PERFORMANCE OPTIMIZATION OF HIERARCHICAL MEMORY SYSTEMS

by

NAGI NASSIEF MEKHIEL, M.A.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

PERFORMANCE OPTIMIZATION OF HIERARCHICAL MEMORY SYSTEMS

DOCTOR OF PHILOSOPHY (1995) McMASTER UNIVERSITY

(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: Performance Optimization of Hierarchical Memory Systems

AUTHOR: Nagi Nassief Mekhiel

SUPERVISOR: Dr. Daniel McCrackin, Department of Electrical and Computer Engineering

NUMBER OF PAGES: xviii,153

ABSTRACT

The gap between processor speeds and memory speeds is increasing. The performance of supercomputers and the scalability of multiprocessor systems is very dependent on the memory system speed.

A cache system helps to narrow the processor/memory speed gap, but cannot completely decouple the processor from slow memory.

The optimization of main memory performance and the use of a deep multi-level cache hierarchy are proposed here to bridge the processor/memory latency gap.

A novel design that combines optimized bank interleaving with several main memory (DRAM) timing modes to increase memory performance is presented. Four different protocols based on this design are proposed and investigated.

Enforcing the inclusion property for multi-level caches is proposed. A new design that uses three level caches is presented and three different models are given.

A design flow graph that makes the design of a multi-level memory system simpler and more flexible is introduced. Selected traces that match real workloads running on a wide range of computers are used to calculate realistic overall system performance.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Daniel McCrackin, who has helped to make this work possible. I appreciate his encouragement, candid advice, ideas, and outstanding guidance.

I also like to thank my committee members, Dr. David Capson and Dr. Skip Poehlman for their support and help.

I would like to thank Dr. S. Boctor, the chairman of the Electrical Engineering at Ryerson, for his continuous encouragement and support. I also thank professor Ken Clowes at Ryerson for his help.

I would like to thank my beloved wife Hala and my children Caroline and Christopher for their patience and support.

Finally, I would like to dedicate this thesis to my late father who had wished me to get the Ph.D.

Contents

List of Figures

List of Tables

LIST OF SYMBOLS

ATUM	Address Traces Using Microcode
CAS	Column Address Strobe signal
CPI	Clock Per Instruction
CS	Chip Select signal
CPU	Central Processing Unit
DEC	Digital Equipment Corporation
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DTMR	Design Target Miss Ratios
MHz	Mega Hertz
MRU	Most Recently Used
NWA	No Write Allocate
RAS	Row Address Strobe signal
RISC	Reduced Instruction Set Computer
SRAM	Static Random Access Memory
T_{AA}	column Address Access Time

T_{CP}	CAS Precharge Time
T_{RCD}	RAS to CAS Delay
T_{CAC}	Column Access Time from CAS
T_{RP}	RAS Precharge Time
W	Word size in bytes
WA	Write Allocate
WB	Write Back
WBTT	Write Back write Through write Through
WBWT	Write Back Write Through
WE	Write Enable signal
WT	Write Through
WTTB	Write Through write Through write Back
WTTT	Write Through write Through write Through
WTWB	Write Through Write Back
WTWT	Write Through Write Through

Chapter 1

Introduction

1.1 Introduction

It is very important for supercomputer designers to make memory systems fast, as the performance of a computer is greatly influenced by its memory speed, further more, multiprocessor systems the scalability of the system depends on the design of the memory.

The introduction of many new fast processors every year makes the design of a memory system that enables full processor speed more challenging. Furthermore, programs have generally grown in their memory requirements by 1.5 to 2 address bits per year, and the typical main memory needed for any computer normally doubles every three years [13].

An ideal memory system should have the same speed as the processor and have the same size as the processor address space. Building fast memory systems could be done using fast Static Random Access Memory (SRAM), but this is only suitable for small memory systems because of the high cost and low density of SRAM chips. For larger memory systems, the system needs extensive address decoding and data buffering which slow the speed of the memory system.

Fortunately there is a solution to the speed/size conflict of the memory system. This solution comes from the principle of locality. All programs favor small parts of their address space at any instant of time. Programs possess two types of locality: Temporal locality (if an item is referenced now, it is more likely to be referenced again soon) and Spatial locality (the neighbor of a currently referenced item is likely to be referenced soon) [19].

Small parts of the program are used frequently. If they are stored in a fast small memory that could be built with SRAM, the overall system speed will not suffer from having the rest of the program stored in a larger, slower memory. This partitioning of the program into small fast memory and slow larger memory is the principle behind hierarchical memory architecture.

In this thesis we examine optimization of hierarchical memory systems. This Chapter gives an overview of the hierarchical memory architecture, presents the scope of this work, and examines related work and trends in memory design.

1.2 Hierarchical Memory Architecture

A hierarchical memory system is built with multiple levels of memory.

A memory hierarchy generally consists of the following:-

- The Cache: The cache is the closest memory level to the processor and is also the fastest and the smallest. Cache operation is based on the program's temporal and spatial locality. Memory references that are made close together in time or space to recent references can be retrieved from a fast cache.

  The hit rate is defined as the ratio of references found in the cache to the total references requested [19]. It is important to maximize the cache hit rate. Usually the latency (or access time) of the first cache level matches the speed of the processor.

- The Main Memory: The main memory is the second level in the memory hierarchy. It affects the processor performance in that its latency determines the cost of the cache misses. Thus the latency of the main memory is an important parameter affecting the performance of the computer memory system.

- The Virtual Memory: It is generally considered impractical to build a memory system that includes all the address space of a 32 or 64 bit processor. Virtual memory divides the existing physical memory into some form of pages, and lets the operating system transfer data between a secondary store (usually implemented using magnetic disk) and main memory whenever it is needed. In effect, the main memory acts as a cache for the much larger disk store.

In this thesis we concentrate on the upper memory levels including the cache and main memory; virtual memory is largely a software subject and beyond the scope of this study.

1.3 The Novelty of this research

We introduce a design flow graph that is used to visually illustrate the design of each memory system. It shows the decisions taken by the system in each operation. This is a new notational tool that helps in making complicated systems simpler to follow and gives us more flexibility to consider different design options. Recently a similar method has been used to illustrate the algorithm of a system [4].

The traces that we use for simulations are tested and qualified against the results of real workloads running on wide range of computers. Traces affect the result of simulation and can cause them to vary by up to 300% [44]. Furthermore the performance of our design is measured in number of clock cycles per instruction, which gives more realistic results than other studies using miss rates to evaluate the performance of systems.

We present four novel interleaving protocols for the main memory design.

The protocols combine multi-bank interleaving with DRAM access modes. An optimal bank mapping for each protocol is found by varying the bank mapping (a new method). Conventional main memory designs use either interleaving or special DRAM access modes to increase system performance, not both. Conventional multi-bank interleaving uses simple low-order (fixed) bank mapping that does not give the system optimal performance.

We use multi-level cache design to increase the cache performance and enforce the inclusion property for simpler multiprocessor cache coherency protocols. Our multi-level cache design is kept simple by limiting the number of write back caches to a single level. A novel three level cache design is introduced and analyzed. Other recent multi-level cache designs use only up to two levels.

In this research, the performance of the overall memory system is optimized which is a fundamentally different approach from other research work that has optimized one level of the memory hierarchy at a time.

1.4 The Scope of this Work

In Chapter 2 we examine different traces for simulation and qualify them against results on a wide class of workloads running on a wide range of computers and workstations.

Chapter 3 presents a new optimized main memory design that uses multiple bank interleaving and several main memory (DRAM) timing modes.

In Chapter 4 we give a review of cache design basics and single level cache design. Chapter 5 discusses the design of a multi-level cache system and evaluates the performance of both a two level cache and a new three level cache design.

Chapter 6 deals with the performance optimization of the complete memory system using the optimized main memory design of Chapter 3 and the single and multi-level cache designs of Chapter 4 and Chapter 5.

Chapter 7 presents conclusions and gives recommendations for future work.

1.5 Related Work

Previous research work on dealing with memory latency has relied largely on the use of a cache system. There have been numerous articles about cache design. All studies show that as cache size or block size or associativity increase, the rate of performance increase shows diminishing returns [3], [45] and [19].

Newer cache designs tend to use two levels of cache, but as the second level cache size is increased, performance comes to be dominated by the smaller first level cache and increasing the second level cache size yields little additional improvement [31] and [43].

With the current trend of processor speed increasing much more quickly than main memory speed, and with little performance improvement gained from new research, we could very well find future processors running at more clock cycles per instruction than present day machines [8].

In this study we use a multi-level cache design with the inclusion property (blocks in higher cache levels are included in lower cache levels) to optimize cache performance. This improves the first level cache miss rate. Enforcing the inclusion property between different cache levels makes the cache coherence protocols for multiprocessor systems simpler [7] and [19].

In this thesis we focus on obtaining optimal system performance by reducing the processor dependency on the main memory. The goal is to close the processor/memory speed gap.

Our study considers performance optimization at all memory hierarchy levels. We also attack one much-neglected level: the performance of the DRAM main memory itself.

The optimization techniques used in this investigation are applied across the whole memory system because optimizing the performance of a single memory level does not guarantee optimal system performance.

In all of our designs, we use a design flow graph technique which is a new and useful method for obtaining and estimating the interactions between each memory level and different program operations. This method gives a simple view of the entire memory system. It also allows the caching algorithm to be specified and the cache performance to be estimated at the same time.

1.6 Trends in Memory Design

Most current research in memory design concentrates on providing methods to improve cache performance. Some recent research uses multi-level, prefetch, victim caches and code restructuring methods to improve cache performance.

It seems fair to say that the intensity of the research in memory system design is growing as the speed gap between processors and memory increases.

Research in cache design will only find use if it produces a significant performance improvement. Our focus for the rest of this thesis is thus upon system performance rather than on simply minimizing the miss rate of the cache system.

The industry focuses in producing faster main memory using new architectures like the cached DRAM and multi-bank DRAM. We share the same opinion with industry and believe that this emphasis on main memory design and memory integration should continue.

Chapter 2

The Choice of Traces for Simulation

2.1 Introduction

The computer systems of today are more complex and rapidly changing than those of even a few years ago. The need for tools and techniques to help in understanding the behavior of these systems is growing. One of the most important techniques is modelling.

There are two main types of computer system modelling. The first is analytical modelling which is inexpensive and fast but may involve simplifications that can make the results unreliable. The second method is to simulate the system by using a program which behaves like the system. If properly done, a simulation gives accurate results, and the system parameters can be easily varied to test different designs.

A simulator needs an input which represents an actual workload. The output of a simulated system depends greatly on the input and a realistic workload should be given to drive the simulator [1][23].

In trace-driven simulation, traces of real program addresses are used as an input to the memory system simulator. Traces are sequences of addresses that the program accesses during its execution. Trace driven simulation is the most popular and reliable method for evaluating the performance of a memory system.

As the results of the simulation depend on the input traces, the selection of the input traces is very important. Input traces should represent real workloads running on real computers. In this Chapter, we discuss the different types of traces that exist and select suitable traces by comparing the results of different traces on our simulator to typical levels of performance for a wide class of workloads and computers from the study done by A.J. Smith in [44].

In Section 2, we explain the characteristics of the available traces. Section 3 discusses a trace qualification method. In Section 4, we apply the trace qualification method to test and select traces.

2.2 The Traces

2.2.1 The SPEC benchmarks

The System Performance Evaluation Cooperative (SPEC) benchmark suite is a set of programs and data designed to provide realistic system performance results [16]. The metric of performance on each program is the ratio of program run time on the target system to the run time of the same program on a DEC VAX 11/780. The geometric mean of the set ratios gives the SPEC figure of merit which is most commonly quoted in advertisements, although all figures for all programs are really needed for a complete picture.

Interestingly, it has been reported that the SPEC benchmarks results are sensitive to cache size and generally yield miss rates lower than they should be [44]. In particular, the integer SPEC benchmarks put very little stress on the memory system.

The SPEC benchmarks have become so popular that computer designers are tuning their design to optimize SPEC benchmark performance regardless of the results of their applications [16].

A study by M.D. Hill [16] concludes that SPEC benchmarks are lacking a significant operating system component, which affects the results of cache simulations in two ways: miss ratios are very low , and the performance of the operating system functions are not tested.

For our study the traces that we use should put greater stress on the memory system and give more realistic miss ratios.

2.2.2 The ATUM traces

Address Traces Using Microcode (ATUM) is one of the most realistic sets of traces [1]. Traces of all addresses (including system addresses) that the processor visits are recorded and stored using the microcode of the machine. The ATUM traces include both the addresses and information about the type of each reference (read, write or instruction fetch).

The ATUM traces have the following advantages with respect to other traces like those from the SPEC benchmarks:

- ATUM traces include operating system code.
- No omission of multitasking; the traces are real traces from a live machine.
- No granularity distortion or implementation distortion.

The only problem the ATUM traces have are that they do not include input/output activities and they have limited trace length.

In conclusion, the ATUM traces provide realistic address traces that include operating system and multitasking effects and are more suitable for memory simulations than the SPEC traces. We use the ATUM traces available from Stanford University to drive our simulations [18]. The following is the list of available ATUM traces from Stanford University for workloads running on VAXes. We use uniprogram-

Trace name	Total Refs	Inst.	Data
dec0	361,982	183,023	178,959
fora	65,291	40,527	24,764
lisp	291,390	169,786	121,604
pasc	422,090	193,025	229,065
ue02	357,810	199,973	157,837
spice	446,701	223,706	222,995
mul2	372,104	208,434	163,670
mul8	429,432	199,455	229,977

Table 2.1: ATUM traces for uniprogramming and multiprogramming

ming (processes are executed in serial fashion) and multiprogramming (processes are executed in a time-multiplexed fashion) ATUM traces.

2.2.3 Hennessy and Patterson Benchmark traces

We also used the traces that are available from the text book "Computer Architecture: A Quantitative Approach" by J. Hennessy and D.A. Patterson. The benchmarks traces for spice, cc1 and Tex are longer than the other traces and are obtained from Stanford University [18]. The traces are of 1,000,000 references long and are Dinero-formatted [23] and [16]. The greater length makes them arguably good candidates for memory system testing.

Trace name	Total Refs.	Inst.	Data
spice'	1000,0001	782,764	217,237
ccl	1000,002	757,341	242,661
Tex	832,477	597,309	235,168

Table 2.2: Benchmarks traces

The following is the list of the obtained benchmarks and some of their characteristics:

2.3 Trace Qualification

2.3.1 Introduction

The characteristics of the used traces affect the results of the simulated model. We want to select traces that represent normal workload conditions.

Smith in [44] studied the effect of workloads on cache miss rates and defined DTMR (Design Target Miss Ratios) for normal loading.

In this section, we will introduce our trace qualification method that uses the values of Smith's DTMR to set a cut off point for the miss rates produced by the tested traces.

2.3.2 The Method

DTMRs are typical levels of cache performance for a wide range of computers [44]. They represent realistic miss ratios on cache systems in typical workstations running realistic workloads. Smith [44] has used hardware monitors, personal experience and trace driven simulation to obtain these miss ratios. The DTMRs for different cache sizes, set associativity and block sizes are presented in [22] and [24].

The miss ratios of trace driven simulations are often optimistic and we need to test our memory system under realistic conditions. DTMRs will be used as guidelines to qualify the traces used in our simulations.

As a lower limit on acceptable miss rates, 50% of the DTMR for the instruction and data caches is chosen. Thus the selected traces must give miss ratios higher than 50% of the DTMR to be used in our memory system simulation. This is reasonable considering the results of instruction and data caches miss ratios may vary widely. For example, the DTMR is 3 times larger than the SPEC miss ratios for instruction caches as measured by [16]. Note that we have set no upper bound for miss ratios on selected traces. This encourages the selection of traces with higher miss ratios which test the memory system under realistic, non-optimal conditions.

Next we will compare cache miss ratios using the different traces with the DTMRs for the same cache parameters, and will select the traces that satisfy the above criterion.

2.4 The Selected Traces

2.4.1 Selecting traces based on miss rates with different cache sizes

This trace evaluating simulation is done assuming a block size of 16 Bytes and a set associativity of two using the ATUM uniprogramming and multiprogramming traces and the Hennessy and Patterson benchmark traces.

The miss ratios for different cache sizes are compared with the DTMR expected values and traces giving miss rates of more than 50% DTMR are selected.

The results for this simulation are shown in figure 2.1 to 2.4.

Lisp, Spice, mul2 and Tex fail the 50% DTMR test.

2.4.2 Selecting traces based on miss rates with different cache block sizes

Simulation is now done assuming a cache size of 32 Kbytes and a set associativity of two for the ATUM traces. The miss ratios for different cache block sizes is compared with the DTMR expected values and traces giving miss rates of less than 50% DTMR are rejected.

Results show that fora, dec0, ue02, mul8 and ccl still qualify.

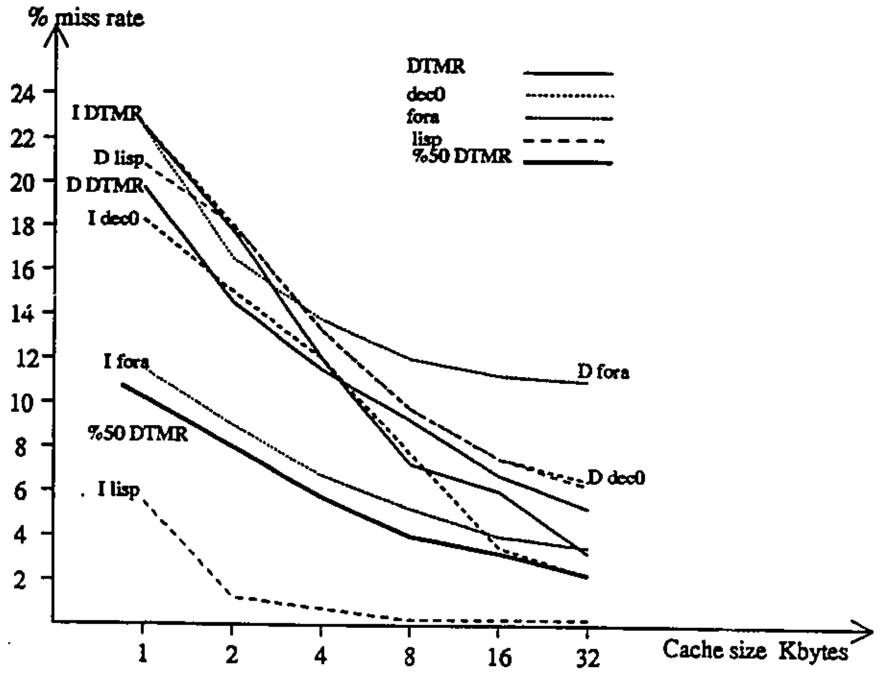


Figure 2.1: Miss ratios versus cache size for dec0, fora and lisp traces

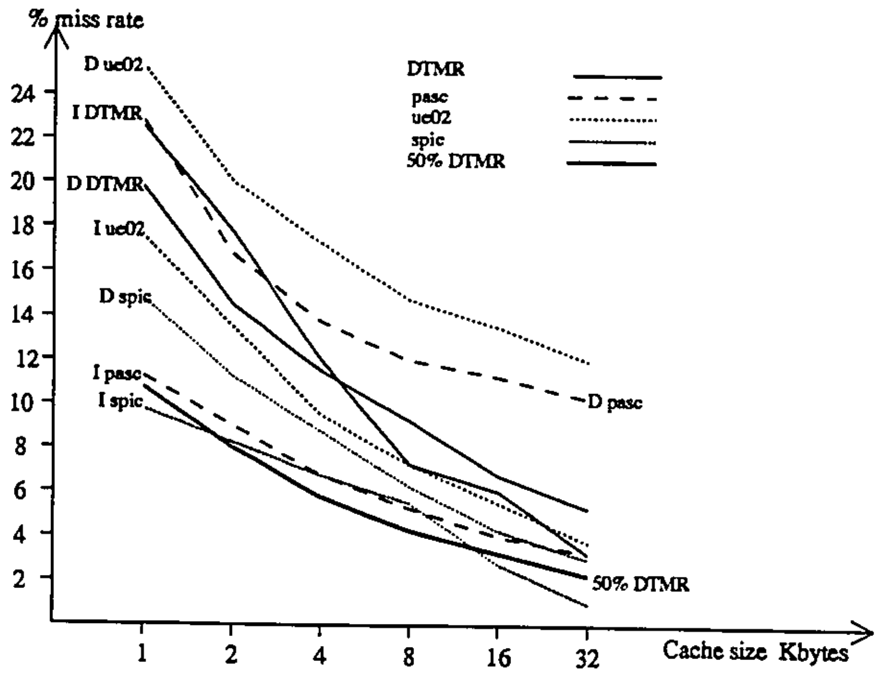


Figure 2.2: Miss ratios versus cache size for pasc, ue02 and spice traces

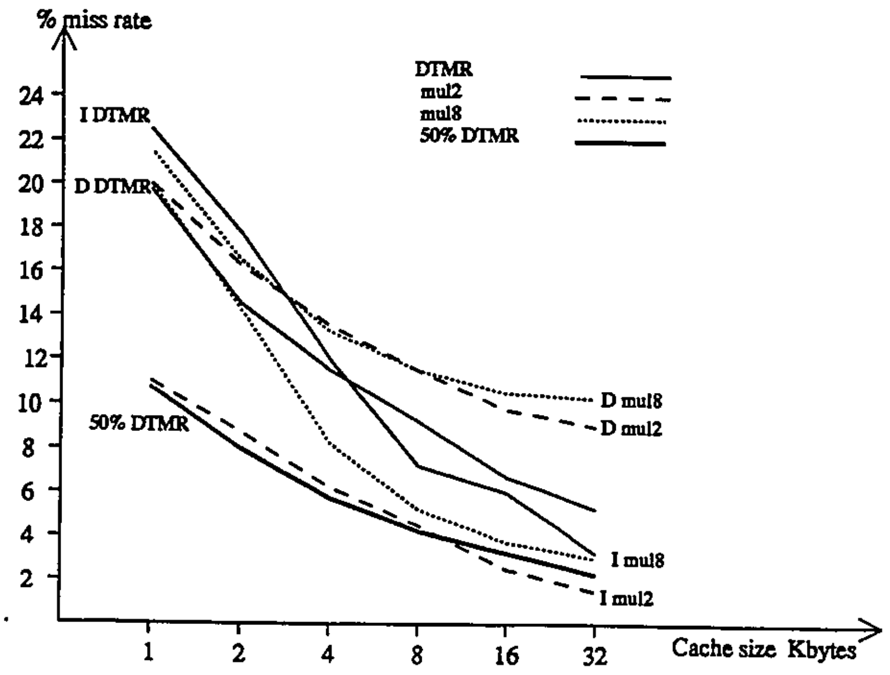


Figure 2.3: Miss ratios versus cache size for mul2 and mul8 traces

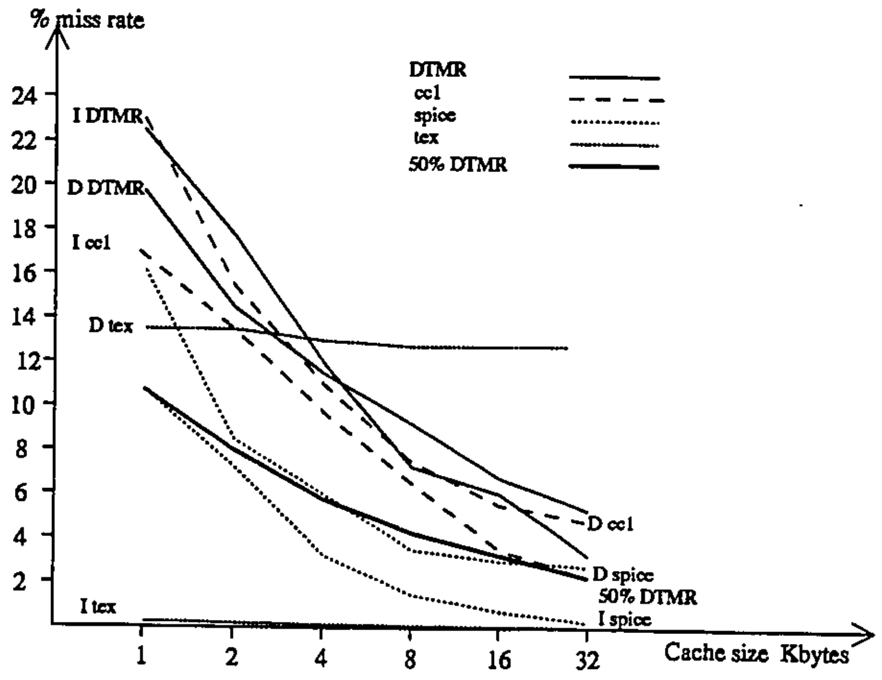


Figure 2.4: Miss ratios versus cache size for cc1, spice and tex traces

2.4.3 Selecting traces based on miss rates with different cache set associativity

Simulation is now done assuming a cache size of 32 Kbytes and a block size of 16 Bytes for the ATUM traces. The miss ratios for different cache set associativity is compared with the DTMR expected values and traces giving miss rates of less than 50% DTMR are rejected.

Results show that fora, dec0, ue02, mul8 and cc1 still qualify.

2.4.4 List of the selected traces

The following are the selected traces which we will use in our simulation in this work.

- Uniprogramming ATUM traces are:
  - fora- 65,291 total number of references, 40,527 instructions and 24,764 data references.
  - dec0- 361,982 total number of references, 183,023 instructions and 178,959 data references.
  - ue02- 357,810 total number of references, 199,973 instructions and 157,837 data references.
- multiprogramming ATUM traces are:

  - mul8- 429,432 total number of references, 199,455 instructions and 229,977 data references.

- Hennessy and Patterson Benchmarks are:
  - cc1- 1000,002 total number of references, 757,341 instructions and 242,661 data references.

All the above selected traces give similar or worse results than DTMR expected values for different cache parameters for both instruction and data caches.

2.5 Summary

We compare the performance of a cache model using the ATUM and Hennessy and Patterson Benchmarks traces, available from Stanford University, to the results of the DTMR [44] (using same cache parameters) and selected the traces that give similar or higher miss rates. We chose 50% of the DTMR as a lower limit for the miss rates of selected traces for any chosen cache parameters.

The traces are qualified against the DTMR for different cache sizes, block sizes and set associativity. The traces dec0, fora, Mul8 and cc1 will be used in this work as they represent uniprogramming, multiprogramming and the Hennessy and Patterson benchmarks, and also pass our DTMR test.

Chapter 3

Optimizing Main Memory Performance

3.1 Introduction

The main memory is typically the second level in the memory hierarchy. The main memory supplies caches, vector units and I/O units with required data. The main memory is generally the most important level in the hierarchy, from the point of view that it is the largest level whose speed is at all comparable to that of the CPU (within 2 orders of magnitude). Even the fastest disk devices are incredibly slow in comparison to the CPU ($>$ 5 orders of magnitude slower).

The main memory is usually built using DRAM (Dynamic Random Access

Memory) which has large capacity and low cost. From an economic point of view it is almost necessary to use DRAM chips in building a typical multiple-mega byte main memory.

It is well known that the need for using larger memory is growing rapidly because of larger software applications, leading to a continuing growth of processor address space. The DRAM industry has partly kept up with this need with DRAM capacity growing by a factor of 4 every 3 years [13].

The only problem with the DRAM is speed. The DRAM is organized as an array of capacitive memory cells, and does not match the speed of the logic circuits that are used in building SRAMs. Also, to reduce the cost of DRAM chips, the pin count is reduced by using a multiplexed address for the columns and rows of the array. The chip has to latch the first half of the address then latch the second half, making access slower.

Thus DRAM speeds are not keeping pace with increasing CPU speeds. There is no main memory available that can allow the newer processors (like the DEC Alpha [47]) to run at full speed.

The processor/main memory speed gap could be narrowed by using a cache. Any cache system has cache misses, in which the processor has to retrieve the data from the main memory at slower speed. Even for the best cache design, cache misses still exist and the performance of the system is limited by the speed of DRAM accesses.

This is a classic application of Amdahl's law [19].

The attention that is being given to improving the main memory speed is very little compared to that given to cache design. Improving the main memory performance is essential for the best overall system performance, and can have a dramatic effect, as we will demonstrate.

The following are the known conventional methods used in improving main memory performance:

- Wider memory- In wider word fetch, multiple words of data can be accessed simultaneously, reducing the cache line fill latency.

  The performance increase obtained by using a wider memory organization is limited by the cache line size, the memory access time and the delay of the multiplexer used between the cache and the processor.

- Interleaving- Interleaved memory uses several memory banks that are arranged so that they can be accessed in a parallel or an overlapped fashion [19], [38] and [28]. The processor can send addresses to multiple banks so that data can be retrieved in parallel.

In this research we consider the interleaved memory organization, since it has many advantages over the wider memory organization and can be easily expanded.

We will first look at how addresses are mapped into different banks and introduce variable address interleaving, a novel interleaving method. In Section 3, we review DRAM access modes like fast page/static column and the precharge mode. A novel method that combines both bank interleaving and these different DRAM access modes will be discussed. We then present four new memory protocols based on the use of the different address modes and bank interleaving. The rest of the Chapter focuses on evaluating the performance of the main memory using the four proposed protocols as a function of the bank mapping, DRAM size and the number of used banks. The characteristics and features of each protocol are examined using the traces of Chapter 2.

3.2 DRAM Interleaving with different bank mapping

3.2.1 Introduction

Memory interleaving is often used in high performance computers like the Cray vector processors [11].

The mapping of addresses to memory banks can affect the performance of the memory system. Consecutive accesses should be to different banks to achieve optimal performance (maximum bandwidth utilization) in simple word interleaving. If one bank has two successive requests, the second request must wait for the first one to complete and the system must generate a wait state [28].

The following are two conventional bank mapping methods:

3.2.2 Low order Address Interleaving

In low order address interleaving, the low order address lines are used to select the different banks. Accesses to consecutive addresses will go to different banks. This is the same as the word interleaving. If the number of banks = N_b, then $log_2 N_b$ of the lower address lines are used to select one of the banks.

3.2.3 High order Address Interleaving

In high order address interleaving, the high order address lines are used to select the different banks. Each bank contains consecutive memory locations of total memory size divided by the total number of banks.

We now introduce a novel bank mapping method:

3.2.4 Variable Address Interleaving

The performance of the interleaved memory system depends on the bank mapping. Low order address interleaving has an advantage for only consecutive accesses. A smart memory design should optimize the performance of both consecutive and non-consecutive accesses.

In this research, we introduce variable address interleaving in which the location of address bits for bank selection is made variable to cover all the memory address space. Variable address interleaving allows us to optimize the performance of the interleaved memory over all the address space.

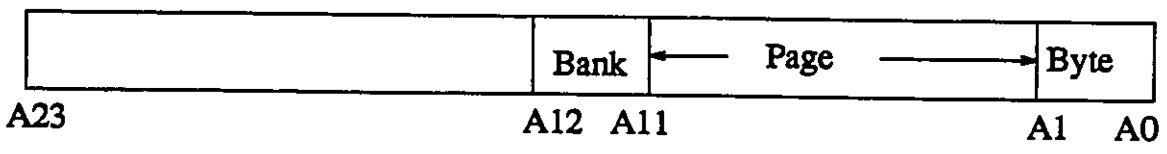
Figure 3.1 shows the variable address interleaving and the mapping of each memory address in different banks. In this example four banks are used and bank select is set at A12-A11. Each bank consists of multiple pages each of which has 0.5K words of consecutive memory locations.

Figure 3.2 shows the correct row address and column address for variable address interleaving assuming that the number of banks equals four. The bank select address takes any value from A3A2 to A23A22. For each location the row address (r) and column address (c) is shown as a function of the overall address space.

