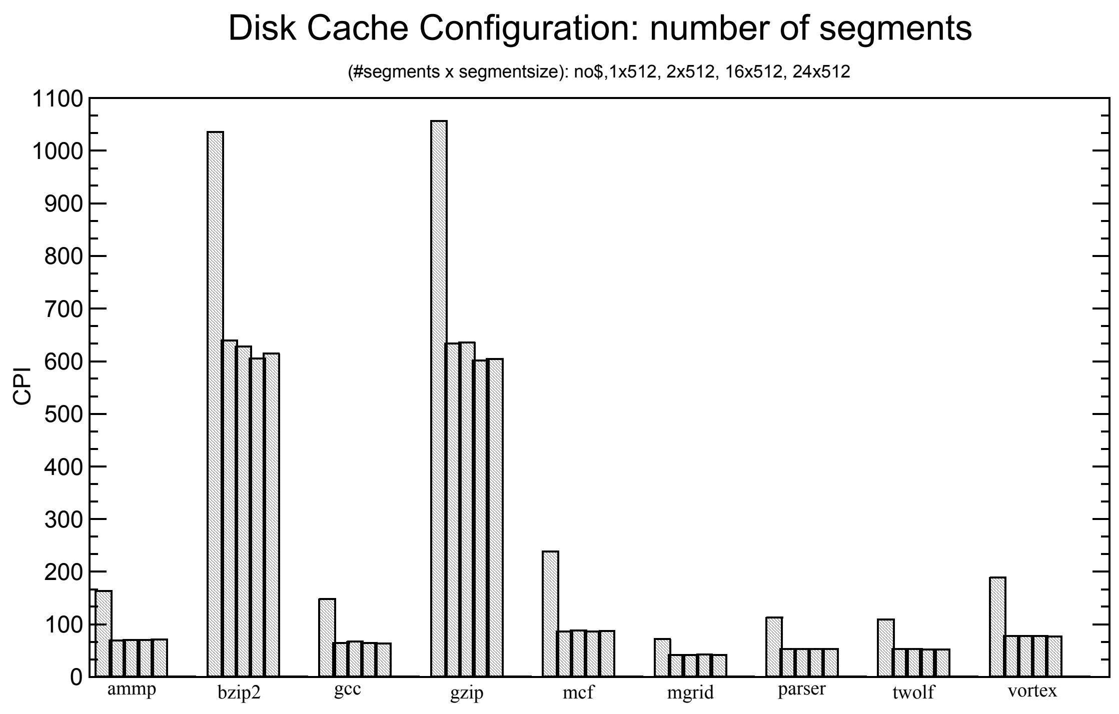
Figure 6.32: The Effects of Disk Cache Size by varying the Segment Size. The figure shows the effects of different segment size with the same number of segments in the disk cache. The system configuration is 128MB of memory with a 12k-RPM disk. There are 7bars for each benchmark, which are (1) no cache, (2) 1segment of 2 sectors each, (3) 1segment of 4 sectors each, (4) 1segment of 8 sectors each, (5) 1segment of 32 sectors each, (6) 1segment of 128 sectors each, and (7) 1segment of 256 sectors each. Note that the CPI graph is in linear scale, and the average response time graph is in log scale.

of 256 sectors each. The top graph shows the CPI and the bottom graph shows the average response time of the disk requests.

Figure 6.33 entitled the Effects of Disk Cache Size by Varying the Number of Segments. The figure shows the effects of different number of segments with the same segment size in the disk cache, which is 512 sectors. The system configuration is set to 128MB of memory with a 12k-RPM disk. There are 5 bars for each benchmark, which are (1) no cache, (2) 1 segment of 512 sectors each, (3) 2 segments of 512 sectors each, (4) 16 segments of 512 sectors each, and (5) 24 segments of 512 sectors each. Note that the CPI graph is in linear scale, and the average response time graph is in log scale.

The effect of the disk cache size is limited to the presence of the cache with a particular size. Meaning, increasing the size of the disk cache, either by segment size or by the number of segments, will not result in a better performance if the disk cache is already large enough as described in [75] and [78]. They simply concluded that, with a reasonably sized file system buffer cache controlled by the operating system, there is very little performance benefit of using a big built-in disk cache. Our study agrees with those observations, but from the system-level point of view. Another interesting behavior to point out is, for the benchmark with disk reads and writes, i.e. bzip2 and gzip, the average response times are always the same while the overall CPIs improve due to the presence of the disk cache. This behavior will be explained in the next section.

Figure 6.34 shows the trade-offs between Memory Sizes and Disk Cache Sizes. The top graph shows the trade-offs between the memory sizes and the disk cache sizes under the assumption that the total in megabytes of memory and disk cache remain the same. We also varied the number of disks in a RAID5 disk systems. In this experiment, the total in



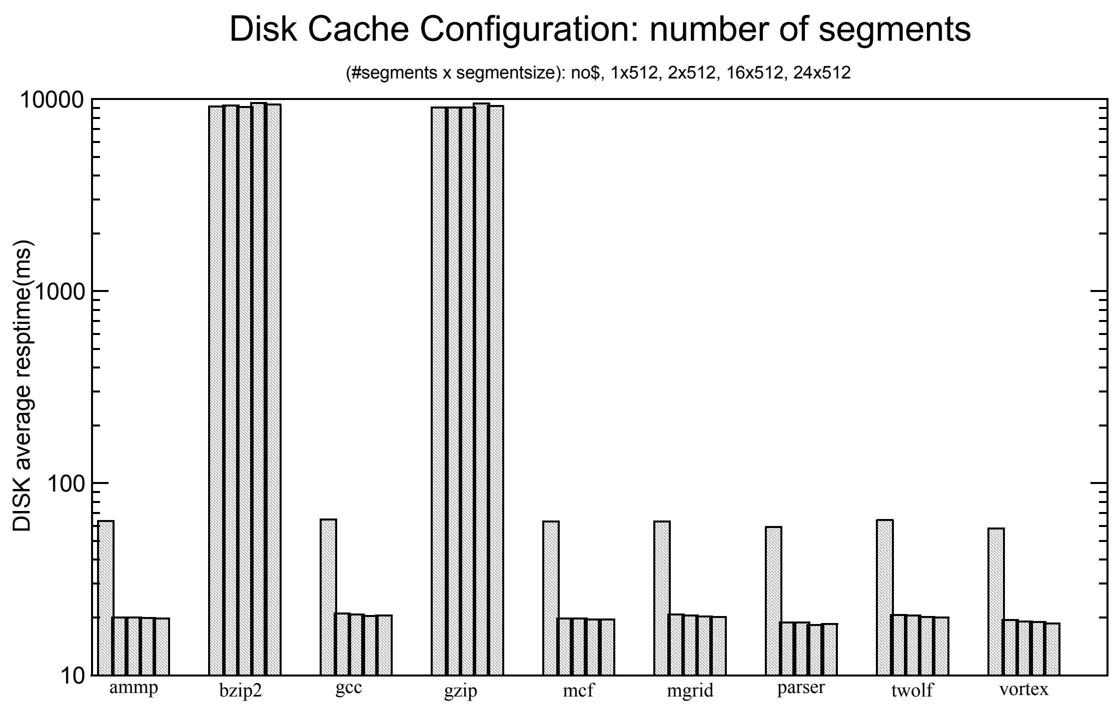
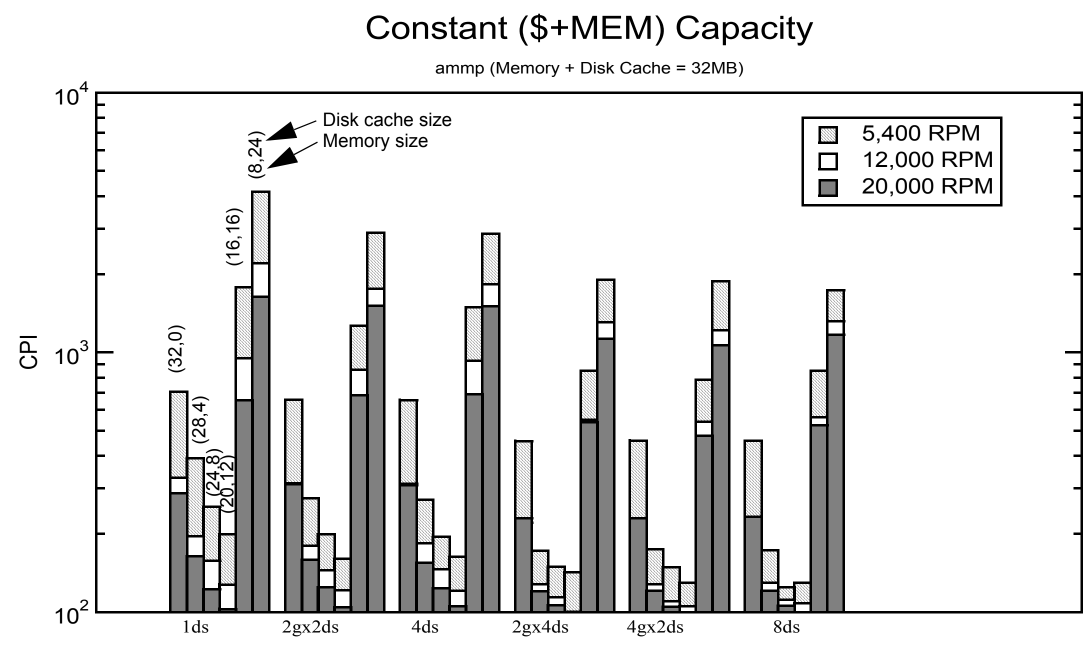


Figure 6.33: The Effects of Disk Cache Size by varying the Number of Segments. The figure shows the effects of different number of segments with the same segment size in the disk cache. The system configuration is 128MB of memory with a 12k-RPM disk. There are 5 bars for each benchmark, which are (1) no cache, (2) 1segment of 512 sectors each, (3) 2 segments of 512 sectors each, (4) 16 segment of 512 sectors each, and (5) 24 segment of 512 sectors each. Note that the CPI graph is in linear scale, and the average response time graph is in log scale.



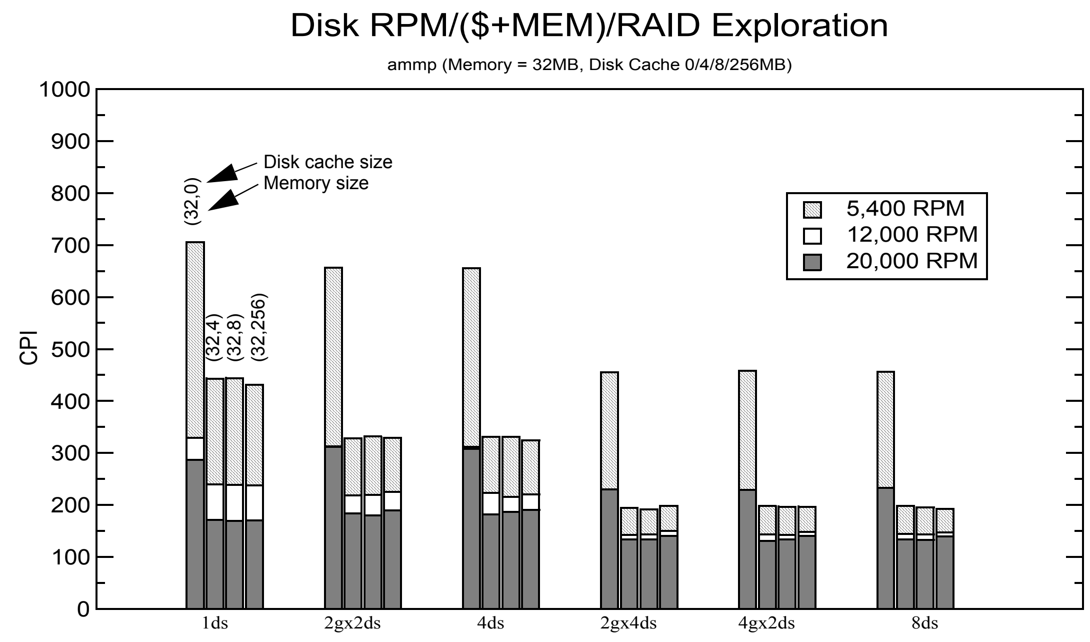


Figure 6.34: The Trade--offs between Memory Sizes and Disk Cache Sizes. The graph above shows the trade-off between the memory size and the disk cache size under the assumption that the total MB of the memory and the disk cache remains the same. In this case, the total MB is 32MB on an ammp execution. The graph below shows the effects of disk cache size over a RAID5 disk system. The system also has 32MB of memory running ammp.

megabytes used on an ammp execution is 32MB. From left to right in each RAID configuration, each bar represents the CPI of (1) 32MB of memory with no disk cache, (2) 28MB of memory with 4MB of disk cache in total, (3) 24MB of memory with 8MB of disk cache in total, (4) 20MB of memory with 12MB of disk cache in total, (5) 16MB of memory with 16MB of disk cache in total, and (6) 8MB of memory with 24MB of disk cache in total. Each bar has 3 bars that are overlapping, which are for 5k-RPM, 12k-RPM, and 20k-RPM disk system. Note, the CPI is in log scale.

The bottom graph shows the effects of disk cache size on RAID5 disk systems. The system is also set to have 32MB of memory constantly and is running ammp while we varied the size of the disk cache on each disk drive. Each bar in each RAID configuration represents (1) 32MB of memory with no disk cache, (2) 32MB of memory with 4MB of disk cache for each disk, (3) 32MB of memory with 8MB of disk cache for each disk, and (4) 32MB of memory with 256MB of disk cache for each disk.

Unlike in Figure 6.33, the experimental results in Figure 6.34 suggest that increasing disk cache size will result in better performance most easily seen in the top graph. Increasing disk cache size is not a step function in this case. The reason is the application in Figure 6.33 has only minor write traffic to the disk, but the write traffic in Figure 6.34 is increasing with the reduced memory size. Therefore, increasing disk cache is beneficial for write traffic. However, the effect remains true until the system reaches the memory limit where the DRAM can no longer contain significant portion of the memory footprint. At that point, the CPI increases rapidly. Also, the bottom graph suggests that, at a particular memory size, only a relatively small amount of disk cache can improve the performance greatly.

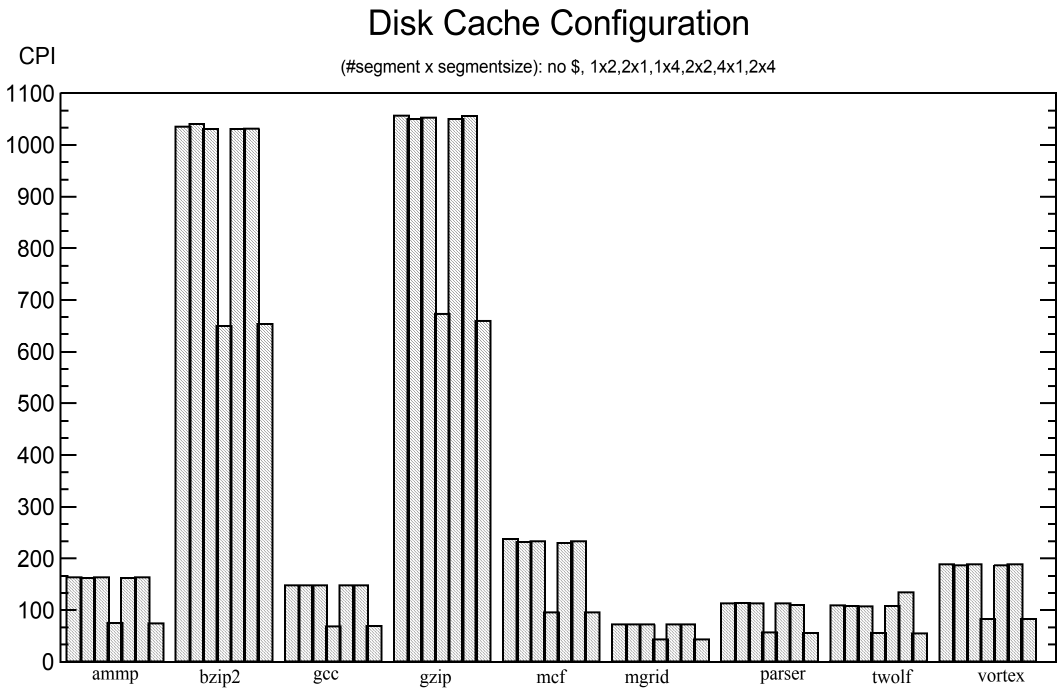
Increasing the disk cache further would not positively affect the performance. These facts are also true with multiple disks in the RAID disk systems.

6.4.5. Disk Cache Organization

In the previous section, we conclude that the size of the disk cache, either by segment size or by the number of segments, does not have significant effects to the system performance. Only a small disk cache size is enough for the disk caching and prefetching. This section explores the choice of cache organization to see if that has any more of an effect than cache size.

To answer this question, we also conducted an experiment to identify the effects of the disk cache organizations. From Figure 6.32 and Figure 6.33, we learned that only a small disk cache, i.e. 1 segment with 4 sectors, can improve the performance. Therefore, we did an experiment to see the effects of the cache organizations around the 4-sector case. Figure 6.35 shows the effects of disk cache organization around this case. The figure shows both the CPI (above graph) and the average disk response time (below graph). The 7 bars on each benchmark in the graph represent the disk system with (1) no cache, (2) disk cache with 1 segment with 2 sectors each, (3) disk cache with 2 segments with 1 sector each, (4) disk cache with 1 segment with 4 sectors each, (5) disk cache with 2 segments with 2 sectors each, (6) disk cache with 4 segments with 1 sector each, (7) disk cache with 2 segments with 4 sectors each.

From the experimental results, we can conclude that only the *1 segment with 4 sectors* and the *2 segments with 4 sectors* cases improved the performance. Both of them gained the same improvements over other cases. For other cases, no significant system performance



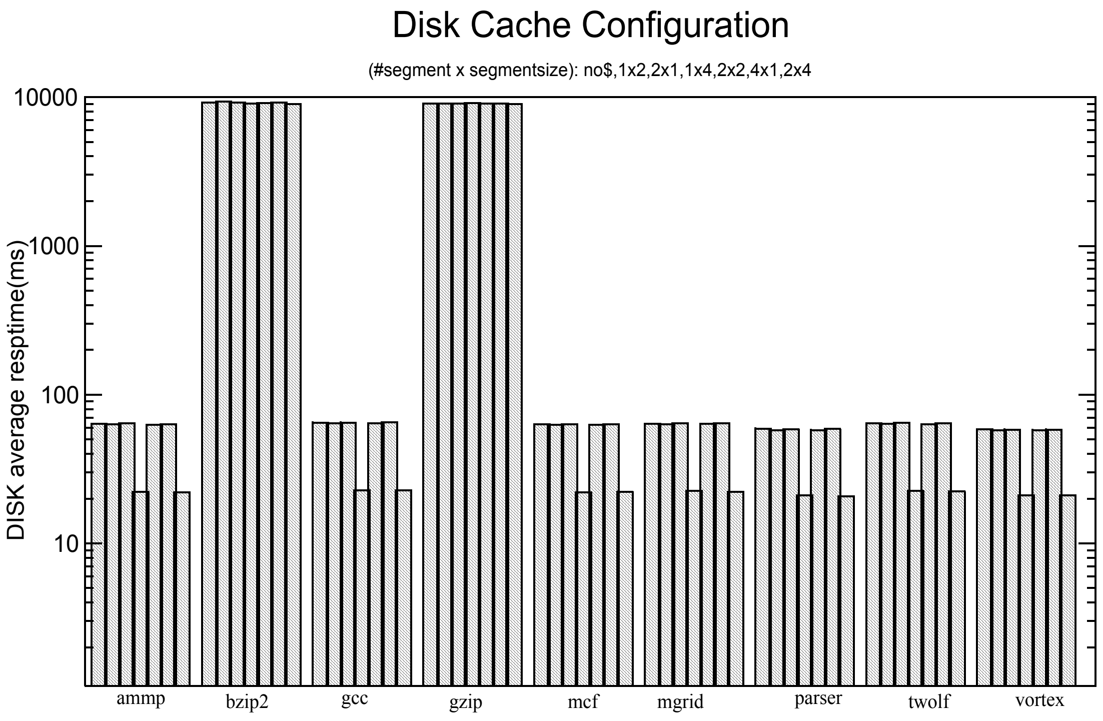


Figure 6.35: Disk Cache Organization. The figure shows both the CPI (above graph) and the average disk response time (below graph). The 7 bars on each group in the graph represent (1)no cache, (2) 1 segment with 2 sectors each, (3) 2 segments with 1 sectors each, (4) 1 segment with 4 sectors each, (5) 2 segments with 2 sectors each, (6) 4 segment with 1 sectors each, (7) 2 segments with 4 sectors each.

effects were exhibited. Therefore, the disk cache with the same size but different configurations, i.e. 1 segment with 4 sectors each, 2 segments with 2 sectors each, and 4 segment with 1 sectors each, perform differently. The configurations exhibiting benefits included the configurations with the segment sizes of 4 sectors. In conclusion, the only cache organization parameter that matters is the size of the segment.

