ANSWERS TO QUESTIONS

CHAPTER 2 PERFORMANCE ISSUES

- Pipelining: The execution of an instruction involves multiple stages of operation, including fetching the instruction, decoding the opcode, fetching operands, performing a calculation, and so on.
- Branch prediction: The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next.
- Superscalar execution: This is the ability to issue more than one instruction in every processor clock cycle.
- Data flow analysis: The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions.
- Speculative execution: Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations.

Performance balance is the adjustment/tuning of the organization and architecture to compensate for the mismatch among the capabilities of the various components.

Multicore: the use of multiple processing units, called cores, on a single chip.

Many integrated core (MIC): a chip containing a large number (50 or more) cores.

General-purpose computing on GPUs (GPGPU): A GPU designed to support a broad range of applications.

Amdahl's law deals with the potential speedup of a program using multiple processors compared to a single processor. The law indicates the amount of speedup as a function of the fraction of code that can be executed in parallel.

Little's law applies to queuing systems. It defines the relationship between arrival rate, average time spent in system, and the average number of items in the system.

MIPS = millions of instruction executions per second. FLOPS = floating-point operations per second.

Arithmetic mean: sum of a set of values divided by the number of values.
Geometric mean: the nth root of the product of a set of n values.
Harmonic mean: for n values, n divided by the product of the reciprocal of the n values.

1. It is written in a high-level language, making it portable across different machines.
2. It is representative of a particular kind of programming domain or paradigm, such as systems programming, numerical programming, or commercial programming.
3. It can be measured easily.
4. It has wide distribution.

This is a collection of benchmark suites is defined and maintained by the Standard Performance Evaluation Corporation (SPEC).

- Base metric: These are required for all reported results and have strict guidelines for compilation. In essence, the standard compiler with more or less default settings should be used on each system under test to achieve comparable results.
- Peak metric: This enables users to attempt to optimize system performance by optimizing the compiler output.
- Speed metric: This is simply a measurement of the time it takes to execute a compiled benchmark.
- Rate metric: This is a measurement of how many tasks a computer can accomplish in a certain amount of time; this is called a throughput, capacity or rate measure.

Answers to Problems

CPI = 1.55; MIPS rate = 25.8; Execution time = 3.87 ms.

a.

$$CPI_A = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(8 \times 1 + 4 \times 3 + 2 \times 4 + 4 \times 3) \times 10^6}{(8 + 4 + 2 + 4) \times 10^6} \approx 2.22$$

$$MIPS_A = \frac{f}{CPI_A \times 10^6} = \frac{200 \times 10^6}{2.22 \times 10^6} = 90$$

$$CPU_A = \frac{I_c \times CPI_A}{f} = \frac{18 \times 10^6 \times 2.2}{200 \times 10^6} = 0.2 \text{ s}$$

$$CPI_B = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(10 \times 1 + 8 \times 2 + 2 \times 4 + 4 \times 3) \times 10^6}{(10 + 8 + 2 + 4) \times 10^6} \approx 1.92$$

$$MIPS_B = \frac{f}{CPI_B \times 10^6} = \frac{200 \times 10^6}{1.92 \times 10^6} = 104$$

$$CPU_B = \frac{I_c \times CPI_B}{f} = \frac{24 \times 10^6 \times 1.92}{200 \times 10^6} = 0.23 \text{ s}$$

b. Although machine B has a higher MIPS than machine A, it requires a longer CPU time to execute the same set of benchmark programs.

a. We can express the MIPs rate as: [(MIPS rate)/10^6] = I_c/T. So that: I_c = $T \times$ [(MIPS rate)/10^6]. The ratio of the instruction count of the RS/6000 to the VAX is [x $\times$ 18]/[12x $\times$ 1] = 1.5.

b. For the Vax, *CPI* = (5 MHz)/(1 MIPS) = 5.
For the RS/6000, *CPI* = 25/18 = 1.39.