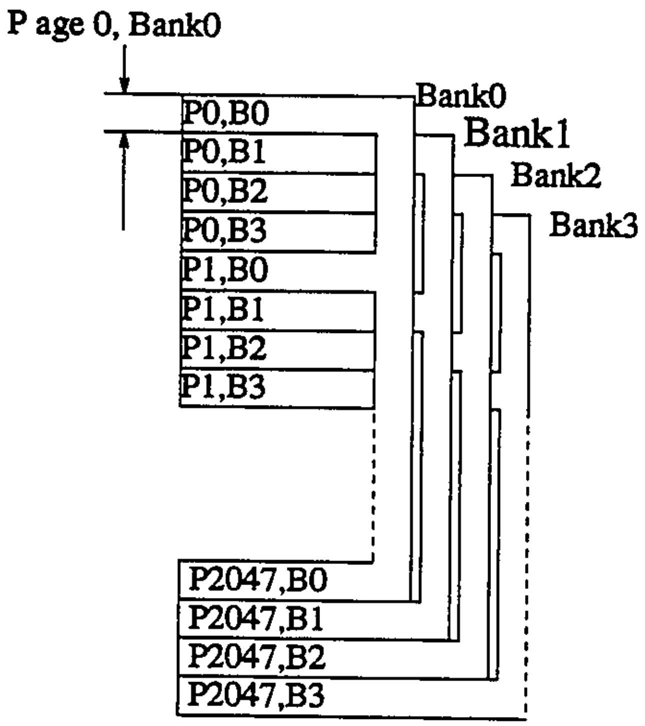
3.3 Improving Main Memory Performance by using different Access modes

There are three common DRAM access methods: random, fast page and RAS precharge. There are several timing parameters of interest, notably:

- t_{RP} – the RAS "row address strobe" precharge time.
- t_{RCD} – the RAS to CAS "column address strobe" delay.



Address space and bank select bits.



Mapping of memory locations in different banks.

Figure 3.1: Variable Address Interleaving

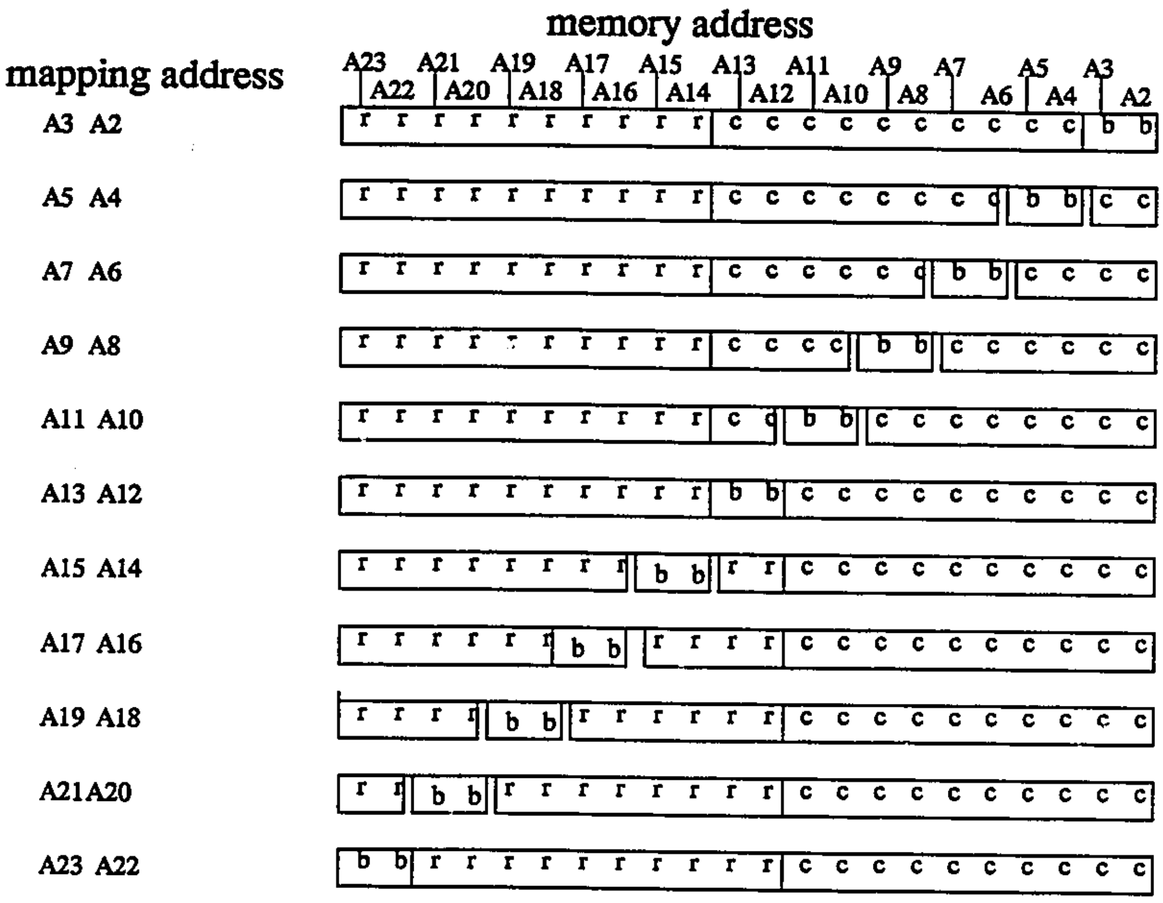


Figure 3.2: Row and Column address in Variable Address Interleaving

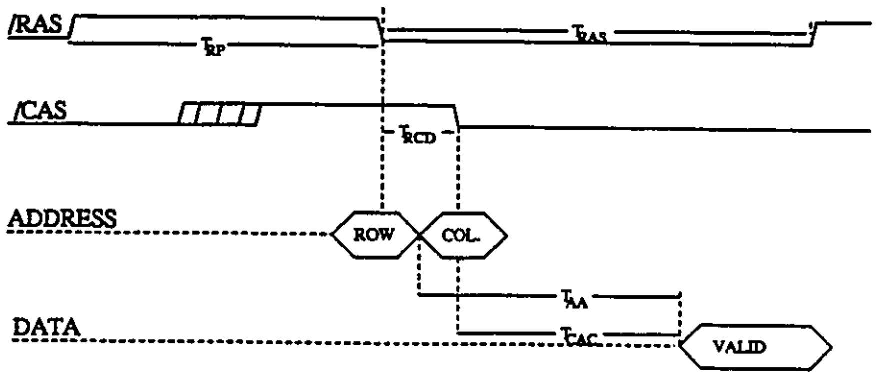


Figure 3.3: Random Access mode

- t_{CP} – the CAS precharge time.
- t_{AA} – the access time from column address.
- t_{CAC} – the access time from CAS.

The different access modes are:

3.3.1 Random access mode

Figure 3.3 shows the random access mode timing. All of the timing requirements (t_{RP}, t_{RCD}, t_{CP}, t_{AA} and t_{CAC}) must be satisfied for this access mode. [15].

3.3.2 Fast page access mode

Figure 3.4 shows the fast page mode access timing. In this mode, the bank RAS is kept active low to latch the previous row address and if the arriving access has the

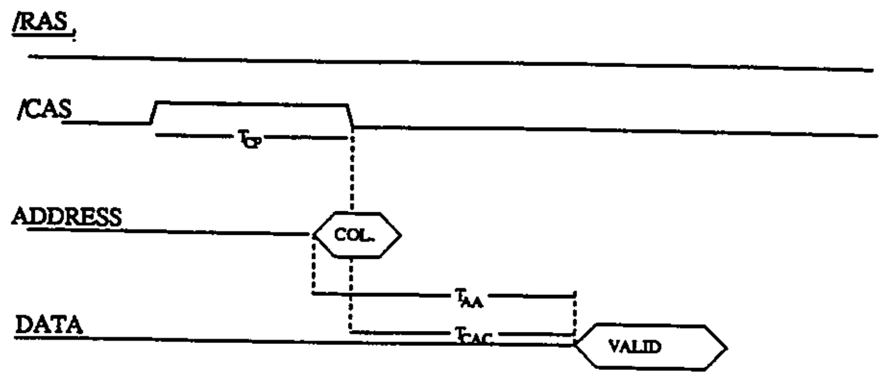


Figure 3.4: Fast Page Access mode

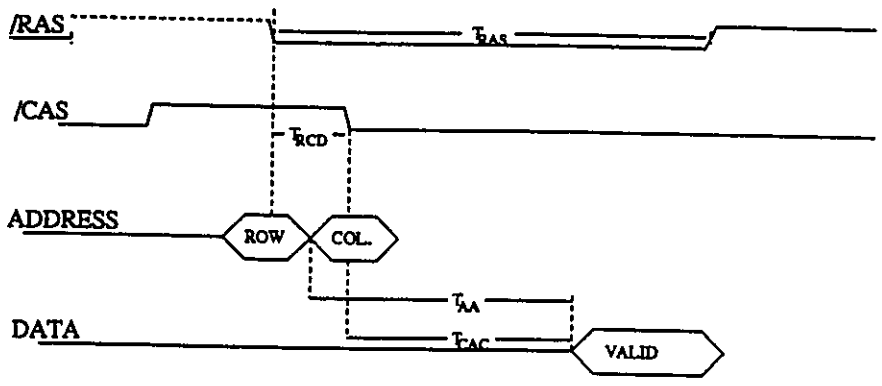


Figure 3.5: Precharge Access mode

same row address then there is no need to re-latch the row; the system need only latch the column address. Thus only t_{AA} and t_{CAC} constrain the access time [15]..

3.3.3 Precharge access mode

Figure 3.5 shows the precharge access mode. In this mode the RAS of the accessed bank is fully precharged and ready to latch the row address of the coming access. The row address must be latched via RAS, satisfying t_{RCD}, then the column address must be latched, satisfying t_{AA} and t_{CAC} [15].

3.4 Four New Protocols for DRAM interleaving

3.4.1 Introduction

In bank interleaving, successive accesses could be overlapped in time. While one bank is finishing the access, the other bank could start a new access.

Combining the different access modes and bank interleaving gives the memory system the advantages of having fast access modes and overlapping of the accesses.

Combinations of both techniques produce new and interesting memory systems. We present four different designs, as described by the following protocols:-

3.4.2 Notation: Design flow graphs

We introduce a design flow graph to visually illustrate the design of each memory system. It shows all decisions taken by the system in each operation. Each node of the graph corresponds to a decision element (like encountering a bank miss or a bank hit).

By assigning probabilities to each arc of the graph and assigning a cost to each operation, a simple and accurate analytical model of the system can be obtained as shown in our publication [33].

This is a new, powerful tool that that helps in making complicated systems simpler to follow. It also gives us more flexibility to consider different design options.

We will use this graph to illustrate the design of each memory system. Interestingly, recently a similar method has been used to illustrate the algorithm of a system [4].

The design flow graph is used to define the DRAM systems described in the following sections:

3.4.3 The precharge protocol

Figure 3.6 shows the flow chart of the conventional precharge protocol. In this protocol the RAS signals of different banks are kept precharged high while an access to another bank is taking place. If the next access is to one of the precharged banks, the system will do the precharge access sequence described above. If the next access is to the same bank, the system has to perform the random access sequence.

3.4.4 The fast page protocol

In this conventional protocol the system uses a comparator and latch for each bank. This requires a large and impractical amount of hardware for systems with large numbers of banks.

Figure 3.7 shows the flow chart of the fast page protocol. In this protocol, the RAS signals of all banks are kept active low, latching different row addresses. When an access occurs, the RAS control unit selects the accessed bank, and if the RAS of the selected bank is latching a row address that is the same as the row address of

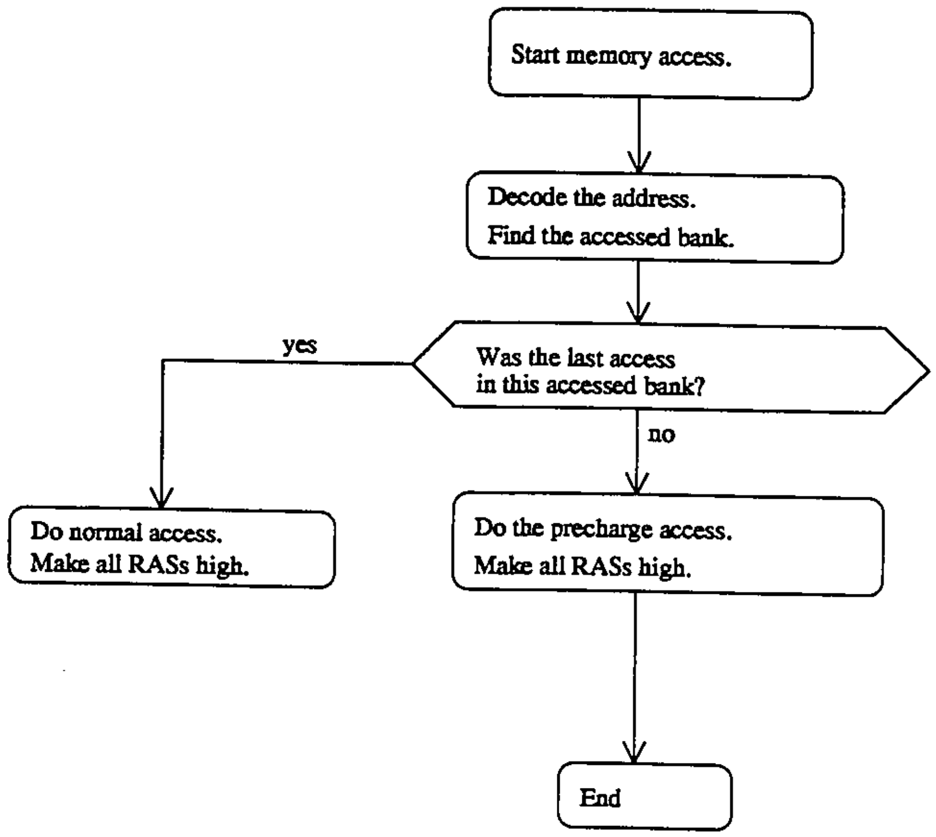
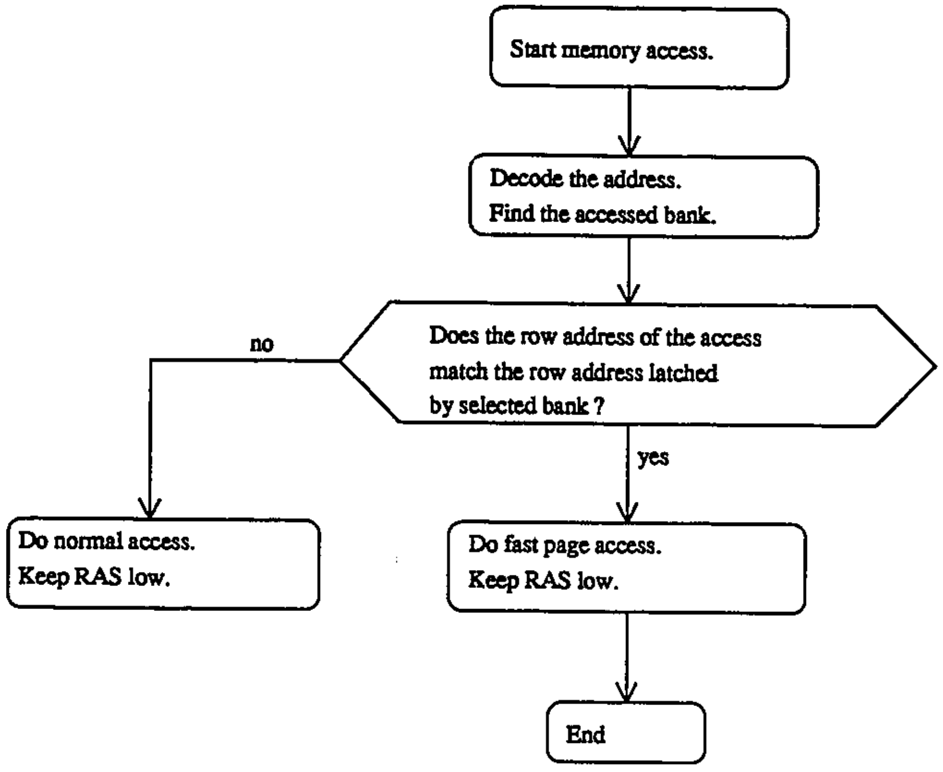


Figure 3.6: Precharge Protocol



Fast page protocol

Figure 3.7: Fast Page Protocol

the access, the system performs a fast page mode access. If the selected bank has a row address that is different from the row address of the incoming access, the system performs a random mode access. The bank will then use RAS to hold the new row address.

3.4.5 The fast page protocol with limited number of active banks

Figure 3.8 shows the flow chart of our novel limited fast page protocol. Some fraction of the the most recently used (MRU) RAS signals is kept active low (50%, in this case), to reduce the number of row comparators and row latches. An access to MRU bank that has same row address is a fast page access , while an access to a MRU bank with different row address is a random access and an access to a LRU (least recently used) bank is always a random access.

3.4.6 The fast page precharge protocol

Figure 3.9 shows the flow chart of our novel fast page precharge protocol. In this protocol, 50% (the MRU) of the RAS signals are kept active like the limited fast page protocol and the other 50% of the RAS signals are kept precharged. This protocol uses half the comparators and latches of the fast page protocol. However, new accesses to a LRU bank take only precharge access time, an access to a MRU bank that has the same row address takes a fast page access time, and an access to a MRU bank with a different row address takes a random access time.

It is thus possible for this protocol to have greater performance than both the limited fast page and the fast page protocols, as not all bank misses result in random (long) accesses.

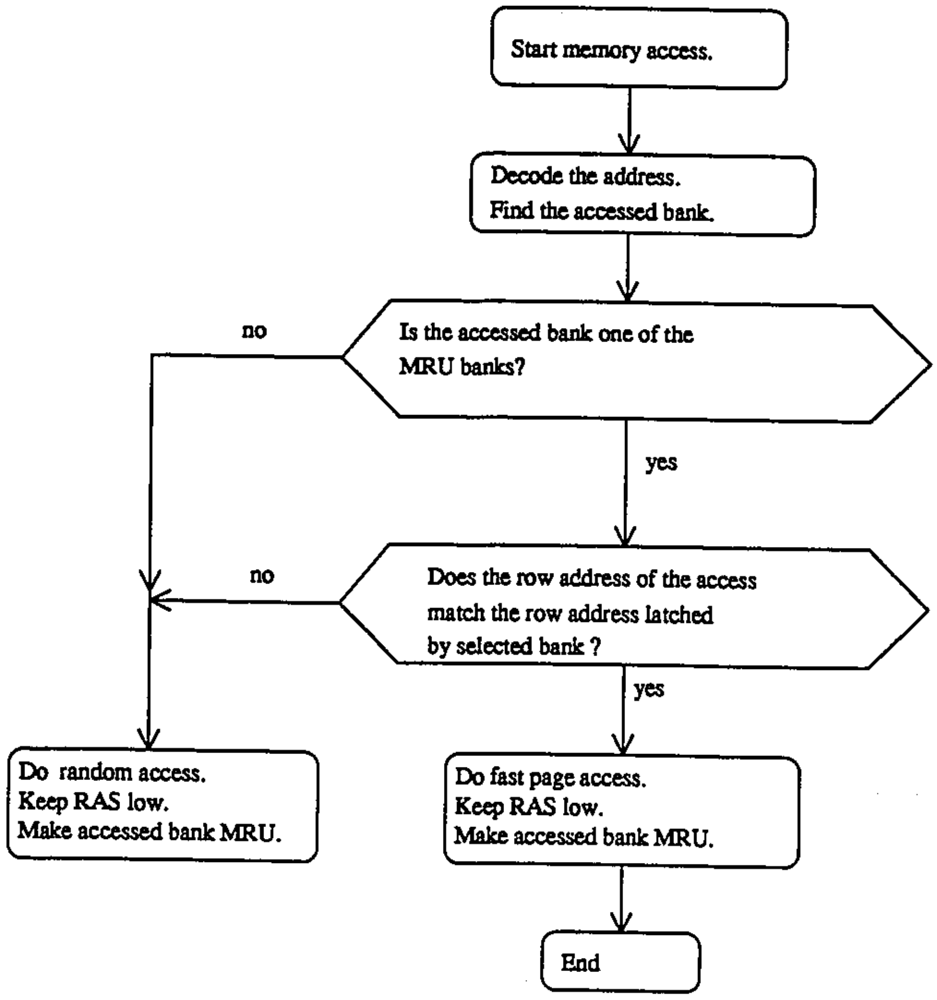


Figure 3.8: Limited Fast Page Protocol

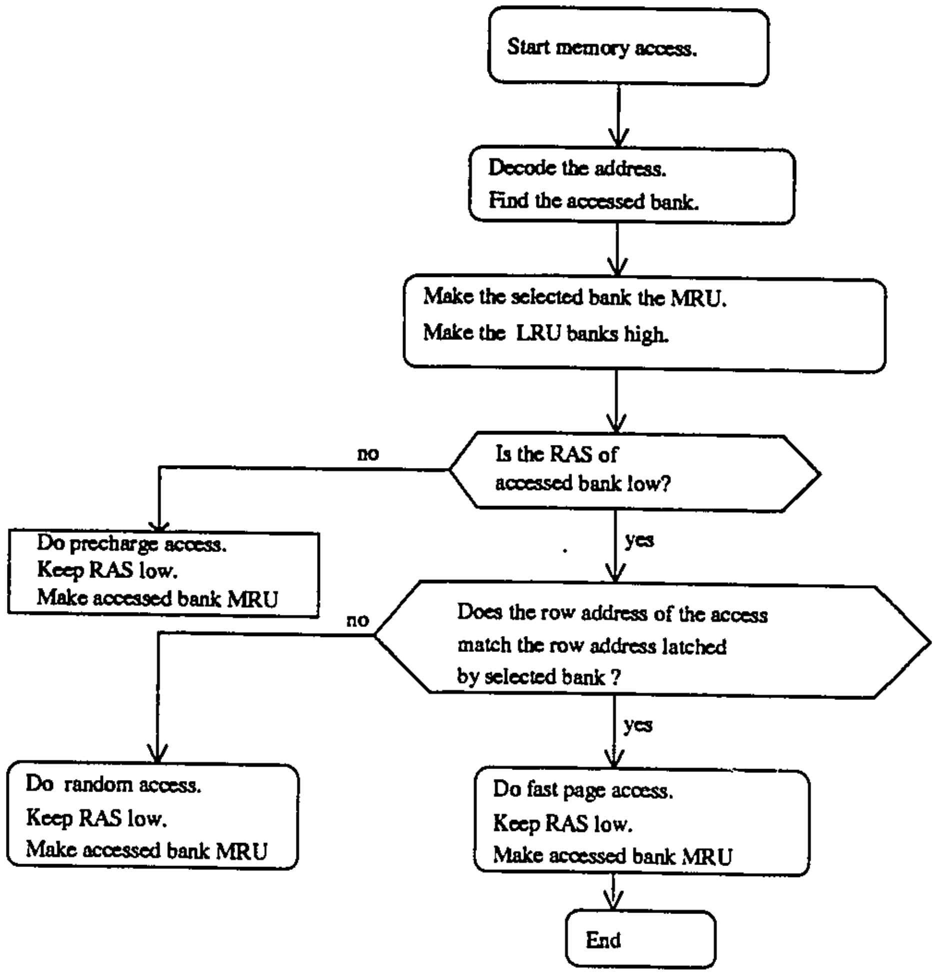


Figure 3.9: Fast Page precharge Protocol

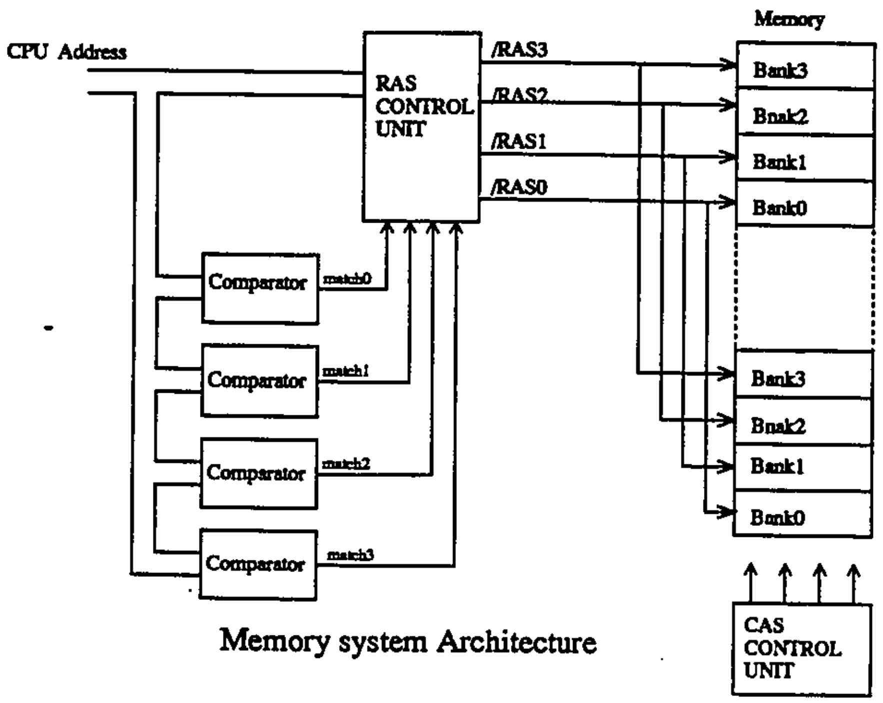


Figure 3.10: Memory System Architecture

3.5 The System Architecture

The system that we now simulate consists of a main memory using multiple bank interleaving. Figure 3.10 shows an example of a system that uses 4 banks. The system consists of the memory (divided into 4 separate banks,) the RAS control unit that generates the 4 RAS lines for the different banks, four row address comparator units and a CAS control unit.

3.5.1 The RAS control unit

The RAS control unit is used to decode the address bits and map addresses into the banks of the memory. We assume that this unit supports the above three different access modes. For normal accesses, the unit allows the RAS to precharge and to go active low to latch the new row address. For precharge accesses, the unit allows the RAS to stay high while the system is using other banks so that the precharge time is satisfied. The RAS unit tracks the value of each RAS so that it can decide to do a precharge access if it finds that the accessed RAS is precharged.

For the fast page mode the RAS is kept active low, latching the last row address in the DRAM. The unit examines the value of the 4 match signals, coming from the comparator units, to do the fast page access.

3.5.2 The row address comparator

The row address comparator is used to detect if an access has the same row as the one that is being latched in a specific memory bank. The comparator has a latch that keeps the last row address that the memory bank uses. For each access, the incoming row address is compared with the latched one and the comparator signals a match if they are equal.

3.5.3 The CAS control unit

The CAS control unit generates the required CAS for the memory. It allows for CAS precharge and the other CAS timing requirements based on the type of access being performed.

3.6 Simulation Model

Figure 3.11 shows the block diagram of the simulation model.

The simulator reads the trace input file to find the current address. The bank decode function gives the current bank number from the address. The address comparators compare the latched row number of the accessed bank with the current access row number and signal a match if they are the same.

The bank comparators distinguish between a RAS precharge or random access by examining the accessed bank's RAS value. If the RAS line is still active, then the row comparators determine if the current access is on the same page (fast page access) or not (random access).

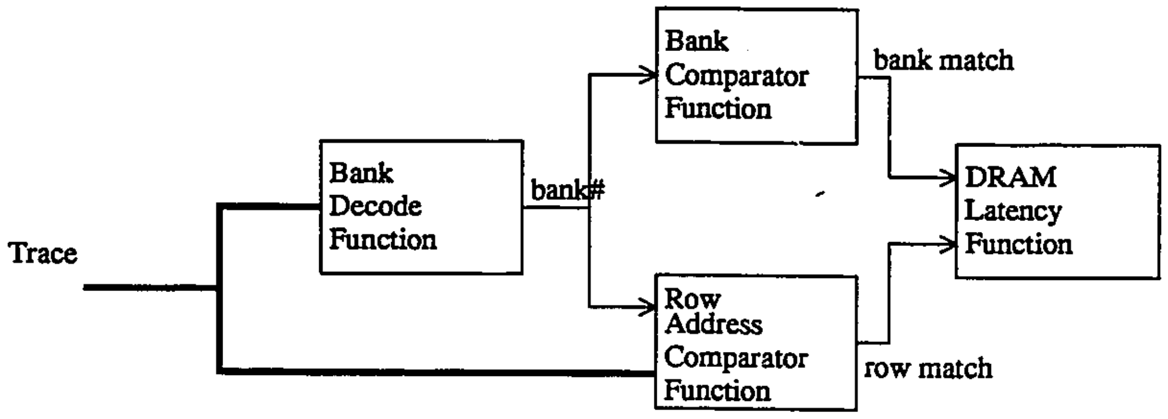


Figure 3.11: The Simulation Model

3.7 The results of Main memory using the four New Protocols

The following traces were used to simulate the DRAM performance: dec0, the ccl benchmark, fora and mul8. The processor is assumed to run at 100 MHz; this makes the clock cycle time equal to 10 ns.

The DRAM chips are chosen to be 4 M bits with 80 ns access times and support fast page mode access [15]. The total time for a random access to the DRAM is 155 ns. We assume that this takes a total of 16 cycles for random access considering that the cycle is 10 ns [15]. For fast page mode, the access takes 6 cycles and for the precharge access it takes 9 cycles according to the DRAM access timing. We use N banks with variable bank select address and variable DRAM size.

Performance improvement is measured by dividing the time taken when the system uses the random access mode by the time when the system uses the protocols under test.

3.7.1 Varying the bank select position

The location of address bits for bank select are varied to cover all the memory address space. Figures 3.12, 3.13, 3.14 and 3.15 show the performance speed up of each protocol versus the bank select address bits for the dec0, cc1, fora and uml8 traces.

Figure 3.16 shows the range of changes in speed up due to the different traces versus the bank mapping address.

From the results, we find the following:

- The performance of all protocols vary considerably from varying the address interleaving (up to 260%).
- The precharge protocol performs badly with high address interleaving and gives its best performance when the banks are interleaved around a word size.
- The fast page, fast page precharge and limited fast page protocols give the best performance when the banks are mapped at A12 (page interleaving). This is a 2 K word bank size, and exactly the size of a page for the 4 M DRAM.
- The new fast page precharge gives the best performance of all protocols. This is particularly interesting as it actually requires less hardware than the fast page

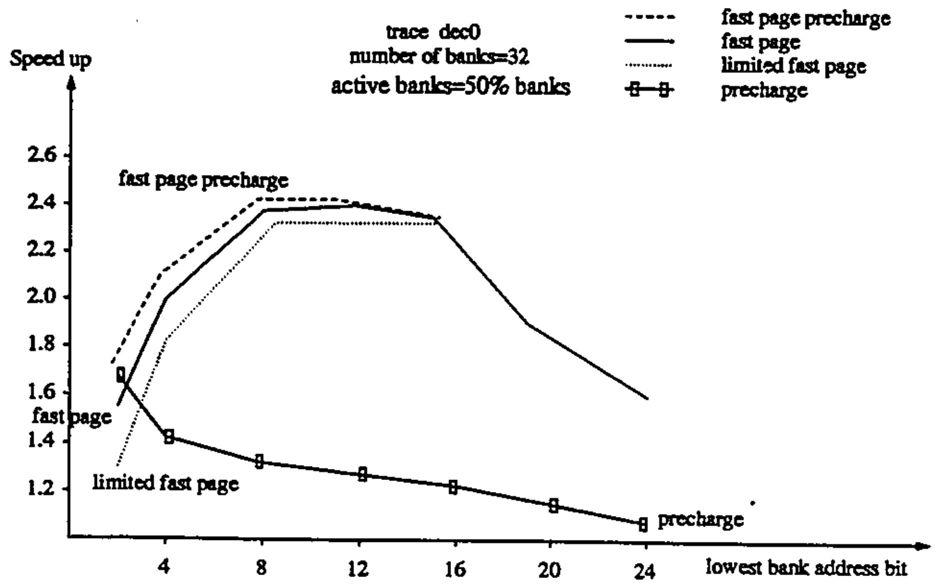


Figure 3.12: Variable address interleaving for dec0 trace (4M DRAM)

protocol.

- the results are insensitive to workload variations.

3.7.2 Varying the DRAM size

The results for the rest of this Chapter are done using all the above traces. We choose only to display the results of one trace (mul8 trace) as the results are insensitive to workload variations (at the point of optimal performance) , and this reduces the number of figures considerably.