6.4.6. Bus Transmission Latency

As suggested in [35], the bus transmission latency in the DRAM system has a significant effect on the overall system performance. We conducted an experiment to identify the effects of the bus transmission latency on the disk system. Figure 6.36 shows the

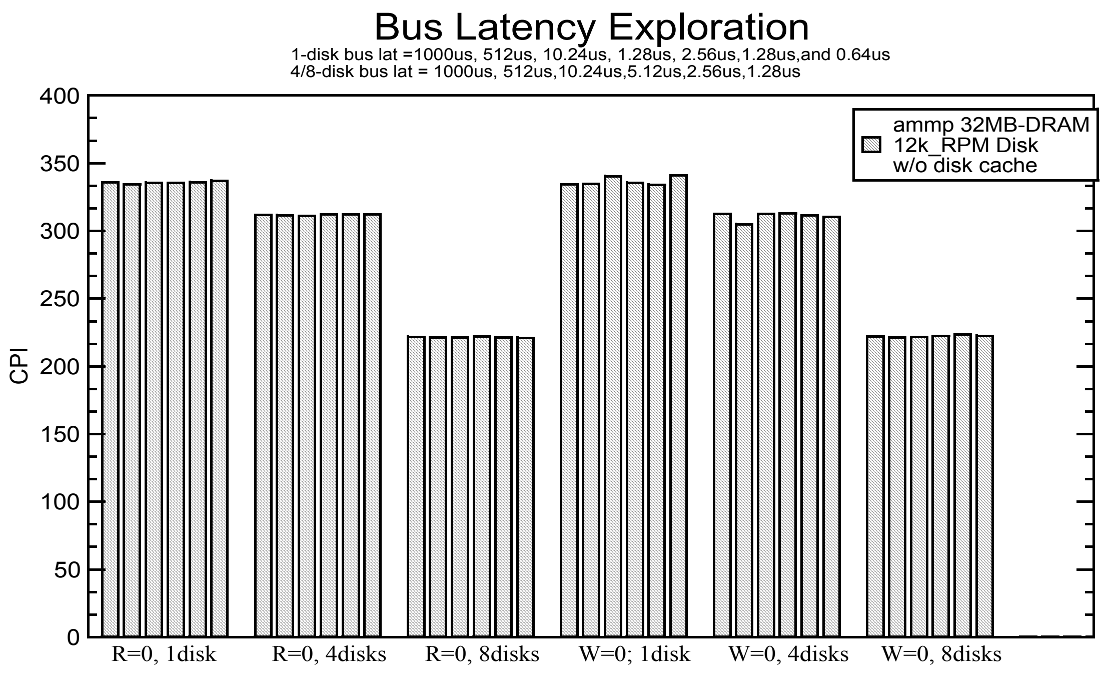


Figure 6.36: Bus Latency Exploration. The graph shows the effects of the bus latency variation to the total system CPI. The system configuration is 32MB of memory running ammp with a 12k-RPM RAID disk system without disk cache. The groups marked as "R=0" represent the read bus latency (data from the disk) taking no time but the write bus latency (data to the disk) taking varied latency, and the groups marked as "W=0" represent the write bus latency to the disk take no time but the read bus latency taking varied latency. The groups marked as "1 disk" has a single disk system,"4 disks" has a 4-disk RAID system, and "8 disks" has an 8-disk RAID system. The bus latency varies from 1 millisecond to 0.64 microseconds for a single disk system, and from 1 milliseconds to 1.28 microseconds for a 4-disk and 8-disk system.

Bus Latency Exploration. The graph shows the effects of the bus latency variation in regards to the total system CPI. The system is configured with 32MB of memory running ammp with a 12k-RPM RAID disk system without disk cache. The groups marked as "R=0" represent the read bus latency (data from the disk) taking no time but the write bus latency (data to the disk) taking varied time lengths to transfer one sector of data. The groups marked as "W=0" represent the write bus latency to the disk taking no time but the read bus latency taking varied time lengths to transfer one sector of data. The groups marked as "1 disk" equipped with a single disk system,"4 disks" equip a 4-disk RAID system, and "8 disks" equipped with an 8-disk RAID system. The bus latency varies from 1 millisecond to 0.64 microseconds for a single disk system, and from 1 millisecond to 1.28 microseconds for a 4-disk system and an 8-disk system. These bus latency values are set according to the range of the latency in the latest disk interface latencies showing in Table 4.2.

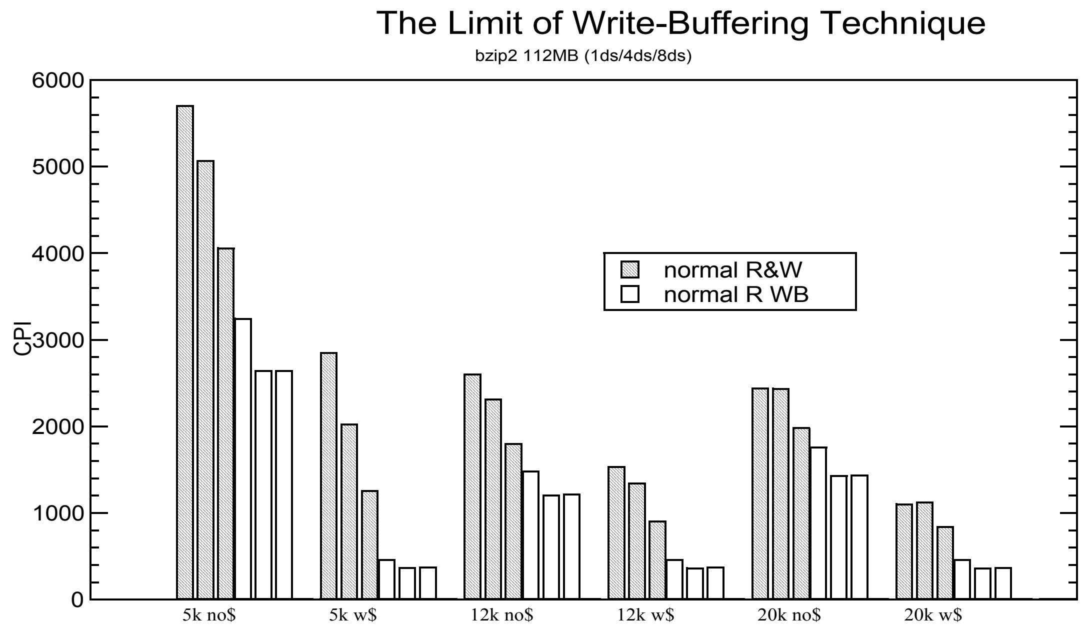
Even though the bus latency has a significant effect on the overall performance in the case of the DRAM systems, there are no significant effects of the bus latency variation on the disk system in regards to the total system CPI. The reason is the bus latency in the DRAM system is comparable to the DRAM latency. However, the bus latency in the disk system, which is in the unit of microseconds, is insignificant compared to the disk latency, which is in the unit of milliseconds.

6.4.7. Perfect Write-Buffering

The I/O subsystem is becoming a bottle-neck in the computer system due to the rapid growth in the processor speed and technology. Disk cache has been used to fill this gap, but the benefit is limited to the read operations as the write I/Os are usually committed to disk to maintain consistency and to allow for crash recovery. Therefore, disk writes basically are clogging the disk system. There are many publications [76, 77, 79, 82] suggesting to use the write-buffering techniques. Such techniques are aimed at hiding the disk write latency by writing to a buffer instead of writing to the disk immediately. To maintain the non-volatile concept of the disk, the buffer is required to withstand possible failures that could happen before the data are written to the disk. Using non-volatile RAM (NVRAM), a disk cache disk (DCD), or a NAND Flash memory are possible options for these techniques. All write-buffering publications measure the techniques against the disk subsystem metric such as the disk request response time and the disk system throughput. We conducted an experiment to exhibit the limit of such techniques to the overall system performance, by modeling a perfect write buffering, which can completely hide the write latency. The system was configured with 112MB of memory running bzip2 with a choice of 1, 4, or 8-disk RAID system. We also varied the existence of the disk cache to isolate the effects.

Figure 6.37 shows the limited effects of write-buffering, assuming that all writes to the disk system are buffered perfectly to eliminate the need to write to the disk immediately. The top graph shows the total system CPI, and the bottom graph shows the disk average response time. The CPI graph compares the CPI of the perfect write-buffering techniques with the system with normal disk reads and writes. For the disk average response time graph, the total average response time and the read response time are similar since the write latency to the disk system is hidden perfectly.

Our study shows that using some techniques to buffer the writes can improve the performance greatly, up to a factor of 10 in the case of a single 5k-RPM disk with disk caching. Additionally, buffering write requests decreases the needs for many disks to exist in a RAID system since both 4-disk and 8-disk systems perform similarly. Note, this may not be true in multiprocessor environment.



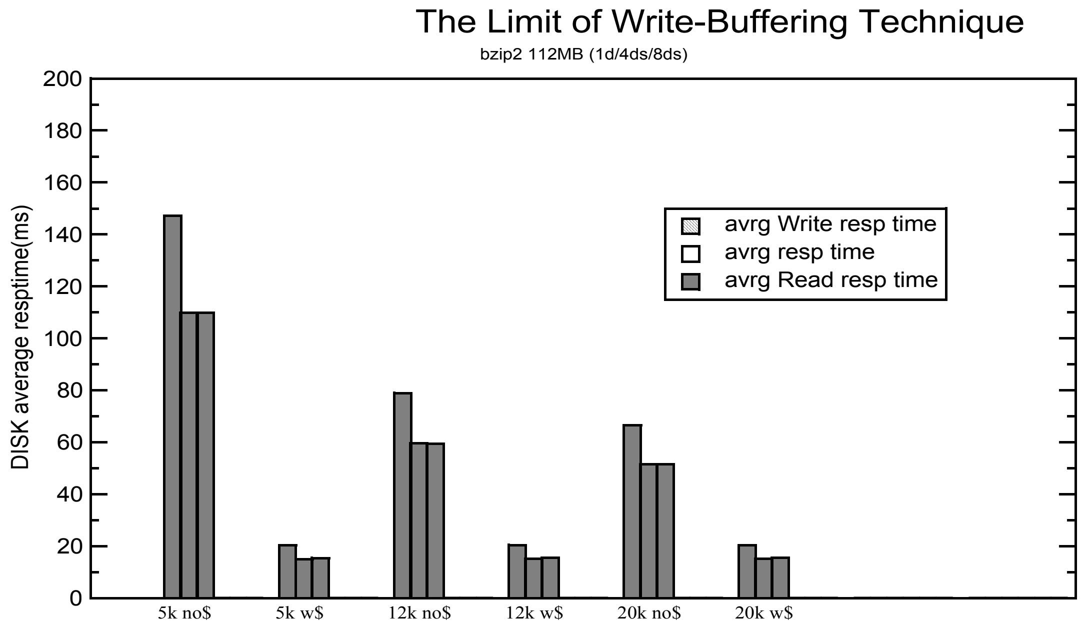


Figure 6.37: The Limit of Write-Buffering Technique. The figure shows the limited effects of write-buffering technique. The above graph show the CPI of both a normal systems and a write-buffering system, and the graph shows the disk average response time of only the write-buffering system. The CPI graph compares the limit with the system with normal disk reads and writes. For the disk average response time graph, the total average response time and the read response time are the same since the writes to the disk system are eliminated.

The graph also shows that if we can perfectly buffer write requests to the disk along with obtaining a small amount of cache to the disk, the performance will significantly be improved up to an order of magnitude. Furthermore, all the CPIs of any RPM disks remain the same no matter how fast the disks are. In conclusion, write-buffering technique can improve the performance immensely without the cost of multiple fast and expensive disks.

To be compared with our base case in Figure 6.26, Figure 6.38 shows the interaction of the system with a single disk with perfect write buffering. Figure 6.39 shows the interaction of the same configuration with a 4-disk RAID system, and Figure 6.40 shows the interaction of the configuration with 8-disk RAID system. All system configurations have 112MB of memory running bzip2. The write buffering eliminates the needs to write to disks. Therefore, reads can be performed immediately. This improves the execution time from 325 seconds, in the case of the system with a single disk with disk cache, to 100 seconds, in the case of the single-disk system with disk cache and write buffering. The rest of the components in the hierarchy remain dissipating low power.

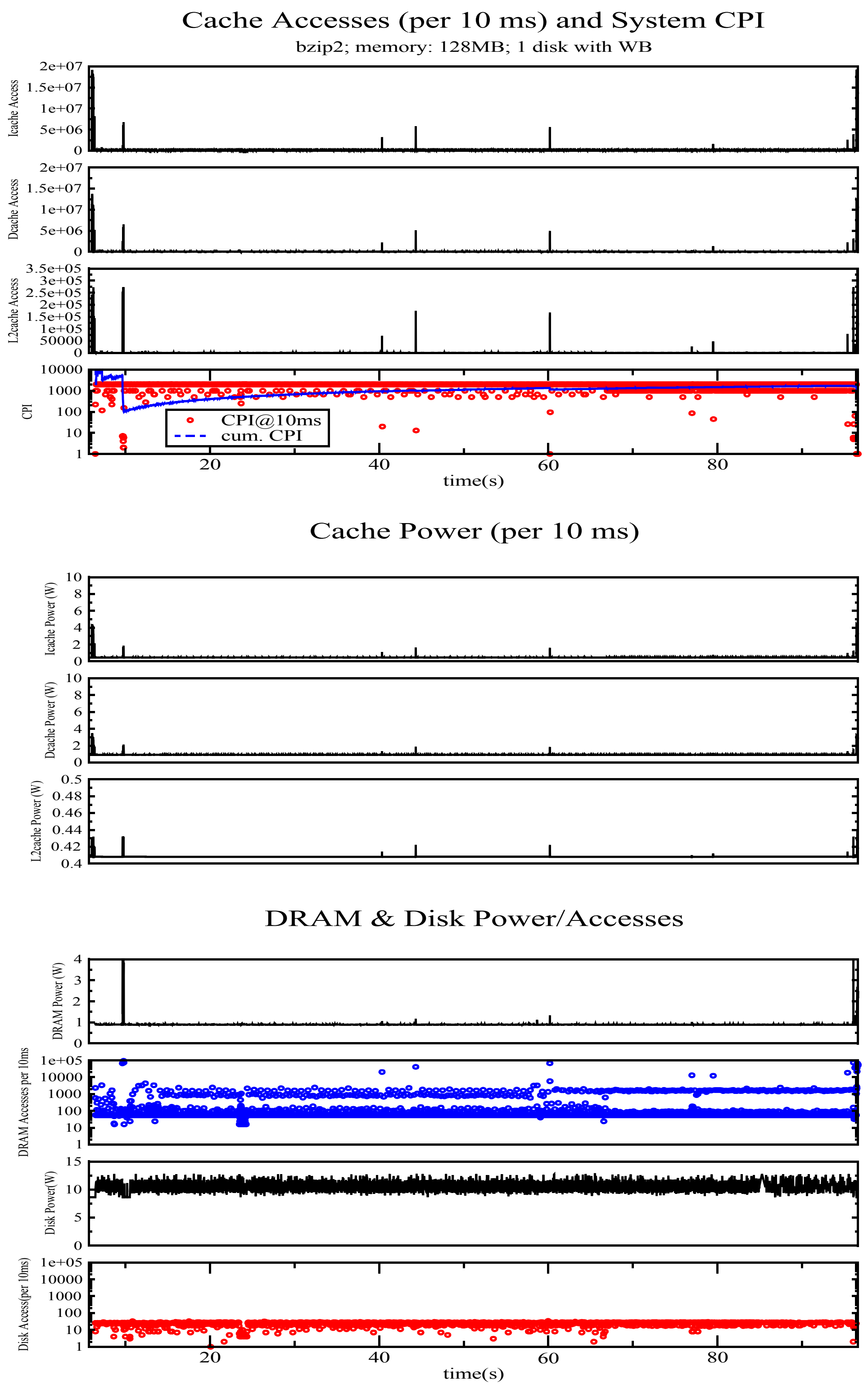


Figure 6.38: The interaction of the memory components in the hierarchy in a single disk system with perfect write buffering.

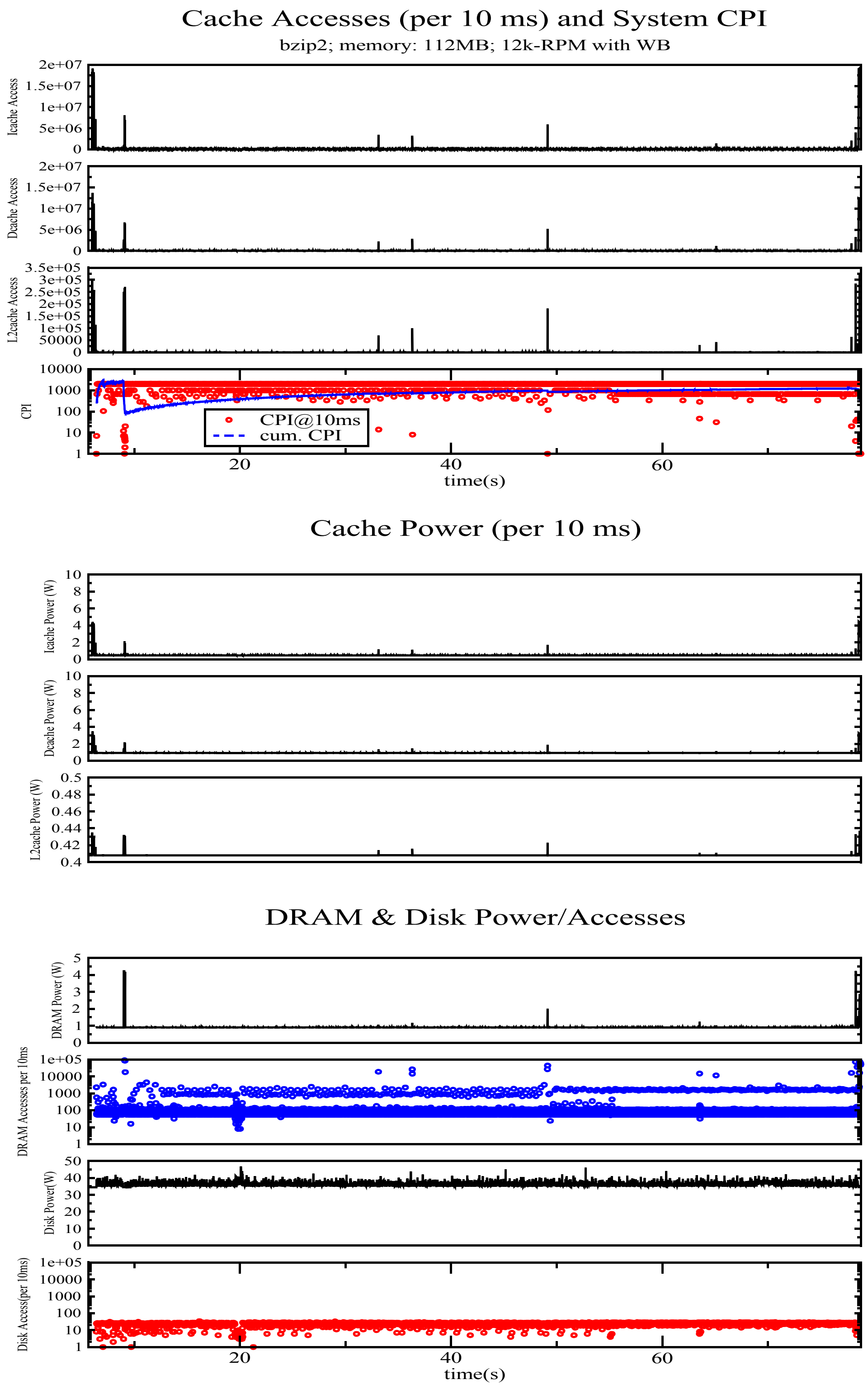


Figure 6.39: The interaction of the memory components in the hierarchy in a system with 4-disk RAID disk subsystem along with perfect write buffering.