From Equation (2.3), MIPS = $I_c/(T \times 10^6)$ = 100/*T*. The MIPS values are:

	Computer A	**Computer B**	**Computer C**
Program 1	100	10	5
Program 2	0.1	1	5
Program 3	0.2	0.1	2
Program 4	1	0.125	1

	Arithmetic mean	**Rank**	**Harmonic mean**	**Rank**
Computer A	25.325	1	0.25	2

Computer B	2.8	3	0.21	3
Computer C	3.25	2	2.1	1

a. Normalized to R:

Benchmark	Processor		
	R	**M**	**Z**
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Arithmetic mean	1.00	1.31	1.50

b. Normalized to M:

Benchmark	Processor		
	R	**M**	**Z**
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00
Arithmetic mean	1.01	1.00	1.10

c. Recall that the larger the ratio, the higher the speed. Based on (a) R is the slowest machine, by a significant amount. Based on (b), M is the slowest machine, by a modest amount.

d. Normalized to R:

Benchmark	Processor		
	R	**M**	**Z**
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Geometric mean	1.00	1.15	1.18

Normalized to M:

Benchmark	Processor		
	R	**M**	**Z**
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00
Geometric mean	0.87	1.00	1.02

Using the geometric mean, R is the slowest no matter which machine is used for normalization.

a. Normalized to X:

Benchmark	Processor		
	X	**Y**	**Z**
1	1	2.0	0.5
2	1	0.5	2.0
Arithmetic mean	1	1.25	1.25
Geometric mean	1	1	1

Normalized to Y:

Benchmark	Processor		
	X	**Y**	**Z**
1	0.5	1	0.25
2	2.0	1	4.0
Arithmetic mean	1.25	1	2.125
Geometric mean	1	1	1

Machine Y is twice as fast as machine X for benchmark 1, but half as fast for benchmark 2. Similarly machine Z is half as fast as X for benchmark 1, but twice as fast for benchmark 2. Intuitively, these three machines have equivalent performance. However, if we normalize to X and compute the arithmetic mean of the speed metric, we find that Y and Z are 25% faster than X. Now, if we normalize to Y and compute the arithmetic mean of the speed metric, we find that X is 25% faster than Y and Z is more than twice as fast as Y. Clearly, the arithmetic mean is worthless in this context.

b. When the geometric mean is used, the three machines are shown to have equal performance when normalized to X, and also equal performance when normalized to Y. These results are much more in line with our intuition.

a. Assuming the same instruction mix means that the additional instructions for each task should be allocated proportionally among the instruction types. So we have the following table:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	12	10%

$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (12 \times 0.1) = 2.64$. The CPI has increased due to the increased time for memory access.

b. $MIPS = 400/2.64 = 152$. There is a corresponding drop in the MIPS rate.

c. The speedup factor is the ratio of the execution times. Using Equation 2.2, we calculate the execution time as $T = I_c/(MIPS \times 10^6)$. For the single-processor case, $T_1 = (2 \times 10^6)/(178 \times 10^6) = 11$ ms. With 8 processors, each processor executes 1/8 of the 2 million instructions plus the 25,000 overhead instructions. For this case, the execution time for each of the 8 processors is

$$T_8 = \frac{\frac{2 \times 10^6}{8} + 0.025 \times 10^6}{152 \times 10^6} = 1.8 \text{ ms}$$

Therefore we have

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{11}{1.8} = 6.11$$

d. The answer to this question depends on how we interpret Amdahl's' law. There are two inefficiencies in the parallel system. First, there are additional instructions added to coordinate between threads. Second, there is contention for memory access. The way that the problem is stated implies that none of the code is inherently serial. All of it is parallelizable, but with scheduling overhead. One could argue that the memory access conflict means that to some extent memory reference instructions are not parallelizable. But based on the information given, it is not clear how to quantify this effect in Amdahl's equation. If we assume that the fraction of code that is parallelizable is $f = 1$, then Amdahl's law reduces to $Speedup = N = 8$ for this case. Thus the actual speedup is only about 75% of the theoretical speedup.

a. Speedup = (time to access in main memory)/(time to access in cache) = T_2/T_1.

b. The average access time can be computed as $T = H \times T_1 + (1 - H) \times T_2$