The performance speed up from the new protocols also depends on the size of the DRAM chip. Figures 3.17 and 3.18 show the speed up for a low density DRAM

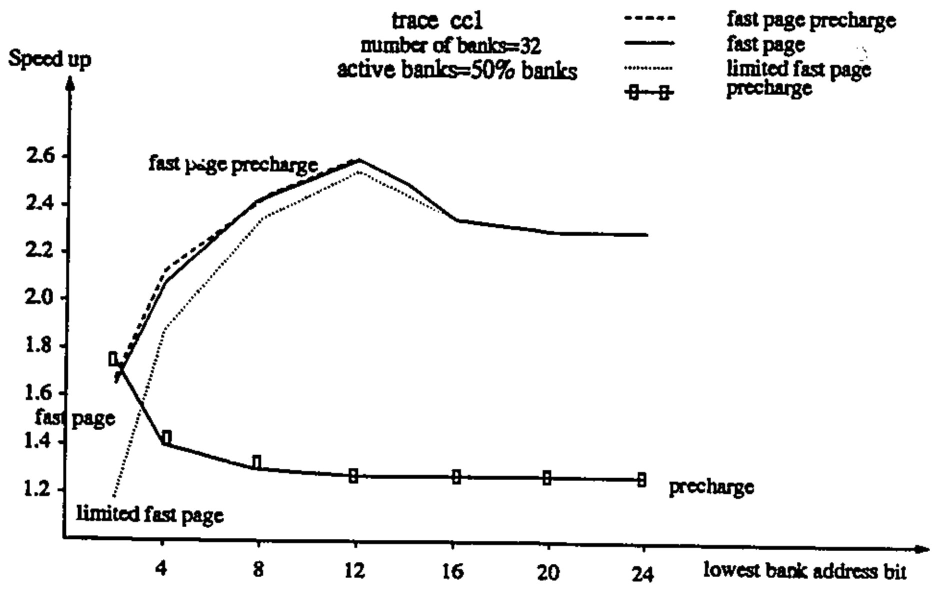


Figure 3.13: Variable address interleaving for cc1 trace (4M DRAM)

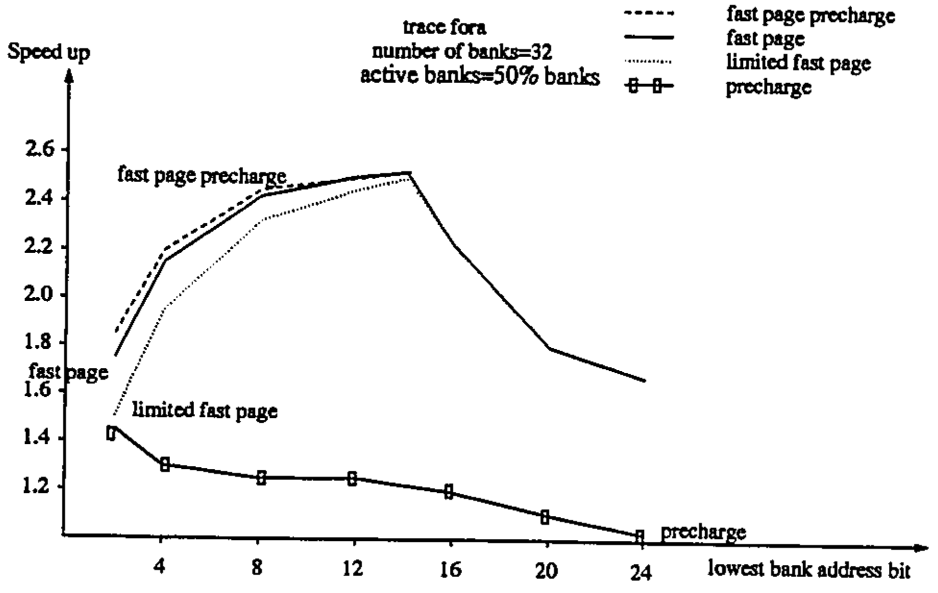


Figure 3.14: Variable address interleaving for fora trace (4M DRAM)

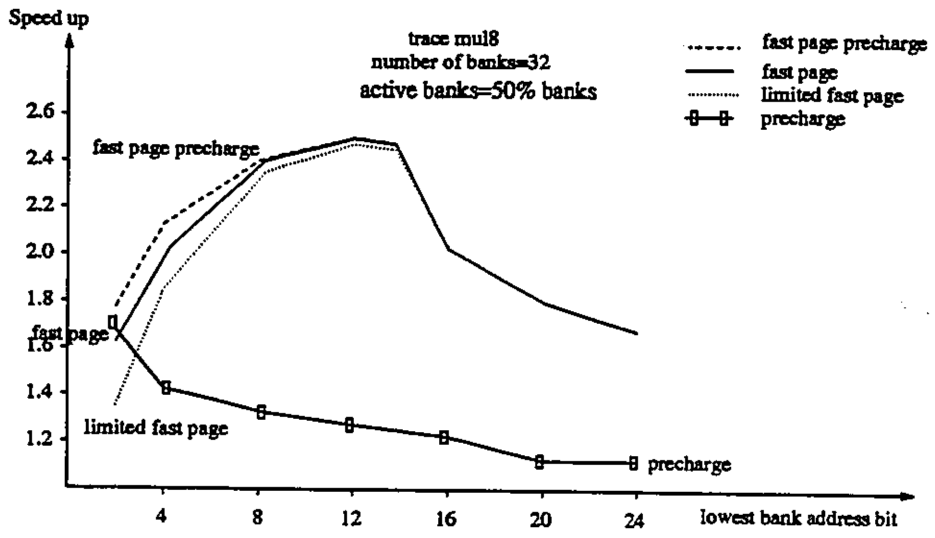


Figure 3.15: Variable address interleaving for mul8 trace (4M DRAM)

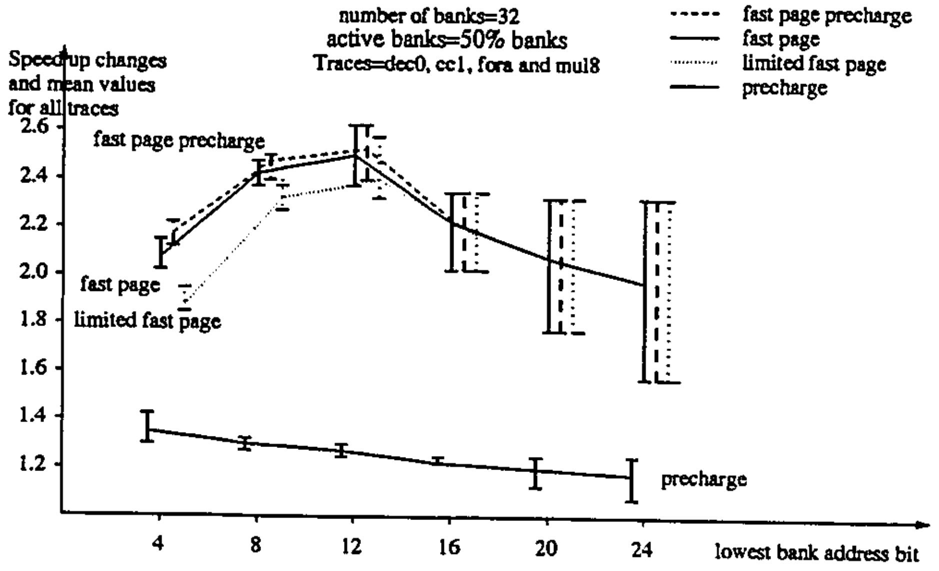


Figure 3.16: Speed up changes versus variable interleaving for 4M DRAM with different traces

that has a size of 256 K bit and the speed up of the 4M DRAM for the different protocols. From the figures, we conclude the following:-

- The performance of the precharge protocol does not depend on DRAM size.
- The optimum performance for the three other protocols occurs around A8, which is the size of a page for the 256 K bit DRAM.
- The performance of fast page, limited fast page and fast page precharge protocols is smaller than that of 4M DRAM, which indicates that larger DRAM size is an advantage when using those schemes (larger DRAM size means larger number of columns in one row and has an effect similar to using larger cache sizes).
- Future DRAM chips, with large sizes, should gain from using the four protocols and page interleaving.

3.7.3 Low Order Address Interleaving versus number of banks

Figure 3.19 shows the performance speed up of each protocol versus the number of banks for the low order address interleaving for the 4 M DRAM.

From the results, we find the following:

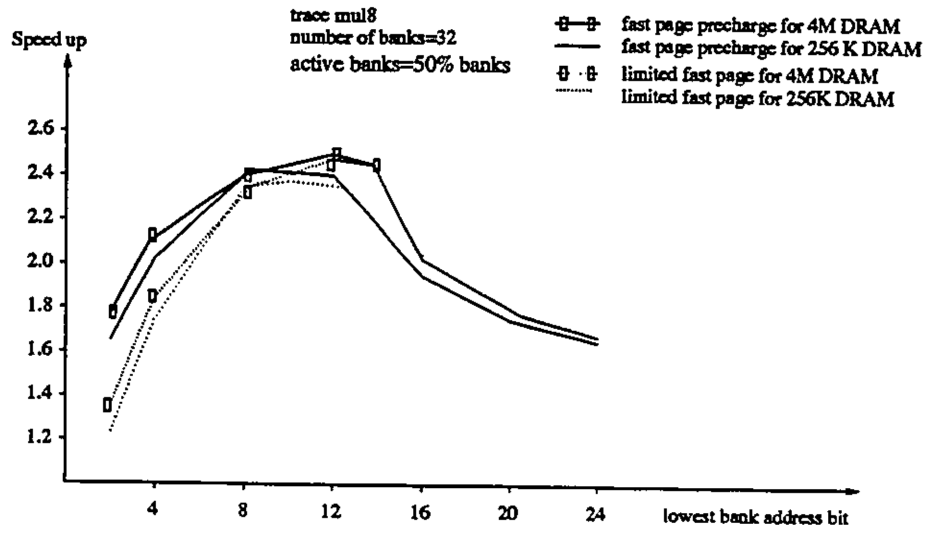


Figure 3.17: Variable address interleaving for limited fast page and fast page precharge (256K and 4M DRAMs)

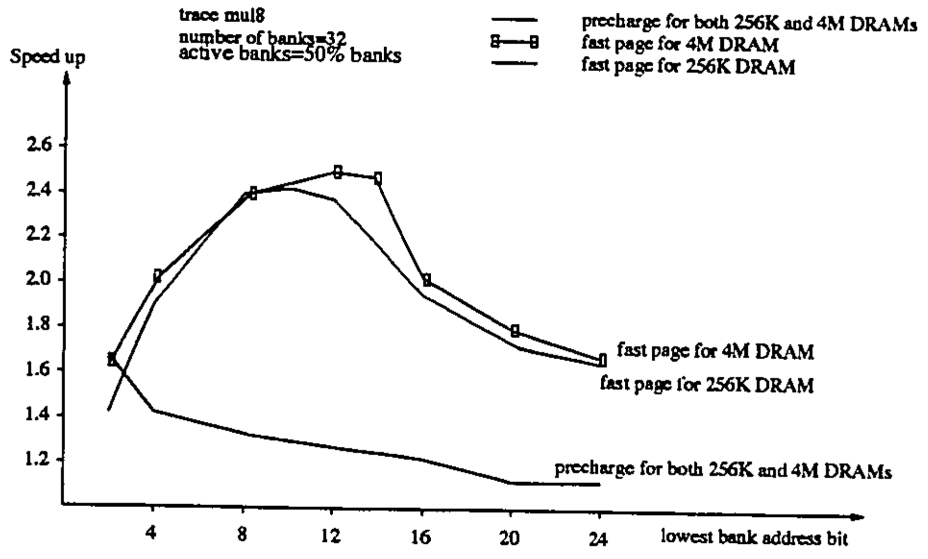


Figure 3.18: Variable address interleaving for precharge and fast page (256K and 4M DRAMs)

- the precharge protocol gives a very reasonable performance for a small number of banks. Increasing the number of banks for the precharge protocol helps the performance up to 8 banks and any more increase in number of banks does not produce an increase in performance.
- The fast page precharge gives the best performance of all protocols. The performance of the fast page precharge protocol increases with the increase in number of banks.
- The fast page protocol gives good performance with a very large number of banks.
- The limited fast page protocol gives only moderate performance and needs a large number of banks.

3.7.4 Performance of High Order Address Interleaving versus number of banks

Figure 3.20 shows the performance speed up of each protocol versus the number of banks for the high order address interleaving for the 4 M DRAM.

From the results, we find the following:

- The performance of fast page, fast page precharge and limited fast page are the same for bank numbers larger than 4.

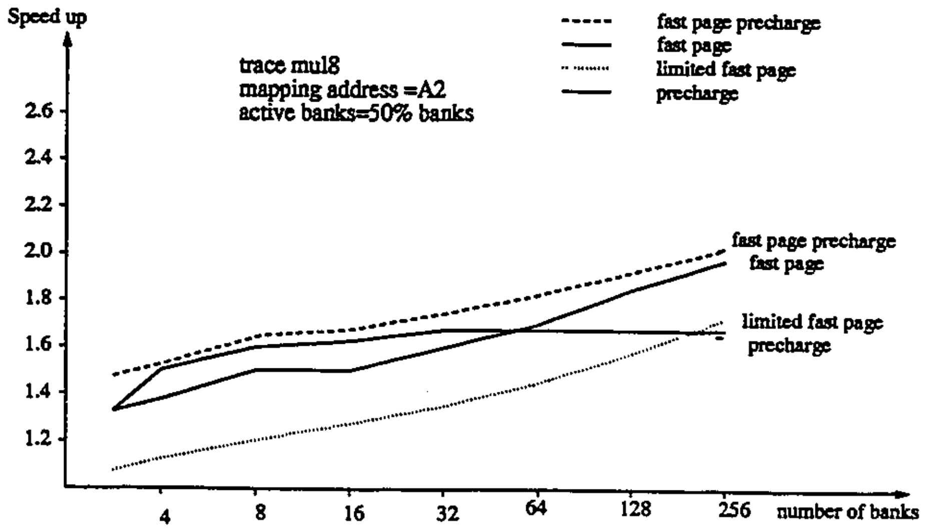


Figure 3.19: Low order address interleaving

- The precharge protocol gives the worst performance because the large bank size (most programs run in a single bank) forces most accesses to use the random access mode.

3.7.5 Page interleaving versus number of banks

Figures 3.21 shows the performance speed up of each protocol versus the number of banks for the page interleaving for the 4 M DRAM. The page interleaving is when banks are interleaved around a page size; in this example, it is address A12.

From the results, we find the following:

- The precharge protocol gives the lowest performance.

- For large number of banks, the fast page, the fast page precharge and the limited

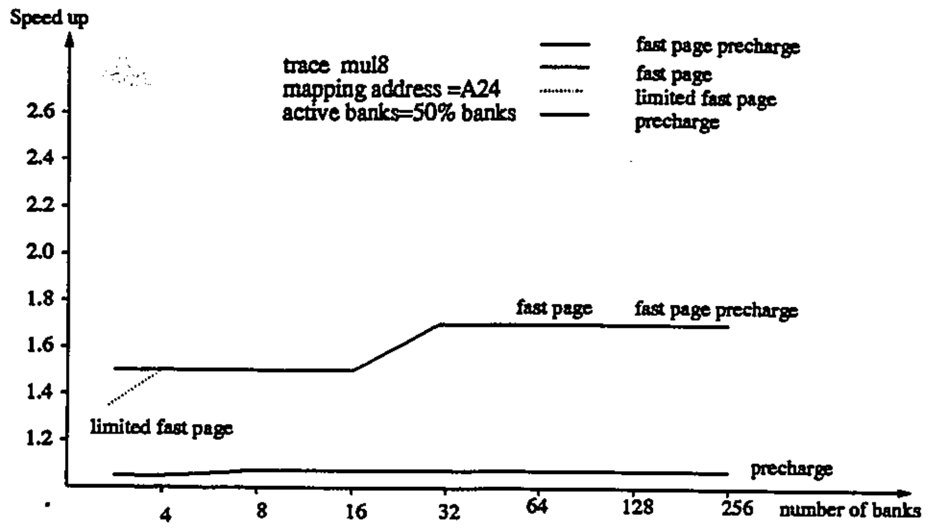


Figure 3.20: High order address interleaving

fast page protocols give similar performance.

- Increasing the number of banks improves performance. The increase in bank number gives diminishing returns if the number of banks is larger than 16.

- For a small number of banks, the fast page precharge and the fast page protocols give the best performance.

3.8 Summary

In this Chapter we have investigated the effect of multi-bank DRAM interleaving with different access modes on the performance of main memory system. We present four interleaving protocols based on the use of the different DRAM access modes.

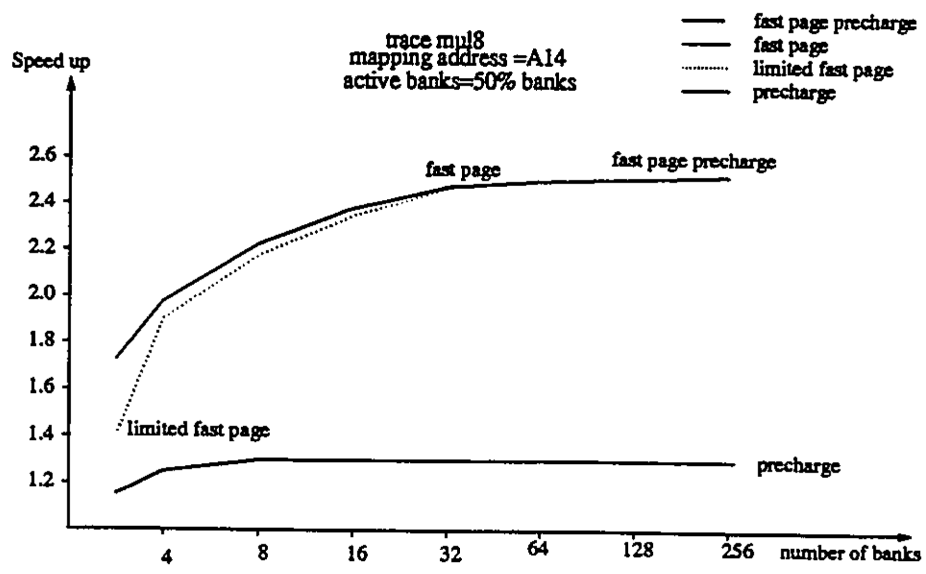


Figure 3.21: Page interleaving

The effect of bank mapping on system performance using the four protocols is studied. It is found that for interleaving based on the fast page access mode, best system performance occurs when banks are interleaved around a DRAM page size. With the RAS precharge access mode, best performance occurs when banks are interleaved around one word size.

A new fast page precharge protocol is introduced that offers the best performance with less banks, fewer row comparators and is suitable for future larger size DRAMs.

The performance of the fast page precharge is better than the performance of the fast page protocol but it uses less hardware (row comparators and latches).

Chapter 4

Single Level Cache Basics and Design

4.1 Review of Cache Design Basics

4.1.1 Introduction

It is generally impossible to find fast enough main memory to match the speed of newer processors. One way of dealing with this speed gap is to add wait states to each processor memory access to match the DRAM speed. This would represent a significant reduction in system performance.

The better solution to this processor/main memory speed mismatch is to use a cache system. Cache memory uses a small, fast SRAM to provide the processor with no wait state accesses. Information from the main memory (DRAM) is placed into the fast cache (SRAM) system and can be retrieved with no delay if it is needed again.

Cache operation is based on the concept of program locality. Programs tend to reference memory locations that are close to recently accessed locations (spatial locality), and tend to reference items that have been recently referenced (temporal locality).

In this Chapter, we review basic cache design and review the effect of cache parameters on system performance. In Section 2, we define cache terminology and discuss cache organization. Section 3 discusses the effect of cache parameters on miss rate. In Section 4, we review the different write strategies, present two designs (write through and write back), and examine the performance of each.

4.1.2 Cache Terminology and Organization

Terminology

The following are the definitions of the most commonly used cache terms:

- Cache tag (directory): A memory device that is used to keep the addresses of memory locations that exist in the cache.

- Cache hit/miss: The cache has a comparator that compares the tag addresses to the CPU addresses and gives a cache hit if a match occurs. If there is no match the comparator signals a cache miss.

- Hit rate: The number of cache hits compared to the total number of memory references made by the CPU.

- Blocks: The amount of data loaded from the main memory and stored in cache in a given access. Block size is always a multiple of the processor word size. For each block in cache, the tag keeps the base address for data in the block.

- Valid bit: For each block in the cache, the tag uses an extra bit to indicate whether or not the block has valid data (i.e., data in cache is the same as in main memory).

- Associativity: If a given block can be stored in only one place in the cache, the cache is known as direct mapped (associativity=1).

  If a block can be stored in any place in the cache, the cache is known as fully associative.

  If a block can be placed in a restricted set of places in the cache, the cache is known as set associative.

Cache Organization

Figure 4.1 shows the organization of a N way set associative cache. The cache system consists of the following parts:

- Cache memory: This example uses N cache memories. The total size of each cache memory= 2^m blocks (m points to the accessed block in the cache).
- Cache Blocks: The number of words in each block= 2^B words (B points to the selected word in each block).
- Tag memory: The size of the tag memory= $2^m \times t$ bits (t is used to determine if the CPU access exists in the cache).
- N sets: The N way set associativity is implemented by duplicating the cache memory and tag memory N times.
- multiplexers: each multiplexer is used to select one word from the selected cache block.

Note that for simplicity we have omitted the following :

- Byte accesses.
- Other ways of mapping the address to cache lines (hash, hash-rehash, and skewed).

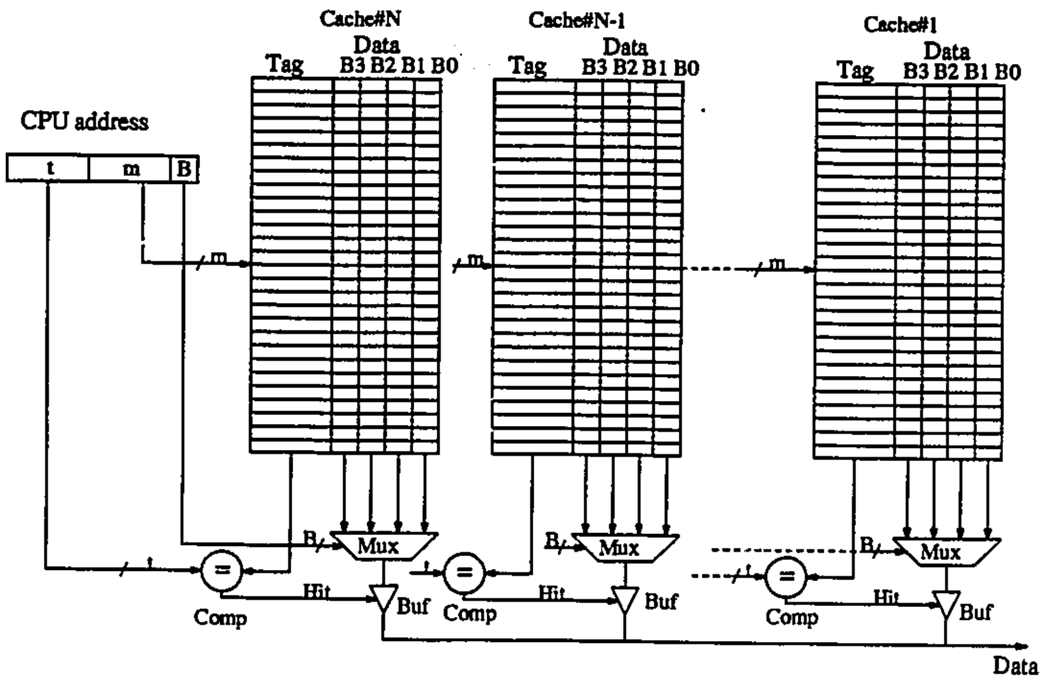


Figure 4.1: Cache organization

4.1.3 Cache Parameters and Miss rate

It is useful to briefly review the effect of cache design parameters on single-level cache miss rates. Since this is a set of well-known phenomenon, we will use the data from Hennessy and Patterson's book " Computer Architecture: A Quantitative Approach" page 421-424 [19]. The results are for a trace that has a total length of over a million addresses that was generated on a VAX-11 runniug Ultrix.

The effect of cache size and cache associativity on miss rate

Figure 4.2 shows the effect of cache size on miss rate of a cache system with variable set associativity (according to [19]). The results show the following:

- Increasing the cache size, improves the cache hit rate.
- Increasing the cache associativity improves the cache hit rate.
- Increasing cache size influences miss rate much more than increasing cache associativity.
- The improvement in hit rate shows diminishing returns with larger cache sizes or larger cache associativity.

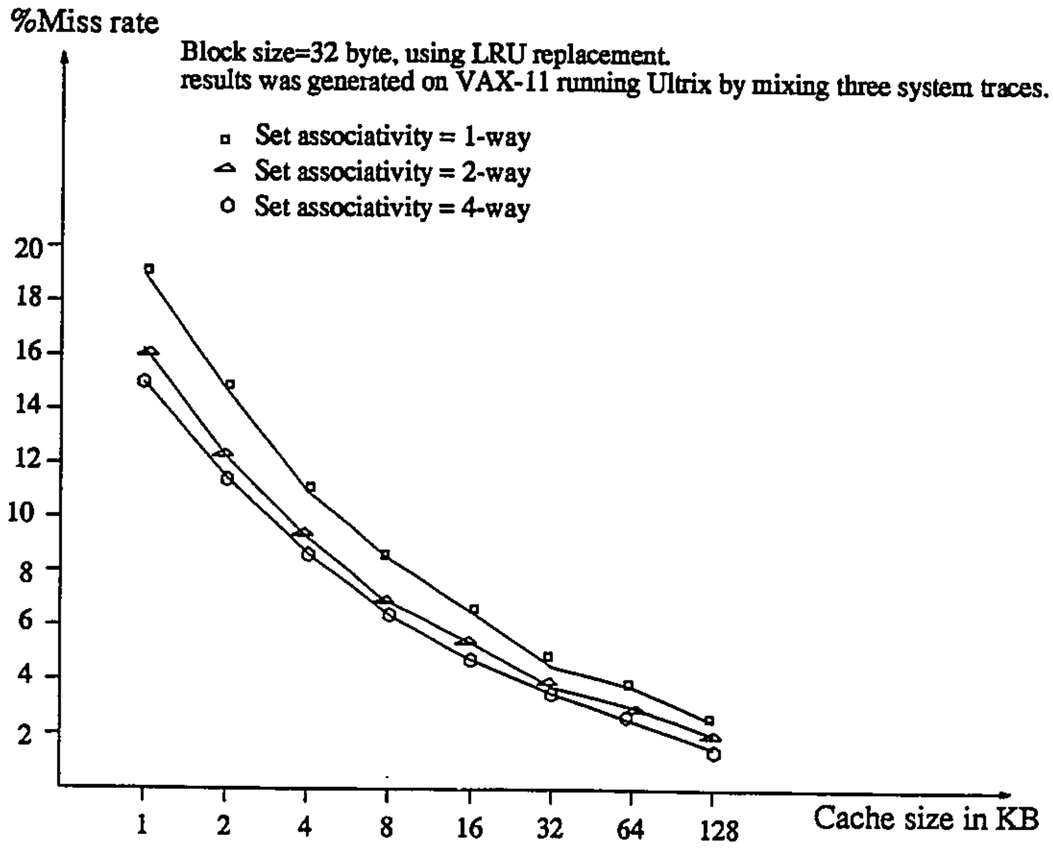


Figure 4.2: Cache size and associativity versus miss rate

The effect of cache block size on miss rate

Figure 4.3 shows the effect of cache block size on cache miss rate [19]. Figure 4.4 shows the value of the average access time versus cache block size [19].

The results show the following:

- Larger block sizes generally reduce miss rate.
- The optimal miss rate for 1 KB, 8KB and 16 KB cache sizes occurs at a block size of 64 bytes.
- The optimal average access time for these cache sizes occurs at a block size of 16 bytes.
- The difference between the block size values for optimal miss rate and optimal average access time indicates that the miss rate should not be used to evaluate the cache performance, since average access time is a better predictor of system performance.

4.1.4 Write Strategies

In any cache design, read operations are handled in the same way. On a read hit, the processor gets the data directly from the cache. On a read miss, the cache system gets the data from memory and updates the cache. It is the write strategy that distinguishes many cache designs.

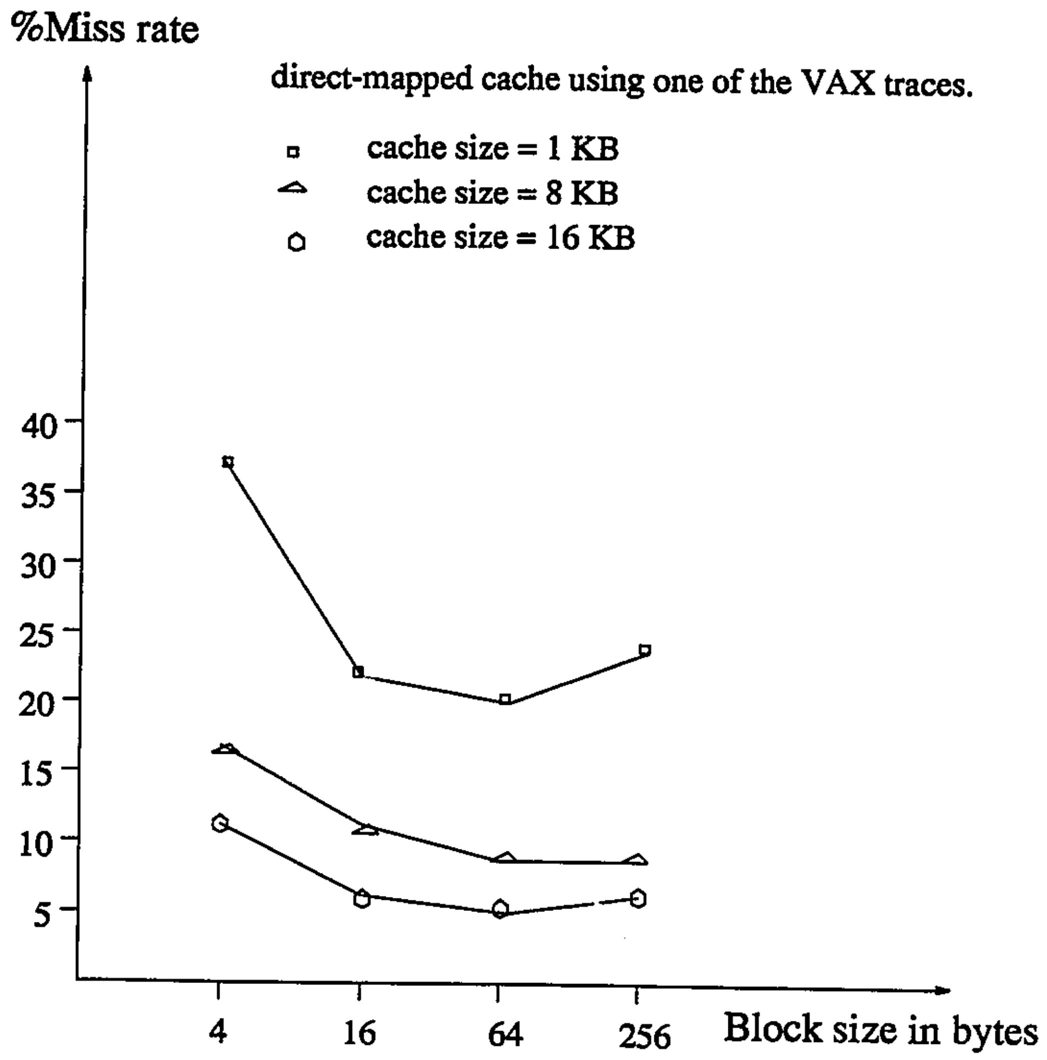


Figure 4.3: Cache block size versus miss rate

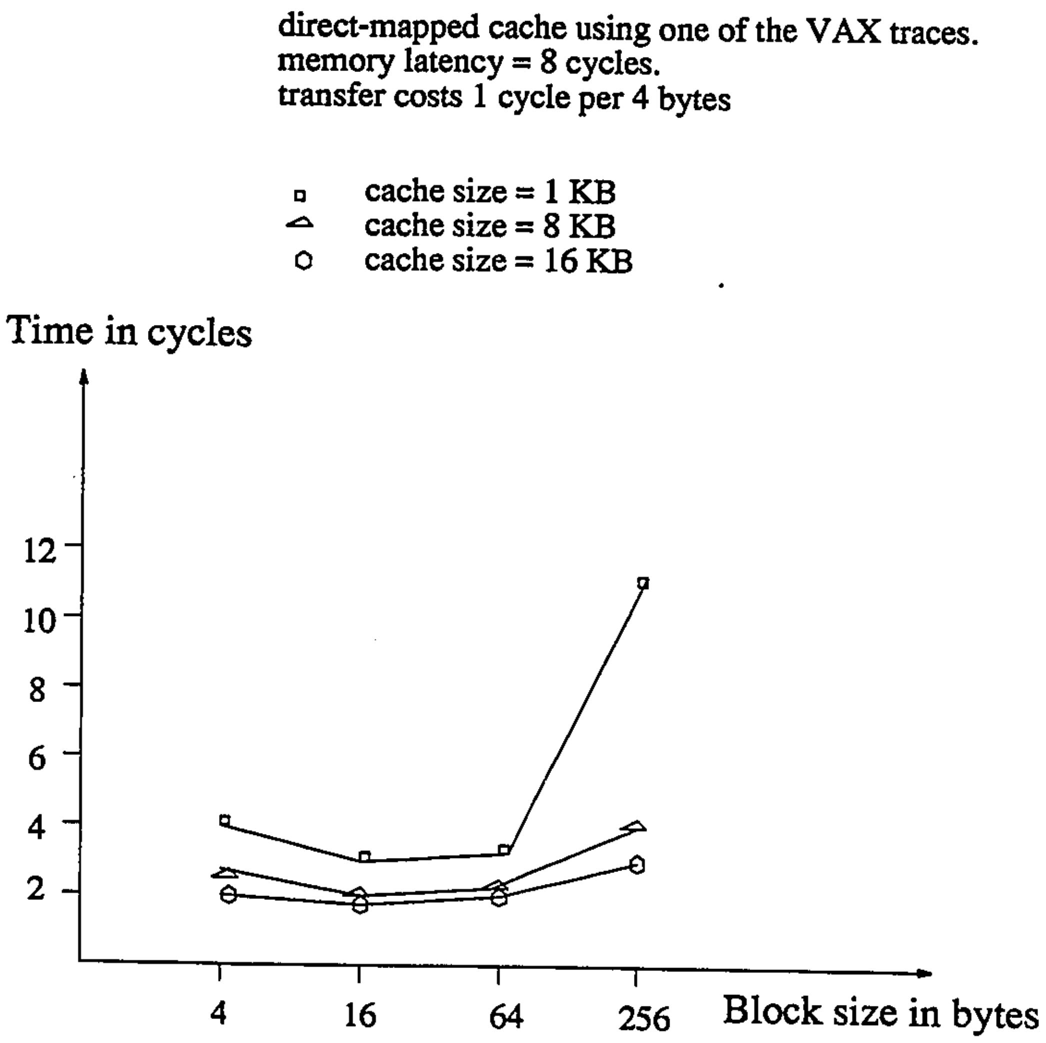


Figure 4.4: Cache block size versus average access time

Policies for Writing to the Cache

There are two different ways for handling write operations in a cache system :

- Write Through (WT): data written by the processor is always written to the main memory and cache at the same time.
- Write Back (WB): the data written by the CPU is stored in the cache and a bit (the dirty bit) in the cache tag is set to indicate that the cache has data that is different from the main memory contents. The write back of this block to memory is deferred as long as possible.