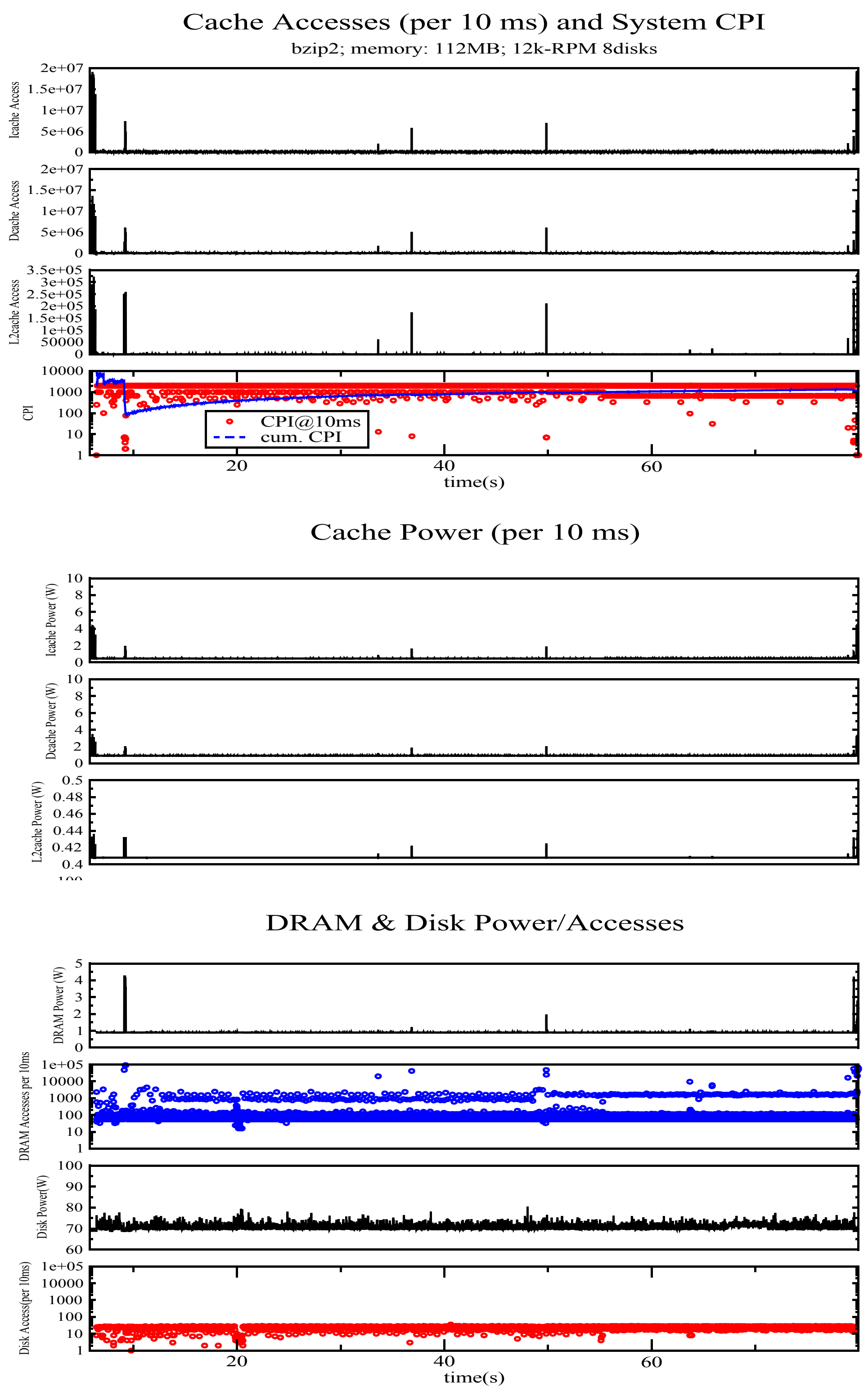


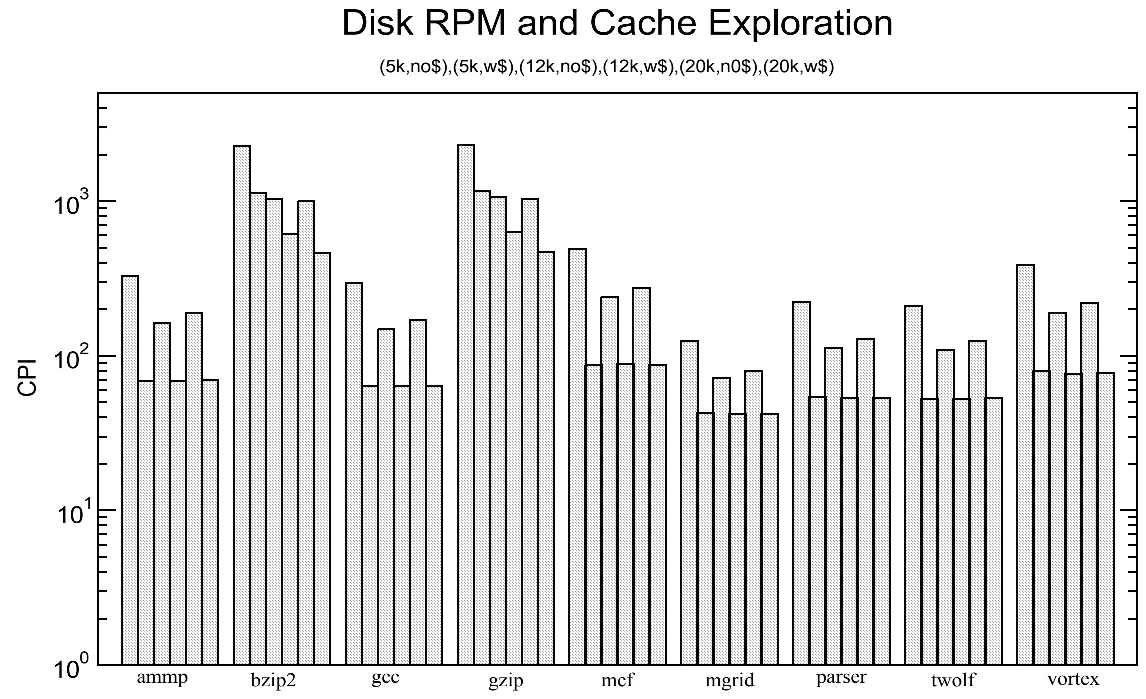
Figure 6.40: The interaction of the memory components in the hierarchy in a system with 8-disk RAID disk subsystem along with perfect write buffering.

6.5. Total CPI v.s. Disk Response Time

Most publications in the disk research community use the average disk response time and/or throughput as the metrics to measure the performance of the system. Especially, in the case of a single-user environment, the user pays attention to only a single process response. Therefore, average disk response time is the metric in this case.

However, as we have noticed this behavior in the previous section, the total CPI does not track the total average disk response time. We conducted an experiment to identify the relationship between the total CPI and the disk response time. We ran several benchmarks on a system with 128MB of memory utilizing a single disk. We varied the RPM of the disk and the existence of the disk cache. Figure 6.41 shows the CPI and Disk Average Response Time of the systems. The 6 bars in each group represent (1) a 5k-RPM disk without disk cache,(2) a 5k-RPM disk with disk cache,(3) a 12k-RPM disk without disk cache,(4) a 12k-RPM disk with disk cache,(5) a 20k-RPM disk without disk cache, and (6) a 20k-RPM disk with disk cache. The top graph shows the total CPI of the system, and the bottom graph shows the average disk response time for reads, writes, and overall for both.

During the I/O intensive phase which consists of both disk reads and writes, the average CPI tracks only average read response time, not the overall average R/W response time. This is true even for the benchmarks with read and write activities or even write-intensive benchmarks as portrayed with bzip2 and gzip. Therefore, the total disk average response time may not be an accurate metric to evaluate a disk technique as a representative to the total performance of a system. A better representative to the total system performance should be the read response time.



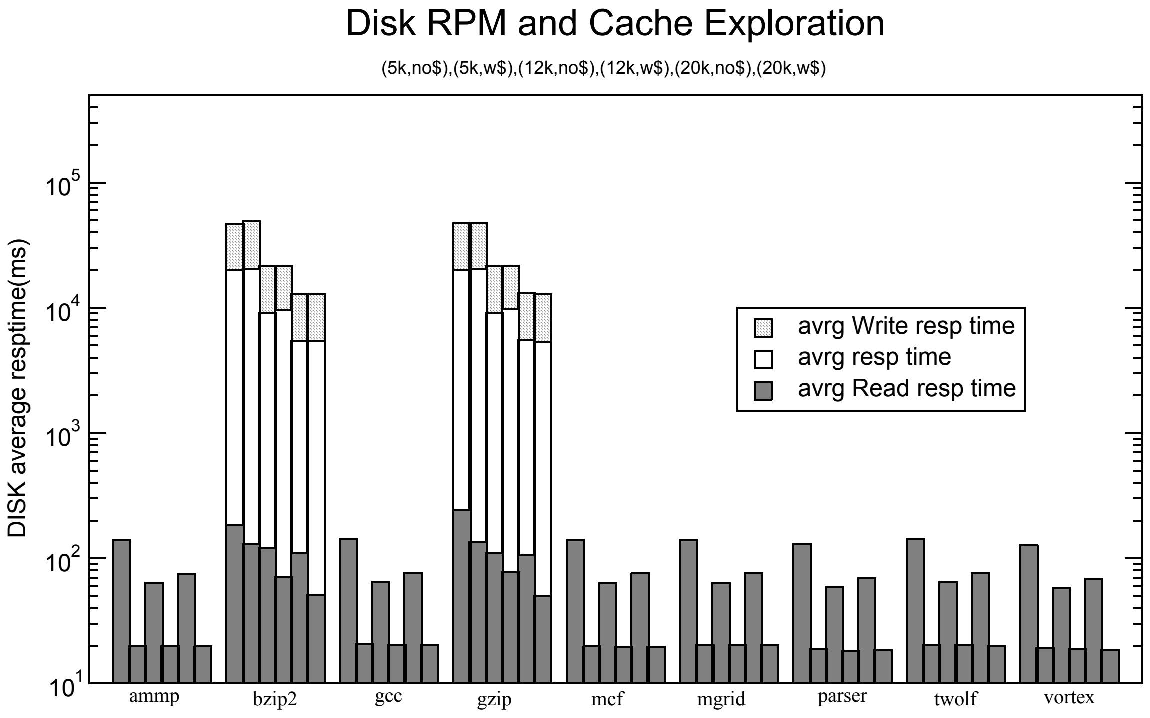


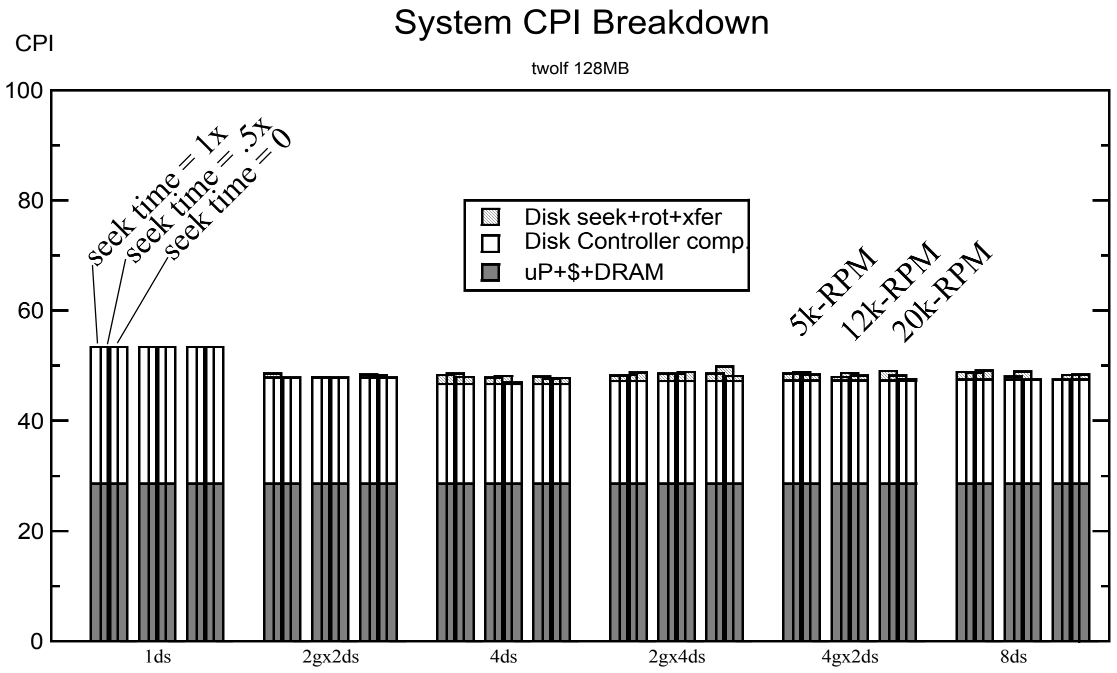
Figure 6.41: CPI v.s. Disk Average Response Time. The 6 bars in each group represent (1) a 5k-RPM disk without disk cache,(2) a 5k-RPM disk with disk cache,(3) a 12k-RPM disk without disk cache,(4) a 12k-RPM disk with disk cache,(5) a 20k-RPM disk without disk cache, and (6) a 20k-RPM disk with disk cache. The above graph shows the total CPI of the system, and the below graph shows the average disk response time for reads, writes, and total.

6.6. The CPI Breakdown

Figure 6.42 shows the System CPI Breakdown. The figure shows the breakdown CPI for 2 benchmarks, twolf and bzip2; both experimental system configurations utilized 128MB of memory with different disk systems. The top graph is for twolf, which does not have disk write requests, and the bottom graph is for bzip2, which has both reads and writes. The graphs show the CPI breakdown portions for (1) the processor, caches, and DRAM, (2) the controller computation which includes queuing, scheduling, and parity calculation, and (3) the disk mechanism, which include seek, rotation, and transfer. We experimented with 6 RAID5 configurations. Each RAID configuration contained 9 bars, divided into 3 groups. The first group uses 5400 RPM disks, the second group uses 12k RPM disks, and the last group uses 20k RPM disks. Each group consisted of 3 bars, which represented normal seek time, half seek time, and zero seek time. “Half seek time” means the seek times were computed then scaled down by half. “Zero seek time” means the seek times for all accesses are assumed to be zero. Note that the graphs have different y-axis.

From the twolf CPI, the CPI remains the same for all configurations due to only disk reads exhibited in the benchmark. RAID systems improve performance with a small margin due to less queuing and less scheduling time. The disk systems rarely use the disk mechanism since most of the requests hit the disk cache due to their sequential nature. In this case, the CPI portion of the processor, caches, and DRAM is significant.

In the case of bzip2, there are both disk read and write requests. Interestingly, the 4-disk systems have a bigger portion of controller computation CPI--this is due to the complexity of the scheduling and the parity calculations. However, due to the parallelism of multiple



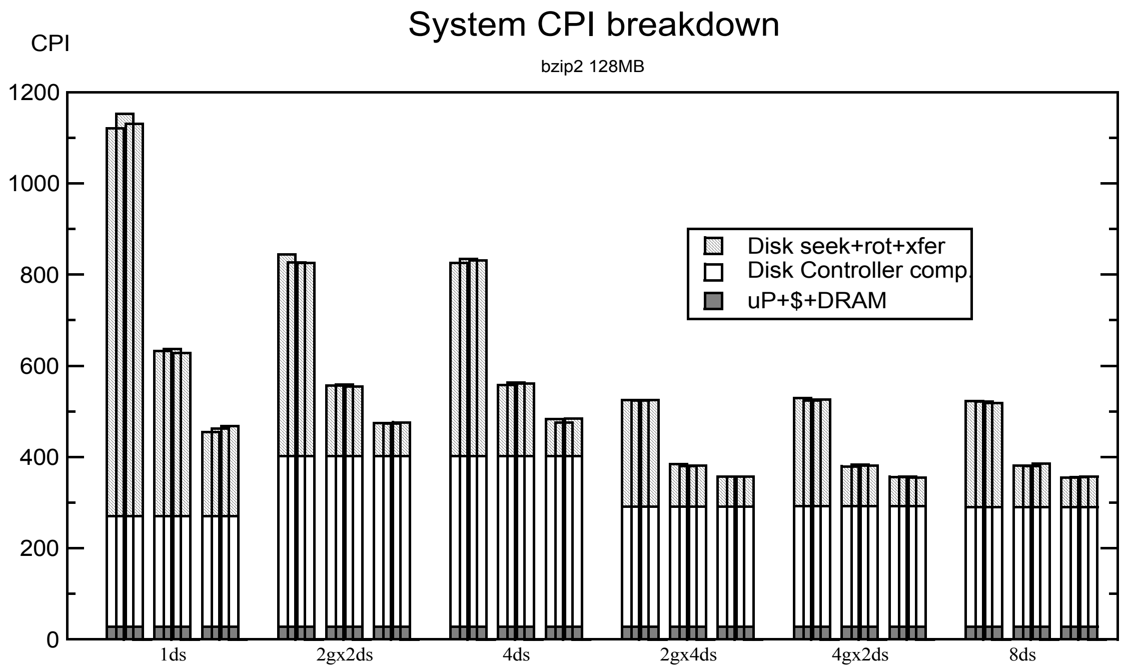


Figure 6.42: System CPI Breakdown. The figure shows the breakdown CPI for 2 benchmarks, twolf and bzip2, both with 128MB of memory. The graph above is for twolf which does not have disk write requests, and the graph below is for bzip2, which has both reads and writes. The graphs show the CPI breakdown portions for (1) the processor, caches, and DRAM, (2) the queuing and scheduling, and (3) the disk mechanism, which are seek, rotation, and transfer. We run the experiment on 6 RAID5 configurations. Each RAID configuration has 9 bars, divided into 3 groups. The first group uses 5400 RPM disks, the second group uses 12k RPM disks, and the last group uses 20k RPM disks. Each group consists of 3 bars, which are for normal seek time, half seek time, and zero seek time.

disks, the overall performance of the 4-disk system is better than the single disk system with the same configuration. The 8-disk systems amortize the complexity of the disk controller computation well enough to reduce the queuing and scheduling CPI. In both benchmarks, varying the seek time has no effect on the CPI due to the largely sequential nature of the requests. This seems not in agreement to the claim that seek time is very significant to the performance. The reason is seek time would be important in the access streams with little sequentiality as in multiprocessor systems, not in uniprocessor systems in our experiments.

In conclusion, CPI portions spent in the processor, caches, and DRAM represent only a secondary effect in comparison with the Disk parameter effects. The reason that the DRAM and Cache CPI are not as significant compared to the Disk CPI portion is that the access time due to the DRAM and cache is insignificant compared to the Disk access time. Additionally, most DRAM and Cache enhancements will affect at most only less than 2X their CPI portions. On the other hand, the Disk Parameter settings can change the total system performance over an order of magnitude.