Using Equation (2.8):

$$\text{Speedup} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}} = \frac{T_2}{T} = \frac{T_2}{H \times T_1 + (1-H)T_2} = \frac{1}{(1-H) + H\frac{T_1}{T_2}}$$

c. $T = H \times T_1 + (1 - H) \times (T_1 + T_2) = T_1 + (1 - H) \times T_2)$

This is Equation (4.2) in Chapter 4. Now,

$$\text{Speedup} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}} = \frac{T_2}{T} = \frac{T_2}{T_1 + (1-H)T_2} = \frac{1}{(1-H) + \frac{T_1}{T_2}}$$

In this case, the denominator is larger, so that the speedup is less.

2.9 $T_w = w/\lambda = 8/18 = 0.44$ hours

a. A = CW

b. A = LT

c. CW = LT $\Rightarrow$ L = W(C/T)

but λ = C/T

a. Number the rectangles from bottom to top:

Rectangle	Width	Height	Area
1	2	1	2
2	2	1	2
3	2	1	2
4	1	1	1
5	3	1	3
6	1	1	1
		Total	11

b. Number the rectangles from left to right:

Rectangle	**Width**	**Height**	**Area**
1	1	1	1
2	.5	2	1
3	.5	3	1.5
4	1	2	2
5	0.25	1	0.25
6	0.25	2	0.5
7	0.75	1	0.75
8	2.5	1	2.5
9	0.5	2	1
10	0.5	1	.5
		Total	11

a. We have FLOPSref = I/Tref, FLOPSsut = I/Tsut
So Tref = I/FLOPSref, Tsut = I/FLOPSsut. Therefore:

$$r_i = \frac{Tref_i}{Tsut_i} = \frac{I_i / FLOPSref_i}{I_i / FLOPSsut_i} = \frac{FLOPSsut_i}{FLOPSref_i}$$

b. $\frac{FLOPSref_i - FLOPSsut_i}{FLOPSsut_i}$ $\qquad$ $\frac{Tref_i - Tsut_i}{Tsut_i}$

Machine	**Execution time**	**Speedup**	**Relative Change**
A	480	1	0
B	360	1.33	+0.33
C	540	0.89	−0.11
D	210	2.29	+1.29

Looking at column 2, it is easy to see that system D is the fastest system, followed by B, A, and C. Column 3 gives a more precise indication of exactly how much fast one system is than the others. Column 4 provides the same level of precision and clarity as column 3.

Machine	**Execution time**	**Speedup**	**Relative Change**
D	210	1	0
A	480	0.44	−0.56
B	360	0.58	−0.42
C	540	0.39	−0.61

No change to the relative rankings.

	Computer A time (secs)	Computer B time (secs)	Computer C time (secs)	Computer A rate (MFLOPS)	Computer B rate (MFLOPS)	Computer C rate (MFLOPS)
Program 1 (10^8 FP ops)	2.0	1.0	0.2	50	100	500
Program 2 (10^8 FP ops)	4	2.0	4.0	250	50	25
Total execution time	2.4	3.0	4.2			
Arithmetic mean of times	1.2	1.5	2.1			
Inverse of total execution time (1/sec)	0.42	0.33	0.24			
Arithmetic mean of rates				150	75.00	262.5
Harmonic mean of rates				83.33	66.67	47.6

Again, the arithmetic mean of rates does not reflect the speed ranks, whereas the harmonic mean does.

a. We use two identities: $e^X e^Y = e^{(X + Y)}$ and $\ln(X^p) = p \ln x$

b. The intermediate values in the calculation are not as large.

Getting Started with the SimpleScalar Tool Set Version 2.0

Introduction

This document contains everything that you need to know to work the SimpleScalar problems in the textbook. Section 1 is a summary that will familiarize you with SimpleScalar and the use of simulation tools in computer architecture research. Section 2 tells you how to get the tool set from the Web and set up the part that is needed for the textbook problems. Section 3 describes each of the simulators, including the information supplied by each of them. Section 4 describes the SPEC95 benchmark binaries that can be used with the sim ulators. Section 5 consists of a complete example of the use of one of the simulators. Section 6 briefly describes the complete tool set for people interested in going beyond the problems in the textbook.