4.1.5 Policies on a Write Miss

There are two policies on a write miss:

- Write allocate (WA): The cache is updated with the new data on a cache write miss.
- No write allocate (NWA): The data is written only to the main memory on a cache write miss.

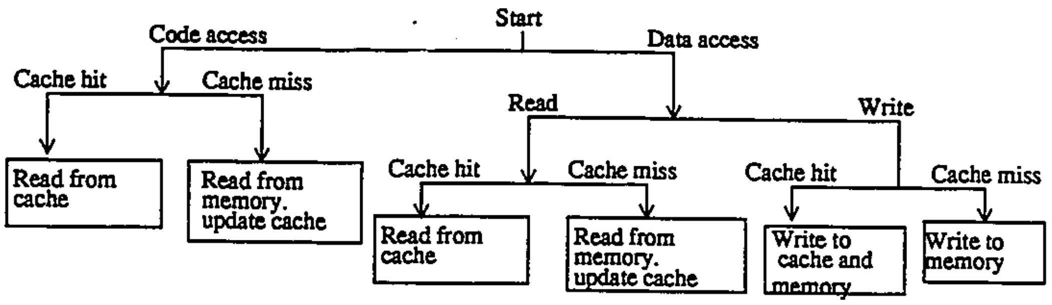


Figure 4.5: Write Through Cache with NWA

4.2 Single Level Cache Design

In the following, we present two models of a single level cache design based on using these different write strategies. The first model, WT-NWA, uses write through with the NWA policy. The second model, WB-NWA, uses write back with the NWA policy.

We use the flow graph method that was introduced in Chapter 3 to present the two models. The flow graph explains the design method, and defines precisely what happens in each operation (read, write, hit and miss) and also gives an overview of the whole system.

Figure 4.5 and figure 4.6 show the flow graphs of our WT and WB models.

4.2.1 Results and limitations of the WT-NWA and WB-NWA models

In this study we assume the following:

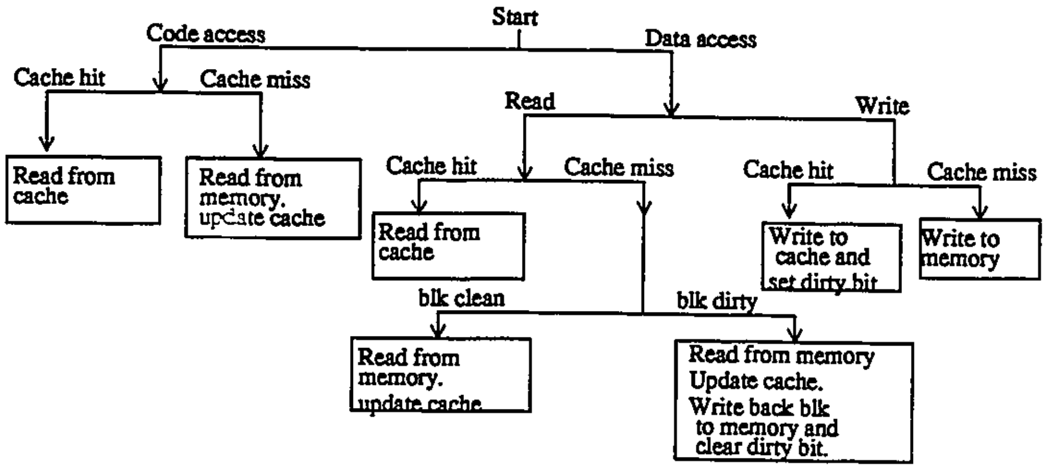

Start
Code access
Data access
Cache hit
Cache miss
Read from cache
Read from memory. update cache
Read
Write
Cache hit
Cache miss
Cache hit
Cache miss
Read from cache
Write to cache and set dirty bit
Write to memory
blk clean
blk dirty
Read from memory. update cache
Read from memory Update cache. Write back blk to memory and clear dirty bit.

Figure 4.6: Write Back Cache with NWA

- The processor has separate data and instruction buses and uses separate instruction and data caches.
- The cache latency is 1 cycle.
- The main memory latency is 16 cycles.
- The cache uses no write allocate and implements the LRU replacement policy.
- The effect of the block size is to add 1 cycle per extra word size. The word size is 4 Bytes and the bus is 1 word wide.
- The CPU is a 1- clock-per-instruction Harvard type architecture.

The performance of the system is measured in the number of cycles per instruction (CPI), which is a more accurate representation of system performance than cache miss rate.

The following are the results of our WT-NWA and WB-NWA models using simulation with the cc1 benchmark trace (defined in Chapter 2).

- The effect of cache size:

  Figures 4.7 shows the performance of the WT-NWA and the WB-NWA cache designs versus the cache size in KBytes.

  The WB-NWA cache design gives better performance than the WT-NWA design. This is a natural consequences of the cost of writes to a slow memory and the high frequency of write operations in the workload. The increase in cache size improves the performance up to a size of 16 KB, then the rate of performance improvement diminishes.

- The effect of cache block size:

  Figures 4.8 shows the performance of the WT and the WB cache designs versus the cache block size in Bytes.

  Optimal WT-NWA and WB-NWA performance occurs with block size in the range of 16- 32 Bytes.

- The effect of cache set associativity:

  Figures 4.9 shows the performance of the WT and the WB cache designs versus the cache set associativity.

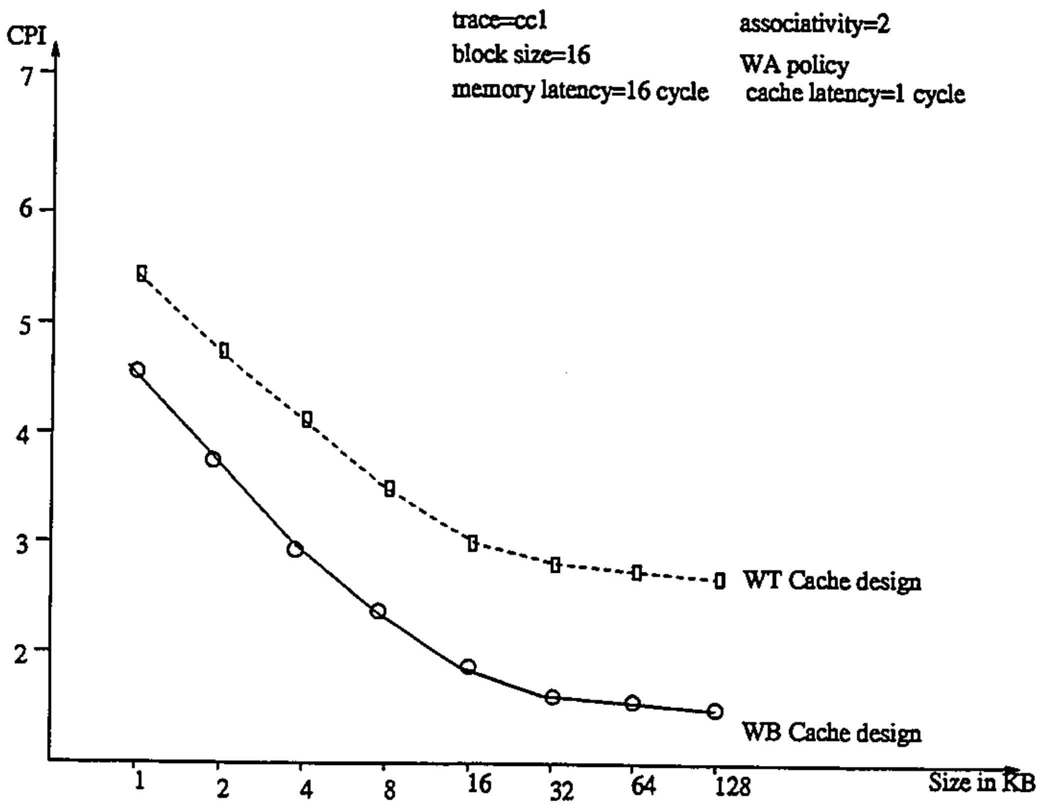


Figure 4.7: Cache size

The increase in cache set associativity improves the performance of the WT-NWA and WB-NWA models. The performance improvement diminishes with associativity larger than 4.

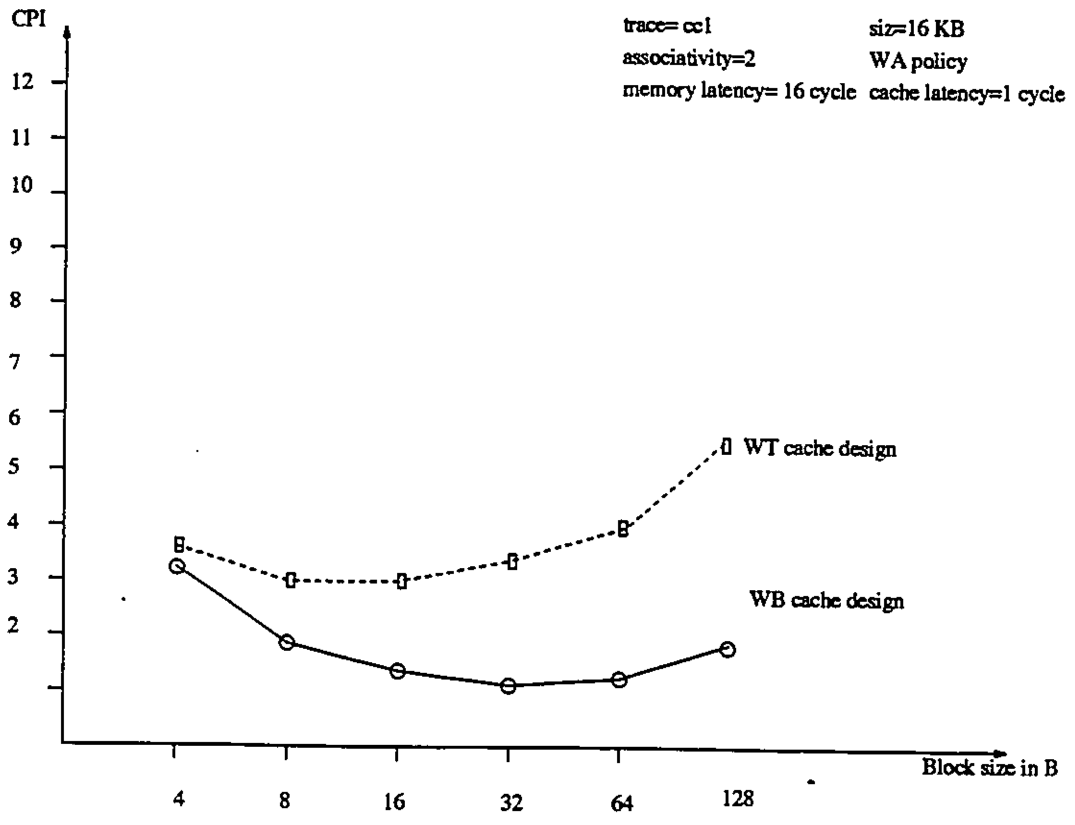


Figure 4.8: Cache block size

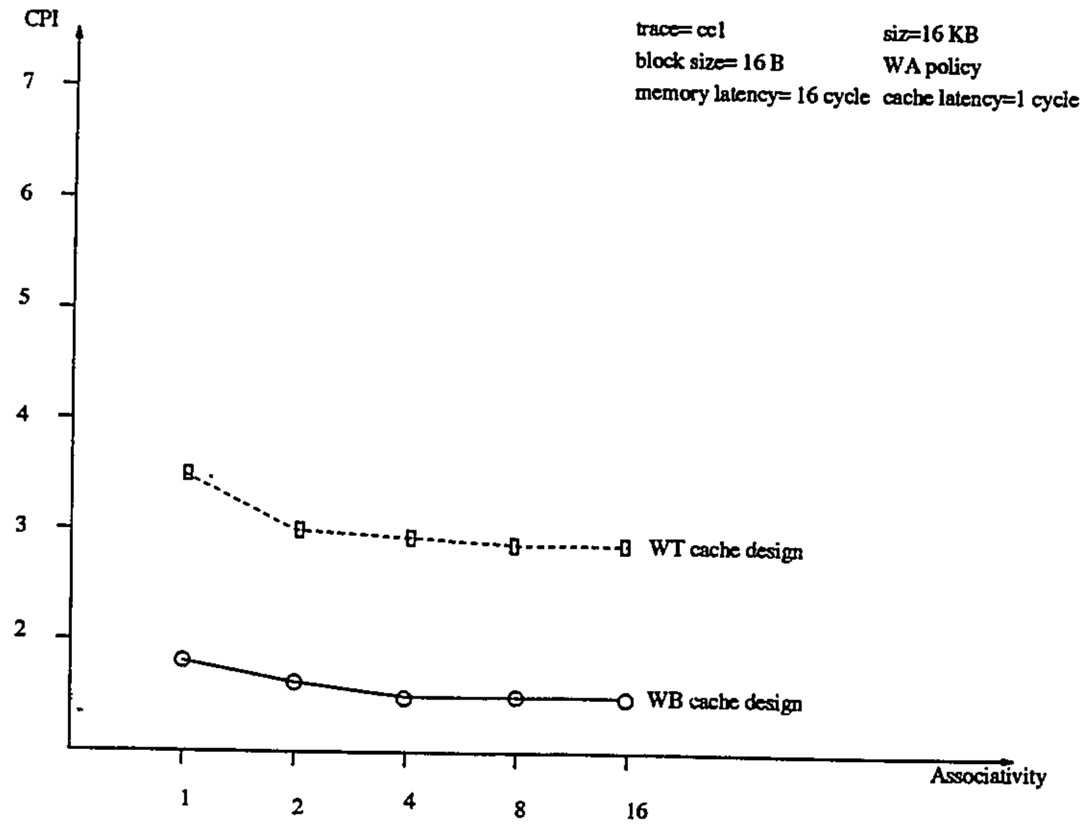


Figure 4.9: Cache associativity

4.2.2 summary

In this Chapter we have reviewed cache design and cache terminology. We discuss the effect of cache parameters on miss rate and present two typical designs for single level caches (WT-NWA and WB-NWA).

Results show that there are limitations in using cache size, set associativity and block size to increase the performance of a single level cache system. The performance of a single level cache system cannot be improved arbitrarily by changing the cache parameters.

The limitations on single level cache performance improvement is becoming increasingly important due to the production of very fast processors with small on-chip caches. Current cache designs use multi-level cache to improve on obtainable single level cache performance.

In the next Chapter, we introduce the multi-level cache design. We present a new concept that both improves performance and makes the multi-level cache suitable for multiprocessor systems.

Chapter 5

Optimizing Multi-level Cache Performance

5.1 Introduction

The results of Chapter 4 show that the performance of a cache system depends on parameters like cache block size, cache size and set associativity. Increasing the block size improves the miss rate. Performance may decrease for larger block sizes, since the increase in block size increases the block transfer time. Increasing the cache size improves the miss rate, but at some point will begin to yield diminishing returns. Associativity helps the miss ratio by reducing cache conflicts. The increase in associativity improves the miss ratio but this improvement diminishes for set associativity greater than eight. These limitations of single level cache performance can be improved by using a multi-level cache system [19] and [37].

The basic principle behind the design of multi-level cache systems is the reduction of the first level cache miss penalty. References that miss the first level cache can be retrieved from the second level cache much faster than from main memory.

The multi-level cache system uses more than one cache. The first level cache, or primary cache, is the closest to the processor and also the fastest. The second level cache is further from the processor, larger and usually slower than the first level cache. A third level cache can be added to further improve performance if the processor/memory speed gap is large and the system runs large software applications.

In this Chapter, we optimize the performance of multi-level cache hierarchies by using a novel design that is based on enforcing the inclusion property. In Section 2, we introduce the multi-level cache design goals. Section 3, discusses the concept of the inclusion property and how it is enforced in our design. In Section 4, we evaluate the cost of inclusion. In Section 5, we present three alternatives for two level cache designs that preserve the inclusion property. Section 6 describes the design of a deep cache hierarchy that uses three cache levels. Section 7, summarizes this chapter.

5.2 Multi-Level Cache Design Goals

The following are our design goals for a multi-level cache:

- We enforce the inclusion property between the first level and the second and third level caches. This is important for cache coherence in multiprocessor systems and is useful for DMA Input/Output and caches in single processor systems [19].

- We allow the first level cache to have optimal performance which is done by enforcing the inclusion property [19].

- We simplify the design of the multi-level cache system by limiting the use of the write back policy to one cache level.

5.3 Enforcing the Inclusion Property

The inclusion property means that all data in the first level cache exists in the second level cache and all data in second level cache exists in the third level cache [7], as is illustrated by figure 5.1. We achieve this by using a special algorithm (defined here by means of a flow graph) for each design and satisfying the necessary conditions for the cache size, block size and set associativity as mentioned in [7].

The algorithm guarantees inclusion through the following:

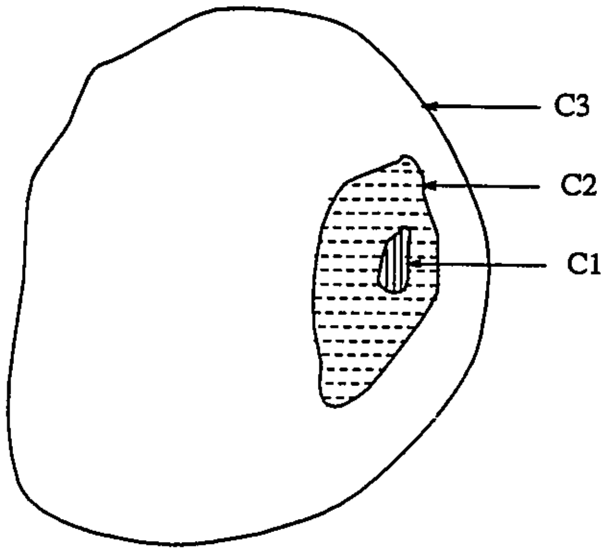


Figure 5.1: Multi-Level Cache with Inclusion Property

- On a read miss to any cache level, the data is loaded to this cache level and all higher levels (i.e., on a cache miss to level 2 the data is loaded to level 2 and level 1 caches).
- The use of NWA "no write allocate" on a write miss.

5.4 Evaluating the Cost of Inclusion

The inclusion property affects the miss rates of the lower level caches. In a two level cache system, the second level cache contains all the references of the first level cache.

With no inclusion the global miss rate for a two level cache would be of a single level cache of size = level 1 + level 2, and with inclusion the global miss rate would be of a single level cache of size = level 2.

Usually the size of the first level cache is much smaller than the size of the second level cache and the miss rate of a large size caches tends to saturate with respect to the increase in their size, thus including the first level cache in the second level cache should have minimal effect on the global cache miss rate.

5.5 Two Level Cache Design and Organization Using Inclusion

Figure 5.2 shows the architecture of a two level cache system. C1 is the first level cache, C2 is the second level cache.

For a simple design, we limit the use of write back to at most one cache. The three options for the two level cache design are the write-through/write-through, write-through /write-back and write-back/write-through. The write-back/write-back has been neglected because we limit the use of write back to one cache. We will compare and study the effect of each cache parameter on each of these different designs.

The following are the different models for the two level cache system that enforces the inclusion property:-

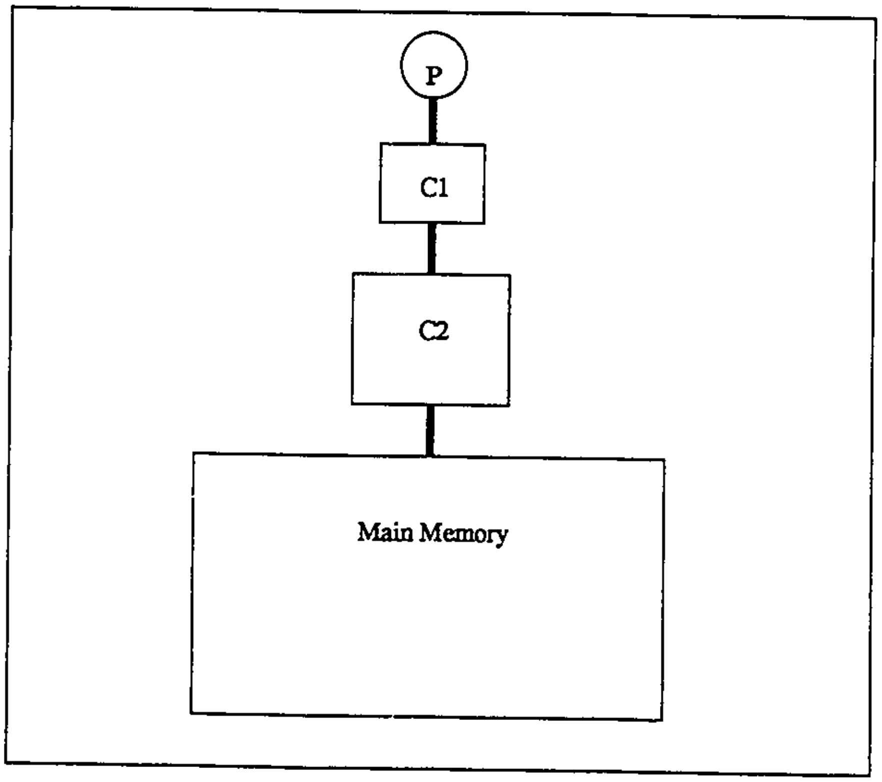


Figure 5.2: Two level cache architecture

5.5.1 The write-through/write-through (WTWT) model

In the WTWT design, the system uses the write through method on both the C1 and the C2 cache levels. This represents the simplest of the possible write handling combinations. Figure 5.3 shows a flow graph detailing what happens on memory access for WTWT using the Not Write Allocate (NWA) policy.

5.5.2 The write-through/write-back (WTWB) model

This model uses the write through method for cache C1 and the write back method for cache C2. The model uses the NWA policy.

Figure 5.4 shows the flow graph of the data memory access section of the WTWB. The code access is identical to the WTWT case.

5.5.3 The write-back/write-through (WBWT) model

In this design the model uses write back for C1 and simple write through method for C2. The model uses the NWA policy.

Figure 5.5 shows the flow graph of the data memory access section of this model.

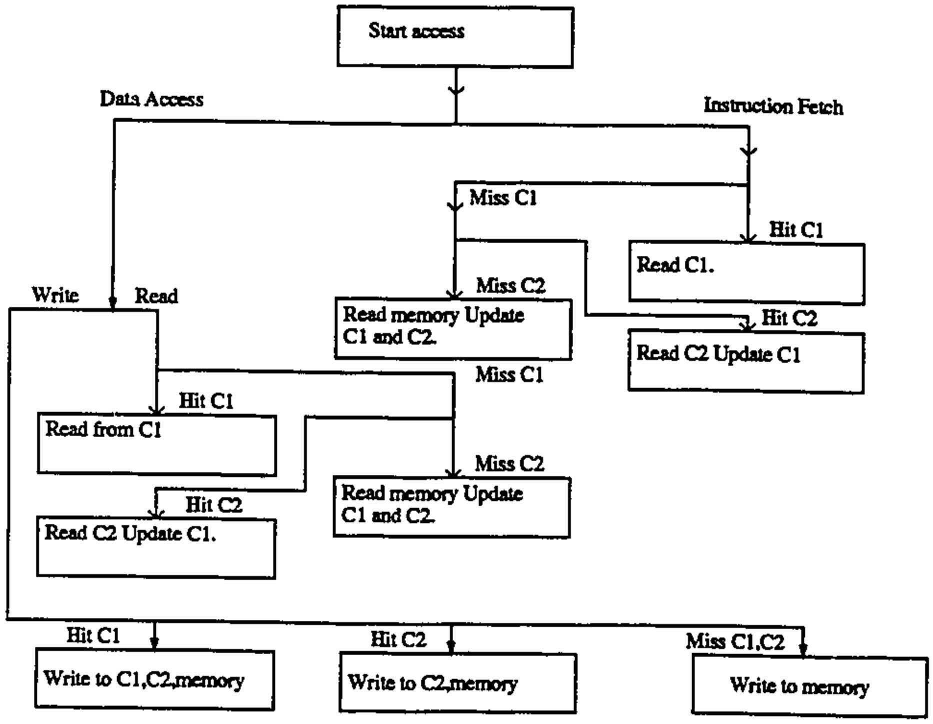


Figure 5.3: WTWT Design

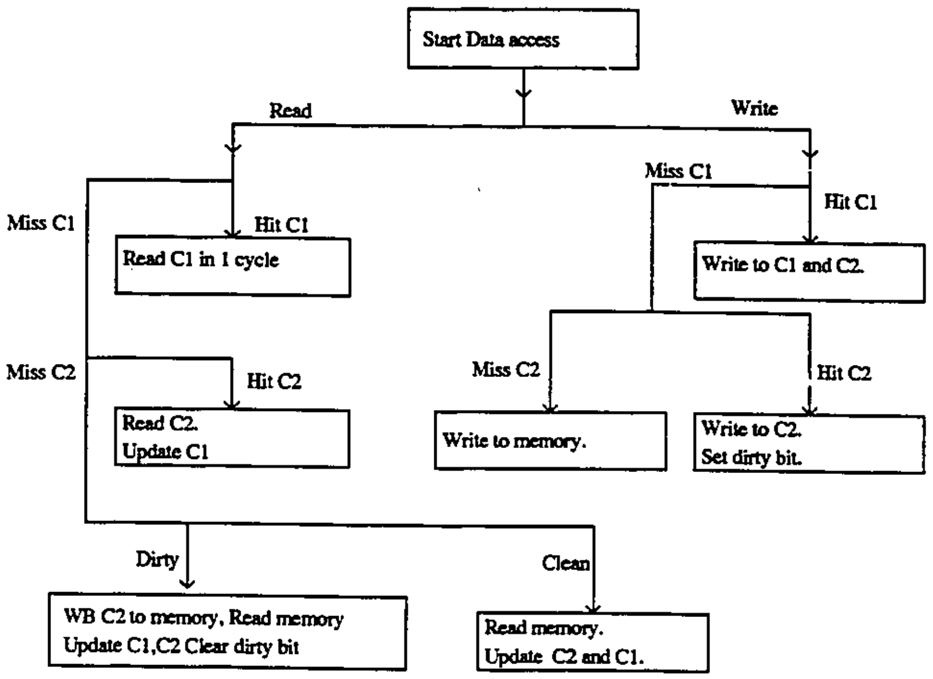


Figure 5.4: WTWB Design (Data Access Section)

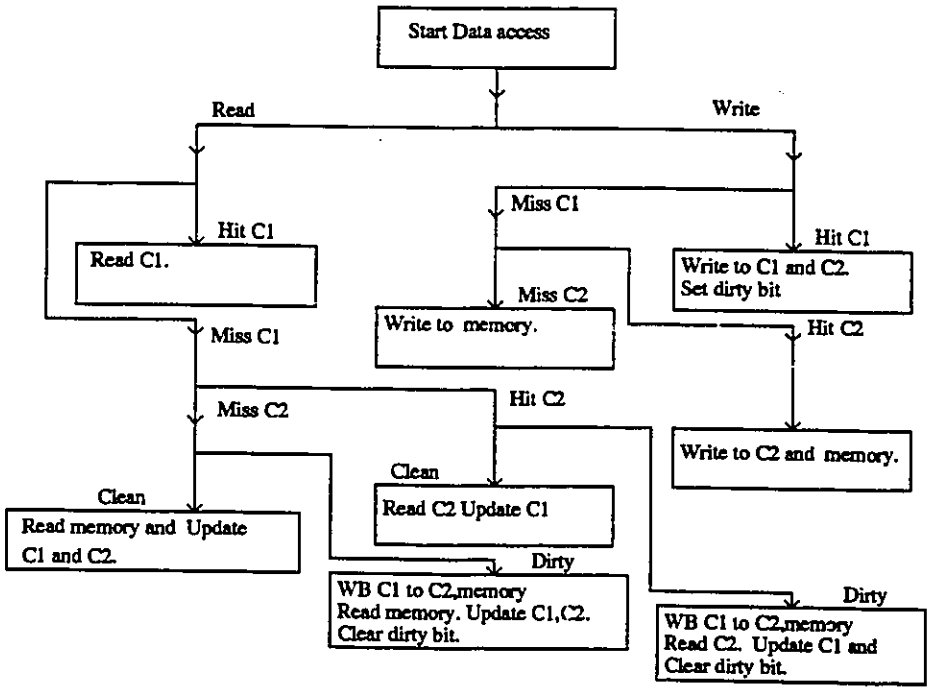


Figure 5.5: WBWT Design (Data Access Section)

5.5.4 Results of the two level cache models

We use the following traces to simulate the two level cache system:

- The H. nessy and Patterson Benchmarks cc1 trace.
- fora ATUM trace (uniprogramming).
- mul8 ATUM trace (multiprogramming).

In this study, the C1 cache latency is assumed to be 1 cycle. The C2 cache latency is assumed to be 4 cycles. The main memory latency is assumed to be 16 cycles. The above values are a reasonable figures for a typical system that uses a processor running at 100 MHz clock speed, C1 latency of 10 ns, C2 latency of 40 ns and main memory (DRAM) normal access time of 160 ns. We assume that the caches use the no write allocate policy, and are direct mapped.

The effect of the block size transfer is to add 1 cycle per extra word. The word size is assumed to be 4 Bytes. The bus is 1 word wide. The CPU is a 1-clock-per-instruction Harvard type architecture.

The performance of the system is measured in number of cycles per instruction (CPI) which, as we have stated, is clearly a better indicator of system performance than the miss rate.

5.5.5 The effect of first level cache size

Figures 5.6, 5.7 and 5.8 show the performance of the WTWT, WTWB, WBWT (two level), WT and WB (single level) cache designs versus the C1 cache size in KBytes.

from the results, we find the following:

- The cycles per access decreases with the increase in level 1 cache size, as one would expect. As the size of the first level cache approaches the size of the second level cache, the rate of performance increase is less, as the limit on the memory system becomes the size of the second level cache.