6.7. Power/Energy Consumption

This section discusses the power dissipation and the energy consumption of the system as functions of memory size and disk enhancements mentioned in previous sections. First, the energy and power consumption of the system with different memory capacities are illustrated in Figure 6.43. Figure 6.43 corresponds to the performance graphs in Figure 6.12. The top graph shows the total power dissipated in the memory system, including caches, DRAM, and disk. The middle graph shows the energy consumption, and the bottom graph

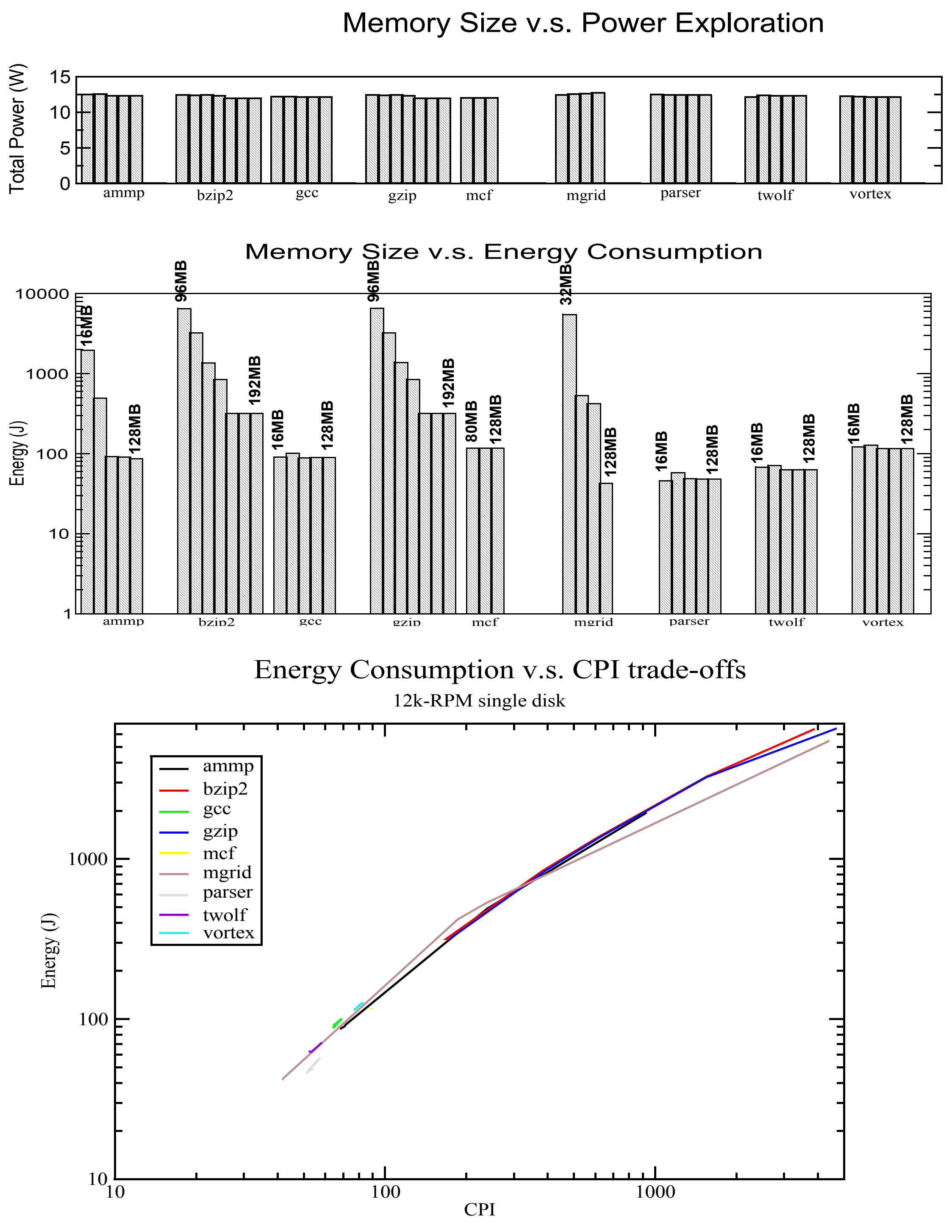
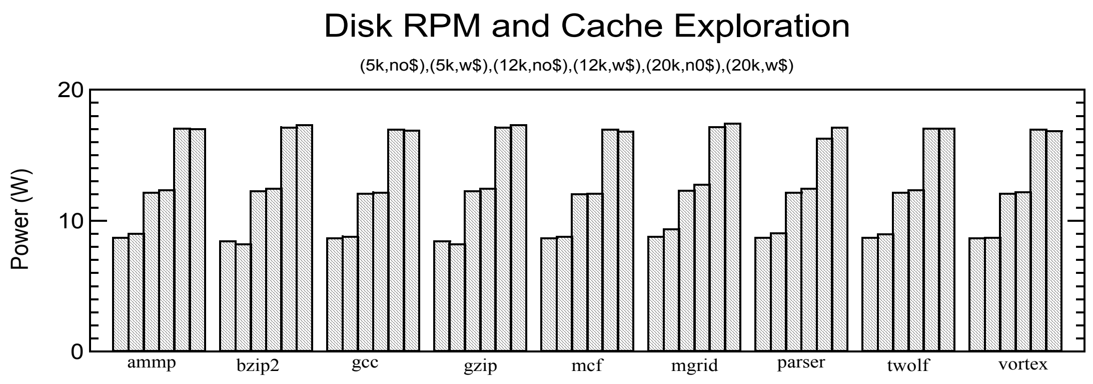
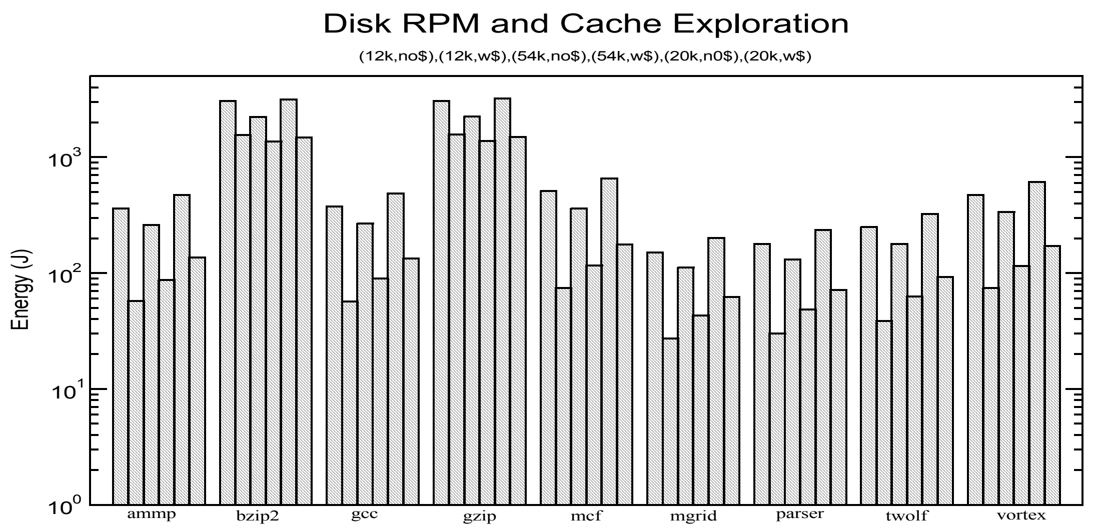


Figure 6.43: Power and Energy Consumption in the system with different memory size. This figure is corresponding to the performance graphs in Figure 6.11. The top graph shows the total power dissipated in the memory system, including caches, DRAM, and disk. The middle graph shows the energy consumption, and the bottom graph shows the pareto optimal of the CPI and energy. The power and energy of nine SPEC benchmarks are reported.

shows the Pareto plot of the CPI and energy. The power and energy of nine SPEC benchmarks are reported. While the total power dissipation in the systems remains at maximum for all memory sizes, the total energy consumed can differ by two orders of magnitude. The total power for the benchmarks demonstrating memory page swapping should be lower when the DRAM capacity increases. However, since the disk power dominates the total system power, and the difference between the active power (11.26W) and idle power (8.62W) of the disk is marginal, the power of the systems with lower memory capacity is not much lower than the systems with large memory. Additionally, despite the large memory capacity, the disk is still the key component as it is accessed actively during the I/O intensive phase. On the other hand, the energy consumption in those systems are extremely different. Compared with Figure 6.12, the energy consumption tracks closely with the CPI and the number of disk requests, and can reach as high as two orders of magnitude of the energy in a system with large memory. Interestingly, by plotting the CPI and the energy trade-offs, all benchmarks end up with having the same relationship between the CPI and the energy consumption. This reflects the realistic behavior of our simulator because the very same hardware-based systems reported the same total performance would consume the same amount of energy, no matter what type of applications the system is running. The reason is the overall performance and the total system energy consumption would account for all activities in all components in the system.

The power and energy of the systems with different disk RPMs and disk cache is illustrated in Figure 6.44. The graph shows the power and energy corresponding to the experiment results in Figure 6.20. The system memory is 128MB running nine different benchmarks. The top graph shows the power dissipation, and the middle graph shows the





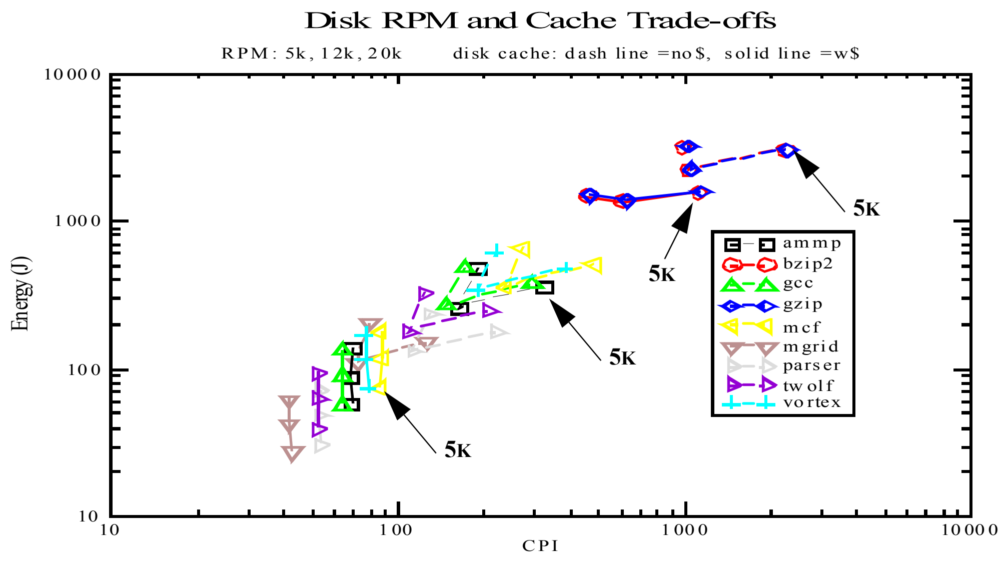
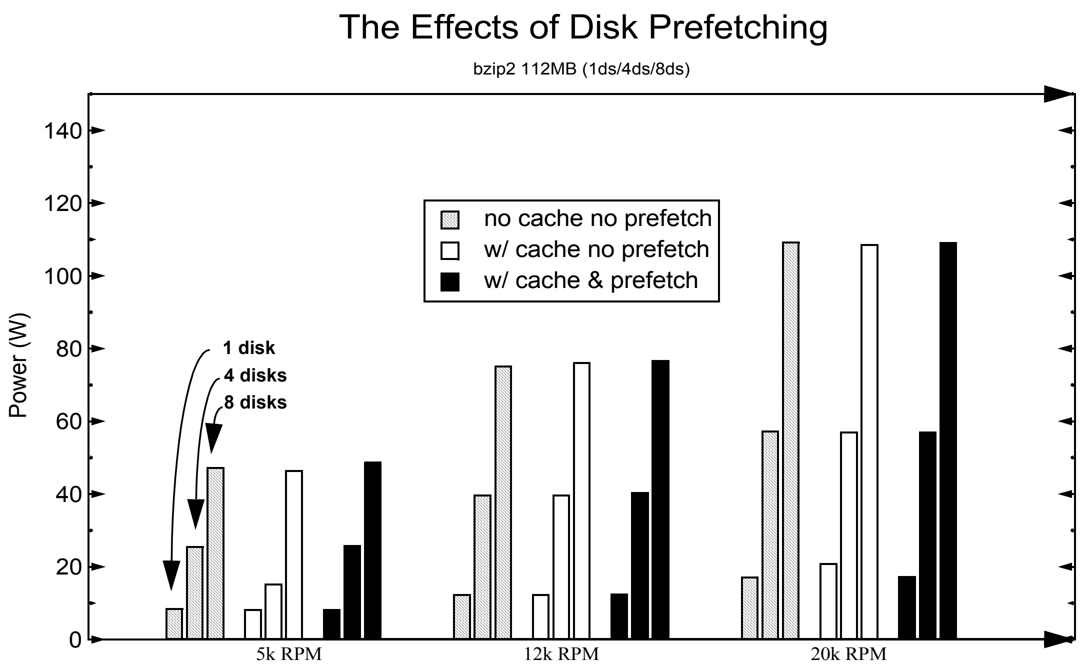


Figure 6.44: Power and Energy Consumption for the system with different RPM and the presence of disk cache. The top graph shows the power dissipation, and the middle graph shows the energy consumption. The bottom graph shows the CPI and total energy trade-offs of the systems with different disk RPM and the presence of disk cache. For the bottom graph, The graph shows the CPI and energy for different benchmark on the systems with varied RPM and the existence of the disk cache. The lines connect the data points with the same disk cache configuration with different RPMs (5k, 12k, 20k): The dash lines represent no-cache configuration, and the solid lines represent with-cache configuration.

energy consumption. The bottom graph shows the CPI and total energy trade-offs of the systems with different disk RPMs and the presence of disk cache. In the bottom graph, For the bottom graph, The graph shows the CPI and energy for different benchmark on the systems with varied RPM and the existence of the disk cache. The lines connect the data points with the same disk cache configuration with different RPMs (5k, 12k, 20k): The dash lines represent no-cache configuration, and the solid lines represent with-cache configuration. Again, the power dissipation remains the same among the systems with the same features, i.e. the same disk rotational speed in this case. The power also increases with the disk rotational speed because the higher RPM disk dissipates more power. Unlike the systems with the same disk RPM, the energy consumption does not track the system CPI. The reason is the power varies in different rotational speed; therefore, the system with the same CPI but equipped with different RPM disks consumes different energy. Interestingly, the systems without disk cache prefer 12k-RPM disk over other rotational speed disk. The systems implementing disk cache prefer lower RPM for the benchmarks with only read requests since the requests are mostly serviced by the disk cache, so the disk mechanical parts are mostly idle in these benchmarks. Additionally, the disks with disk cache consume the same amount of energy when the request stream is a mix of reads and writes, such as bzip2 and gzip because slow disks compensate slowness with lower power. Like Figure 6.43, the CPI and energy relationships for all systems lie on the same projected band with different slopes due to different disk RPMs. Moreover, now we actually have an interesting Pareto plot: more than one optimal points are exhibited in case of the benchmarks with both reads and writes. For those benchmarks with both reads and writes, regardless of disk cache, both 12k and 20k RPM are optimal points.

Figure 6.45 shows the power and energy consumption of the systems as a function of



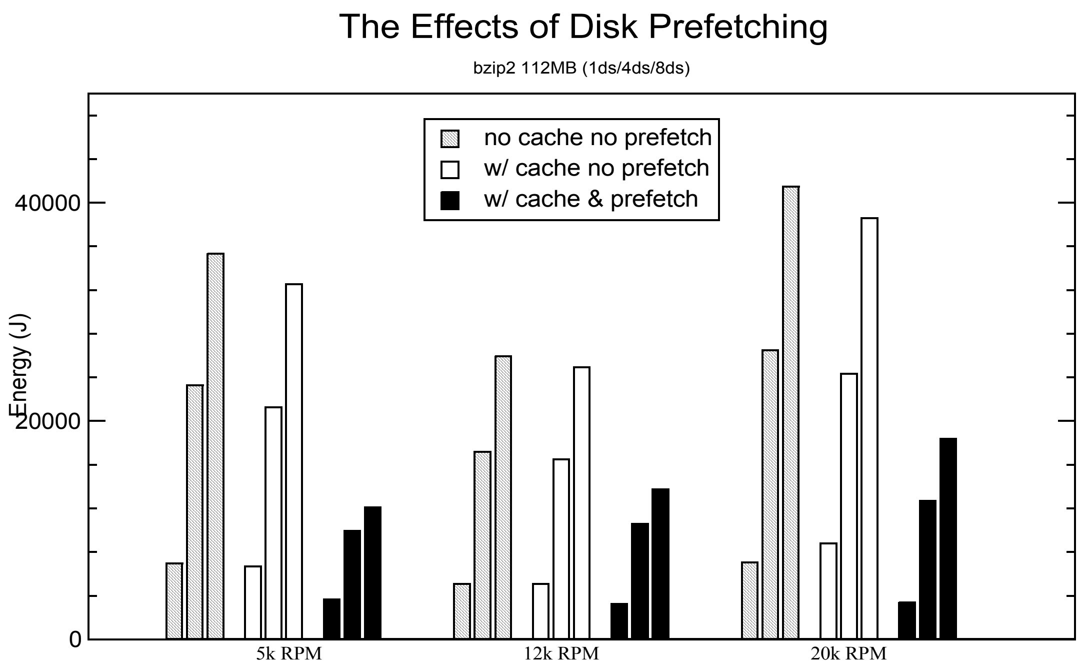
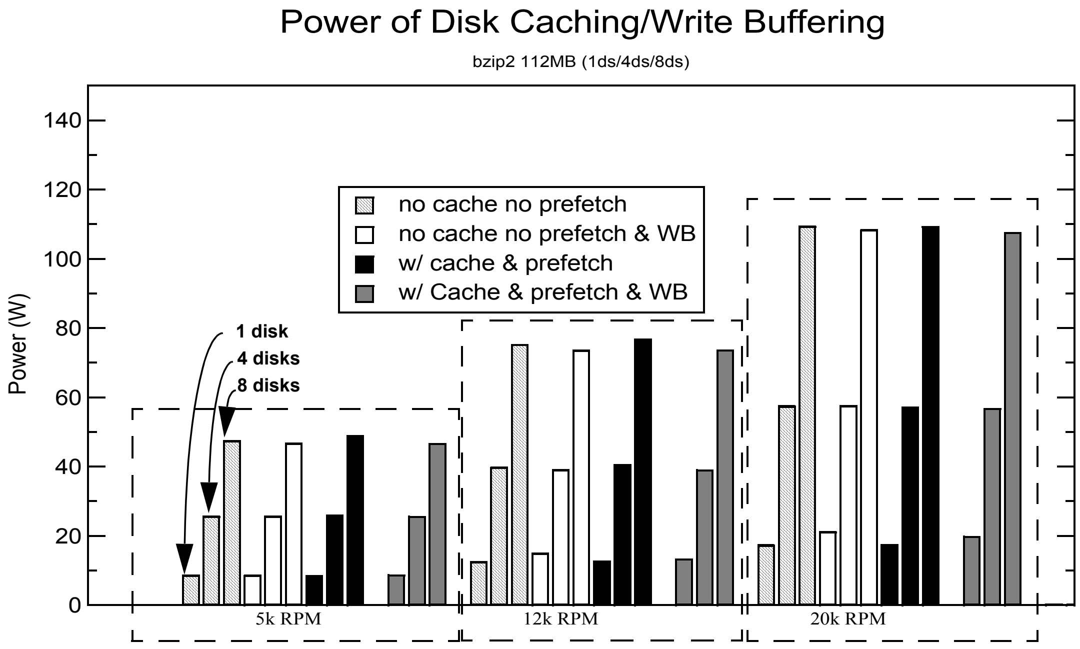


Figure 6.45: Power and Energy Consumption of the system with Disk Caching/Prefetching. This figure shows the power and energy consumption corresponding to Figure 6.22. The memory is 112MB running bzip2. The number of RAID disks, the disk RPM, the presence of disk cache and prefetching were varied.

disk caching and prefetching. This figure shows the power and energy consumption corresponding to Figure 6.23. The system configuration is set to 112MB of memory running bzip2. The other experimental system configuration settings such as the number of RAID disks, the disk RPM, the presence of disk cache and prefetching were set to various increments. In contrast to Figure 6.23 where more RAID disks gain better performance, the power and energy is proportionally increased with the number of disks. The same pattern repeats here again where the power dissipation in the systems with similar features remains the same across all different disk caching/prefetching organizations. On the other hand, the energy numbers of those systems with different disk caching/prefetching organizations are different. The system with better performance in Figure 6.23, i.e. one with both disk caching and prefetching, consumes less energy. Considering only energy, the systems prefer 12k-RPM disk system over other RPMs. The reason is the 12k-RPM disk systems with lower active and idle power perform as comparable as the 20k-RPM disk systems, so the 20k-RPM consumes more energy. The 5k-RPM disk systems perform much slower despite lower power, so the final total energy is higher.