1 SimpleScalar and Simulation in Computer Architecture

When computer architecture researchers work to improve the performance of a computer system, they often use an existing system to simulate a proposed system. Although the intent is not always to measure raw performance (estimating power consumption is one alternative), performance estimation is one of the most important results obtained by simulation. The SimpleScalar tool set is designed to measure the performance of several parts of a superscalar processor and its memory hierarchy. This document describes the SimpleScalar simulators. Other simulation systems may be similar or very different.

Overview of SimpleScalar Simulation

The SimpleScalar tool set includes a compiler that creates binaries for a non-existent processor. The binaries can be executed on one of several simulators that are included in the tool set. This section describes the goals of processor simulation.

The execution of a processor can be modeled as a series of known states and the time (or other costs, i.e., power) required to make the transition between each pair of states. The state information may include all or a subset of:

- The values stored in all memory locations.
- The values stored in and the status of all cache memories.
- The values stored in and the status of the translation-lookaside buffer (TLB).
- The values stored in and the status of the branch prediction table(s) or branch target buffer (BTB).
- All processor state (i.e., the pipeline, execution units (integer ALU, load/store unit, etc.), register file, register update unit (RUU), etc.)

A good way to evaluate the performance of a program on a proposed processor architecture is to simulate the state of the architecture during the execution of the program. By simulating the states through which the processor will pass during the execution of a program and estimating the time (or other measurement) necessary for each state transition, the amount of time that the simulated processor will need to execute the program can be estimated.

The more state that is simulated, the longer a simulation will take. Complex simulations can execute 100s of times slower than a real processor. Therefore, simulating the execution of a program that would take an hour of CPU time on an existing processor can take a week on a complex simulator. For this reason, it is important to evaluate what measurements are desired and limit the simulation to only the state that is necessary to properly estimate those measurements. This is the reason for the inclusion of several different simulators in the SimpleScalar tool set.

Profiling

In addition to estimating the execution time of a program on the simulated processor, profile information may be of use to computer architects. Profile information is a count of the number or frequency of events that occur during the execution of a program. One common example of profile data is a count of how often each type of instruction (i.e., branch, load, store, ALU operation, etc.) is executed during the running of a program.

Profile information can be used to gauge the relative importance of each part of a processor's implementation in determining its performance when executing the profiled program.

The SimpleScalar Base Processor

The SimpleScalar tool set is based on the MIPS R2000 processor's instruction set architecture (ISA). The processor is described in MIPS RISC Architecture by Gerry Kane, published by Prentice Hall, 1988. The ISA describes the instructions that the processor is capable of executing—and therefore the instructions that a compiler can generate—but it does not describe how the instructions are implemented. The implementation is what computer architects change inorder to improve the performance of a processor.

An existing processor can be chosen as a base processor for several reasons. These may include:

- The architecture of the processor is well known and documented.
- The architecture of the processor is state-of-the-art and therefore it is likely to be useful as a base for the study of future processors.
- The architecture of the processor has been implemented as a real processor, allowing simulations to be compared to executions on a real, physical processor.

An important consideration in the choice of the MIPS architecture for the SimpleScalar tool set was the fact that the GNU GCC compiler was available in source-code form, and could compile to the MIPS architecture. This allowed the use of this public-domain software as part of the SimpleScalar tool set.

2 Setting-Up the Simulators

The SimpleScalar simulators are part of the complete SimpleScalar tool set that is described in Section 6. The simulators are the easiest part of the tool set to set up, and they can be used to simulate existing SimpleScalar binaries without the need for any of the other parts of the tool set.

The simulators are available in a file that is available on the *SimpleScalar Tools Home Page* at

on the Web. The file is called `simplesim.tar`. You can get it directly from the Web page or via ftp (the ftp file is named `simplesim.tar.gz`) as described on the Web page. At the time that this document was written, the current release of SimpleScalar was version 2.0.