- The WBWT model gives the best performance of the three two-level cache organizations considered here.

- The two level cache designs give higher performance than a single level cache for C1 sizes up to 16 KB. This means that the two level cache design is useful for systems with small first level caches.

- Adding a second level cache to a system with a large first level cache could decrease the system performance as shown in the three figures. This agrees with the results in [43].

- For very small C1, the WTWB model could outperform the other models. WTWB uses the WT method for C1 making the design of first level cache simpler. Processors with smaller on-chip caches should use the WTWB model.

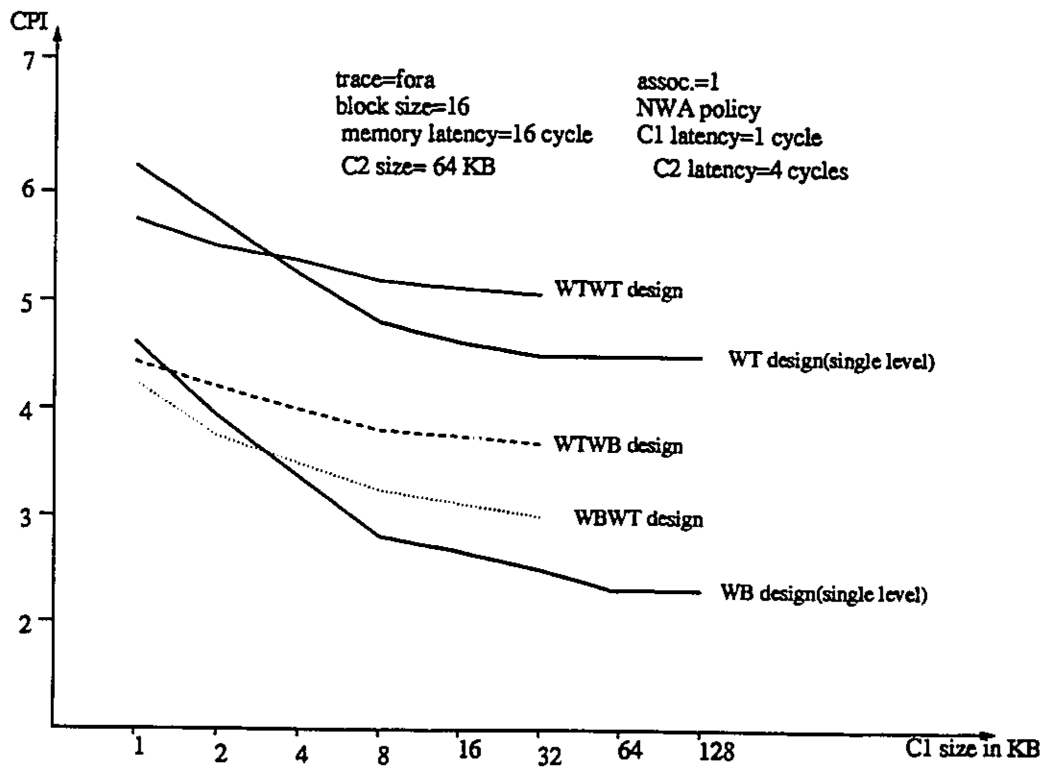


Figure 5.6: CPI versus first level cache size for fora trace

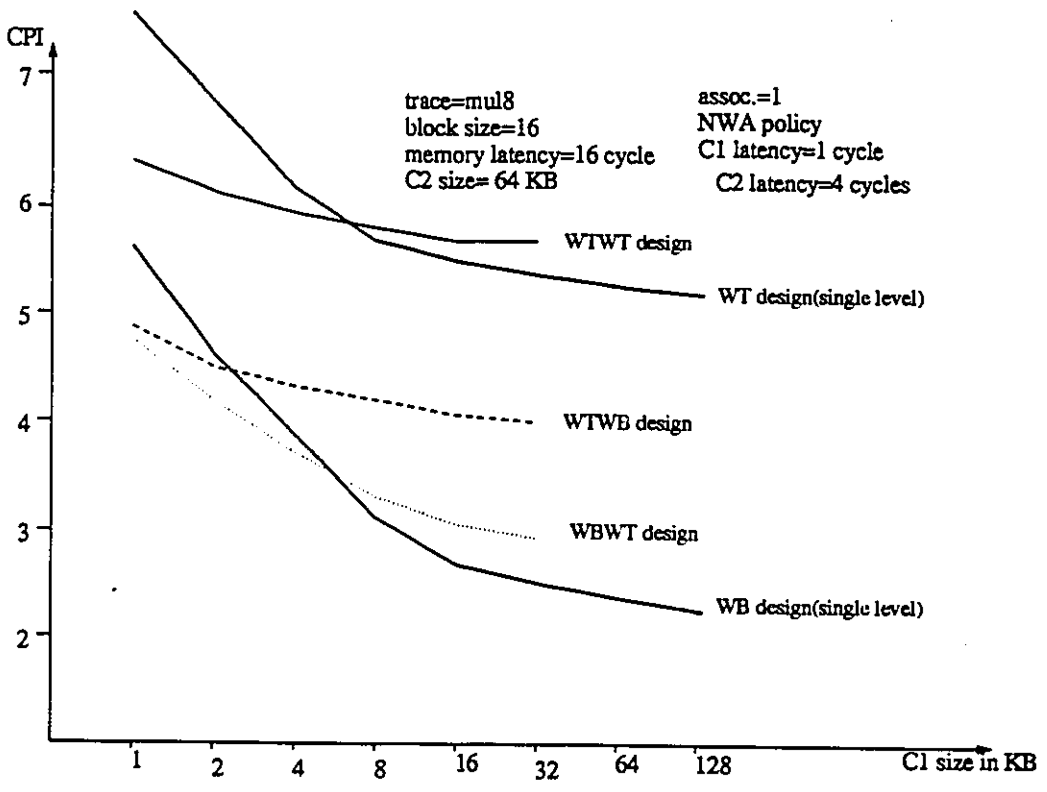


Figure 5.7: CPI versus first level cache size for mul8 trace

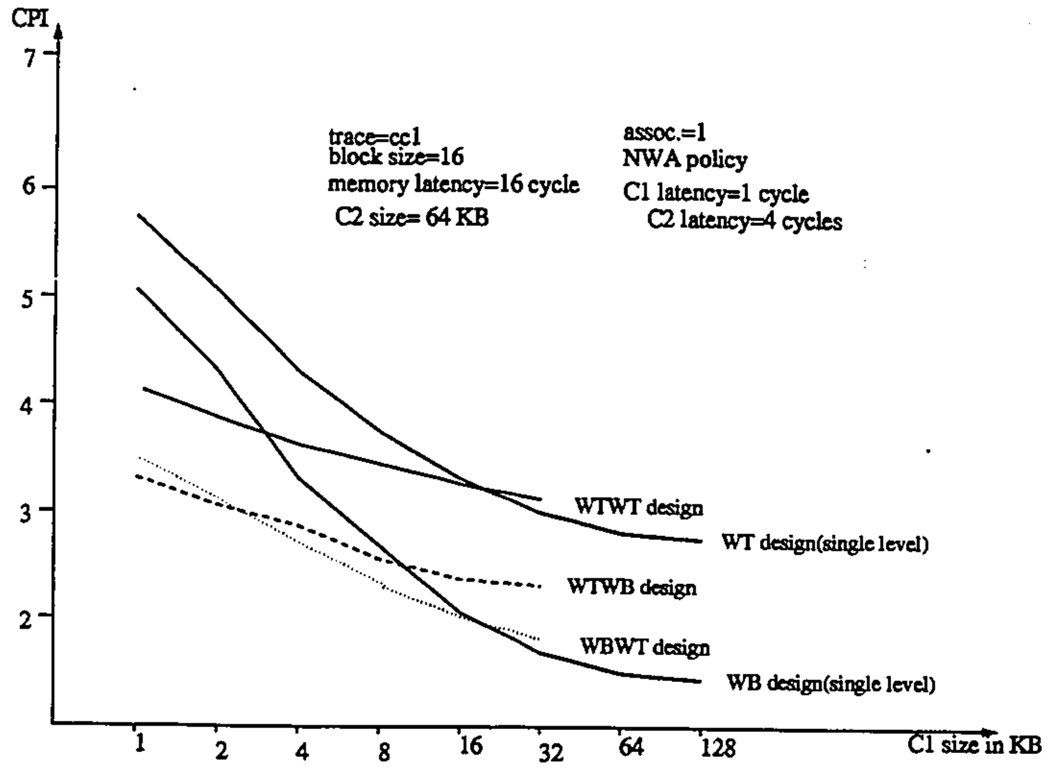


Figure 5.8: CPI versus first level cache size for cc1 trace

5.5.6 The effect of second level cache size

Figure 5.9 shows the performance of the WTWT, WTWB, WBWT cache designs versus the C2 cache size in KBytes for C1 of 1 KBytes and 8 KBytes.

From the results, we find that the rate of performance increase due to the increase in C2 size depends on the size of C1. This rate is higher for larger C1 sizes.

5.5.7 The effect of second level cache latency

Figure 5.10 shows the performance of the WTWT, WTWB, WBWT cache designs versus the second level cache latency in cycles.

From the results, we find the following:

- As expected, longer level 2 cache delays degrade overall performance.

- WTWB shows the most performance decrease with increasing C2 latency. Systems with slow second level caches could use the simple WTWT design and outperform the WTWB design.

  This is because the cost of writing back dirty blocks to memory in the WTWB design on a read miss exceeds the cost of writing on a write hit to C1 and C2 in the WTWT design.

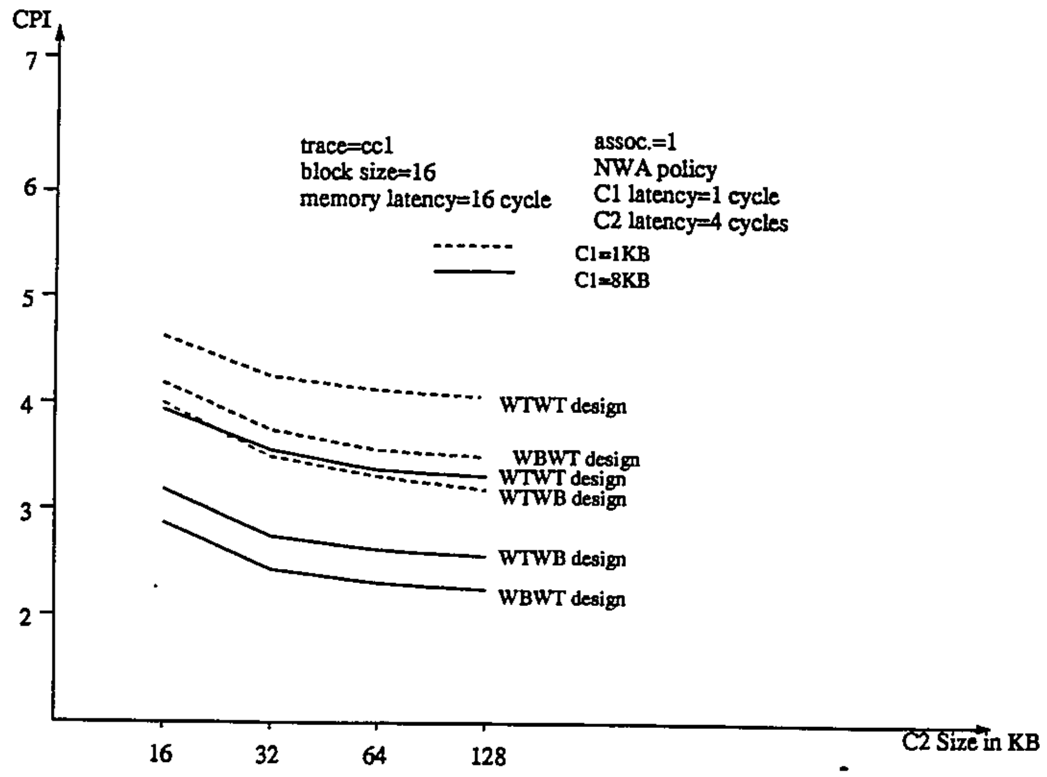
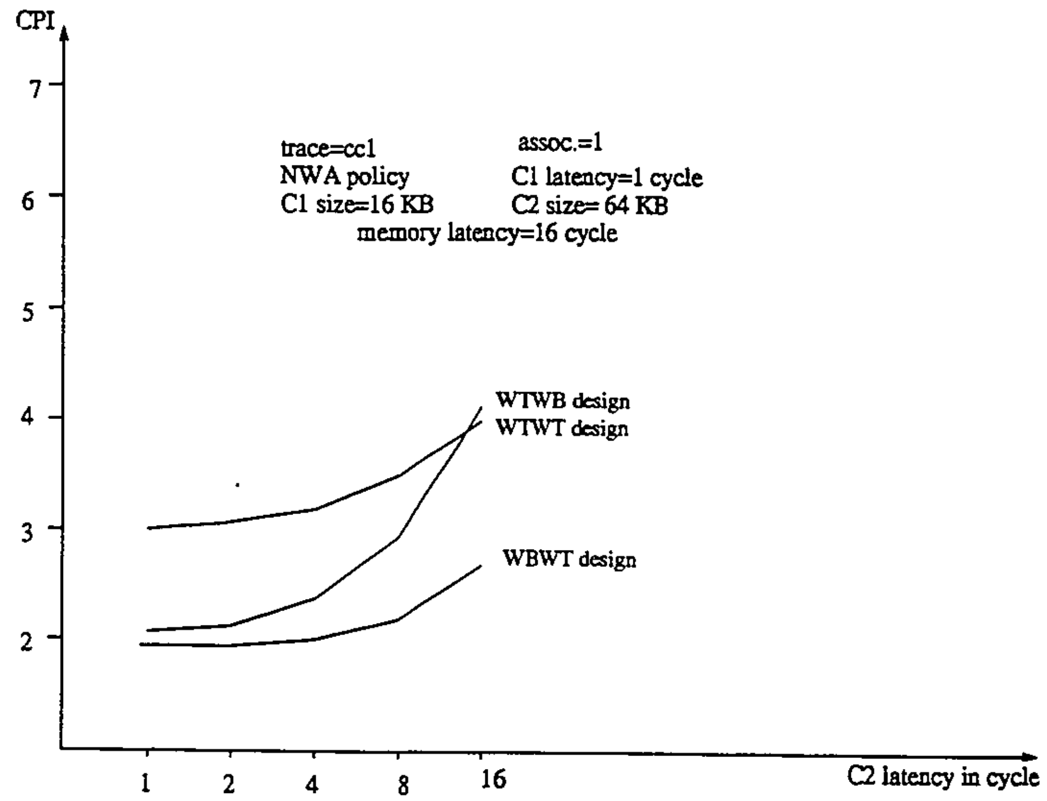


Figure 5.9: CPI versus second level cache size



1

Figure 5.10: CPI versus second level cache latency

5.6 Three Level Cache

For a simple implementation with the inclusion property, we limit the use of the write back method to one cache just as before. We also apply the write through method to two levels of cache that are close to each other.

Three options for the three level cache design are then write-through/write-through/write-through, write-through /write-through/write-back and write-back/write-through/write-through. We have neglected design options like write-back/write-back/write-through because it uses more than one cache for write back. We also have not consider options like write-through/write-back/write-through because it does not apply the write through method to two levels close to each other. We will compare and study the effect of each cache parameter on the different designs, and study the feasibility of adding a third level cache to the memory system. It should be noted that, to the best of our knowledge, this is the only work that discusses the design of a three level cache system.

5.6.1 The write-through/write-through/write-through "WTTT" model

Figure 5.11 shows the flow graph of the WTTT model.

In the WTTT, model we use the write through method for all three caches.This design is the simplest of all designs.

5.6.2 The write-through/write-through/write-back "WTTB" model

Figure 5.12 shows the flow graph of the WTTB model. The code fetch is the same as in the flow graph for the WTTT model. The WTTB model uses write through for C1 and C2 and write back for C3 .

5.6.3 The write-back/write-through/write-through "WBTT" model

Figure 5.13 shows the flow graph of the WBTT model.

The system uses write back for C1 and write through for C2 and C3.

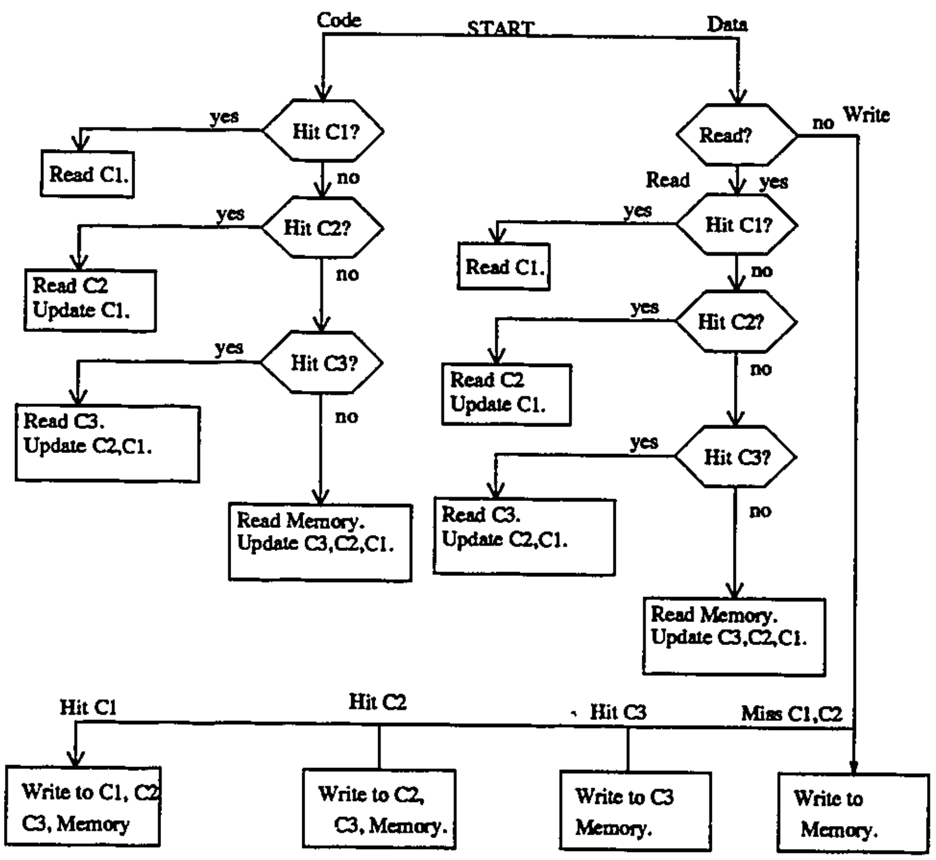


Figure 5.11: WTTT Design

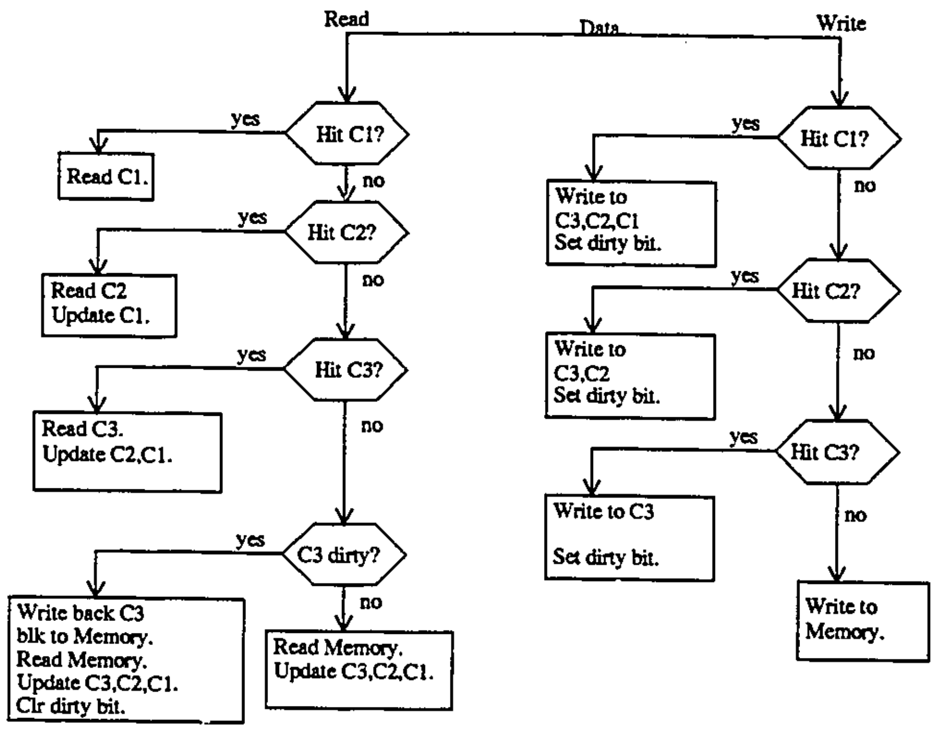


Figure 5.12: WTTB Design (Data Section)

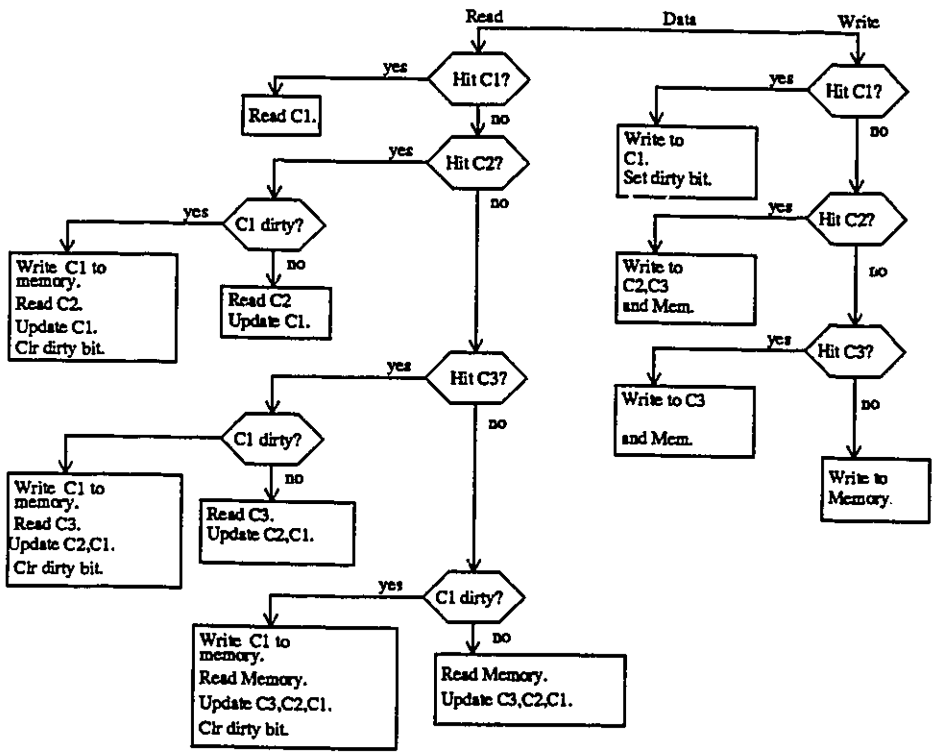

Read
Data
Write
Hit C1?
yes
no
Read C1.
Hit C2?
C1 dirty?
Write C1 to memory. Read C2. Update C1. Clr dirty bit.
Read C2 Update C1.
Hit C3?
Write C1 to memory. Read C3. Update C2,C1. Clr dirty bit.
Read C3. Update C2,C1.
Write C1 to memory. Read Memory. Update C3,C2,C1. Clr dirty bit.
Read Memory. Update C3,C2,C1.
Write to C1. Set dirty bit.
Write to C2,C3 and Mem.
Write to C3 and Mem.
Write to Memory.

Figure 5.13: WBTT Design (Data Section)

5.6.4 Results for three level cache systems

We use fora, mul8 and cc1 traces to simulate the three level cache design. In some graphs we show the results of one trace because the results are very much workload independent, and would overlap on the graph.

The C1 cache latency is assumed to be 1 cycle. The C2 cache latency is assumed to be 4 cycles. The C3 cache latency is assumed to be 16 cycles. The main memory latency is assumed to be 64 cycles.

The above values are reasonable figures for a typical system that uses a processor running at 400-500 MHz clock speed, C1 latency of 2-2.5 ns, C2 latency of 10 ns, C3 latency of 40 ns (one could use fast DRAM) and main memory normal access time of 160 ns.

5.6.5 The effect of first level cache size on miss rate of each cache

Figures 5.14 shows the effect of C1 size on the miss rate of each cache level for all models.

From the results, we find the following:

- The miss rate of C1 is obviously the same miss rate of a single level cache system. I.e., the addition of other cache levels to the hierarchy does not affect

C1 miss rate.

- C2 miss rate increases as C1 size increases because the blocks that are in C1 exist also in C2, and having C2's size fixed, the increase in hit rate for C1 causes fewer blocks to be found in C2.
- The combined miss rate of C1 and C2 is equal to a miss rate of a single level cache that has a size equal to C2. This is also a result of the inclusion property.
- The miss rate of C3 is independent on C1 size, because of the inclusion property between C2 and C1. C2 is shielding C3 from C1.

5.6.6 The effect of second level cache size on miss rate of each cache

Figures 5.15 shows the effect of C2 size on the miss rate of each cache level.

From the results, we find the following:

- The miss rate of C1 does not depend on C2 size, as expected.
- The miss rate of C2 decreases when C2 size increases which causes the miss rate of C3 to increase because of the inclusion property between C2 and C3.
- The combined miss rate of C2 and C3 is equal to a miss rate of a single level cache that has a size equal to C3. This is also a result of the inclusion property.

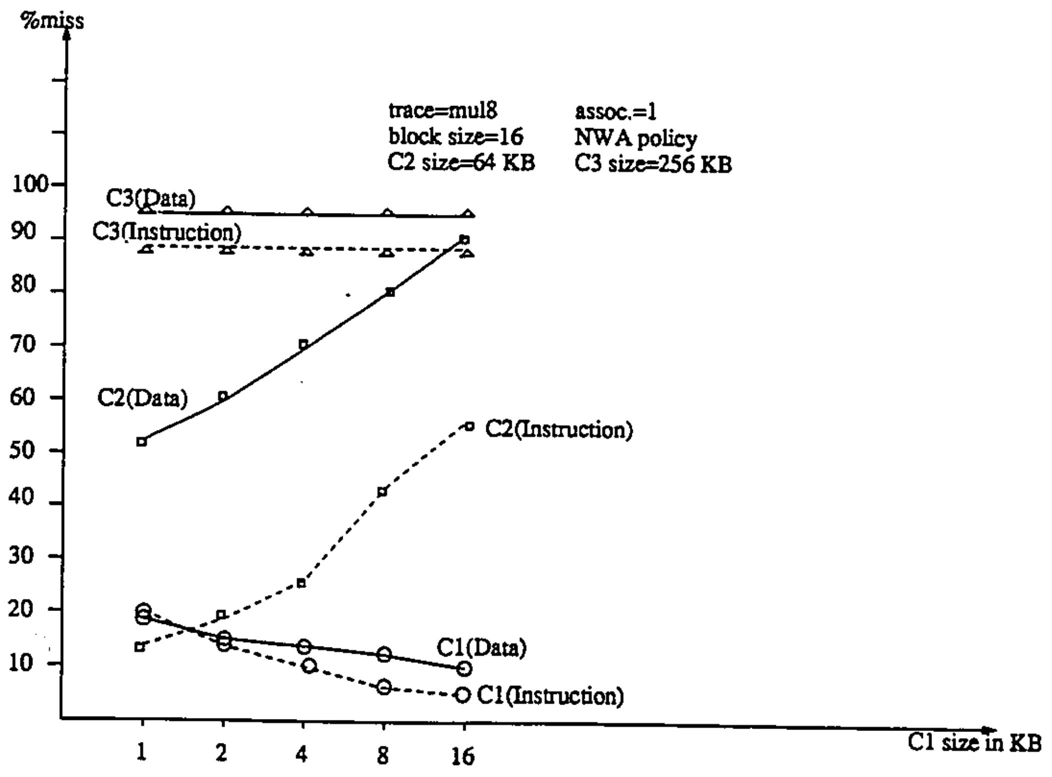


Figure 5.14: The effect of first level cache size on miss rate of each cache

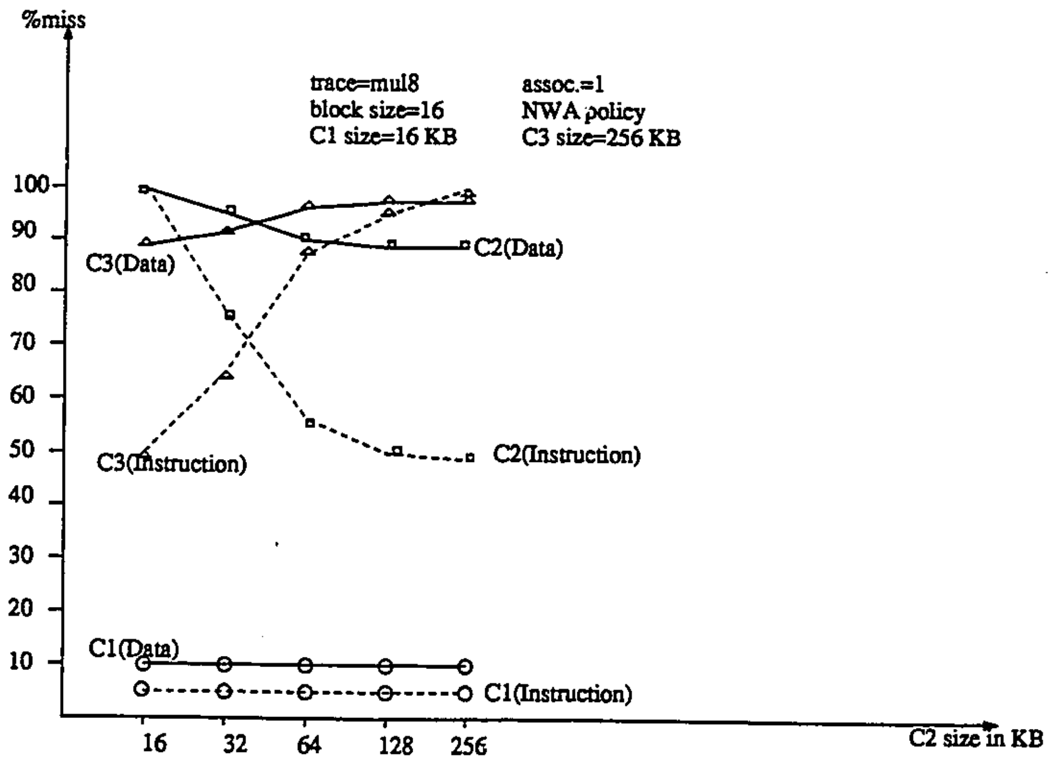


Figure 5.15: The effect of second level cache size on miss rate of each cache

5.6.7 The effect of first level cache size on performance

Figures 5.16, 5.17 and 5.18 show the effect of C1 size on the system performance for a single level, two level and a three level cache systems.

From the results, we find the following:

- The WBTT model gives the best performance compared to the WTTT and WTTB designs.
- The WBTT model has a strong dependency on C1 size.
- The two level cache designs gives higher performance than a single level cache for C1 size up to 16 KB. This means that the two level cache design is useful for systems with small first level caches as we concluded from the studying of the two level cache.
- Adding a third level cache to the system improves the performance of the system and gives higher performance than a single or two level cache systems.
- The improvement in performance due to adding a third cache increases if the trace is longer(cc1 and mul8), which means that the three level cache system is helpful with systems using large (or more address-diverse) software applications. This agrees with Agarwal's results for simulation of steady-state cache performance versus the trace length [2].