The results for the systems with perfect write buffering are shown in Figure 6.46. The figure shows the power and energy corresponding to Figure 6.37. Like the previous case, the power dissipation in the systems with the same disk-system configurations remain the same across different disk caching and write buffering choices. The energy consumption is different depending on the system performance. Despite the RPM, the systems with both disk caching and write buffering prefer a lower RPM disk system because of its lower power. This is true because disk caching and write buffering eliminate the need to wait for the disk’s mechanical parts; thus the system no longer requires fast RPM disks to improve



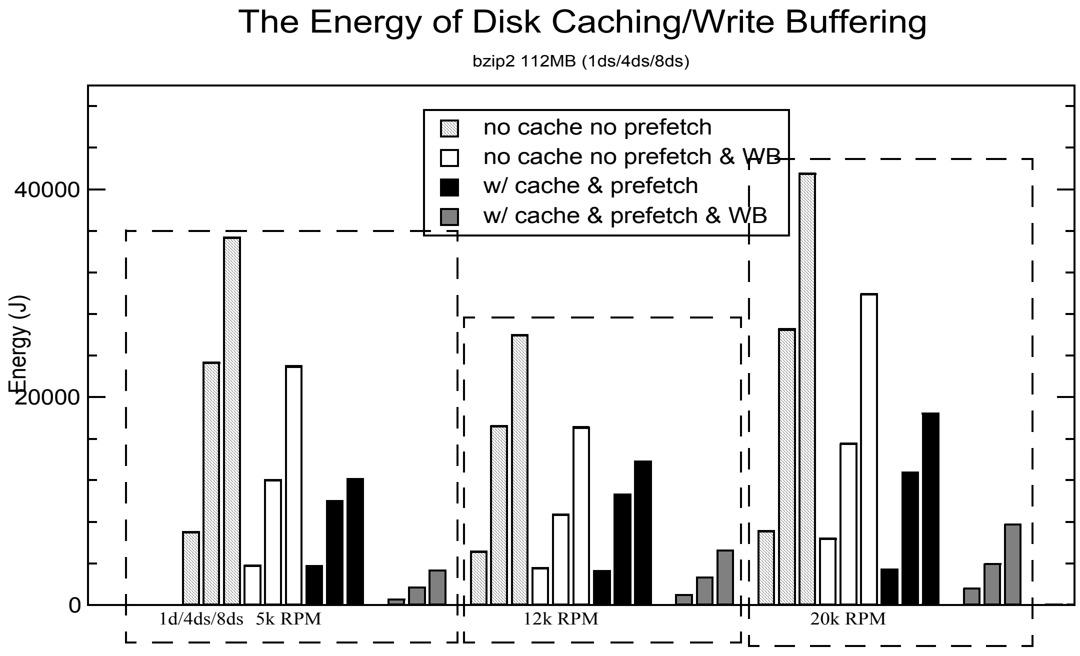


Figure 6.46: Power and Energy Consumption for Caching and Perfect Write Buffering. The figure shows the power and energy corresponding to Figure 6.36. The memory is 112MB running bzip2. The number of RAID disks, the disk RPM, the presence of disk caching/prefetching and write buffering were varied.

performance. Therefore, slower but lower powered disks are more energy efficient at this point.

The energy and power results for the system with constant megabytes of the sum of DRAM and disk cache capacity are also included in Figure 6.47. The energy and power reported corresponds to the top graph in Figure 6.34. The graph shows the energy and power trade-offs between the memory size and the disk cache size under the assumption that the total MB of the memory and the disk cache remains the same. In this case, the total MB is 32MB on an ammp execution. The bars represent the system power, and the lines represent the system energy. The total power remains the same across all systems with the same number of disks and disk RPM. On the other hand, the energy consumption tracks the CPI shown in Figure 6.34, while the last two data points in each RAID configuration, which are (16,16) and (8,24), increase rapidly because the memory is not large enough for the

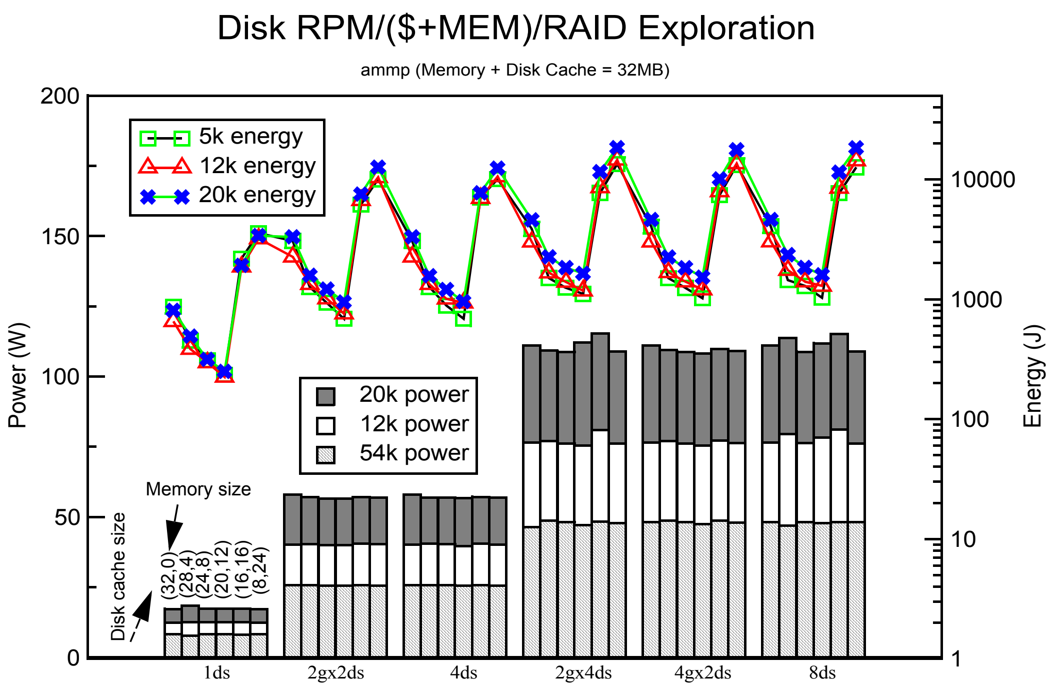


Figure 6.47: Power and Energy Consumption for the system with constant sum of memory size and disk cache.

application's footprint. All disk RPMs consume relatively the same amount of energy in this case, typically within 50% of each other.

The energy and power for the systems with different disk cache size is shown in Figure 6.48. The system memory is 32MB running ammp. The disk cache size varies from no cache, 4MB, 8MB, and 256MB of disk cache. The figure is corresponding to the bottom graph of Figure 6.34. The power remains the same across the systems with similar number of RAID disks. The energy consumption tracks the system CPI in the configurations with the same number of RAID disks and RPM. However, the systems with faster RPM disks consume more energy as well as being superior in performance.

Finally, we conducted an experiment to investigate the trade-off between the power consumption and the performance of several disk technology improvements and enhancements. Figure 6.49 shows the trade-offs Chart of the Power Dissipation/Energy

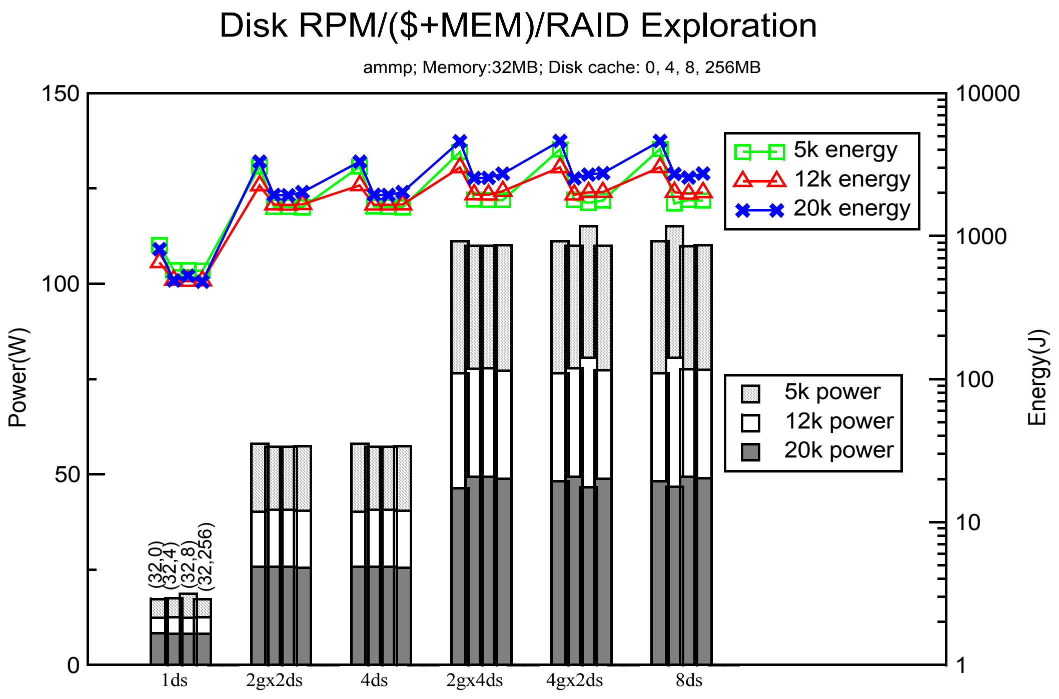


Figure 6.48: Power and Energy Consumption for the system with different size of disk cache.

Consumption versus CPI. The top graph is the Power Dissipation versus the CPI, and the bottom graph is the Energy Consumption versus the CPI. The system configuration is 128MB running bzip2. We varied the number of disks to one, four, and eight RAID disks with 5k, 12k, and 20k RPM. We also varied the existence of the disk cache. The dash line is

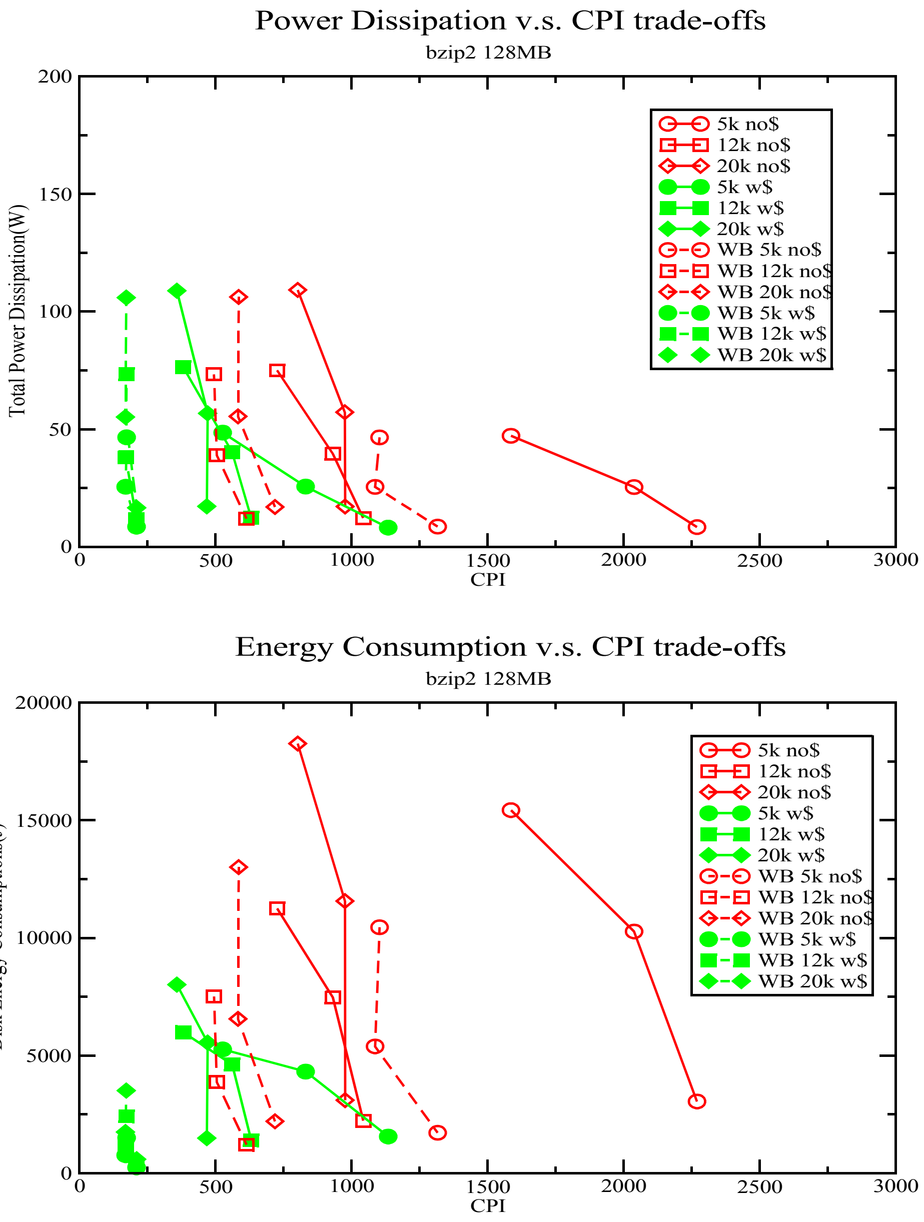


Figure 6.49: Power Dissipation/Energy Consumption v.s. CPI trade-offs. The above graph is Power Dissipation v.s. CPI, and the lower graph is the Energy Consumption v.s. CPI trade-offs. The system configuration is 128MB running bzip2. We varied the number of disks to one (the lowest data point on the line), four (the middle data point), and eight RAID disks (the highest data point) with 5400, 12k, and 20k RPM. We also varied the existence of the disk cache. The dash line is for the disk system with write-elimination technique. Therefore, the dash line is the limit of energy/power saving.

for the disk system with perfect write-buffering technique marked as “WB”. Therefore, the dash line can be considered as the limit of energy/power saving of the write-buffering technique.

For the power dissipation, except for 5k RPM with no disk cache, all other data points are clustered in the region or CPI 500-1000. We can conclude that the techniques, such as increasing RPM and disk caching and prefetching, can improve the performance only to a factor of 2. The write-buffering technique can also improve the performance by a factor of 2, and the combination of write-buffering and caching/prefetching can improve the performance greatly without the requirement of multiple fast disks. However, the power dissipation remains the same among the systems with the same number of disks.

On the other hand, the energy consumption graph gives us more of an insight. Obviously, unlike the power dissipation, the systems with the same number of disks do not consume the same amount of energy. For example, the system with 8 20k-RPM RAID disks without disk cache consumes more energy and performs worse than the same configuration with disk caching. The system with 5k-RPM disks consumes more energy than other different RPM-disk systems containing more disks. Nevertheless, the write-buffering technique on a slow disk system in conjunction with disk cache produces the optimal effect in this case.

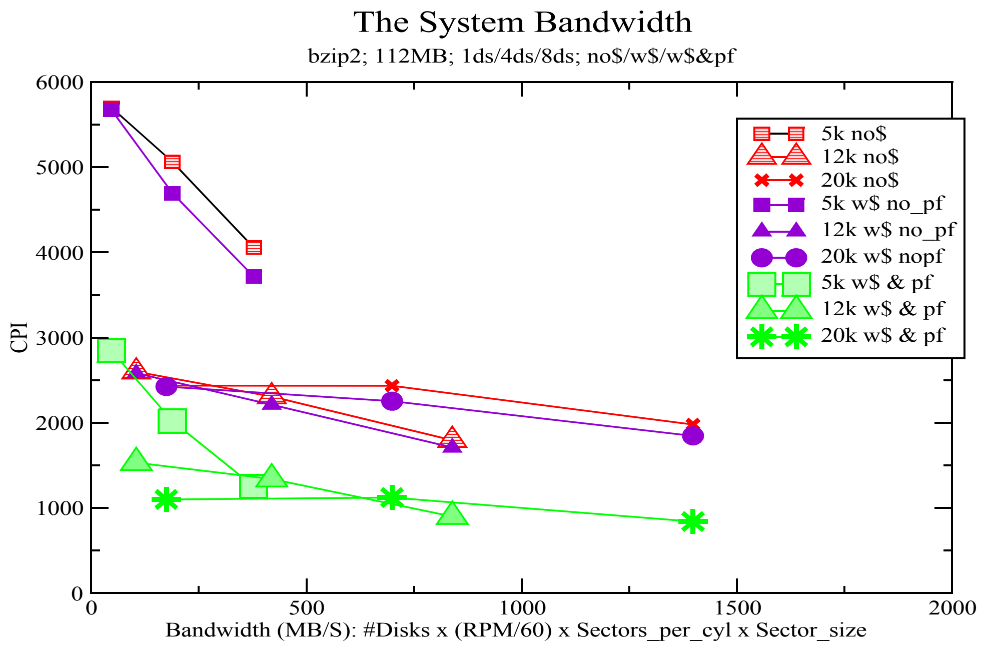
To sum up, systems with *N* RAID disks do not directly improve the performance by a factor of *N*, while they typically consume *N* times more energy and power. On the other hand, increasing the RPM of an already fast disk system will not gain any obvious benefits, and only increases the energy consumption, which varies with the number of disks. Using a low RPM disk does not save energy in most case. The disk enhancements, i.e. disk caching/prefetching and write-buffering, can improve the performance by a factor of 2 while reducing the energy approximately the same rate. Care should be taken into the disk enhancements rather than attempting to increase only the disk bandwidth parameters, such as the number of RAID disks and the RPM.

6.8. The System Bandwidth

To sum up, the figure 6.50 shows how the total System Bandwidth of configurations on different disk systems compares to the total system performance. The figure shows the CPI versus the system bandwidth, which is calculated by multiplying the number of disks, the rotation speed, the number of sectors per cylinder (1024), and the sector size (512 bytes). We varied the disk RPM, the existence of the disk cache, prefetching, and write-buffering technique. We also varied the number of disks in the RAID5 disk system. Each line connects systems with the same RPM disks; therefore, there are 3 data points on each line, which represent 1-disk, 4-disk, and 8-disk system, respectively from left to right. The top graph shows only the configuration with caching and prefetching, which are already explicitly implemented in today's disk drives. The bottom graph shows the same graph along with the perfect write-caching configurations represented as dotted lines. We ran bzip2 on all system configurations with 112MB of memory.

Interestingly, the total performance of the system with the same system bandwidth can vary over an order of magnitude, depending on which enhancements have been applied. In some cases, the disk system with comparable bandwidth and employing the same techniques can have the total performance as different as a factor of 2 due to different configurations.

For example, the case of the 8 5k-disk system without disk cache and the 4 12k-disk system without disk cache exhibit this behavior. On the other hand, with different enhancements,



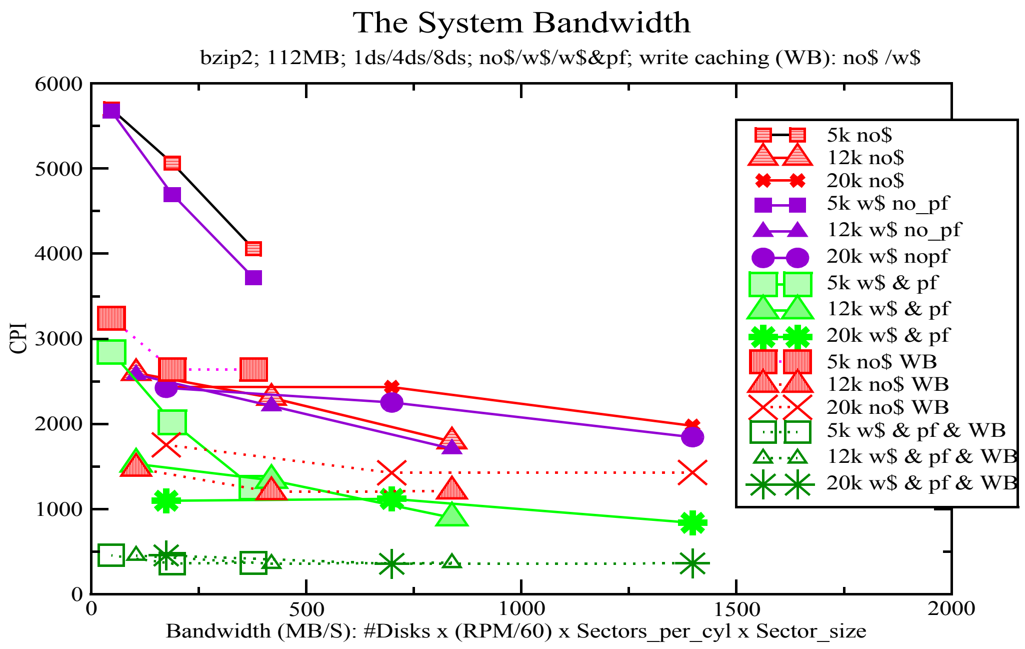


Figure 6.50: The System Bandwidth. The figure shows the CPI versus the system bandwidth, which is calculated by multiplying the number of disks, the round- per-second, the number of sectors per cylinder, and the sector size. We varied the disk RPM, the existence of the disk cache, prefetching, and write-buffering technique. We also varied the number of disk in the RAID5 disk system. Each line connects the system with the same RPM disks; therefore, there are 3 data points on each line, which represent 1-disk, 4-disk, and 8-disk system, respectively from left to right. The top graph shows only the configuration already implemented in today's disk drives. The bottom graph shows the configuration with perfect write caching in dotted lines.

carelessly choosing a configuration only to increase the bandwidth may cause the system to perform worse. For example, choosing a system with 8 20k-RPM disks with no cache rather than a system with 4 12k-RPM disks with cache will not benefit the system as suggested by the system bandwidth.