Using standard Unix commands, you can unpack the archive. All that remains is to build the simulators by moving to the directory containing them (probably `simplesim-2.0`) and building them by typing `make`. Note that the default optimization is `-O`. The simulators will run much faster (i.e., 10 times) if you modify the make file (`Makefile`) section “Choose your optimization level here” so that the “for optimization” flags are used as the `OFLAGS`. If you have built the simulators and want to rebuild them with different optimizations, type `make clean` and then `make`.

Note: Building the most complex simulator, `sim-outorder`, may require 64MB of memory when it is compiled with all optimizations. If the building process is stalled for several minutes on the line that compiles `sim-outorder`, you may want to try changing the optimizations. You may also want to simply wait for the build to finish, as the simulator will run much faster with the optimizations included.

After you have built the simulators, either move the executables to a directory in your execution path or add the directory in which you built the executables to your execution path. The simulator executable files include:

- `sim-bpred`
- `sim-cache`
- `sim-cheetah`
- `sim-fast`
- `sim-outorder`
- `sim-profile`
- `sim-safe`

Each of these is described in the next section.

3 Description of the Simulators

Each of the simulators appears here in a separate section. Each section includes the name of the simulator and the output that is generated when you ask for the simulator's help screen by executing it with no input file or parameters.

These help screens—written by the SimpleScalar authors—include a short description of the function of the simulator, a short description of why you would want to use the particular simulator, and a description of the input parameters for the simulator.

CodeBlue Wars

Introduction

In the late 1960's, Robert Morris, the future National Security Agency (NSA) Chief Scientist, along with Dennis Ritchie and Victor Vyssotsky, created a simple computer game called *Darwin*. Later, *Darwin*, evolved and became *Core War*, a game in which two or more opponents write computer programs in a simple assembly language that execute in memory and try to "kill" their opponents by forcing them to execute an illegal instruction. The programs replicate themselves to move around memory, and try to disrupt the memory space of the other programs. The last program left alive is the winner. If after a set number of iterations, multiple programs remain, it is a tie. We will be writing programs in a simplified version of the language and competing against each other in a visual simulation of memory.

Environment

The "playing field" consists of a linear array of memory locations that can be considered circular. That is, the first memory location follows the last. Each memory location is either a data value or a single instruction. A program consists of a sequence of instructions and data values that reside somewhere in the memory. Multiple programs exist simultaneously in memory. Each competing program has an associated Program Counter (PC) that keeps track of the next instruction to execute for that program.

The game is played by loading at least two of the competing programs into memory at random starting locations such that they do not initially overlap. The game loops by executing one instruction from each program at each instruction cycle, always in the same order. If a program tries to execute an illegal instruction (such as a DATA value), it terminates and is out of the game.

The Language

The original *Core War* used an assembly language called *Redcode* which consisted of eleven simple instructions. We have created an even simpler version, called *CodeBlue*, that only has eight instructions and a modified format to make for easier coding. The basic format of a *CodeBlue* instruction is:

```
<label>: <instruction> <parameter1>, <parameter2> ; <comment>
```

where:

a) <label>: and ;<comment> are optional
b) the number of parameters (one or two) depends on the instruction
c) the parts of the instruction are separated with one or more spaces or tabs
d) comments are ignored
e) everything is case-insensitive
f) execution of an illegal instruction (such as a DATA location) halts execution

SIMPLIFIED CORE WAR FOR INTRODUCING LOW-LEVEL CONCEPTS[*]

ABSTRACT

Assembly language programming is often used to teach students low-level hardware components and computer architecture basics. To introduce some of the basic concepts of program execution in memory, we have developed a very simple language based on Core War, a programming game developed in the 1980's pitting competing programs against each other in a simulated memory. We have built a visual development environment that allows students to create a program, see its representation in memory, step through the program's execution, and simulate a battle of competing programs in a visual memory environment. This paper will describe the language, the environment, the competition simulator, and our experience.

INTRODUCTION

Computer architecture and organization is one of the recognized Body of Knowledge elements in the 2001 ACM Curriculum for undergraduate Computer Science [3]. Core topics within architecture include machine level representation, assembly level machine organization, and memory system organization and architecture. Most undergraduate CS programs teach at least one course in architecture, many of which use assembly language as a tool to introduce low-level concepts.