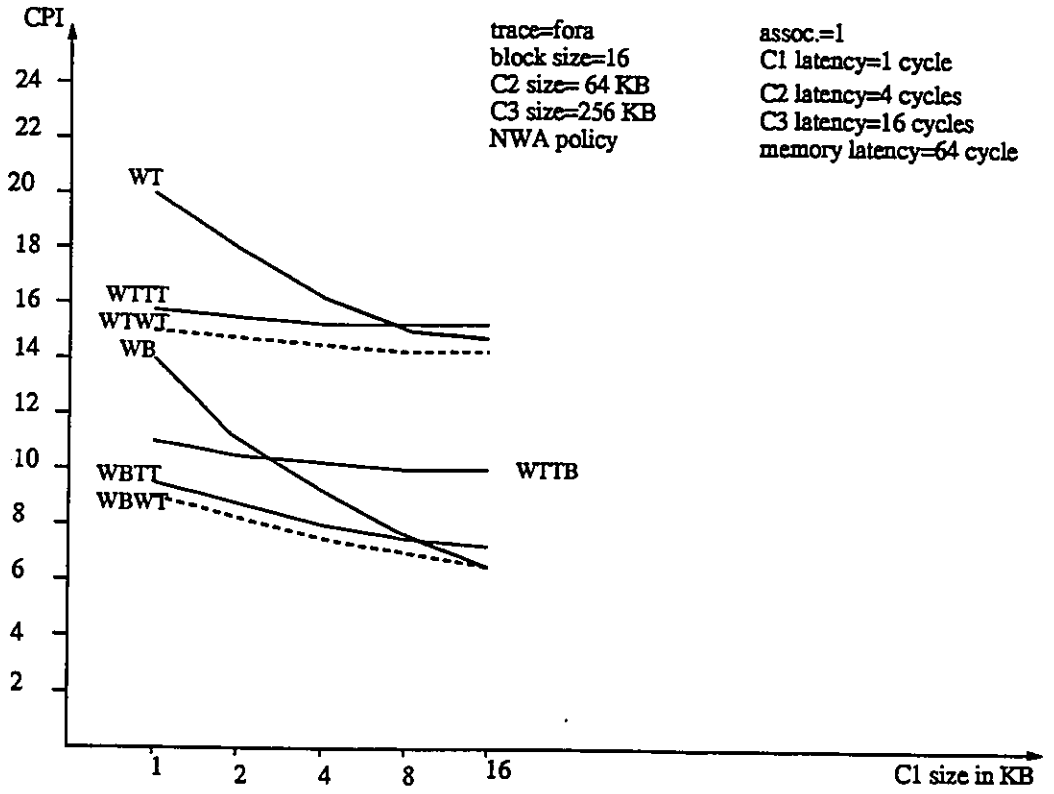


Figure 5.16: The effect of first level cache size on performance for fora trace

- The WTTB model gives a performance that is better than a single level WB cache and similar to a WBWT design. The WTTB model uses the WT method for C1 and C2 which makes the design of the first and second level cache simpler.

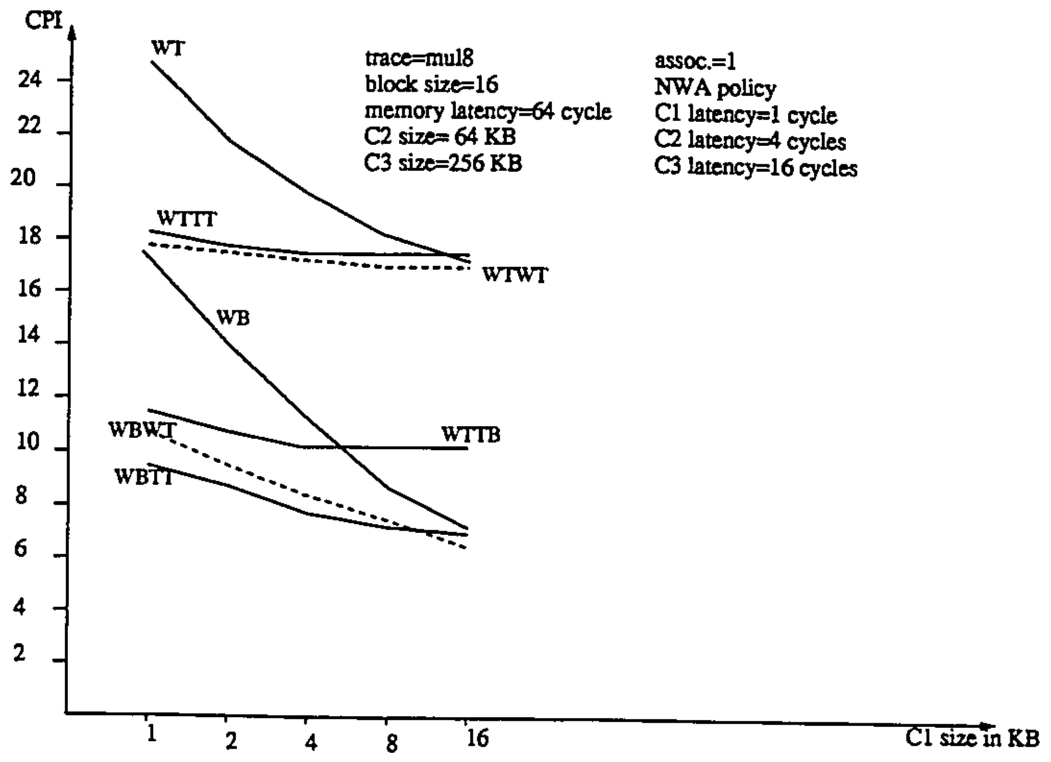


Figure 5.17: The effect of first level cache size on performance for mul8 trace

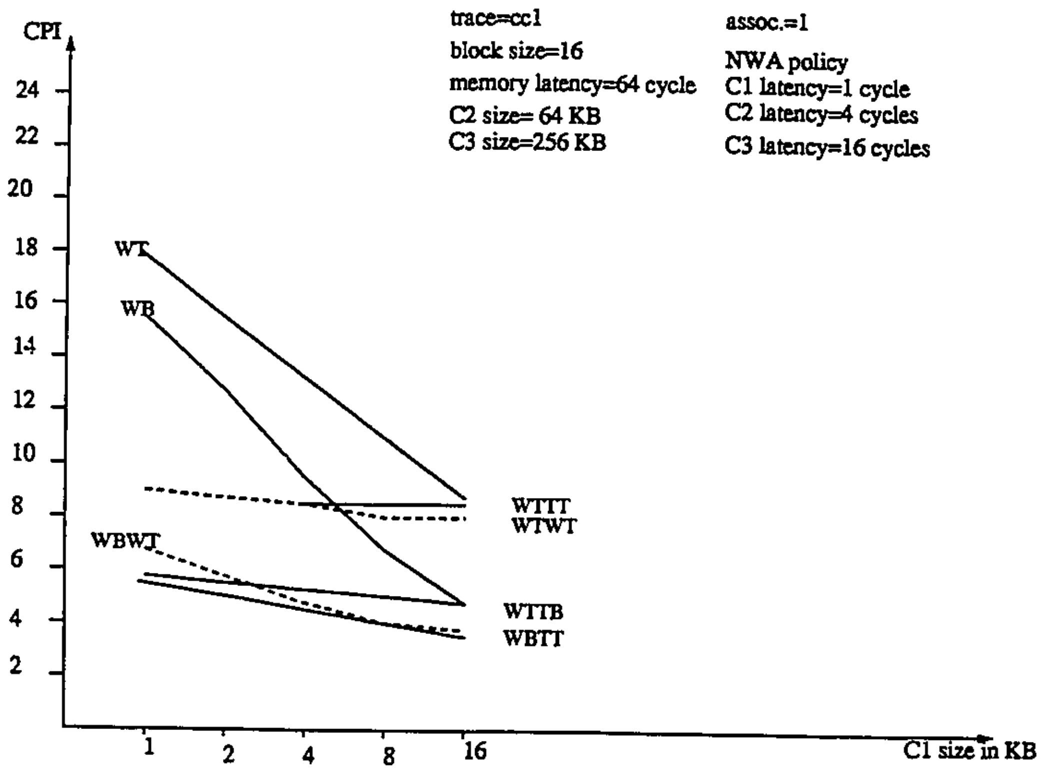


Figure 5.18: The effect of first level cache size on performance for cc1 trace

5.6.8 The effect of second level cache size on performance

Figures 5.19 shows the effect of C2 size on performance.

From the results, we find the following:

The performance gain for all models with increasing C2 size is small, which suggests that we could use a smaller second level cache without affecting the system performance.

The small effect of C2 size on performance could be explained by the following:

If C2 size increases, the C2 hit rate increases but the hit rate of C3 will decrease because of inclusion. If C2 size decreases then C2 hit rate decreases but C3 hit rate will increase because of inclusion. Any change in C2 size will have an opposite effect on C3 performance and the change in overall system performance is minimal.

5.6.9 The effect of third level cache size on performance

Figures 5.20 shows the effect of C3 size on performance.

From the results, we find the following:

The results show that the increase in C3 size helps to improve the performance of all models. The best performance gain is achieved from the WBTT and WTTB designs.

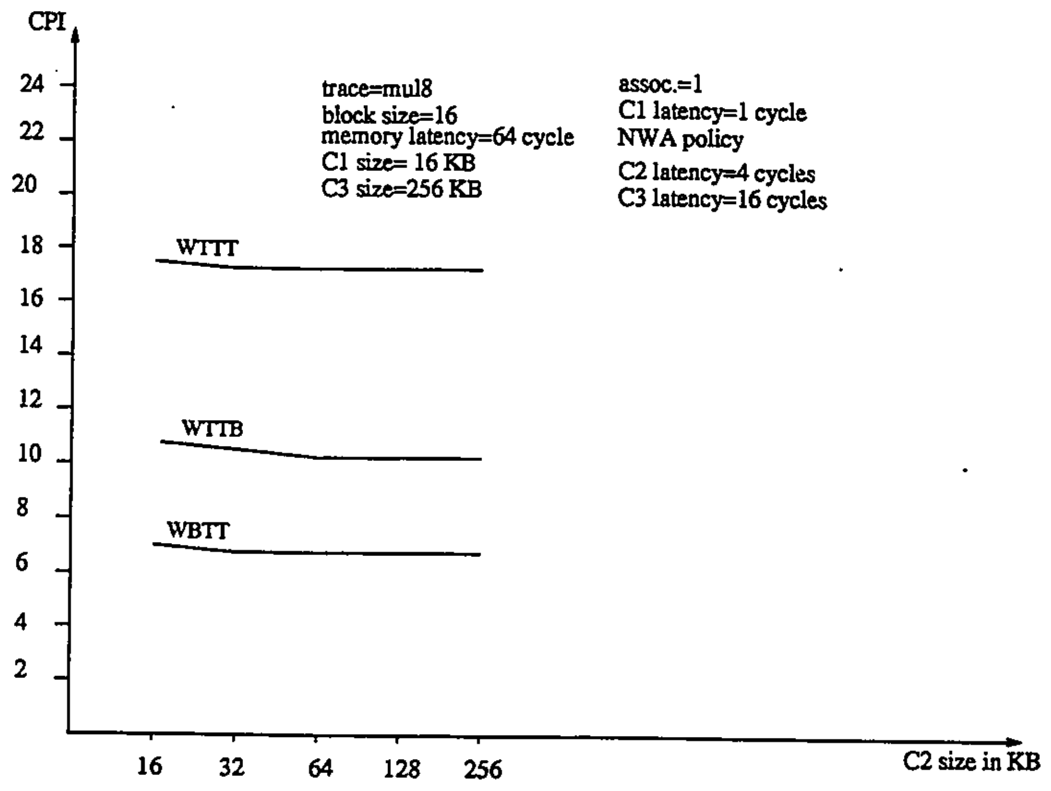


Figure 5.19: The effect of second level cache size on performance

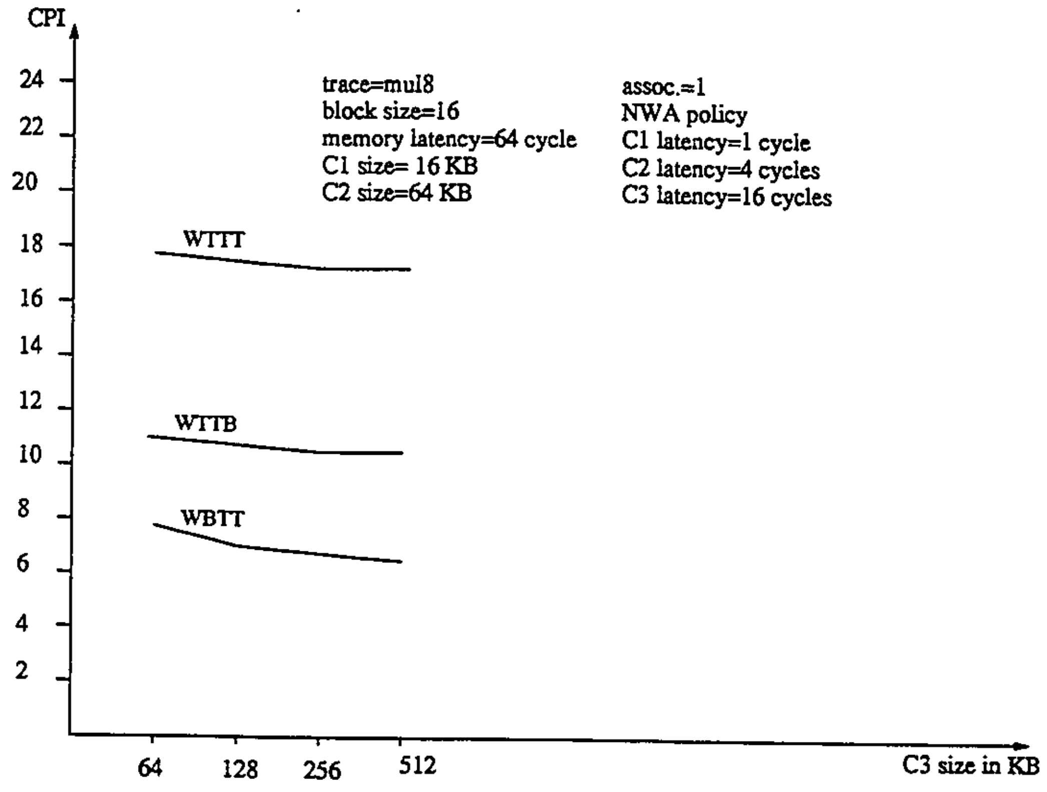


Figure 5.20: The effect of third level cache size on performance

5.6.10 The effect of second level cache latency on performance

Figures 5.21 shows the effect of C2 latency on performance of the three models.

From the results, we find the following:

The 16 times increase in C2 latency decreases the performance of WTTT and WTTB only by 2% and the WBTT performance by 3%. This small change in performance with the huge increase in C2 latency suggests that the 3 level cache design could use a slower second level cache with very little performance degradation.

This can be explained by considering the total CPU execution time versus the miss rate and latency of the different caches and given as:

$$T = L1 + L2 \times M1 + L3 \times M1 \times M2 + Lm \times M1 \times M2 \times M3.$$

L1, L2 and L3, Lm are C1,C2,C3 and memory latency. M1,M2 and M3 are the miss rates of C1,C2 and C3 caches.

By substituting the values of L1=1, L2=4, L3=16 and Lm= 64 and that $M1 \times M2$= the miss rate of C2= M(C2) and $M1 \times M2 \times M3$ = miss rate of third level cache= M(C3) because of the inclusion property.

then T= 1+ 4M(C1) + 16 M(C2) + 64 M(C3) which shows that C3 has more effect on T than C2 because it has a weight of 64 compared to the weight of 16 for C2. C1 has a large effect on performance because the value of M(C1) is much higher than M(C2) and M(C3).

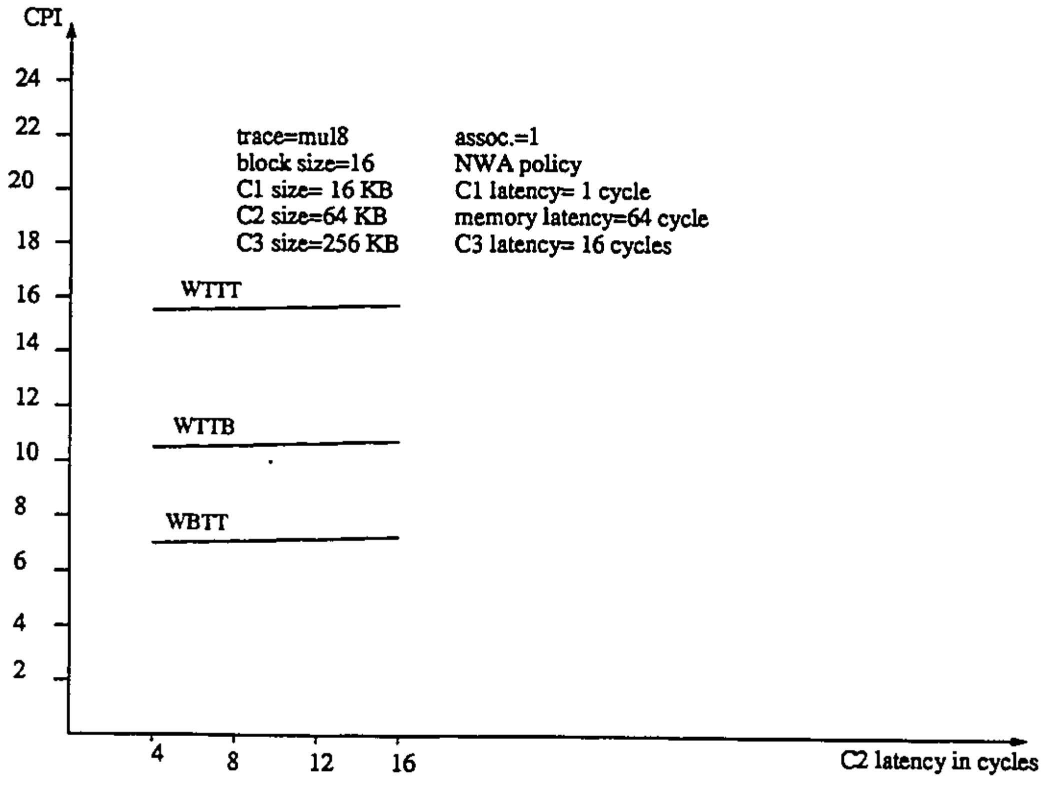


Figure 5.21: The effect of second level cache latency on performance

5.6.11 The effect of third level cache latency on performance

Figures 5.22 shows the effect of C3 latency on performance of the three models.

From the results, we find the following:

The increase in C3 latency decreases the performance of the three different models. The performance of the WTTB model deteriorates badly from the increase in C3 latency. The WTTB model is very dependent on the C3 cache size and latency.

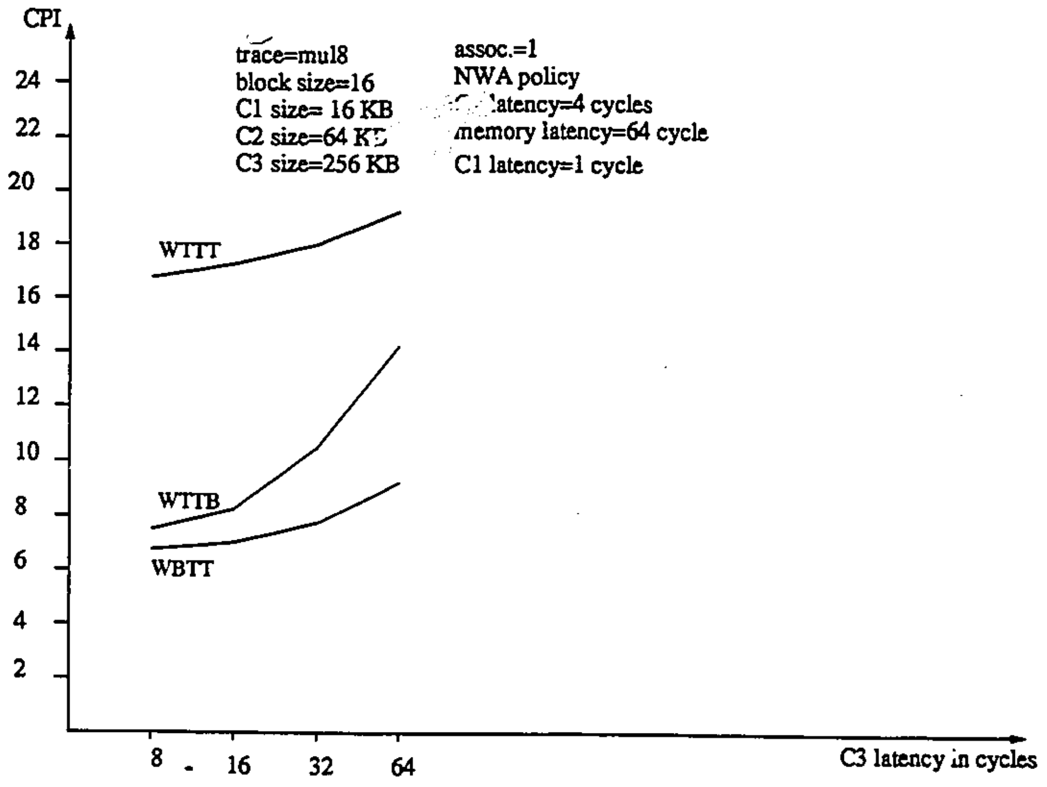


Figure 5.22: The effect of third level cache latency on performance

5.7 Summary

In this Chapter, we have presented novel multi-level cache hierarchy designs based on enforcing the inclusion property. Enforcing the inclusion property guarantees optimal first level cache performance and simplifies multiprocessor cache coherency protocols.

Limiting the use of the write back method to one cache level simplifies the system design and helps in enforcing the inclusion property.

A multi-level cache design should be carefully evaluated before using it. For a large first level cache, using a multi-level cache design could actually decrease performance.

We have introduced a novel method to optimize the design of the memory cache system by using the multi-level cache concept presented in this chapter. In Chapter 3, we have presented a novel method to optimize the performance of the main memory. In the next chapter, we will combine both methods to produce a memory system that is optimal for each memory level and we will further optimize the performance of the overall memory hierarchy.

Chapter 6

Optimizing Overall Memory Performance

6.1 Introduction

In Chapter 5, we presented a novel method for multi-level cache design. Multi-level caches help system performance most dramatically if the first level cache is small. However, even with the use of a multi-level cache system, the performance improvement is limited. Increasing the second or third level cache sizes rapidly reaches diminishing returns.

In Chapter 3, we presented novel multi-bank DRAM interleaving protocols that can greatly increase the performance of the main memory.

In this chapter, we combine the DRAM protocols (from Chapter 3) and the cache systems of Chapter 4 and Chapter 5 to produce a complete memory hierarchy. This composite system has optimal main memory performance and optimal cache performance. In this chapter, we focus on further tuning the performance of this memory complete system.

We study the effect of DRAM parameters and cache parameters on the overall memory system performance using simulation of the complete memory system(main memory and cache).

In Section 2, we present the architecture of the complete memory system. Section 3 discusses the performance tuning of the memory system with a single level of cache. In Section 4, we discuss the performance tuning of the memory system with multi-level caches. Section 5 discusses the effect that memory designs have on system performance.

6.2 The Complete Memory System Architecture

Figure 6.1 shows the block diagram of a complete memory system, with 1 level of cache. It consists of the cache unit,cache control, cache tag comparator, DRAM, RAS control, row comparator and CAS unit.

We assume that the main memory uses multi-bank interleaving with one of the four protocols defined in Chapter 3. The cache unit uses one of the cache models

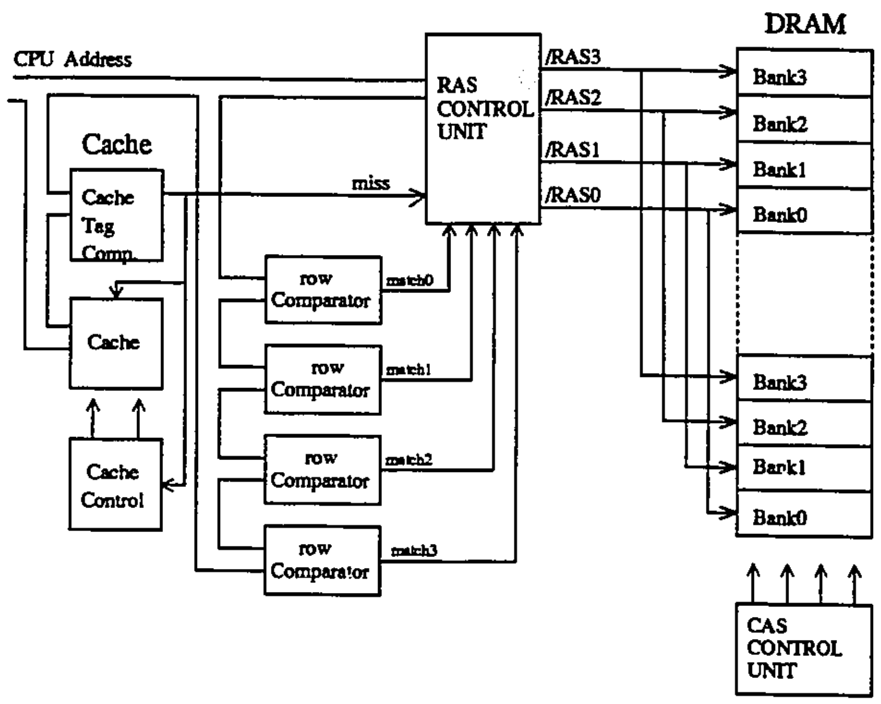


Figure 6.1: The System Architecture

of Chapter 4, or one of the the multi-level cache models of Chapter 5.

6.3 Optimizing the Performance of the Complete Memory System (with a Single Cache Level)

6.3.1 Introduction

The memory system uses the write through or the write back models (WT and WB of Chapter 4) with one of the following 5 main memory designs: the Random access protocol, the Precharge protocol, the Fast Page protocol, the Limited Fast Page protocol and the Fast Page Precharge protocol [Chapter 3]. In this section, we optimize the performance of the overall memory system with respect to the cache and the DRAM design parameters.

6.3.2 The results of the complete memory system (with single level cache)

The system is assumed to have the same parameters as before. The system has the following:

The DRAM chips are chosen to be 4 M bits with 80 ns access times and support fast page mode. The total time for random access for this DRAM is 155 ns. Assume that this access takes a total of 16 cycles considering that the cycle time is 10 ns. For fast page mode, the access takes 6 cycles and for the precharge access it takes 9 cycles according to the DRAM access timing [15].

The number of MRU banks for the limited fast page protocol or for the fast page precharge protocol is 1/2 of the total number of banks.

The processor has separate data and instruction buses and uses separate instruction and data caches. The cache latency is 1 cycle, and the cache uses the NWA policy with LRU replacement.

The effect of a block transfer is to add 1 cycle per word size. The word size is assumed to be 4 Bytes. The bus width is equal to 1 word.

The performance of the system is measured by the number of cycles per instruction (CPI).

We use fora and mul8 traces to simulate the memory system design. In some conditions we show the results of one trace because the results are very much workload independent.

6.3.3 The effect of variable address interleaving on the performance of a complete memory system (with single level cache)

Figure 6.2, Figure 6.3, Figure 6.4 and Figure 6.5 show the performance of the different memory designs versus the bank select address.

The location of bank select bits changes to cover the whole address space.

From the results:

- The cache with the precharge protocol gives the best performance when the banks are mapped in to A2-A4 (which is a word size) and performs badly with high address interleaving. This is the same as the results for the DRAM using precharge without a cache (Chapter 3).
- The cache with fast page, limited fast page and fast page precharge protocols give the best performance when the banks are mapped in to the A12-A14 range. This is a 2 K word and exactly the size of a page for the 4 M bytes DRAM. We conclude that the system gives it's best performance by using banks that are interleaved around a page size. This is the same result as for the three DRAM protocols with no cache (Chapter 3).
- The memory system has the best performance when it uses the fast page precharge protocol with banks mapped around a page size. In this case, the performance increases by 60% for WT cache and 35% for WB cache (relative to the cache system using random DRAM accesses).
- In high order address interleaving, the memory system performance is the same for the fast page, limited fast page and fast page precharge protocols.
- The system with a write back cache gives better performance than with write through as we would expect (as in Chapter 4).

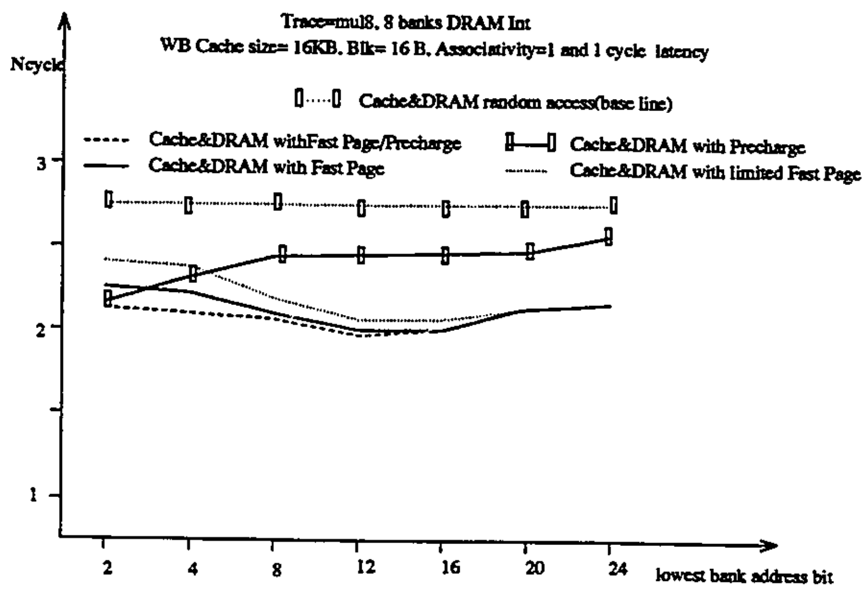


Figure 6.2: Variable address interleaving for WB and DRAM using mul8 trace

- The performance improvement for WT cache using DRAM variable address interleaving is double the performance improvement for WB cache with DRAM variable address interleaving. This is because the write through cache uses the DRAM in every write operation.

6.3.4 The effect of cache size on the performance of a complete memory system (with single level cache)

Figure 6.6 and Figure 6.7, show the performance of the complete memory system versus the cache size.