Another trend also demonstrated in the graph is the trend that increasing only the system bandwidth does not directly translate into improvement in total system performance. When the bandwidth is low, increasing the bandwidth will improve the performance significantly, except the cases where employing all disk caching/prefetching and write-buffering techniques. We noticed that as we continue increasing the system bandwidth without applying further enhancement, the CPI exhibits relatively no improvement. As a result, new enhancements for disk systems are required to improve the total system performance.

6.9. Configuration Comparison

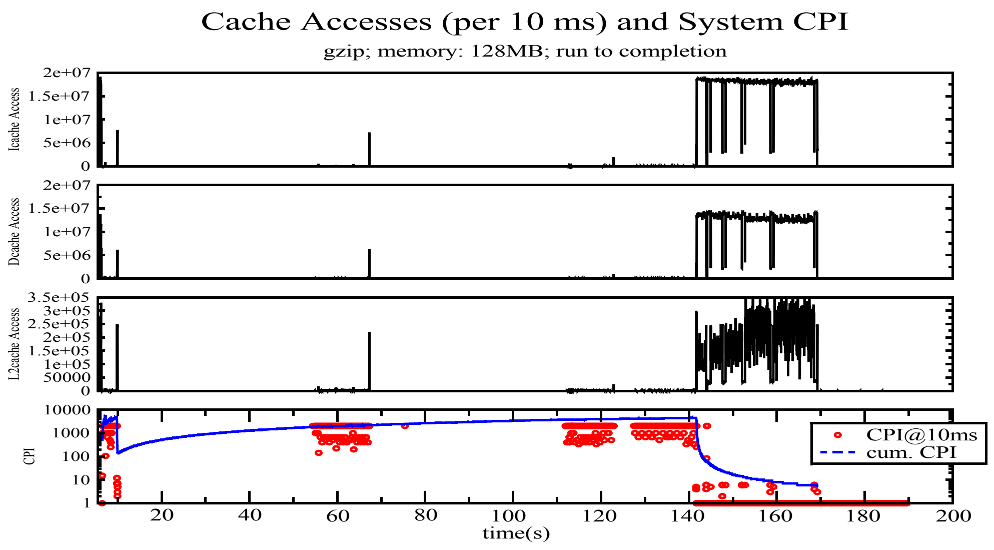
In this dissertation, we tried to answer this question:

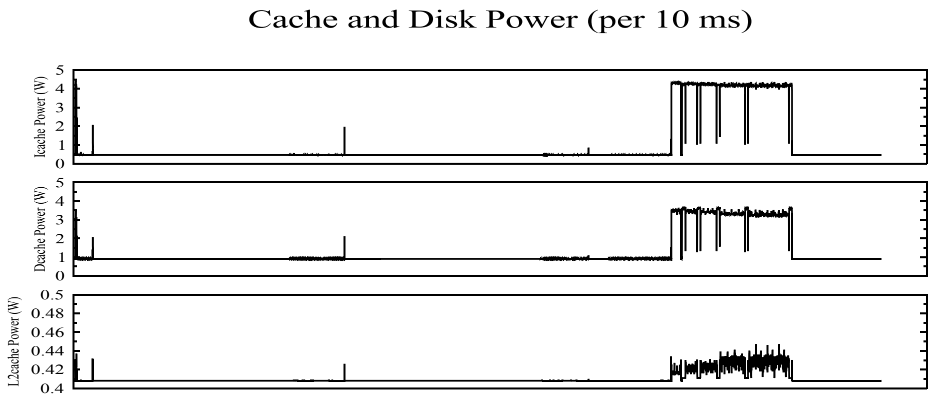
What is the best solution, in terms of both total system performance and power/energy consumption, for a single processing system whose I/O intensive phase is exposed to occupy a significant portion of the entire execution time?

And, from our experiments with SYSim, we conclusively proposed two answers:

- increasing the memory size, and/or
- using a single disk system with disk cache and significant attention paid to write buffering.

The interactions between all components in the entire memory hierarchy and the system CPI of both solutions are shown in the last figures. To compare with the interaction in Figure 1.1 which is also shown here again for comparison, Figure 6.52 shows the interaction in the case of increasing the memory size to 512 MB, and Figure 6.53 shows the interaction in the case of 128MB of memory with perfect disk write buffering. Both systems run gzip with a 12k-RPM disk drive with a small disk cache (4MB). The figure shows the interaction between all components in the memory hierarchy including the level-1 instruction cache, the level-1 data cache, the level-2 unified cache, DRAM, and a disk drive. Notice that the initialization time reduced from 140 seconds in Figure 1.1 to 40 seconds in Figure 6.52 and to 48 seconds in Figure 6.53. The first solution would solve the problem under the condition where the memory is always big enough to hold the application memory footprint. However, the energy consumption of the first solution may increase significantly if using next generation DRAM, i.e. an FBDIMM system. In contrast, the second solution would be less sensitive to the application characteristics due to the I/O latency hiding nature of the approach. However, one would suggest using a RAID disk system to improve parallelism in the disk system. As shown in the RAID studies, using RAID in single user mode does not improve the performance as much as its costs because the energy and power consumption of the RAID system is proportional to the number of disks, and RAID performance does not directly scale with the number of the disks. Figure 6.54 shows the interaction in the system with an 8-disk RAID system equipped with cache and write buffer. The execution time of the RAID system improves only 5 seconds--less than 7% improvement over a single disk with write buffer while the user has to pay the cost of 8 disks. As a result, RAID is not recommended to a single process environment during the I/O intensive phase.





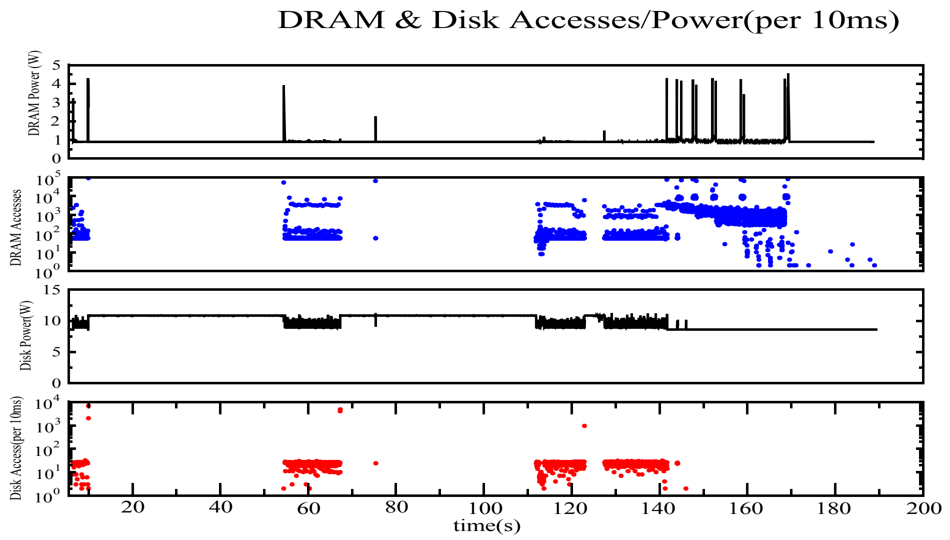


Figure 6.51: The interaction in memory hierarchy in our base configuration with 128MB of memory. The figure shows the System CPI over the entire run of gzip. The system configuration is a 2-GHz processor with 128MB of memory and a 12k-RPM disk. The CPI graph shows 2 CPI values: one is the instant CPI for every 10ms, another is the accumulated average CPI. The duration having no data point means no instructions are executed due to the I/O latency. The course of execution when the accumulated CPI is over 100 is the I/O intensive phase, and the course of execution when the CPI is below 100 is the computation phase.

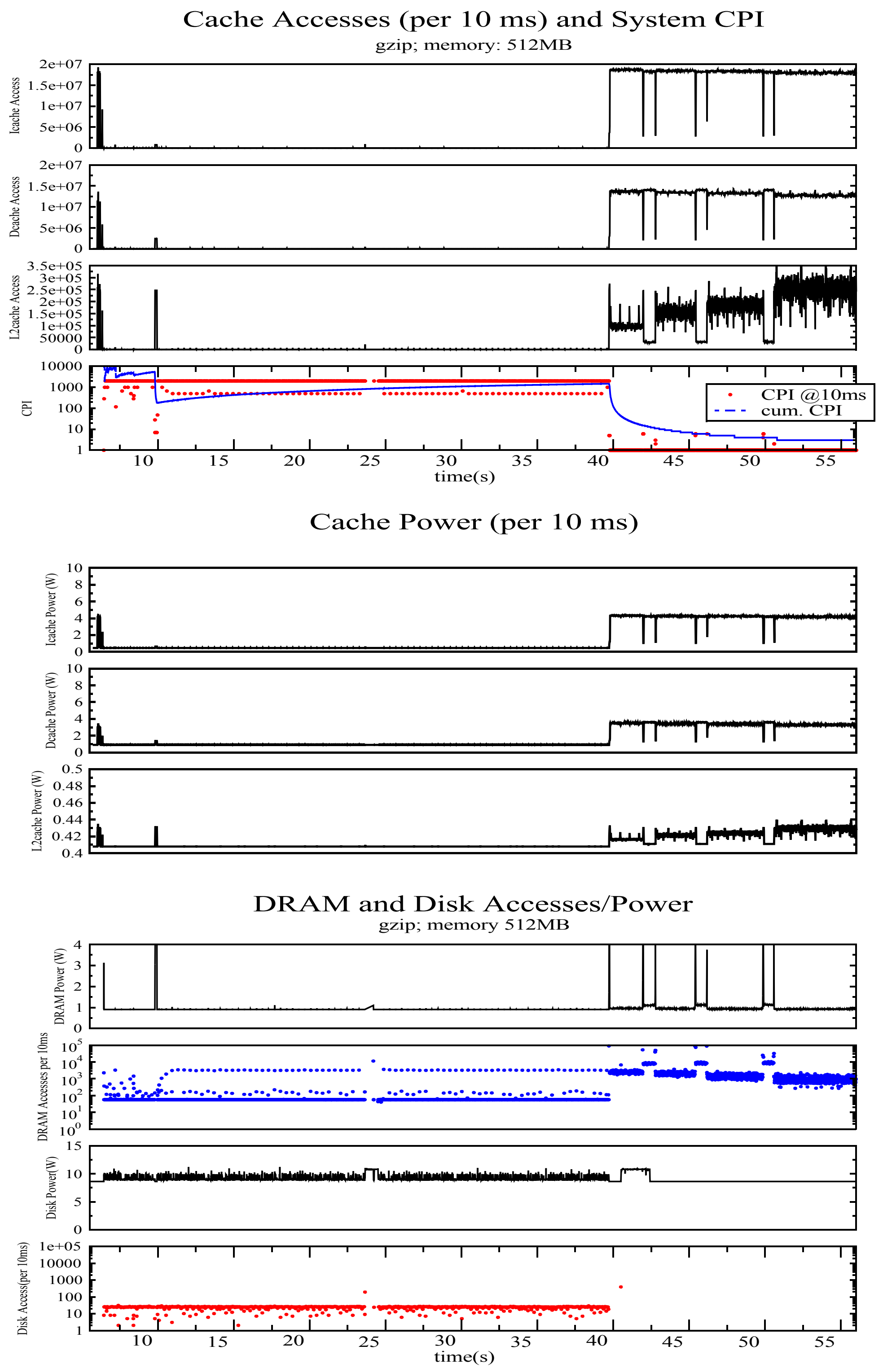


Figure 6.52: The interaction in memory hierarchy in a system with 512MB of memory. The figure shows the interaction between all components in the memory hierarchy including level-1 instruction cache, level-1 data cache, level-2 unified cache, DRAM, and a disk drive. Notice that initialization time reduces from 140 seconds in Figure 1.1 to 40 seconds in this figure.

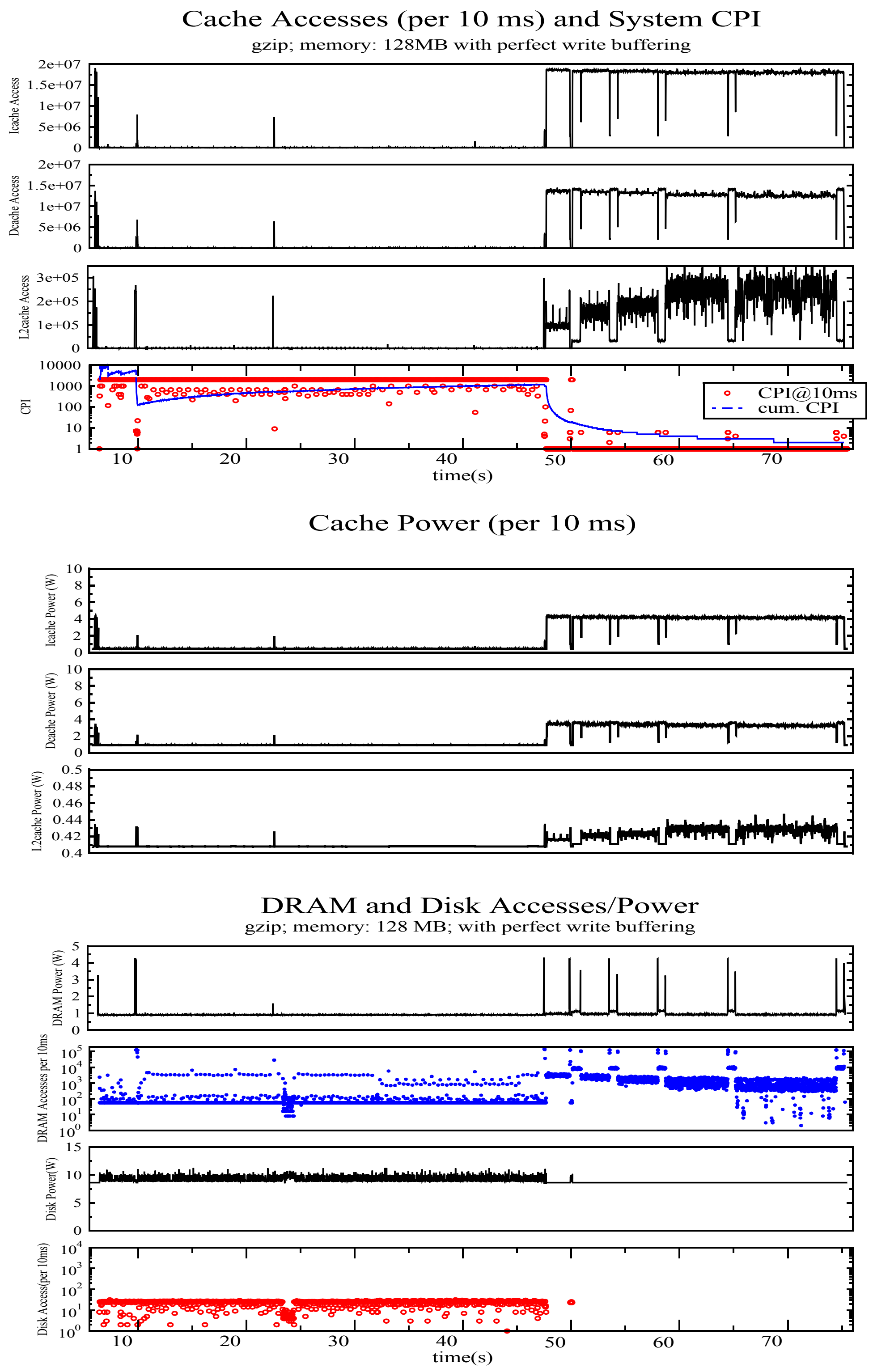


Figure 6.53: The interaction in memory hierarchy in a system with 128MB of memory and a disk drive with perfect write buffering. The figure shows the interaction between all components in the memory hierarchy including level-1 instruction cache, level-1 data cache, level-2 unified cache, DRAM, and a disk drive. Notice that initialization time reduces from 140 seconds in Figure 1.1 to 48 seconds in this figure.

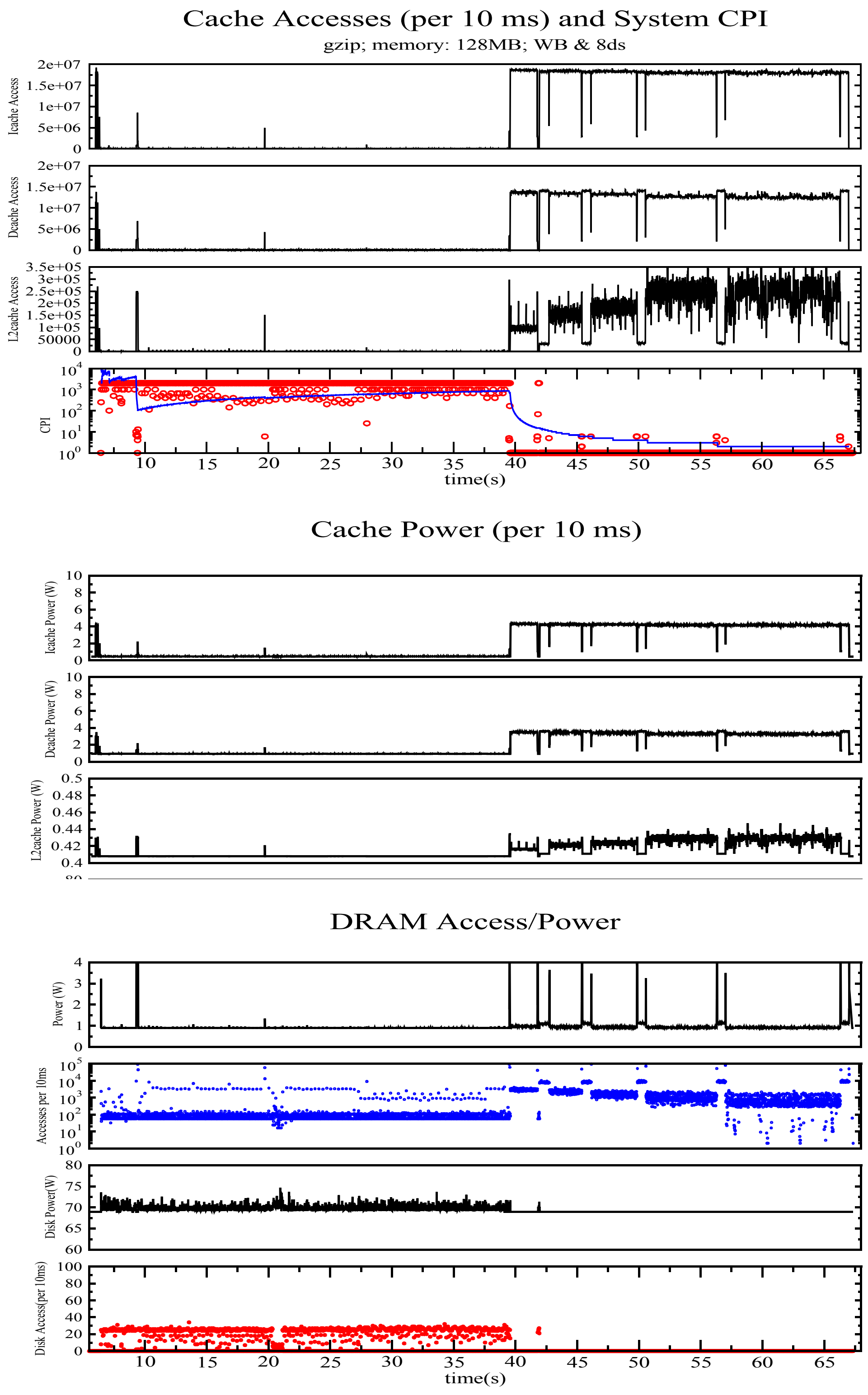


Figure 6.54: The interaction in memory hierarchy in a system with the same configuration with RAID disk system. The figure shows the interaction between all components in the memory hierarchy including level-1 instruction cache, level-1 data cache, level-2 unified cache, DRAM, and a disk drive. Notice that initialization time reduces from 140 seconds in Figure 1.1 to less than 40 seconds in this figure

CHAPTER 7: CONCLUSIONS

Most studies focus on the computation phase during which the most repeated instructions are executed. The argument for focusing on the computation phase is to make the most repeated case fast. However, we followed a different path from those studies. The course of the entire execution consists of I/O intensive phase as well as the main computation phase. We have shown that a program spends significant amount of the time in the I/O intensive phase due to the I/O latency, especially in the single processing environment mostly found in personal computers. Therefore, the I/O has been exposed as a significant component with respect to total execution time.

To obtain a system with more balanced phases, we require more understanding in the effects of I/O configurations to the entire system. Therefore, we are forced to extensively investigate the I/O effects to the full-system scale. The system total execution time can be improved to an order of magnitude by the previously mentioned enhancements in disk systems, i.e. using disk caching/prefetching and write-buffering techniques.

Memory performance and power are now the key challenges in system design. With respect to the processor, memory accesses become slower and consume more power with increasing memory size. Most of the total power consumption of the systems is dominated by the entire memory hierarchy. Hence, memory power and access time significantly affect total power and performance for computations with large storage requirements, and memory becomes the main bottleneck

Disks in general have been widely used as secondary, non-volatile storage and as a low-level memory in virtual memory hierarchy. It is accepted as an indispensable part of the general-purpose computer system. So far, no studies demonstrate the complete picture of the virtual memory hierarchy including disk. One of the reasons is that there are no proper tools available in the public domain for such studies.

Therefore, we created SYSim, an open-source complete-system simulator aiming at complete memory hierarchy studies. SYSim focuses on demonstrating the detailed interactions in the entire memory hierarchy. SYSim has the ability to instantaneously collect the statistic information in both performance and power consumption.

With SYSim, we extensively conducted the complete-system experiments. We explored disk drive design space, including several disk drive enhancements and technology improvements, during the I/O intensive phase. The experimental results are reported in terms of total system performance (CPI) and power/energy consumption for many SPEC 2000 benchmark applications. We captured unquestionably fascinating behaviors as follows:

- The disk research community uses average response time as the metric to measure the disk system performance, which includes both disk read and write response time. However, we found that, during the I/O intensive phase, the average CPI tracks only average disk read response time and not overall average disk response time. This behavior stays true with the disk request stream consisting of any ratio of reads and writes. Therefore, average read response time should be a better representative for measuring the disk system performance and relating it to the entire system performance.

- The effect of the size of the disk cache is limited to the presence of the cache with a particular size. Meaning, with constant DRAM capacity, increasing the size of the disk cache will not result in better performance if the disk cache is already large enough. The disk cache organization does not have any impact to the performance. Only one segment of disk cache is sufficient for our case. This behavior is in agreement with the disk-level simulation results in [78]. However, increasing disk cache size benefits increasing write traffic.
- In disk read-dominating applications, Disk Prefetching is more important than increasing the disk RPM. That is, rotational latency and bandwidth can be overcome by simple prefetching mechanism. In such applications, the request stream is often a stream of sequential reads. Therefore, requests mostly hit the prefetched data in the disk cache. By hitting the cache, there is no need to access the physical disk and move the disk mechanical parts, which are the major reason for long I/O latency.
- In applications with both disk reads and writes, the disk RPM matters. The reason is the disk maintains the concepts of non-volatile storage, so when the write comes in, it is required to write to the disk immediately if there is no sufficient space in the cache. Therefore, if a long write burst is scheduled before a read, even if the read is a cache hit, the read has to wait for the write burst to complete before it hits the data in the cache. The waiting time can be very long since the write burst has to move the disk mechanical parts. As a result, the RPM affects the read response time, which represents the overall system performance. The experiment shows that using some techniques to eliminate the writes may improve the performance significantly in this case.

- The cost of writing in a RAID system is significant as the RAID system usually suffers from small writes [80]. The reason is that a disk write in a RAID system requires parity calculation, so a disk write in RAID system takes longer than a write in a single disk system. If the cost of a write is reduced, such as by implementing write buffer mechanism, the overall system performance will be improved by potentially an order of magnitude.
- Individual DRAM chips dissipate little power, but a system must have a substantial amount of DRAM to reduce disk traffic and thus prevent the disk from dissipating significant power. Since the total system performance is related to DRAM capacity more strongly than disk RPM, and an active disk dissipates more power than individual active DRAMs, it is wise to increase the DRAM capacity rather than increase the disk RPM. However, when there is enough amount of DRAM in the system, the total DRAM power can be significant and can approach that of the disk system. In this case, higher disk RPM also increases system energy without performance benefit, so using high RPM disk with sufficiently large DRAM capacity is a bad design point.
- The energy consumption has more significance than the power dissipation. While the power stays constant in most systems with similar features, the energy consumed can change significantly with different disk parameters. This is because the I/O latency, resulted from different disk parameters, substantially prolongs the program execution time. The difference in energy in different systems can be as much as a factor of 10.
- In systems with high RPM disks, techniques aiming at increasing the system

bandwidth alone, such as increasing the number of RAID disks and RPM, fail to improve the total system performance directly. In some cases, the systems with higher bandwidth perform worse than the systems with lower bandwidth. To significantly improve total system performance further, the disk enhancement techniques are required in the systems with fast disks.

APPENDIX: SPEC CPU2000

SPEC CPU2000 [13] is the industry-standardized CPU-intensive benchmark suite. SPEC designed CPU2000 to provide a comparative measure of computation-intensive performance across the most extensive practical range of hardware. The benchmark suite is comprised of source code benchmarks developed from real user applications. The benchmarks in the suite have a goal to measure the performance of the processor, memory and compiler on the tested system.

The SPEC CPU2000 benchmarks are intended to exercise the CPU, the memory hierarchy, and the compilers. The data collected show that SPEC CPU2000 met its goals for memory footprint. Meaning, most benchmarks are larger than common cache sizes, many are larger than 100MB, and none are larger than 200MB.

This section provides details about a set of SPEC CPU2000 suite used in the experiments in this dissertation. A selection of seven benchmarks from integer suite and a selection of two benchmarks from floating-point suite are explained.

A.1. CFP2000 (Floating Point Suite of SPEC CPU2000):

A.1.1.188.ammp

188.ammp is classified as a Computational Chemistry program. It models large systems of molecules usually associated with Biology. The benchmark runs molecular dynamics on a protein-inhibitor complex, which is embedded in water. The energy is approximately calculated by a classical potential or "force field". The protein in the complex is HIV protease complexed with the inhibitor indinavir. There are 9582 atoms in the water and the protein, making the benchmark a representative of a typical large simulation. This 188.ammp benchmark is a derivation from published work on an understanding of drug resistance in HIV. The problem traces how the atoms move from an initial coorinates and initial velocities. The output is the energy of the final configuration of atoms. It is written in C.

A.1.2.172.mgrid

172.mgrid is a Multi-grid Solver program, which is a 3D Potential Field program. The 172.mgrid benchmark demonstrates the capabilities of a simple multigrid solver in computing a three dimensional potential field. The benchmark is adapted by SPEC from the NAS Parallel Benchmarks with modifications for portability and a different workload as follows

1. It solves only a constant coefficient equation, and only on a uniform cubical grid.
2. It solves only a single equation, representing a scalar field rather than a vector field.

The output includes echoing some of the inputs and the smoothed approximate inverse. The main part of the output is from the smoothed approximate inverse. However, only a small portion of the smoothed output is printed. This output is only enough to assure that all work is being done and to check intermediate results for accuracy. Additionally, the L2 norm and Inf norms are used as a checksum of the output. The benchmark is written in Fortran 77.

A.2. CINT2000 (Integer Component of SPEC CPU2000)

A.2.1.164.gzip

164.gzip (GNU zip) is a popular data compression program written by Jean-Loup Gailly for the GNU project. The 164.gzip benchmark uses Lempel-Ziv coding (LZ77) as its compression algorithm. However, SPEC's version of gzip performs only reading I/O for input, but no file I/O for output. All compression and decompression are computed entirely in memory. This is to differentiate the work done in the CPU from the work done in the memory subsystem. Reference workload of 164.gzip includes five components: a large TIFF image, a web server log, a program binary, random data, and a source tar file. The random data is selected to test gzip's worst-case behavior. Beside the random data, the rest of the workload components were selected as a realistic representative set of general inputs that gzip might encounter regularly. Every input set is compressed and decompressed at several different blocking factors or compression levels. Then, the end result of the process is compared against the original data after each step. The output files are generated to include a brief outline of the benchmark activities during execution. Output sizes for each compression and decompression are included to facilitate validation. To validate, the results of decompression are compared against the input data to ensure that they match. The benchmark is written in C.

A.2.2.176.gcc

176.gcc is a C Language optimizing compiler. The 176.gcc benchmark is based on gcc Version 2.7.2.2 from GNU. It generates code for a Motorola 88100 processor. The benchmark executes as a compiler with multiple optimization flags enabled. Unlike GNU gcc, 176.gcc has its inlining heuristics altered slightly. Therefore, more code can be inlined than it would be typical on a Unix system in 1997. The reason is the expectation that this feature would be more typical for compiler usage in 2002. The change was done so that 176.gcc would spend more time analyzing it's source code inputs, and use more memory. Despite of this effect, 176.gcc would have done less analysis, and required more input workloads to achieve the run times specification for SPECint2000. There are five input workloads included in 176.gcc. All of them are preprocessed C code (.i files). First, integrate.i and expr.i come from the source files of gcc itself. 166.i is produced by concatenating the Fortran source files of a SPECint2000 candidate benchmark, then using the f2c translator to produce C code, and then pre-processing. 200.i is produced with the same method from a previous version of the SPECfp2000 benchmark Finally, 200.sixtrack and scilab.i are produced with the same method from a version of the Scilab program. All output files are 88100 assembly code files. The code of 176.gcc is in C.

The known portability issues for the 172.gcc benchmark are as follows:

1. The code requires the knowledge of the platform endian of the host it runs on. The default endian for 176.gcc is set to little endian. To run correctly on a big- endian machine, the flag HOST_WORDS_BIG_ENDIAN must be defined when the benchmark is compiled (eg -DHOST_WORDS_BIG_ENDIAN).
2. Some of the optimizations 176.gcc performs require platform-dependent calculation of floating point constants. These requirements form an insignificant amount of computation time, depending on IEEE floating point format to produce a correct result.

3. 176.gcc is not an ANSI C program. It uses GNU extensions.
4. 176.gcc is inherently a 32-bit program. SPEC has successfully ported 176.gcc to many 64-bit UNIX implementations. However, use of high optimization levels with a 64 bit system in conjunction with inlining of procedures from different source files may reveal some 64-bit portability issues with 176.gcc.
5. SPEC has changed176.gcc slightly in order to build properly with newer versions of GCC. If you're using an old gcc (~2.6 or older) to build 176.gcc, you should define the __OLDANDBUGGY__GNUC__ flag.

A.2.3.181.mcf

181.mcf is a Combinatorial optimization / Single-depot vehicle scheduling. It is a benchmark derived from a program used for single-depot vehicle scheduling in public mass transportation. The benchmark is written in C, and The benchmark version uses almost entirely integer arithmetic. The program is designed to solve single-depot vehicle scheduling (sub-)problems occurring in the planning process of public transportation companies. It take into account one single depot and a homogeneous vehicle fleet. It is based on a line plan and service frequencies, so-called timetabled trips with fixed departure/arrival locations and times derived. Each of this timetabled trip has to be served by exactly one vehicle. The links between these trips are called dead-head trips. Additionally, there are pull-out and pull-in trips for leaving and entering the depot, respectively. Cost coefficients are provided for all dead-head, pull-out, and pull-in trips. The purpose is to schedule all timetabled trips such that the number of necessary vehicles is minimized and, secondarily, the operational costs among all minimal fleet solutions are also minimized. For simplification, the benchmark assumes that each pull-out and pull-in trip is defined implicitly with a duration of 15 minutes and a cost coefficient of 15. For the considered single-depot case, the problem can be formulated as a large-scale minimum-cost flow problem that can be solved with a network simplex algorithm accelerated with a column generation. The main calculation of the benchmark 181.mcf is the network simplex code "MCF Version 1.2 -- A network simplex implementation", which is embedded in the column generation process. The network simplex algorithm is a specialized version of the prominent simplex algorithm for network flow problems. The linear algebra of the general algorithm is replaced by simple network operations, such as finding cycles or modifying spanning trees that can be performed very rapidly. The main work of our network simplex implementation is pointer and integer arithmetic.

The input file includes the followings:

- the number of timetabled and dead-head trips,
- its starting and ending time for each timetabled trip,
- its starting and ending timetabled trip and its cost for each dead-head trip.

Worst case execution time is pseudo-polynomial in terms of the number of timetabled and dead-head trips and in the amount of the maximal cost coefficient. However, the expected execution time is in the order of a low-order polynomial. The benchmark memory footprint is approximately 100 and 190 megabyte for a 32 and a 64 bit architecture, respectively. The benchmark generates two output files, inp.out and mcf.out. The inp.out output file consists of log information and a checksum while the mcf.out output file contains check output values describing an optimal schedule computed by the program.

A.2.4.197.parser

197.parser is a Word processing program. The Link Grammar Parser is a syntactic parser of English, based on link grammar, an original theory of English syntax. The program assigns a syntactic structure to a given sentence. The syntactic structure consists of set of labeled links connecting pairs of words.

The parser includes a dictionary of about 60000 word forms. The dictionary covers a wide variety of syntactic constructions, including many rare and idiomatic ones. The parser is robust. It has the capability to skip over portions of the sentence that it cannot understand, and assign structure to the rest of the sentence. It can handle unknown vocabulary, and intelligently guess from context about the syntactic categories of unknown words. The input is a sequence of proposed sentences, one per line. It is sensitive to punctuation and case. The output is an analysis of each input sentence. The analysis output consists of a set of links capturing the grammatical structure of the sentence, a labelling of each word with an appropriate part of speech tag, and a judgement of the grammaticality of the input sentence. Words in square brackets are determined superfluous by the parser. The parser is written in ANSI C.

A.2.5.255.vortex

255.vortex is a Database program. The benchmark 255.vortex is a subset of a full object oriented database program called VORTEx. VORTEx stands for "Virtual Object Runtime EXpository". It is a single-user object-oriented database transaction benchmark, exercising a system kernel coded in integer C. The VORTEx benchmark is a derivative of a full OODBMS that has been customized to conform to SPEC CINT2000 guidelines.

Transactions operated on the database are translated through a schema. The function of a schema is to provide the necessary information to generate the mapping of the internally stored data block to a model viewable in the context of the application. The benchmark schema is pre-configured to manipulate three different database, including mailing list, parts list, and geometric data. Both little-endian and big-endian binaries for the schema are provided in the benchmark.

The 255.vortex benchmark builds and manipulates three separate, but inter-related databases based on the schema. The size of the database is scalable, but has been restricted to about 200 Mbytes for CINT2000 guidelines. However, this version of VORTEx benchmark has been modified to prevent committing transactions to memory in order to remove input-output activity from the benchmark.

The workload of VORTEx has been modeled to reflect general object-oriented database benchmarks with modifications to vary the mix of transactions.

The 255.vortex benchmark executes three different times with different sequences of transactions. Each time a different combination of database insert, delete and lookup transactions is used to simulate different database usage patterns.

The benchmark 255.vortex use three different workloads, simulating different dataset sizes and access patterns. Each run, one for each workload, produces one output file. Each output file (vortex1.out, vortex2.out, and vortex3.out) is a log of all transactions occurring during the execution of the benchmark. These transactions include creating entries in the database, deleting entries, and entry lookups. 255.vortex is written in C.

A.2.6.256.bzip2

256.bzip2 is a compression program. The 256.bzip2 benchmark is a derivative of Julian Seward's bzip2 version 0.1. Like SPEC version of gzip, the only difference between bzip2 0.1 and 256.bzip2 is that SPEC's version of bzip2 performs no file I/O rather than reading the input. All compression and decompression occurs entirely in memory to help isolate the work done to only the CPU and memory subsystem. The output files consist of a brief outline of what the benchmark is doing during its execution. Output sizes for each compression and decompression are printed to facilitate validation. To validate the execution, the results of decompression are compared against the input data to ensure that they match. The 256.bzip2 benchmark is written in ANSI C.

A.2.7.300.twolf

TimberWolfSC is a placement and global routing application package. The TimberWolfSC package is used to create the lithography artwork needed for the production of microchips. Especially, it determines the placement and global connections for standard cells, which constitute the microchip. The standard cell is usually a group of transistors. The placement problem is a permutation. Meaning, an exploration of the state space would take an execution time proportional to the factorial of the input size. For example, To solve a problems with 70 cells, a brute force algorithm would take the execution time of the factorial of 70, which is an unacceptable amount of time even on the world's fastest computer. Instead, the TimberWolfSC program implements simulated annealing as a heuristic to find relatively optimal solutions for the row-based standard cell design style. In this implementation, transistors are grouped together to form standard cells. These standard cells are placed in rows each or which share power and ground connections by abutment. The simulated annealing algorithm has found the relative optimals to a large group of placement problems. After the placement step, the global router interconnects the microchip design. It is implemented with a constructive algorithm followed by iterative improvement.

The basic simulated annealing algorithm has been widely used in many applications since its first introduction in 1983. The SPEC suite version is the most numerically intensive version. Recently, the newer versions have reduced runtimes by intelligent reductions in the search space. However, the solution search strategy and cost functions remain the same to later versions.

SPEC has customized the version of TimberWolfSC so that it would capture the flavor of many implementations of simulated annealing. The submitted version spends most of its time in the inner loop calculations. With this behavior, this version often creates cache misses due to traversing memory. In fact, the execution of small jobs on this version is similar to later simulated annealing versions executing on large jobs. The reason is to insure the applicability of the benchmark in the future versions of the program running large instances. The submitted version should be extremely computer-intensive, but realistic for future problems.

Three test problems are provided for the SPEC 300.twolf benchmark. The first problem is a small synchronous circuit which is being placed and routed as a subchip. The second test circuit is the MCNC primary one benchmark circuit. It is one of the most frequently executed benchmark circuits. The third test case is a structured circuit found in the MCNC benchmark suite. In all test problems, the TimberWolf program is required to determine the position of the standard cells and determine the interconnection of the netlist. Additionally, the global router must add extra cells, called feedthrus, to complete the route if not enough space is present between two adjacent standard cells. The input files are composed of the block description file, the netlist file, the net weighting file, and the parameter file. The block description file describes the number and position of the rows, where standard cells are to be placed. A valid placement is defined as the placement all of the cells are placed within the specified rows without any overlap between cells. The netlist file describes the standard cells and the connection network between cells. At this moment, the physical location of these connections has not been determined.

Two output files are created for each test circuit: the placement file and the global routing file. The benchmark is written in C.

References

[1] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, L.K. John, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-8), Cambridge, MA, pages 141-150, February, 2002.

[2] Jianwei Chen, Michel Dubois, and P. Stenstrom, "SimWattch: An Approach to Integrate Complete-System with User-Level Performance/Power Simulators," IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2003), March 2003.