Note that the Precharge Protocol uses 8 banks that are mapped in the A2-A4 address range (optimal for it, from Chapter 3) and other protocols use 8 banks that

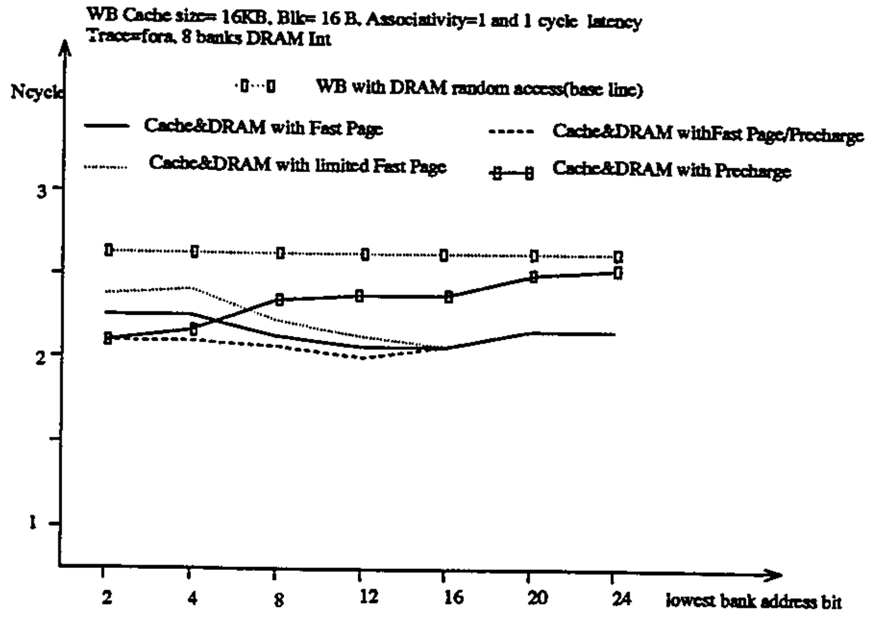


Figure 6.3: Variable address interleaving for WB and DRAM using fora trace

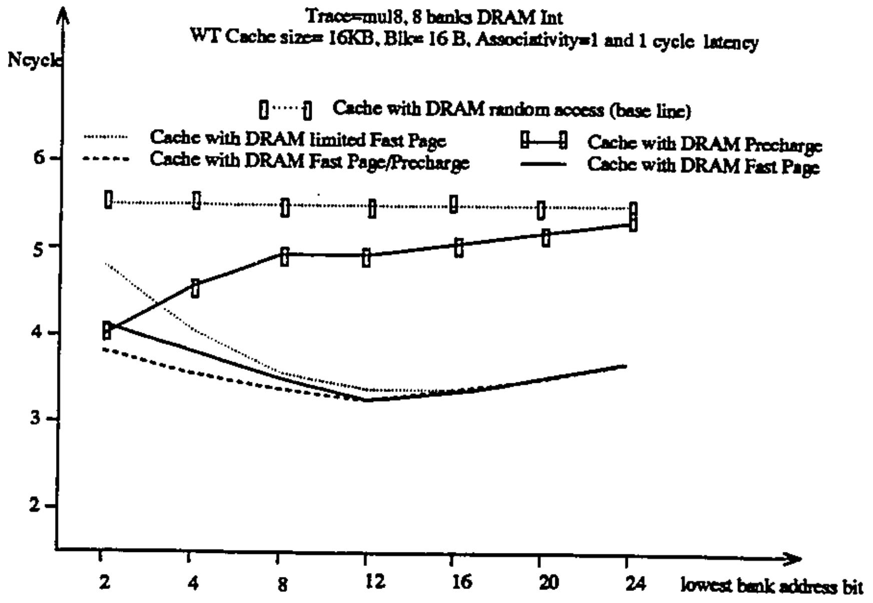


Figure 6.4: Variable address interleaving for WT and DRAM using mul8 trace

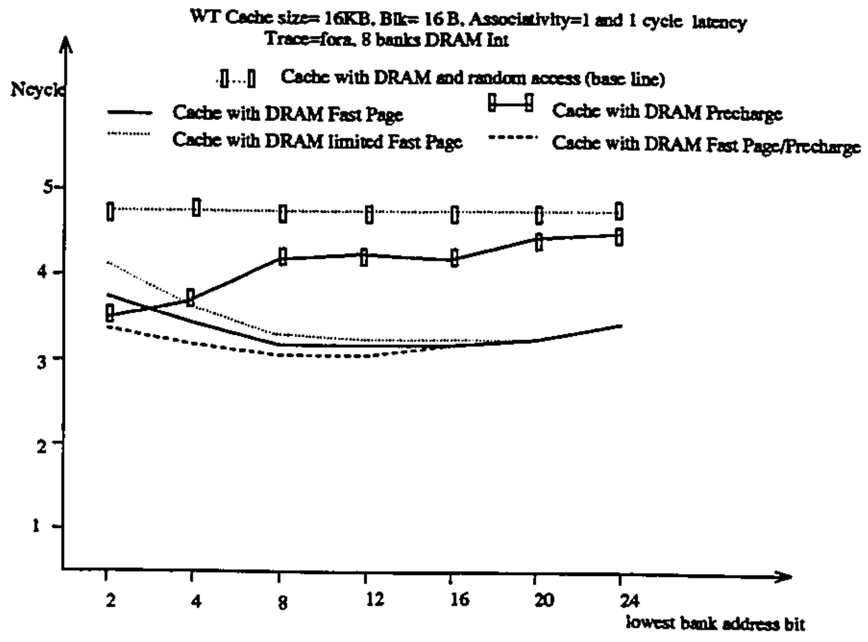


Figure 6.5: Variable address interleaving for WT and DRAM using fora trace

are mapped in the A12-A14 address range (optimal for them, from Chapter 3).

From the results, we find the following:

- Increasing the cache size improves the system performance as expected.
- the memory system with the fast page precharge protocol gives the best performance.
- The increase in system performance due to the use of the optimized bank interleaving is so profound that the system out performs others using 4 times larger cache and no bank interleaving.
- The system with write through cache benefits (from the optimized bank inter-

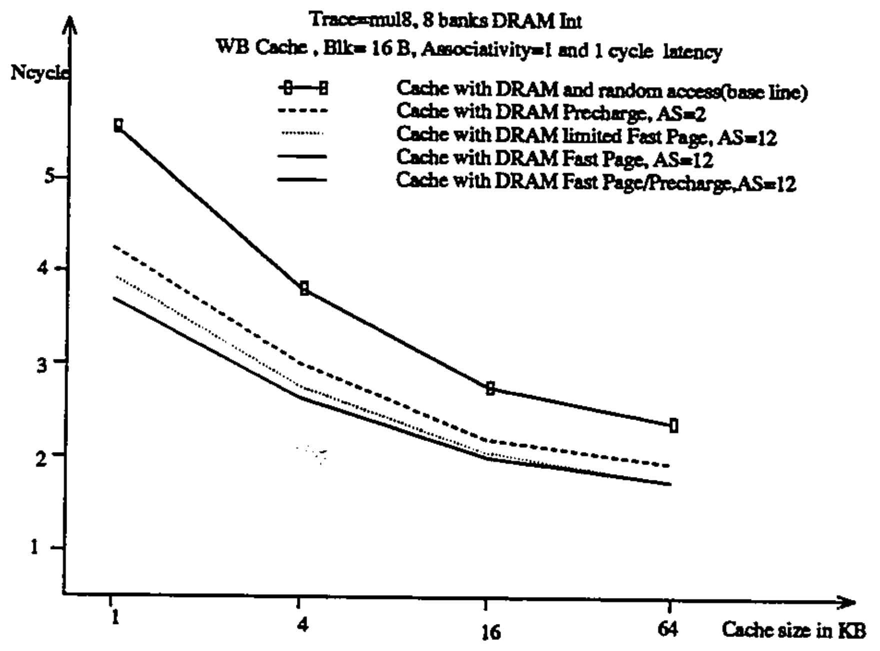


Figure 6.6: The effect of the WB cache size on performance

leaving) by up to 50% for all cache sizes. The write back cache performance increase is 45% for a small cache and 25% for a large cache. This means that it is necessary to use the optimized DRAM bank interleaving for all caches, especially for write through caches and small write back caches.

6.3.5 The effect of block size on the performance of a complete memory system (with single level cache)

Figure 6.8 and figure 6.9 show the performance of the memory system versus the cache block size.

From the results, we find the following:

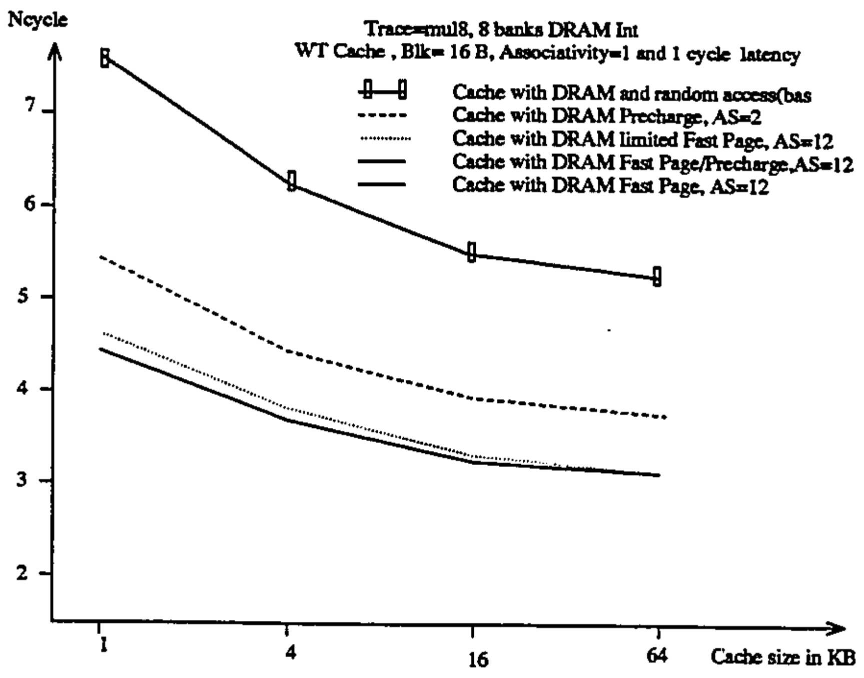


Figure 6.7: The effect of the WT cache size on performance

- The optimal performance for the complete memory system ,with no interleaving, occurs at a block size of 16-32 bytes. The system with write back cache gives its best performance for a block size of 8-16 bytes. The system with write through cache gives its best performance for a block size of 4 bytes. The optimized DRAM bank interleaving causes the system's optimal performance to occur at lower block sizes.

- The shift in the point of optimal performance occurs because it is less probable for large blocks to be in the same row address, causing reduction in the use of fast page accesses, and because the cost of block transfer increases with larger block sizes.

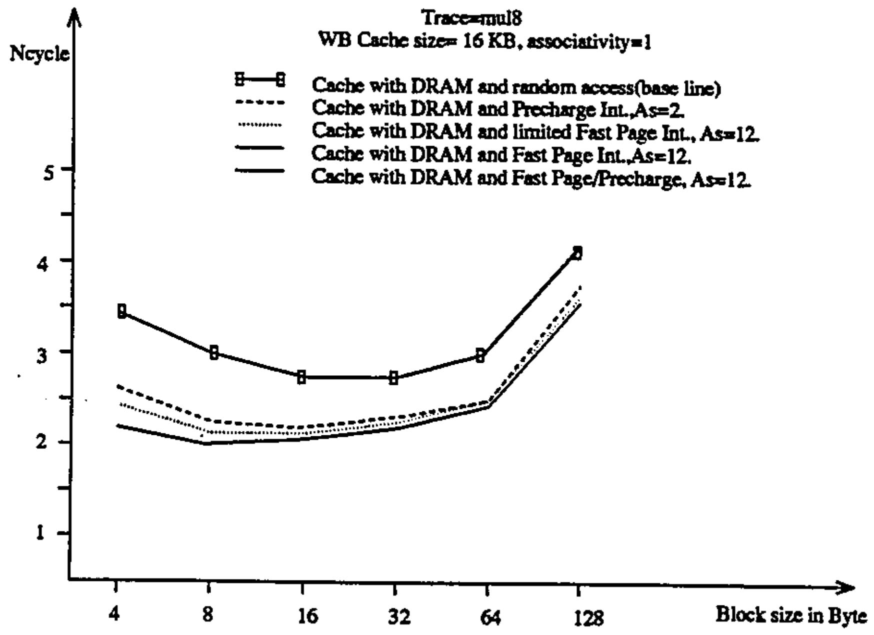


Figure 6.8: The effect of the WB cache block size on performance

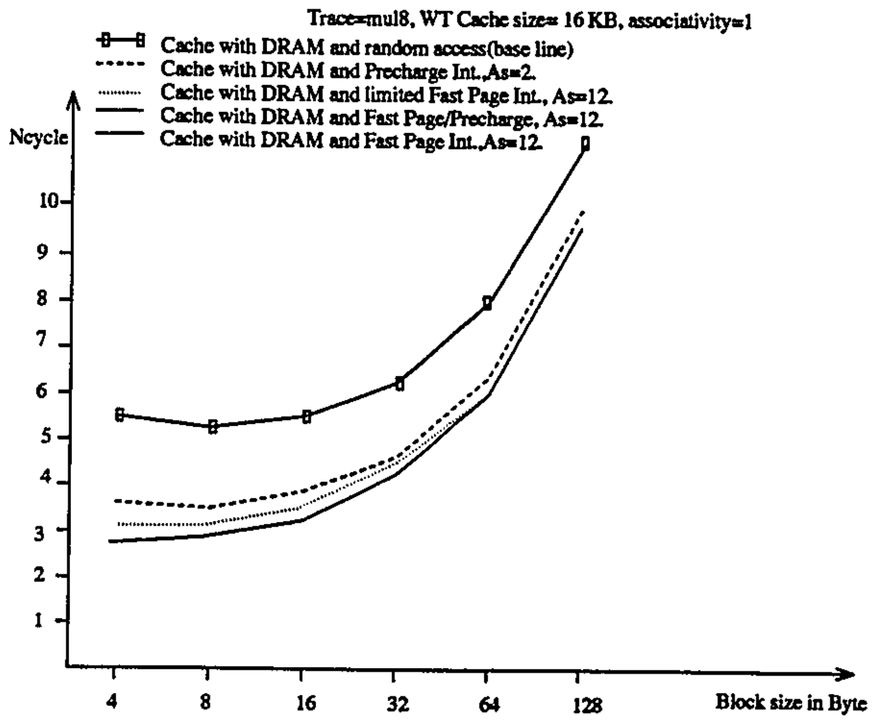


Figure 6.9: The effect of the WT cache block size on performance

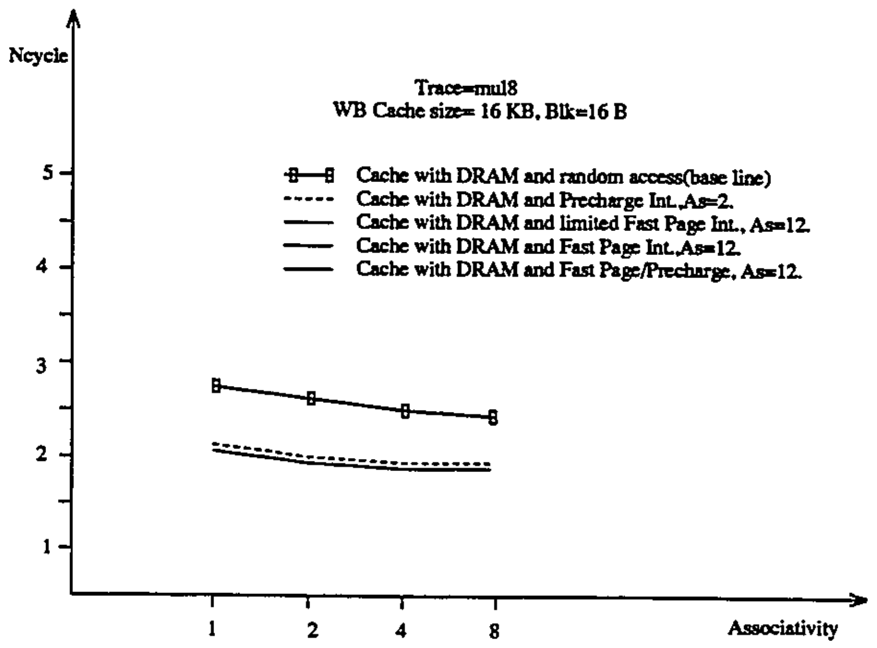


Figure 6.10: The effect of the WB cache associativity on performance

6.3.6 The effect of associativity on the performance of a complete memory system (with single level cache)

Figure 6.10 and figure 6.11 show the performance of a cache system that uses the different DRAM designs versus the associativity.

From the results, we find that the increase in associativity improves the performance of any system only by about 10%. The system gains much more from using the different DRAM protocols [Chapter 3] than from using larger associativity.

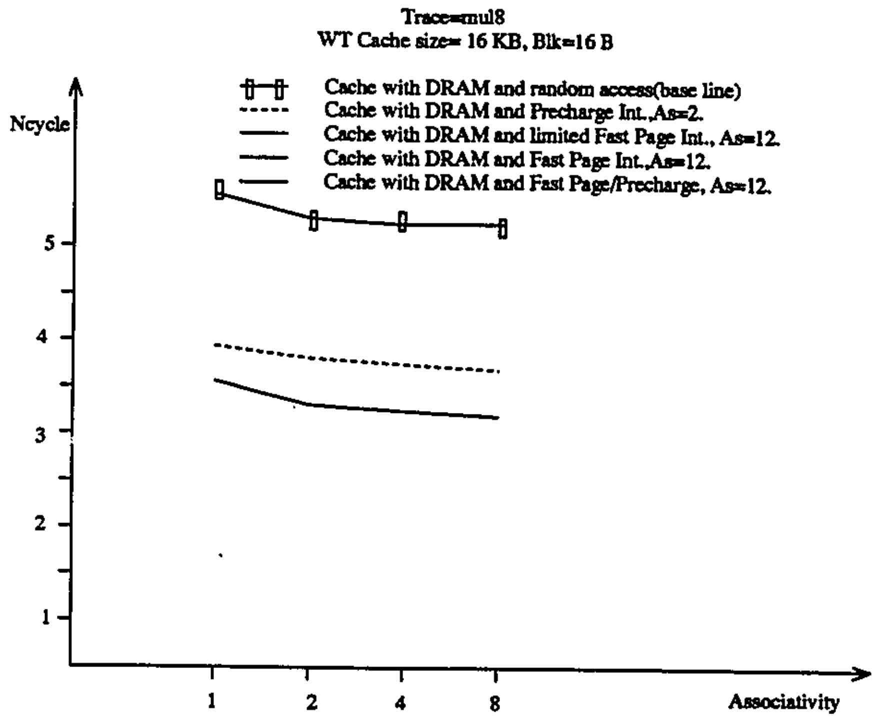


Figure 6.11: The effect of the WT cache associativity on performance

6.4 Optimizing the Performance of a Complete Memory System (with multi-level cache)

6.4.1 Introduction

The results of Chapter 5, suggested that the use of multi-level cache could improve system performance under some conditions.

We limit the discussion of the complete memory system with multi-level cache to the use of the WBWT two level cache [Chapter 5] and the fast page precharge DRAM protocol [Chapter 3]. The reason for this selection is that WBWT gives the best two level cache performance and fast page precharge gives the best main memory performance.

6.4.2 Performance of a complete memory system with two level cache

For this set of experiments, the main memory uses the fast page precharge protocol [Chapter 3], while the cache is of the WBWT multi-level cache design [Chapter 5]. As before, we assume the following:

The DRAM chips are chosen to be 4 M bits with 80 ns access times and support fast page mode access. The total time for random access is 155 ns. We assume that this access takes a total of 16 cycles considering that the cycle is 10 ns. For fast page mode, the access takes 6 cycles and for the precharge access it takes 9 cycles according to the DRAM access timing. The number of banks=8 and the number of MRU banks=4.

We use the WBWT two level cache design with C1 size=16 KB, C2 size= 64 KB, ,Block size=16 B, C1 latency = 1 cycle and C2 latency= 4 cycles. This is similar to most recent processor architectures (ie. DEC Alpha 21064 [34]).

Again, the performance of the system is measured by the number of cycles per instruction (CPI).

6.4.3 Variable address interleaving and the performance of a complete memory system (with two level cache)

Figure 6.12 and Figure 6.13 show the performance of the complete memory system versus the bank select address.

The results show the following:-

- The system with the precharge protocol gives the best performance when banks are mapped in to A2-A4 (which is a word size) and performs badly with high address interleaving . This is the same result obtained in Chapter 3 for the DRAM using precharge without a cache.

- The system with fast page, limited fast page and fast page precharge protocols give the best performance when the banks are mapped into the A12-A14 range. This is a 2 K of bank size and exactly the size of a page for the 4 M bytes DRAM. We conclude that the system gives it's best performance by using banks that are interleaved around a page size, which is the same result as that obtained in Chapter 3 for a system with no cache.

- The system has the best performance when it uses the fast page precharge protocol and the banks are mapped around a page size.

- The fast page precharge protocol can increase the multi-level cache system performance by an extra 25%.

- The increase in system performance for the two level WBWT system is the same for the WB single level system, which means that the optimized address interleaving benefits all types of caches in a similar way. It is important that this memory organization benefits all system configurations rather than one particular hierarchy design.

- Any improvement in the speed or architecture(larger size) of the DRAM chips will be reflected to the performance of the memory system that uses this optimized bank interleaving.

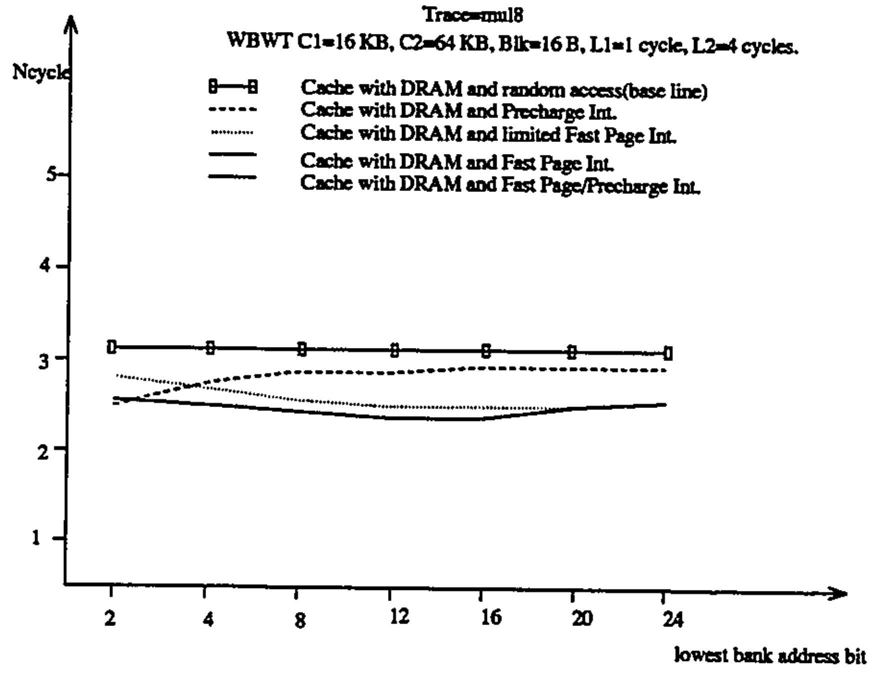


Figure 6.12: The WBWT Variable Address Interleaving for mul8 trace

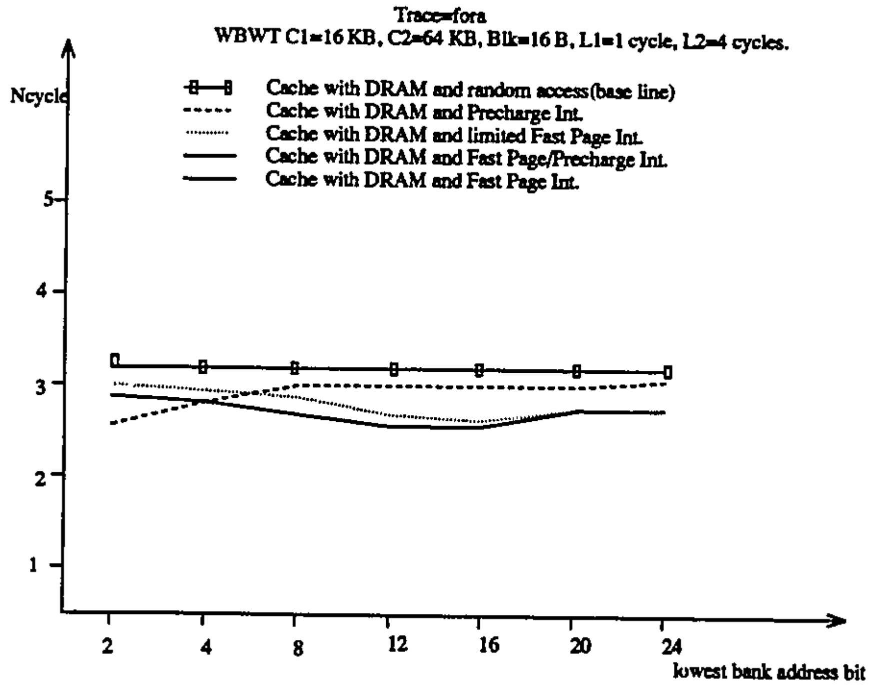


Figure 6.13: The WBWT Variable Address Interleaving for fora trace

6.5 The Effect of Using Different Memory Designs on Overall System Performance

Figure 6.14 shows the effect of the different memory designs on the system performance, when using a WBWT cache or precharge fast page protocol or both (assuming 8 banks mapped around A12-A14).

From the results, we find the following:-

- the memory performance increases by about 260% by using the fast page precharge protocol. This improvement is achieved with minimum hardware requirements.
- The multi-level cache improves memory system performance by 520%.
- The use of fast page precharge protocol can further improve the performance of the memory system with WBWT cache by about 25%.

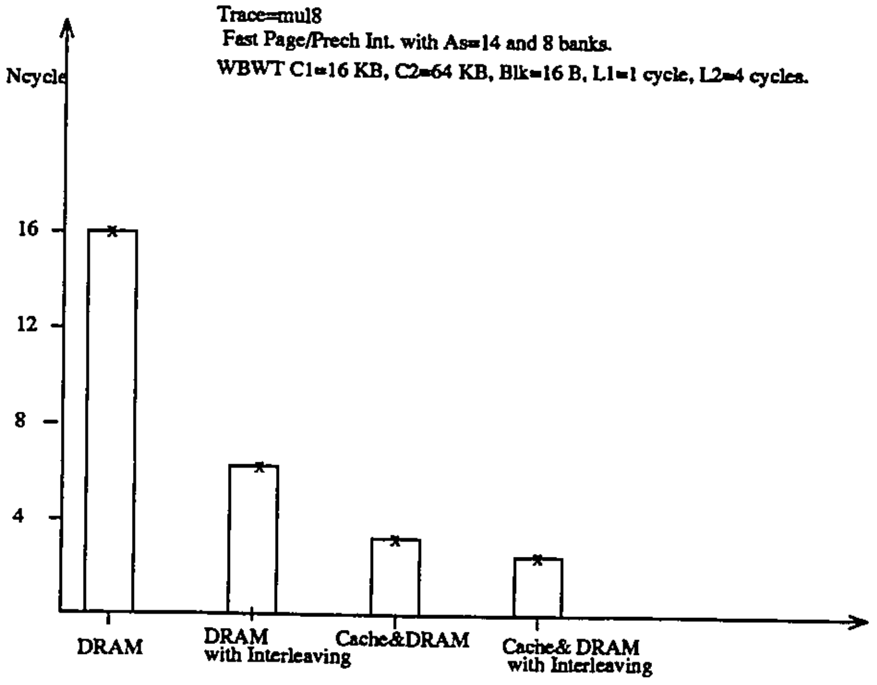


Figure 6.14: The effect of different memory designs on performance

6.6 Summary

In this chapter, we have evaluated the performance of the overall memory hierarchy (including the main memory) using simulation.

The memory systems that we have considered use the 4 multi-bank interleaving protocols (Chapter 3) for the main memory and the single level cache or the multi-level cache for the cache system (Chapter 4 and Chapter 5).

The results show the following:-

- The memory system has the best performance when it uses the fast page precharge protocol and the banks are mapped around a page size.
- The increase in system performance due to the use of the optimized bank interleaving is so profound that the system out performs systems using 4 times larger cache and no bank interleaving.
- The system gains more from using an optimized DRAM design than from using larger associativity.
- A multi-level cache hierarchy can use multi-bank interleaving to considerably increase system performance.

Chapter 7

Summary and Conclusions

7.1 Summary

The performance of fast computers and the scalability of multiprocessor systems depend on the speed of their memory systems. In this research, we present and optimize the design of hierarchical memory systems.

Trace driven simulation is used to evaluate the memory system performance. The traces are selected and qualified against values of typical levels of cache performance for a wide class of work loads using a wide range of computers (Chapter 2).

In all our measurements of system performance, we use the total number of processor clock cycles per instruction. This more accurately reflects machine performance than using the miss rate [37]

We evaluate the design of the main memory and introduce a novel design for optimal main memory performance that combines optimized bank interleaving with different DRAM timing modes, and present the following 4 protocols for main memory systems (Chapter 3) :

- **Fast page** The Fast page protocol combines the DRAM fast page timing mode and bank interleaving (Chapter 3). In this protocol, the maximum performance occurs when banks are mapped around a DRAM page size. A system using the fast page protocol needs a comparator and a latch for each bank, which makes the cost of implementing this protocol the highest of the four protocols.

- **Precharge** The Precharge protocol combines the DRAM precharge timing mode and bank interleaving (Chapter 3). In this protocol, the maximum performance occurs when banks are mapped around a word size. A system using the precharge protocol does not need latches or comparators and has the lowest cost of the four protocols.

- **Precharge Fast page** The Precharge Fast page protocol combines the DRAM precharge timing mode, DRAM fast page timing mode and bank interleaving (Chapter 3). In this protocol, maximum performance occurs when banks are mapped around the DRAM page size. The Precharge Fast page protocol consistently gives the best performance of all protocols. It uses 50% of the number of latches and comparators that the Fast page protocol uses.

- **Limited fast page** The Limited fast page protocol combines DRAM fast page timing and bank interleaving (Chapter 3). In this protocol, maximum performance occurs when banks are mapped around the DRAM page size. The limited fast page protocol gives lower performance than the fast page protocol and the precharge fast page protocol, although it uses the same number of latches and comparators as the precharge fast page protocol.

In chapter 4, we reviewed cache design and evaluated the performance for single level cache designs. The following are the two models for the single level cache (Chapter 4):

- **Write Through (WT)** This model uses the write through method. WT cache design is simple to implement but does not give the best performance.
- **Write Back (WB)** This model uses the write back method. WB cache gives better performance than the WT cache but its implementation is more complicated.

We have discussed multi-level cache designs (Chapter 5). In our novel multi-level cache design, we enforce the inclusion property which simplifies the cache coherency protocols for multiprocessor applications. We allow only one cache level to use the write back method to simplify the design of multi-level cache.

The following models are presented for the two level cache (Chapter 5):

- **Write Through Write Through (WTWT)** WTWT uses write through for the level 1 and the level 2 caches. WTWT design is simple, but gives the lowest performance of the two level cache models.
- **Write Through Write Back (WTWB)** WTWB uses write through for level 1 and write back for the second level cache. The WTWB design is reasonable in complexity and performance. For a very small first level cache, the WTWB could outperform any other model.
- **Write Back Write Through (WBWT)** WBWT uses write back for the level 1 cache and write through for the second level cache. WBWT gives the best cache performance of the three models considered here.

Three level caches are useful for large software applications and large processor to main memory speed differences.

The following models are presented for three level caches (Chapter 5):

- **Write Through Write Through Write Through (WTTT)** WTTT uses write through for all three cache levels. WTTT design is simple, but gives the lowest performance of the three level cache models.

- **Write Through Write Through Write Back (WTTB)** WTTB uses write through for the level 1 and the level 2 caches, and uses write back for the level 3 cache. WTTB design is reasonable in complexity and performance.
- **Write Back Write Through Write Through (WBTT)** WBTT uses write back for the level 1 cache and write through for the level 2 and level 3 caches. WBTT gives the best performance of all 3 level cache models.

For each of these models the effect of each cache parameter on system performance and the parameter for optimal performance have been obtained by simulation.

In chapter 6 we combined the different main memory protocols with the different cache designs to produce complete memory hierarchies, and apply the optimization techniques to the overall memory hierarchy rather than a single level. This differs from previous research work which typically considers only one level of the memory hierarchy at a time.

The effect of different DRAM parameters on the overall system performance have been studied. For a single level memory system (only main memory), the fast page precharge protocol gives the best performance at reasonable cost. For two level memory systems (cache and main memory), the write back cache with the fast page precharge main memory protocol gives the best performance. Finally, for a memory system with a multi-level cache and main memory we have evaluated the performance of the write back write through cache with the fast page precharge main memory protocol.

7.2 Conclusions

The design flow graphs have proved useful in aiding visualization of each memory system designs. This is a new powerful tool that helps in making complicated systems simpler to follow. It also gives us more flexibility to consider different design options.

It is important to use traces for simulation that are qualified against the results of real workloads and is also necessary to measure memory system performance by using the number of clock cycles per instruction and not by using the miss rate.

It is essential to optimize the performance of the main memory for maximum overall system performance. The four novel DRAM protocols give optimal performance because they combine multi-bank interleaving with DRAM access modes. Conventional main memory designs use either of interleaving or the special DRAM access modes to increase system performance. The variable bank mapping helps to find the optimal performance for each protocol. Conventional multi-bank interleaving uses fixed bank mapping and does not give the system it's optimal performance.

In a multi-level cache system, enforcing the inclusion property between different levels helps to reduce the complexity of cache coherency for multiprocessor systems. Limiting the use of write back to one level is also important to simplify the complexity of multi-level cache design. A novel three level cache design is useful to further improve system performance if the processor/memory speed gap is large.

In this research, the performance of the overall memory system is optimized which is of fundamental difference from all other research work that optimizes one level of the memory hierarchy. The memory system that combines the optimized DRAM interleaving protocols and the multi-level cache designs (presented here) is optimal in performance and complexity.

7.3 Recommendations For Further Work

The application of these designs to multiprocessors must be studied thoroughly. In particular, we recommend studying the effect of using this optimized hierarchical memory for multiprocessors, exploring the effect of limiting the write back to a single level on performance and cache coherency, and studying the effect of having low effective memory latency on the throughput and scalability of the multiprocessor system. The application of these designs to multiprocessors must be studied thoroughly.