[3] H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H. Lipasti, "Precise and Accurate Processor Simulation," In Proceedings of the Fifth Workshop on Computer Architecture Evaluation Using Commercial Workloads, pages 13–22, Feb. 2002.

[4] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: The SimOS Approach," IEEE Parallel and Distributed Technology: Systems and Applications, 3(4):34–43, 1995.

[5] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," IEEE Computer, 35(2):50–58, Feb. 2002.

[6] R. C. Bedichek, "Some Efficient Architecture Simulation Techniques," Winter 1990 USENIX Conference, pages 53–63, Jan. 1990.

[7] P. S. Magnusson, "A Design For Efficient Simulation of a Multiprocessor," First International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 69–78, Jan. 1993.

[8] R. C. Bedichek, "Talisman: Fast and accurate multicomputer simulation," In Proceedings of the 1995 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, pages 14–24, May 1995.

[9] Carl J. Mauer, Mark D. Hill and David A. Wood, "Full System Timing-First Simulation," In Proceedings of the 2002 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems June, 2002

[10] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy, T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. Van Hensbergen, and Lixin Zhang, "Mambo -- A Full System Simulator for the PowerPC Architecture," ACM SIGMETRICS Performance Evaluation Review, Volume 31, Number 4, March 2004.

[11] The Bochs IA-32 Emulator Project. http://bochs.sourceforge.net

[12]S. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-chip Caches," In WRL Research Report 93/5, DEC Western Research Laboratory, 1994.

[13]Systems Performance Evaluation Cooperative. SPEC Benchmarks. http://www.spec.org.

[14]G. Ganger, B.Worthington, and Y. Patt, "The DiskSim Simulation Environment Version 2.0 Reference Manual," http://www.ece.cmu.edu/ ganger/disksim/.

[15]S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, "DRPM: Dynamic Speed Control for Power Management in Server Class Disks," In the Proceedings of the International Symposium on Computer Architecture (ISCA), pages 169-179, June, 2003.

[16]IBM Hard Disk Drive - Ultrastar 36ZX. http://www.storage.ibm.com/ hdd/ultra/ ul36zx.htm.

[17]David T. Wang, "Modern DRAM Memory systems: Performance Analysis and Scheduling Algorithm," Ph.D. Dissertation, Electrical and Computer Engineering, University of Maryland at College Park, 2005.

[18]Jeff Janzen, The Micron System-Power Calculator.
http://www.micron.com/products/dram/syscalc.html

[19]D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," In Proceedings of the 27th Annual International Symposium on Computer Architecture, June 2000.

[20]W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "The Design and Use of SimplePower: A cycle-accurate energy estimation tool," In Proceedings of the Design Automation Conference (DAC), June 2000.

[21]G. Cai and C.H. Lim, "Architectural Level Power/Performance Optimization and Dynamic Power Estimation," in Proceedings of Cool Chips Tutorial, in conjunction with MICRO32, Nov. 1999, pp. 90-113.

[22]K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob, "The performance and energy consumption of embedded real-time operating systems," IEEE Transactions on Computers, vol. 52, no. 11, pp. 1454-1469. November 2003.

[23]T. L. Cignetti, K. Komarov, and C. S. Ellis, "Energy Estimation Tools for the Palm," In Proceedings of ACM MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Modile Systems, August 2000.

[24]M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno and A. Sangiovanni-Vincentelli, "Efficient Power Estimation Techniques for HW/SW Systems," In Proceedings of the

IEEE VOLTA'99 International Workshop on Low Power Design , pp. 191-199, Como, Italy, March 4-5, 1999.

[25]J.R. Lorc, “A complete picture of the energy consumption of a portable computer,” Master's thesis, University of California, Berkeley, December 1995.

[26]Robert P. Dick , Ganesh Lakshminarayana , Anand Raghunathan , Niraj K. Jha, “Power analysis of embedded operating systems,” Proceedings of the 37th conference on Design automation, p.312-315, June 05-09, 2000, Los Angeles, California, United States

[27]Tajana Šimunic , Luca Benini , Giovanni De Micheli, “Cycle-accurate simulation of energy consumption in embedded systems,” Proceedings of the 36th ACM/IEEE conference on Design automation, p.867-872, June 21-25, 1999, New Orleans, Louisiana, United States

[28]D. Burger and T. M. Austin. “The SimpleScalar Tool Set, Version 2.0,” Computer Architecture News, pages 13-25, June 1997.

[29]Mirko Loghi, Massimo Poncino, Luca Benini, “Cycle-accurate power analysis for multiprocessor systems-on-a-chip,” Proceedings of the 14th ACM Great Lakes symposium on VLSI, April 2004.

[30]Giovanni Beltrame, Gianluca Palermo, Donatella Sciuto, Cristina Silvano, “Plug-in of power models in the StepNP exploration platform: analysis of power/performance trade-offs,” Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems, September 2004

[31]Gilberto Contreras, Margaret Martonosi, Jinzhan Peng, Roy Ju and Guei-Yuan Lueh. “XTREM: A Power Simulator for the Intel XScale Core,” The 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04), June 2004.

[32]R. Y. Chen, M. J. Irwin, and R. S. Bajwa, “Architecture-level power estimation and design experiments,” ACM Transactions on Design Automation of Electronic Systems, 2001.

[33]N. An, S. Gurumurthi, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, “Energy-Performance Trade-Offs for Spatial Access Methods on Memory Resident Data,” In The VLDB Journal,11(3):179-197, November, 2002.

[34]Tajana Simunic, Luca Benini, Giovanni De Micheli, “Energy-efficient design of battery-powered embedded systems,” International Symposium on Low Power Electronics and Design, Stanford University, 212-17, August, 1999.

[35]Vinodh Cuppu and Bruce Jacob, "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?," In Proc. 28th International Symposium on Computer Architecture (ISCA 2001), pp. 62-71, Goteborg Sweden, June 2001.

[36]Luca Benini, Giovanni de Micheli, "System-level power optimization: techniques and tools," ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 5 Issue 2 , April 2000.

[37]D. Lidsky and J. Rabaey, "Low-power design of memory intensive functions," Proceedings of the IEEE Symposium on Low Power Electronics (Sept.), IEEE Computer Society Press, Los Alamitos, CA, 16-17, 1994.

[38]F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, "Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design," Kluwer Academic, Dordrecht, Netherlands, 1998a.

[39]J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," Morgan Kaufmann, Second edition, 1996, pp. 487.

[40]M. Kandemir , N. Vijaykrishnan , M. J. Irwin , W. Ye, "Influence of compiler optimizations on system power," Proceedings of the 37th conference on Design automation, p.304-307, June 05-09, 2000, Los Angeles, California.

[41]F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, "Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design," Kluwer Academic, Dordrecht, Netherlands, 1998.

[42]F. Catthoor, S. Wuytack, E. De Greef, F. Franssen, L. Nachtergaele, and H. De Man, "System-level transformations for low-power data transfer and storage," In Low-Power CMOS Design, R. Chandrakasan and R. Brodersen, Eds. IEEE Press, Piscataway, NJ, 1998.

[43]C.-l. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: a case study," In Proceedings of the 1995 International Symposium on Low Power Design (ISLPD-95, Dana Point, CA, Apr. 23–26), M. Pedram, R. Brodersen, and K. Keutzer, Eds. ACM Press, New York, NY, pp. 63–68, 1995.

[44]M. B. Kamble and K. Ghose, "Analytical energy dissipation models for low-power caches," In Proceedings of the 1997 International Symposium on Low Power Electronics and Design (ISLPED '97, Monterey, CA, Aug. 18–20), B. Barton, M. Pedram, A. Chandrakasan, and S. Kiaei, Eds. ACM Press, New York, NY, pp. 143–148, 1997.

[45]W. Shiue and C. Chakrabarti, "Memory exploration for low power, embedded systems," In Proceedings of the Conference on Design Automation (June), pp. 140–145, 1999.

[46]Luca Benini , Alberto Macii , Enrico Macii , Massimo Poncino, "Synthesis of application-specific memories for power optimization in embedded systems," Proceedings of the 37th conference on Design automation, p.300-303, June 05-09, 2000, Los Angeles, California, United States.

[47]Peter Grun , Nikil Dutt , Alex Nicolau, "APEX: access pattern based memory architecture exploration," Proceedings of the 14th international symposium on Systems synthesis, September 30-October 03, 2001, Montreal, P.Q., Canada.

[48]A. H. Farrahi, G. E. Téllez, and M. Sarrafzadeh, "Memory segmentation to exploit sleep mode operation," In Proceedings of the 32nd ACM/IEEE Conference on Design Automation (DAC '95, San Francisco, CA, June 12–16), B. T. Preas, Ed. ACM Press, New York, NY, pp. 36–41, 1995.

[49]A. H. Farrahi and M. Sarrafzadeh, "System partitioning to maximize sleep time," In Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 452–455, 1995.

[50]Luca Benini , Alberto Macii , Massimo Poncino, "A recursive algorithm for low-power memory partitioning," Proceedings of the 2000 international symposium on Low power electronics and design, p.78-83, July 25-27, 2000, Rapallo, Italy.

[51]Hsien-Hsin S. Lee , Gary S. Tyson, "Region-based caching: an energy-delay efficient memory architecture for embedded processors," In Proceedings of the international conference on Compilers, architectures, and synthesis for embedded systems, p.120-127, November 17-19, 2000, San Jose, California, United States.

[52]J. Kin, M. Gupta, and W.h. Mangione-smith, "The filter cache: an energy efficient memory structure," In Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 30, Research Triangle Park, NC, Dec. 1–3), M. Smotherman and T. Conte, Eds. IEEE Computer Society Press, Los Alamitos, CA, pp. 184–193, 1997.

[53]P. Grun , N. Dutt , A. Nicolau, "Access pattern based local memory customization for low power embedded systems," Proceedings of the conference on Design, automation and test in Europe, p.778-784, March 2001, Munich, Germany.

[54]Jayaprakash Pisharath , Alok Choudhary, "An integrated approach to reducing power dissipation in memory hierarchies," Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems, October 08-11, 2002, Greenoble, France.

[55]Afzal Malik , Bill Moyer , Roger Zhou, “Embedded cache architecture with programmable write buffer support for power and performance flexibility,” Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems, October 08-11, 2002, Greenoble, France.

[56]Chuanjun Zhang , Frank Vahid , Jun Yang , Walid Najjar, “A way-halting cache for low-energy high-performance systems,” Proceedings of the 2004 international symposium on Low power electronics and design, August 09-11, 2004, Newport Beach, California, USA.

[57]Rui Min , Wen-Ben Jone , Yiming Hu, “Location cache: a low-power L2 cache system,” Proceedings of the 2004 international symposium on Low power electronics and design, August 09-11, 2004, Newport Beach, California, USA.

[58]S. Liao, S. Devadas and K. Keutzer, “Code density optimization for embedded DSP processors using data compression techniques,” In IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 17, 7 (July), pp. 601–608, 1998.

[59]H. Lekatsas and W. Wolf, “Code compression for embedded systems,” In Proceedings of the 35th Annual Conference on Design Automation (DAC ’98, San Francisco, CA, June 15–19), B. R. Chawla, R. E. Bryant, and J. M. Rabaey, Eds, ACM Press, New York, NY, pp. 516–521, 1998.

[60]Luca Benini , Alberto Macii , Enrico Macii , Massimo Poncino, “Selective instruction compression for memory energy reduction in embedded systems,” Proceedings of the 1999 international symposium on Low power electronics and design, p.206-211, August 16-17, 1999, San Diego, California, United States.

[61]S. Segars, K. Clarke, and L. Goudge, “Embedded control problems, thumb and the ARM7TDMI,” IEEE Micro 15, 5 (Dec.), pp. 22–30, 1995.

[62]Wen-Tsong Shiue , Chaitali Chakrabarti, “Memory Design and Exploration for Low Power, Embedded Systems,” Journal of VLSI Signal Processing Systems, v.29 n.3, p.167-178, November 2001.

[63]S. Wuytack, F. Catthoor, and H. De Man, “Transforming set data types to power optimal data structures,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 15, 6 (June), pp. 619–629, 1997.

[64]J.L. Da Silva, F. Catthoor, D. Verkest, and H. De Man, “Power exploration for dynamic data types through virtual memory management refinement,” In Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED ’98, Monterey, CA, Aug. 10–12), A. Chandrakasan and S. Kiaei, Eds. ACM Press, New York,NY, pp 311–316, 1998.

[65]C. Gebotys, "Low energy memory and register allocation using network flow," In Proceedings of the 34th Conference on Design Automation (DAC '97, Anaheim, CA, June), pp. 435–440, 1997.

[66]Alvin R. Lebeck , Xiaobo Fan , Heng Zeng , Carla Ellis, "Power aware page allocation," ACM SIGOPS Operating Systems Review, v.34 n.5, p.105-116, Dec. 2000.

[67]Xiaobo Fan , Carla Ellis , Alvin Lebeck, "Memory controller policies for DRAM power management," Proceedings of the 2001 international symposium on Low power electronics and design, p.129-134, August 2001, Huntington Beach, California, United States.

[68]Nam Sung Kim , Krisztián Flautner , David Blaauw , Trevor Mudge, "Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," In Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture, November 18-22, 2002, Istanbul, Turkey

[69]V. Delaluz , M. Kandemir , N. Vijaykrishnan , M. J. Irwin, "Energy-oriented compiler optimizations for partitioned memory architectures," Proceedings of the international conference on Compilers, architectures, and synthesis for embedded systems, p.138-147, November 17-19, 2000, San Jose, California, United States.

[70]Mahmut Kandemir , Ugur Sezer , Victor Delaluz, "Improving memory energy using access pattern classification," In Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, November 04-08, 2001, San Jose, California.

[71]P. Panda and N. Dutt, "Low-power memory mapping through reducing address bus activity," IEEE Trans. Very Large Scale Integr. Syst. 7, 3 (Sept.), 309–320, 1999.

[72]Preeti R. Panda , Nikil D. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," Proceedings of the 1996 European conference on Design and Test, p.63, March 11-14, 1996.

[73]Wei-Chung Cheng , Massoud Pedram, "Low power techniques for address encoding and memory allocation," Proceedings of the 2001 conference on Asia South Pacific design automation, p.245-250, January 2001, Yokohama, Japan.

[74]Naehyuck Chang , Kwanho Kim , Jinsung Cho, "Bus encoding for low-power high-performance memory systems," Proceedings of the 37th conference on Design automation, p.800-805, June 05-09, 2000, Los Angeles, California, United States.

[75]Yingwu Zhu, Yiming Hu, "Can Large Disk Built-in Caches Really Improve System Performance?", in Proceedings of the ACM SIGMETRICS 2002 (extended abstract), Marina Del Rey, California, June 15-19, 2002. pp. 284-285.

[76]Yiming Hu , Qing Yang, "DCD—disk caching disk: a new approach for boosting I/O performance", Proceedings of the 23rd annual international symposium on Computer architecture, p.169-178, May 22-24, 1996, Philadelphia, Pennsylvania, United States

[77]Jung-ho Huh, Tae-mu Chang, "Hierarchical disk cache management in RAID 5 controller", Journal of Computing Sciences in Colleges archive, Volume 19 , Issue 2 (December 2003), P.47 - 59, 2003

[78]W. Hsu , A. J. Smith, "The performance impact of I/O optimizations and disk improvements", IBM Journal of Research and Development, v.48 n.2, p.255-289, March 2004

[79]P. Biswas, K. K. Ramakrishnan, and D. Towsley. "Trace driven analysis of write caching policies for disks", ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 13 23, 1993.

[80]D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Proc. Int'l Conf. Management of Data, ACM, 1989, pp. 109-116.

[81]A. J. Smith, "Disk Cache: Miss Ratio Analysis and Design Considerations." Proceedings of the 5th annual Symposium on Computer Architecture, Apr. 1985, 242-248.

[82]"SAMSUNG Teams with Microsoft to Develop First Hybrid HDD with NAND Flash Memory.", http://www.samsung.com/Products/HardDiskDrive/news/HardDiskDrive_20050425_0000117556.htm, Apr 25, 2005.

[83]A. J. Smith, "On the effectiveness of buffered and multiple arm disks", In Proceedings of the 5th Annual Symposium on Computer Architecture (April 03 - 05, 1978). ISCA '78. ACM Press, New York, NY, 242-248.

[84]A. J. Smith, "Sequentiality and Prefetching and Database Systems", ACM Trans. Database Syst. 3, No. 3, 223-247 (September 1978).

[85]W. W. Hsu, A. J. Smith, and H.C. Young, "I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks--An Analysis at th eLogical Level", ACM Trans. Database Syst. 26, No. 1, 96-143 (March 2001).

[86]R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed Prefetching and Caching," Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain, CO, December 1995, pp. 79 -95.

[87]L. Haas, W. Chang, G. Lohman, M. McPherson, P. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. Carey, and E. Shekita, "Starburst Mid-Flight: As the Dust Clears", IEEE Trans. Knowledge & Data Eng. 2, No. 1, 143-160 (March 1990).

[88]J. Z. Teng and R. A. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance", IBM Syst. J. 23, No. 2, 211-218 (1984).

[89]F. Chang and G. A. Gibson, "Automatic I/O Hint Generation Through Speculative Execution", Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA, February 1999, pp. 1-14.

[90]V. Soloviev, "Pretching in segmented disk cache for multi-disk systems", In Proceedings of the Fourth Workshop on I/O in Parallel and Distributed Systems: Part of the Federated Computing Research Conference (Philadelphia, Pennsylvania, United States, May 27 - 27, 1996). IOPADS '96. ACM Press, New York, NY, 69-82.

[91]S. W. Son and M. Kandemir, "Energy-aware data prefetching for multi-speed disks", In Proceedings of the 3rd Conference on Computing Frontiers (Ischia, Italy, May 03 - 05, 2006). CF '06. ACM Press, New York, NY, 105-114.

[92]F. Chen, S. Jiang, and X. Zhang, "SmartSaver: turning flash drive into a disk energy saver for mobile computers", In Proceedings of the 2006 international Symposium on Low Power Electronics and Design (Tegernsee, Bavaria, Germany, October 04 - 06, 2006). ISLPED '06. ACM Press, New York, NY, 412-417.

[93]M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system", ACM Trans. Comput. Syst. 10, 1 (Feb. 1992), 26-52.

[94]A. Varma and Q. Jacobson, "Destage Algorithms for Disk Arrays with Nonvolatile Caches", IEEE Trans. Comput. 47, 2 (Feb. 1998), 228-235.

[95]J. A. Solworth and C. U. Orji, "Write-only disk caches", In Proceedings of the 1990 ACM SIGMOD international Conference on Management of Data (Atlantic City, New Jersey, United States, May 23 - 26, 1990). SIGMOD '90. ACM Press, New York, NY, 123-132.

[96]B. Hong, F. Wang, S. A. Brandt, D. D. Long, and T. J. Schwarz, "Using MEMS-based storage in computer systems---MEMS storage architectures", Trans. Storage 2, 1 (Feb. 2006), 1-21.

[97]P. M. Chen and E. K. Lee, "Striping in a RAID level 5 disk array", In Proceedings of the 1995 ACM SIGMETRICS Joint international Conference on Measurement and Modeling of Computer Systems (Ottawa, Ontario, Canada, May 15 - 19, 1995). B. D. Gaither, Ed. SIGMETRICS '95/PERFORMANCE '95. ACM Press, New York, NY, 136-145.

[98]Hitachi Global Storage Technologies--HDD Technology Overview Charts, http://www.hitachigst.com/hdd/technolo/overview/storagetechchart.html

[99]C. Ruemmler and J Wilkes, “UNIX disk access patterns”, Proceedings of Winter 1993 USENIX (San Diego, CA, 25--29 January 1993), pages 405--20, January 1993.

[100]TCP: Transaction Processing Performance Council, http://www.tpc.org/default.asp.

[101]Steven A. Przybylski, “Cache and Memory Hierarchy Design, A performancedirected approach”, Morgan Kaufmann Publishers, Inc, 1990.

[102]Bruce Jacob, Spencer Ng, David Wang, Aamer Jaleel, and Samuel Rodriguez, “Memory Systems: Cache, DRAM, Disk —A Holistic Approach to Design”, Morgan Kaufmann Publishers, Inc., to be published in 2007.