With the introduction of the new cached DRAM designs, it is important to optimize the performance and find the best protocol for this architecture. Studying the effect of adding a cache system to the cached DRAM memory and optimizing their overall system performance is strongly recommended.

Appendix A

Simulator Program

The following is the Simulator program for the overall memory system which includes main memory interleaving and multi-level cache.

Simulator For WBWT and Precharge Fast Page Protocol

```
/*** WBWT MODEL   DEC.16,1992 modified timing march 19,93                        *:
/** JULY 28,94 MEM SIMULATION WBWT CACHE&DRAM*****/

#include<stdio.h>
#include<stdlib.h>
#define MAXCOUNT 6000000
#define MAX1 66000
#define MAX2 263000

/** CREATE FUNCTION TAKES INPUT PARAMETERS AND CREATES NEW
    FILES THAT HAS THE TAG AND BLK FOR EACH SET AND  THE
    FIRST LINE CONTAINS THE PARAMETERS OF THE CACHE      **/
create(file,size,blk,assoc,wp,wa,delay)
char *file; int size,blk,assoc,delay,wp,wa;
{ unsigned int nset,i,j;  FILE *fp; nset=(size/blk)/assoc;
fp=fopen(file,"w");
fprintf(fp," %u  %u  %u  %u  %u  %u \n",size,blk,assoc,wp,wa,delay);
for(i=0;i<nset;i++){for(j=0;j<assoc;j++)
       {fprintf(fp,"%u %u   ",-1,-1);  }
fprintf(fp,"\n");
}
fclose(fp);
}
/** GET ADDRESS AND LABEL OF NEXT ACCESS FROM TRACE INPUT
    FILE ONLY ONE AT A TIME.IT GETS N-1 CHAR FROM FILE
    OR UNTIL A NEW LINE  CHANGE NEWLINE TO NULL      **/
getaddr(fp,label,addr,tracecount)
FILE *fp; unsigned  *label,*addr, *tracecount;
{int i, j,label1,addr1; char str[255],*fgets();
str[0]=NULL; fgets(str,255,fp);
i=sscanf(str,"%x %x",&label1,&addr1); *addr=addr1; *label=label1;
if(i==2){(*tracecount)++;
         if(*tracecount==MAXCOUNT) {return(EOF);}else
                                                  {return(~EOF);}
}
else {if(i!=EOF){printf("***error in stdi\n");}
```

```
     return(EOF);
}
}


/** LOAD FUNCTION IS TO LOAD CACHE DIRECTORY FROM A FILE THAT
    CROSPONDS TO THE SPECIFIC CACHE TO AN ARRAY FOR THE TAG AND
    THE BLOCK CONDITION,I.E DIRTY,VALID...                  **/
/**CACHE PARAMETERS ARE SIZE,BLOCK,ASS,DELAY,WP,WA AND STORED IN parm[]**/
load(cache,parm,tag,blk)
char *cache; unsigned int *tag,*blk; int parm[6];
{long int i,j,nset,buf,ass; FILE *fp;
fp=fopen(cache,"r");
for(i=0;i<6;i++){
fscanf(fp," %d ",&buf); /** get cache parameters from 1st line to parm[]**/
parm[i]=buf;
}
ass=parm[2]; nset=(parm[0]/parm[1])/parm[2];
    for(i=0; i<nset;i++) {for(j=0;j<ass;j++){
           fscanf(fp, "%u %u   ",(tag+i*ass+j) /**load tag[],blk[]**/
                                                /** from file**/
                              ,(blk+i*ass+j));
    }
    }
fclose(fp);
}

/**   STORE CACHE TAG AND BLOCK CONDITION IN A FILE             **/
store(cache,parm,tag,blk)
char *cache;unsigned int *tag,*blk; int parm[6];
{  int i,j,ass,nset; FILE *fp; ass=parm[2];
nset=(parm[0]/parm[1])/parm[2]; fp=fopen(cache,"w");
fprintf(fp,"%d %d %d %d %d %d\n",parm[0],parm[1],parm[2],parm[3]
                                  ,parm[4],parm[5]);
for(i=0;i<nset;i++){for(j=0;j<ass;j++){
        fprintf(fp,"%u %u  ",*(tag+i*ass+j),*(blk+i*ass+j));
}
             fprintf(fp,"\n");
}
fclose(fp);
```

```
}

 /** PLACE A BLOCK AT MRU WHICH IS POINTED TO BY SSET LOCO         **/
/** LOCO HAS THE MRU BLOCKS AND LOC [ASS-1] HAS THE LRU**/
/**Nbset=total number of sets in cache, bnset=the accessed set number**/
/**tag = is the tag of accessed location                            **/
 plcatmru(parm,tag,blk,address,blkstat)
unsigned int *tag,*blk,*address;  int parm[6],*blkstat;
 {
long int Nbset,bnset,tag1,  i,j,ass,temp;  ass=parm[2];
 Nbset=(parm[0]/parm[1])/parm[2]; bnset=(*address/parm[1])%Nbset;
 tag1=(*address/parm[1])/Nbset;
  *(tag+bnset*ass+0)=tag1;    /* store tag in MRU              */
  *(blk+bnset*ass+0)=*blkstat; /* blk stat in MRU               */
 }

 /** MAKE THE USED BLOCK A MRU AND UPDATED STACK TO REFLECT
     THE LRU. THIS MEAN THAT THE USED WILL BE POINTED TO BY SSET LOCO
     AS MRU AND THE RESET WILL BE PUSHED ONE STEP.IF BLOCK NOT FOUND
     THE LRU BLOCK IS MOVED                                          **/
 maktmru(parm,tag,blk,address)
 unsigned int *tag,*blk,*address;  int parm[6];
 {
int Nbset,bnset,tag1,   i,j,ass,temp1,temp2,hit;
 ass=parm[2];  Nbset=(parm[0]/parm[1])/parm[2];
 bnset=(*address/parm[1])%Nbset;  tag1=(*address/parm[1])/Nbset;
 hit=0;
 for(i=0;i<ass;i++){ if(*(tag+bnset*ass+i)==tag1){hit=1;
/** FIND IF BLOCK EXISTS**/
                                                 break;
 }
 }
 /* if it is a miss make the LRU  the MRU and update the
    rest, remove the LRU                                 */
 if(hit==0){
                              /*make LRU a MRU swap them*/
                temp1= *(tag+bnset*ass+ass-1);
                temp2= *(blk+bnset*ass+ass-1);
            for(j=ass-1;j>0;j--){
                              *(tag+bnset*ass+j)=
                              *(tag+bnset*ass+j-1);
                              *(blk+bnset*ass+j)=
                              *(blk+bnset*ass+j-1);
   }
                              *(tag+bnset*ass+0)=temp1;
                              *(blk+bnset*ass+0)=temp2;
 }

 /* if it is a hit then make the block MRU  and update the
    rest,                                              */
 if(hit==1) {
                temp1= *(tag+bnset*ass+i);
                temp2= *(blk+bnset*ass+i);
                   for(j=i;j>0;j--){
                                   *(tag+bnset*ass+j)=
                                   *(tag+bnset*ass+j-1);
                                   *(blk+bnset*ass+j)=
                                   *(blk+bnset*ass+j-1);
    }
                             *(tag+bnset*ass+0)=temp1;
                             *(blk+bnset*ass+0)=temp2;
           }
 }

/** MISS FUNCTION IS USED TO FIND MISS OR A HIT. IT RETURNS
    0 IN A MISS AND 1 IN A HIT AND ALSO THE BLOCK STATUS      **/
 miss(parm,tag,blk,address,blkstat,hit)
unsigned int *tag,*blk,*address; int parm[6],*blkstat,*hit;
{
unsigned int Nbset,bnset,tag1; int i,j,ass,temp;
ass=parm[2];Nbset=(parm[0]/parm[1])/parm[2];
bnset=(*address/parm[1])%Nbset; tag1=(*address/parm[1])/Nbset;
*hit=0;                      /* intialize hit                */
for(i=0;i<ass;i++){
               if(tag1==*(tag+bnset*ass+i)){ *hit=1;
                                             *blkstat=
                                             *(blk+bnset*ass+i);
                                            break;
  }
}
```

```
}

/** getblk address from MRU        **/
getblkaddr(parm,tag,address,blkaddr)
unsigned int *tag, *address,*blkaddr;
int parm[6];
{
unsigned int Nbset,bnset,blk1;
Nbset=(parm[0]/parm[1])/parm[2];
bnset=(*address/parm[1])%Nbset;
blk1=bnset*parm[1]+(*(tag+bnset*parm[2]))*Nbset*parm[1];
*blkaddr=blk1;
}

/**** DRAM functions***********/
/** MAKE THE USED RAS THE MRU  ***/
/** Stack has dim=Nb and Stack[0]=the MRU bank, Stack[Nb-1]=LRU bank**/
mru(Cbank,Stack,Nb)
int *Cbank, *Stack, *Nb;
{
int i,j,k,k1,temp,temp1,m; m=0;
for(i=0;i<*(Nb);i++){if(*(Stack+i)==*(Cbank)){ m=1; break;}
}
/** if the bank was not used before, make it MRU**/
if(m==0){ for(j=*(Nb)-1;j>0;j--){*(Stack+j)=*(Stack+j-1);}
          *(Stack)=*(Cbank);
}
/**if the bank has been accessed before, it becomes the MRU **/
if(m==1){for(k=i;k>0;k--){*(Stack+k)=*(Stack+k-1);}
        *(Stack)=*(Cbank);
}
}

/** PRECHARGE THE LRU RASs and make MRU RAS active**/
/** ras[] array of dim=Nb and store the value of each bank ras=1
for precharge and 0 for active. after an access the ras[Cbank]
is made=0 and the LRU RASs are made=1  ***/
rasonoff(Cbank,RAS,rasoff,Stack,Nb)
 int *Cbank,*RAS, *rasoff, *Stack,*Nb;
{
```

```
int i,i1,i2,i3,temp; *(RAS+*(Cbank))=0;
i1=*(Nb);  i2=i1-*(rasoff);
for(i=i2;i<i1;i++){ temp=*(Stack+i); if(temp<i1)    {*(RAS+temp)=1;}
}
}

/** CALCULATE THE ACCESSED BANK NUMBER AND ROW NUMBER ***/
/** Acl=size of columns,    Abyt=number of  byte
    Abnk=number of  banks,    Ast=bank select start address **/
rwbnk(Nb,W,Z,address,As,Cbank,row)
int *Nb,*W,*Z,*As,*Cbank,*row; unsigned int *address;
{
unsigned int i,j,Acl,Abyt,Abnk,row1,row2,At,row1t,row2t;
unsigned int A,Ast;
Acl=*(Z); Abyt=*(W); Abnk=*(Nb); Ast=*(As);
A=*(address)>>Ast;
*(Cbank)=A%Abnk; /** CURRENT ACCESSED BANK  ***/
 /**banks are within column address**/
if((1<<Ast)<=Acl*Abyt){*(row)=*(address)/(Acl*Abyt*Abnk);}
 /** banks are outside column address and within row address**/
if((1<<Ast)>Acl*Abyt){row1t=*(address)/(Acl*Abyt);
                    row1=row1t%((1<<Ast)/(Acl*Abyt));
 row2t=(*(address)/(1<<Ast));  row2t=row2t/Abnk;
   row2=row2t*((1<<Ast)/Acl*Abyt);       *(row)=row1+row2;
}
}


/*** WBWT MODEL.  DEC.16,1992 modified timing march 19,93      ***/
/** JULY 28,94 MEM SIMULATION WBWT CACHE&DRAM*****/
main()
{
unsigned int Nset, NCYCLE ,ADDRESS,BLKADDR,BLKADDR1,
temp ,row,mem,mem1,*rowaddr ;
float I1MISS,I2MISS,D1MISS,D2MISS,NI,NR,NW,pd,npd,TA,NCYCLEI,NCYCLED,tc;
FILE *strm, *fp1;
int SCI1,BCI1,ACI1,PCI1,SCI2,BCI2,ACI2,PCI2,SIM,PIM;
unsigned int *a1,*b1,*a2,*b2;
int SDC1,BDC1,ADC1,WDC1,WADC1,PDC1,SDC2,BDC2,ADC2,WDC2,WADC2;
int PDC2,SDM,PDM,BLKST1,BLKST2,HIT1,HIT2,LABEL,end,k2,k3,i,i1,i2,i3;
```

```
unsigned int thelabel,theaddr,n,c1[6],c2[6],tracecount;
int lastbank,cbank,*stack,*ras,z,w,nb,as,RASOFF,Trm,Tfp,Tpr;
for(i2=1;i2<3;i2++)
{
 ACI1=1; PCI1=1;  ACI2=1; PCI2=4;  PDC2=4;
 ADC1=1; PDC1=1; ADC2=1;  WADC1=1; WADC2=1;  WDC1=1; WDC1=1;
PIM=16; PDM=16; BCI1=16; BCI2=16; BDC1=16; BDC2=16;
SDC2=65536; SCI2=65536; SCI1=16384;  SDC1=16384;
/*** DRAM PARAMETERS***/
z=2048; nb=8;  w=4;  Trm=16-1+BCI1/4; Tfp=6-1+BCI1/4;
Tpr=9-1+BCI1/4; RASOFF=nb/2;
for(i3=0;i3<7;i3++) {if(i3==0){as=2;}else
                              {as=4*i3;}
create("icach1.fp",SCI1,BCI1,ACI1,PCI1,0,0);
create("icach2.fp",SCI2,BCI2,ACI2,PCI2,0,0);
tracecount=0; NCYCLE=0; pd=0;  npd=0;
NI=0;NR=0;NW=0;I1MISS=0;I2MISS=0;D1MISS=0;D2MISS=0;
a1=(unsigned int*)malloc(SCI1); b1=(unsigned int*)malloc(SCI1);
a2=(unsigned int*)malloc(SCI2); b2=(unsigned int*)malloc(SCI2);
rowaddr=(unsigned int*)malloc(nb); stack=(int*)malloc(nb*sizeof *stack);
ras=(int*)malloc(nb*sizeof *ras);
if(i2==1){strm=fopen("/home/eccles/nmekhiel/traces/atum/fora.000.din", "r");
        }
if(i2==2){strm=fopen("/home/eccles/nmekhiel/traces/atum/mul8.003.din", "r");
       }
if(i2==3){strm=fopen("/user/nmekhiel/stanfd/traces/benchmarks/cc1.din","r");
       }
load("icach1.fp",c1,a1,b1); load("icach2.fp",c2,a2,b2);
PIM=PIM-1+BCI1/4; c2[3]=c2[3]-1+BCI2/4;
for(i=0;;i++)
    {
end=getaddr(strm,&thelabel,&theaddr,&tracecount);
if(end==EOF){break;}
LABEL=thelabel; ADDRESS=theaddr;
if(LABEL==2){ NI=NI+1;
miss(c1,a1,b1,&ADDRESS,&BLKST1,&HIT1);
if(HIT1==0){I1MISS=I1MISS+1;
            miss(c2,a2,b2,&ADDRESS,&BLKST2,&HIT2);
            if(HIT2==0){I2MISS=I2MISS+1;}
            }
 if(HIT1==1){maktmru(c1,a1,b1,&ADDRESS);
            plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
            NCYCLE=NCYCLE+c1[3];
           }else{if(HIT2==1){maktmru(c1,a1,b1,&ADDRESS);
                             plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                             maktmru(c2,a2,b2,&ADDRESS);
                             plcatmru(c2,a2,b2,&ADDRESS,&BLKST2);
                             NCYCLE=NCYCLE+c2[3]+c1[3];
                             }else{maktmru(c1,a1,b1,&ADDRESS);
                                   plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                                   maktmru(c2,a2,b2,&ADDRESS);
                                   plcatmru(c2,a2,b2,&ADDRESS,&BLKST2);
                                   NCYCLE=NCYCLE+2*c2[3];
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;   mru(&cbank,stack,&nb);
                    *(rowaddr+cbank)=row;
                   rasonoff(&cbank,ras,&RASOFF,stack,&nb);
}
                     else{
                          if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                     mru(&cbank,stack,&nb);
                          }else {
                                mru(&cbank,stack,&nb); *(rowaddr+cbank)=row;
                                rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                                NCYCLE=NCYCLE+Trm;
                               }
                     }

                                  }
                 }
           }
    }
I2MISS=((I2MISS*100)/I1MISS); I1MISS=((I1MISS*100)/NI);
k3=NI; NCYCLEI=NCYCLE;  TA=i;
fp1=fopen("21mem.fp.res","a");
fprintf(fp1," WBWTS1 FORA TA=%d   #Inst.=%d  M1=%f M2=%f NCL=%u \n"
,i,k3,I1MISS,I2MISS,NCYCLE);
fprintf(fp1," the precharge  for trace#%d loc%d  \n",i2,as); fclose(fp1);
fclose(strm); create("dcach1.fp",SDC1,BDC1,ADC1,WDC1,WADC1,PDC1);
create("dcach2.fp",SDC2,BDC2,ADC2,WDC2,WADC2,PDC2);  tracecount=0;
```

```
if(i2==1){strm=fopen("/home/eccles/nmekhiel/traces/atum/fora.000.din", "r");
         }
if(i2==2){strm=fopen("/home/eccles/nmekhiel/traces/atum/mul8.003.din", "r");
          }
if(i2==3){strm=fopen("/user/nmekhiel/stanfd/traces/benchmarks/cc1.din","r");
         }
load("dcach1.fp",c1,a1,b1); load("dcach2.fp",c2,a2,b2);
PDM=PDM-1+BDC1/4; c2[5]=c2[5]-1+BDC2/4;
k2=0; k3=0; NCYCLE=0; NR=0; NW=0; D1MISS=0; D2MISS=0;
for(i=0;;i++)
 {
end=getaddr(strm,&thelabel,&theaddr,&tracecount);
if(end==EOF){break;} LABEL=thelabel; ADDRESS=theaddr;
if(LABEL==0){NR=NR+1;
             miss(c1,a1,b1,&ADDRESS,&BLKST1,&HIT1);
             if(BLKST1==1){pd=pd+1;}else{npd=npd+1;}
             if(HIT1==0){D1MISS=D1MISS+1;
                         miss(c2,a2,b2,&ADDRESS,&BLKST2,&HIT2);
                         if(HIT2==0){D2MISS=D2MISS+1;}
                         }
             if(HIT1==1){maktmru(c1,a1,b1,&ADDRESS);
                         plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                         NCYCLE=NCYCLE+c1[5];
                        }else{if(HIT2==1){
                                          if(BLKST1==1){NCYCLE=NCYCLE
                                          +c1[5];
                                  maktmru(c1,a1,b1,&ADDRESS);
                              getblkaddr(c1,a1,&ADDRESS,&BLKADDR);
temp=ADDRESS; ADDRESS=BLKADDR;
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;  mru(&cbank,stack,&nb);
                     *(rowaddr+cbank)=row;
                    rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                     }
                      else{
                           if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                     mru(&cbank,stack,&nb);
                           }else    { mru(&cbank,stack,&nb);
                                      *(rowaddr+cbank)=row;
                                      rasonoff(&cbank,ras,&RASOFF,stack,&nb);
```

```
                                        NCYCLE=NCYCLE+Trm;
                                       }
                          }
      BLKST1=0; ADDRESS=temp;
  }
                                                   maktmru(c1,a1,b1,&ADDRESS);
                                                   plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                                                   maktmru(c2,a2,b2,&ADDRESS);
                                                   plcatmru(c2,a2,b2,&ADDRESS,&BLKST2);
                                                   NCYCLE=NCYCLE+c2[5]+c1[5];
                                                  }else{
                                                   if(BLKST1==1)  {NCYCLE=NCYCLE
                                                                   +c1[5]-c2[5];
                                                                  BLKST1=0;
                                                         maktmru(c1,a1,b1,&ADDRESS);
                                      getblkaddr(c1,a1,&ADDRESS,&BLKADDR);
temp=ADDRESS; ADDRESS=BLKADDR;
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;   mru(&cbank,stack,&nb);
                    *(rowaddr+cbank)=row;
                   rasonoff(&cbank,ras,&RASOFF,stack,&nb);
}
                     else{
                          if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                   mru(&cbank,stack,&nb);
                          }else     {
                                     mru(&cbank,stack,&nb);
                                     *(rowaddr+cbank)=row;
                                     rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                                     NCYCLE=NCYCLE+Trm;
                                    }
                     } ADDRESS=temp;

                                                       }
                                                        maktmru(c1,a1,b1,&ADDRESS);
                                                         plcatmru(c1,a1,b1
                                                         ,&ADDRESS,&BLKST1);
                                                         maktmru(c2,a2,b2,&ADDRESS);
                                                         plcatmru(c2,a2,b2
                                                         ,&ADDRESS,&BLKST2);
```

```
                                                NCYCLE=NCYCLE+2*c2[5];
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;
                    mru(&cbank,stack,&nb);
                     *(rowaddr+cbank)=row;
                    rasonoff(&cbank,ras,&RASOFF,stack,&nb);
}
                      else{
                           if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                        mru(&cbank,stack,&nb);
                           }else     {
                                      mru(&cbank,stack,&nb);
                                      *(rowaddr+cbank)=row;
                                      rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                                      NCYCLE=NCYCLE+Trm;
                                     }
                      }
                                                    }

                                      }
             }
 if(LABEL==1){NW=NW+1;
              miss(c1,a1,b1,&ADDRESS,&BLKST1,&HIT1);
              if(BLKST1==1){pd=pd+1;}else{npd=npd+1;}
              if(HIT1==0){D1MISS=D1MISS+1;
                          miss(c2,a2,b2,&ADDRESS,&BLKST2,&HIT2);
                          if(HIT2==0){D2MISS=D2MISS+1;}
                         }
              if(HIT1==1)
                   {maktmru(c1,a1,b1,&ADDRESS);
                    plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                    NCYCLE=NCYCLE+2*c1[5];
                    BLKST1=1;
                    plcatmru(c1,a1,b1,&ADDRESS,&BLKST1);
                   }else {if(HIT2==1)
                            {maktmru(c2,a2,b2,&ADDRESS);
                             plcatmru(c2,a2,b2,&ADDRESS,&BLKST2);
                             NCYCLE=NCYCLE+c1[5];
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;
```

```
                        mru(&cbank,stack,&nb);
                         *(rowaddr+cbank)=row;
                        rasonoff(&cbank,ras,&RASOFF,stack,&nb);
}
                          else{
                                if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                           mru(&cbank,stack,&nb);
                                }else     {
                                           mru(&cbank,stack,&nb);
                                           *(rowaddr+cbank)=row;
                                           rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                                           NCYCLE=NCYCLE+Trm;
                                          }
                          }
                                  }else{NCYCLE=NCYCLE+c1[5];
rwbnk(&nb,&w,&z,&ADDRESS,&as,&cbank,&row);
if(*(ras+cbank)==1){NCYCLE=NCYCLE+Tpr;
                        mru(&cbank,stack,&nb);
                         *(rowaddr+cbank)=row;
                        rasonoff(&cbank,ras,&RASOFF,stack,&nb);
}
                          else{
                                if(*(rowaddr+cbank)==row){NCYCLE=NCYCLE+Tfp;
                                                           mru(&cbank,stack,&nb);
                                }else     {
                                           mru(&cbank,stack,&nb);
                                           *(rowaddr+cbank)=row;
                                           rasonoff(&cbank,ras,&RASOFF,stack,&nb);
                                           NCYCLE=NCYCLE+Trm;
                                          }
                          }

       }

    }
                    }
 }
D2MISS=((D2MISS*100)/D1MISS); D1MISS=((D1MISS*100)/(NR+NW));
k3=NR+NW; k2=NR;  NCYCLED=NCYCLE; tc=(NCYCLEI+NCYCLED)/TA;
fp1=fopen("2lmem.fp.res","a");
```

```
fprintf(fp1,"ta%f SCI%d BCI1%d ACI1%d PCI1%d PIM%d SCI2%d PCI2%d perf%f \n"
,TA,SCI1, BCI1,ACI1,PCI1,PIM,SCI2,PCI2,tc);
fprintf(fp1,"Data=%d   R=%d   M1=%f M2=%f NC=%u \n\n"
,k3,k2,D1MISS,D2MISS,NCYCLE);
remove("icach1.fp"); remove("icach2.fp");
remove("dcach1.fp"); remove("dcach2.fp");
fclose(fp1);  fclose(strm);
free(a1); free(b1); free(a2); free(b2); free(ras); free(stack);
free(rowaddr);
} } }
```

REFERENCES

[1] A. Agarwal, R.L. Sites, and M. Horowitz, "ATUM: a new technique for capturing address traces using microcode", *Proc. 13th Int. Symp. Comput. Architecture*, pp. 119-129, June 1986.

[2] Agarwal, A. " Analysis of Cache Performance for Operating Systems and Multiprogramming", Norwell, MA:Kluwer Academic Publishers, 1989

[3] A. Agarwal, M. Horowitz, J. Hennessy, "An analytical cache model", *ACM Trans. Computer Systems* , vol. 7, pp. 184-215, May 1989.

[4] Anant Agarwal and Steven D. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches", *Intl. Symposium on Computer Architecture*, pp. 179-190, 1993.

[5] R.C. Alford, "Memory caching schemes", *Technical Science*, pp. 87-92, June 1989.

[6] Donald B. Alpert, Michael J. Flynn, "Performance Trade-offs for Microprocessor Cache Memories", *IEEE MICRO* , pp. 44-53, April 1988.

[7] J.L Bear and W. H. Wang, "On the inclusion properties for multi-level cache hierarchies", *Proc. 15th Annual Symp. Computer Architecture*, Honolulu, HI , pp. 73-80, June 1988.

[8] Keith Boland and Apostolos Dollas, "Predicting and Precluding Problems with Memory Latency", *IEEE Micro*, pp. 59-67, August 1994.

[9] Donald Y. Chang, David J. Kuck and Duncan H. Lawrie, "ON the Effective Bandwidth of Parallel Memories", *IEEE Trans., Computers*, pp. 480-489, May 1977.

[10] T.F. Chen and J.L. Baer, "Reducing memory latency via non-blocking and prefetching caches", *Technical Report 92-06-03, Department of Computer Science*, University of Washington, Seattle WA, June 1992.

[11] Tony Cheung and Jmes E. Smith, "A Simulation Study of the Cray X-MP Memory System", *IEEE Trans., Computers*, vol. c-35, pp. 613-622, July 1986.

[12] Chiang M.C., Sohi G.S. " Evaluating Design Choices for shared bus Multiprocessors in a Throughput- oriented Environment", *IEEE Transaction on Computers*, Vol.41, pp.297-317, March 1992.

[13] Child Jeff, " DRAM Vendors juggle with new architectures to increase performance", *Computer Design* , pp. 71-86, March 1995.

[14] Kifung Chung, Gurindar Sohi, Kewal Saluja, "Organization and Analysis of a Gracefully-Degrading Interleaved Memory System", *14th Intl. Symp. on Computer Architecture* pp. 224-231, 1987.

[15] Fujitsu Microelectronics, Inc. "MOS Memory Products", *Data Book*, 1992.

[16] Jeffrey D. Gee, Mark D. Hill and Dionisios N. Pnevmatikatos, "Cache Performance of the SPEC92 Benchmark Suite", *IEEE Micro*, pp. 17-27, August 1993.

[17] V. Hamacher, Z. Vranesic, S. Zaky " Computer Organization", *McGraw-Hill, Inc.*, 1990.

[18] J. Hennessy, D.A. Patterson " Software Distribution for Computer Architecture: A Quantitative Approach", *Stanford University* , California,August, 1990.

[19] J. Hennessy, D.A. Patterson "Computer Architecture: A Quantitative Approach", *Morgan Kaufmann, Palo Alto, California*, 1990.

[20] M.D. Hill, A.J. Smith, " Experimental evaluation of on- chip microprocessor cache memories", *Proc. 11th Intl. Symp. on Comp. Arch.*, New York, pp. 158-166, June 1984.

[21] Mark D. Hill, "A Case for Direct-Mapped Caches", *IEEE COMPUTER*, pp. 25-40, December 1988.

[22] Mark D. Hill AND Alan Jay Smith, "Evaluating Associativity in CPU Caches", *IEEE Transaction on Computers*, , pp. 1612-1629, December 1989.

[23] M.D. Hill, "The Dinero simulator", *internal report*, Computer Science Dept., Univ. of Wisconson at Madison, 1989.

[24] Subhasis Laha, Janak H. Patel AND Ravishankar K. Iyer, "Accurate Low-Cost Methods for Performance Evaluation of Cache Memory System", *IEEE Transaction on Computers*, pp. 1325-1335, November 1988.

[25] Leutenegger S.T., Vernon M.K. "A Mean Value Performance Analysis of a New Multiprocessor Architectures", *Proc. ACM SIGMETRICS*, conf. Measurement and Modelling of Compt. Syst., May 1988.

[26] Jacques Lienfant, "Fast Random and Sequential Access to Dynamic Memories of Any Size", *IEEE Trans., Comp.* Vol. 26, pp. 847-855, September 1977.

[27] Richard E. Matick, "Functional cache chip for improved system performance", *IBM RES DEVELOP*, Vol.33, pp. 15-31, January 1989.

[28] N. Mekhiel, "Speed System Memory By Interleaving DRAM Accesses", *Electronic Design*, pp. 65-72, October 12, 1989.

[29] N. Mekhiel, "DRAM control scheme speeds memory access", *EDN*, pp. 166-168, March 29, 1990.

[30] Mekhiel N., "Easily Upgrade a 68030-Based System With A Clever Cache Design", *Electronic Design*, pp. 59-66, March 14, 1991.

[31] N. Mekhiel, D. McCrackin "Performance analysis of a two level cache system", *26th Asilomar conference on Signals, Systems and Computers* pp.71-77, Oct. 26-28, 1992.

[32] N. Mekhiel and D. McCrackin, "Methods for Improving Main Memory Performance", *ISCA Computer Applications In Industry and Engineering International Conference*, Honolulu, Hawaii, pp 120-125, Dec. 15-17 1993.

[33] N. Mekhiel and D.C. McCrackin, "Performance Modelling of Hierarchical Memory Systems", *1994 Canadian Conference on Electrical and Computer Engineering*, pp.612-616, September 25-28 1994, Halifax Canada.

[34] Stephen Ohr, "Fast cache designs keep the RISC monster fed", *Computer Design* , pp. 67-91, January 1995.

[35] A.V. Pohm, "High-Speed Memory Systems", *IEEE Computer*, pp. 162-171, October 1984.

[36] Steven Przybylski, Mark Horowitz, John Hennessy, "Performance Tradeoffs in Cache Design", *15th Annual Symposium on Computer Architecture*, pp. 290-298, June 1988.

[37] Steven Przybylski, Mark Horowitz, John Hennessy,

"Characteristics of Performance-Optimal Multi-Level Cache Hierarchies", *16th Annual Symposium on Computer Architecture*, pp. 114-121, 1989.

[38] R. Raghavan and J.P. Hayes, "On randomly interleaved memories", *Proc. Supercompu. '90*, pp. 49-58, Nov. 1990.

[39] A. Raghavan and J.P. Hayes, "Reducing Interference Among Vector Accesses interleaved Memories", *IEEE Trans., Computers*, vol. 42, pp. 471-483, April 1993.

[40] B. Ramakrishna Rau, "Program Behavior and the Performance of Interleaved Memories", *IEEE Trans., Comp.* Vol. c- 28, pp. 191-199, March 1979.

[41] C.V. Ramamoorthy, and Benjamin W. Wah, "An Optimal Algorithm for Scheduling Requests on Interleaved Memories for a Pipelined Processor", *IEEE Trans., Comp.* Vol. c- 30, pp. 787-799, October 1981.

[42] Andre Seznec " A case for two-way skewed-associative caches", *Intl. Symposium on Computer Architecture*, pp. 169-178, 1993.

[43] R.T. Short, H.M. Levy "A Simulation study of two-level caches", *15th Annual Symposium on Computer Architecture*,pp. 81-88, 1988.

[44] A.J.Smith, "Cache Evaluation and the Impact of Work Load Choice", *Proc. 12th int'l Symp. Computer Architecture*, IEEE CS Press, pp. 64-75, 1985.

[45] A.J. Smith, "Line size choice for CPU cache memories", *IEEE Trans. Computers*, vol. C-36, pp. 1063-1074, September 1987.

[46] Gurindar S. Sohi, "Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors", *IEEE Transactions On Computers*, Vol 38, pp. 484-492, April 1989.

[47] R. Witek and D. Sites, "Alpha architecture and EV4 – the first implementation", In *Compcon Spring 92*, San Francisco, 1992.