Teaching assembly language can be a challenge and different approaches have been used to make it easier for students to understand. For example, over the past decade, simulators have been widely used for teaching assembly language concepts in computer architecture [9]. The use of simulators provides many benefits over teaching assembly language programming using real processors as the target machines [8]. An example of a simplified simulator is Silverman, Ackerman, and Chesley's SC123 which has a lean 20-instruction architecture [6]. One of the most widely-used simulators in computer architecture courses is James Larus' SPIM. SPIM implements most of the MIPS32 assembler-extended instruction set [5]. While simulators such as SPIM make it easier to teach concepts of computer architecture concepts, the assembly language programming

[*] This paper is authored by an employee(s) of the United States Government and is in the public domain.

1 Introduction

This paper presents a senior study project in the Department of Electrical Engineering at the University of Washington. This project is the work of three senior undergraduate students in the department. The goals of the project were twofold. First, it allowed the students to exercise their knowledge in computer architecture and hardware design. Second, it provided them with practical work experience involving team work and project management. One of the two main objectives of this paper is to present the findings of how well these concepts work in the group project environment. The second objective of this paper is to present the modern design approach that was used in the project.

The scope of the project was to design and simulate a realistic and modern Central Processing Unit (CPU) including both simple and complex instructions for a microcomputer system in a ten-week quarter time frame. This setting provides several challenges. First, a ten-week period is a very short time to design and simulate a complete processor. Therefore decisions had to be made to simplify the design to keep the task achievable. Second, attempts to provide realistic and modern architecture add extra burdens to the heavy work-load of the project. Third, division of work among team members and following an agreed-upon aggressive time table was a real struggle as well.

After consulting with the project advisor, Dr. Arun K. Somani, the design team decided that the only way to complete the project successfully was to follow a modern design approach. After defining the architectural goals, the team developed a very aggressive and tight project schedule (see Appendix A). In addition, coding and documentation standards were introduced to facilitate communication among team members. Moreover, many design and work concepts taken from previous projects and/or work experience were applied in this project. Some of these concepts worked well in this environment and some of them did not.

As the project progressed, the team encountered many additional problems. For example, one team member had to leave for a job in the middle of the quarter. The team had to be restructured and the project schedule had to be redefined. The restructured schedule turned out to be even more aggressive. Fortunately, the flexible nature of the modern design approach helped the team recover from the drawbacks and enabled the team to finish the project successfully on time.

Our goal as the authors of this paper is to summarize our experiences in applying many modern design and work concepts as well as to discuss the steps we took to resolve problems in a team work environment.

We used several modern design and work concepts while doing this project. The main design and work concepts used included the following.

CHAPTER 4: CACHE SIMULATOR

1) Consider the input sequence of length 120 given below.

220 66 131 35 94 172 126 217 73 176 250 84 114 187 201 116 4 102 84 22 44 87 114 82 144 28 211 131 25 192 12 134 176 157 197 211 223 67 199 203 30 154 51 123 140 172 218 249 27 91 5 51 202 59 196 240 238 71 100 217 49 231 226 12 118 233 204 222 220 31 220 66 173 5 6 94 62 126 124 250 21 81 74 116 233 9 167 62 20 4 161 35 152 102 79 73 86 84 182 22 92 44 66 159 187 240 167 100 169 201 174 114 232 82 187 87 175 131 156 301

	2- way			4 – way			8 - way		
	LRU	FIFO	RAND	LRU	FIFO	RAND	LRU	FIFO	RAND
8 blocks									
16 blocks									
32 blocks									
64 blocks									

(a) Analyze the effectiveness of different block replacement techniques by listing down the miss rate in each case.
(b) What other block replacement technique can be used and is proved to be the ideal? Explain.

2) In a N-way set-associative cache, blocks are mapped to different sets when N changes. Also, for a particular sequence, the number of compulsory and conflict misses change with the cache type. Consider the following sequence 4 0 9 7 8 11 7 5 2 1 12 6 8.

(a) List the compulsory and conflict misses for different replacement techniques for the caches below.

MISSES	**LRU**		**FIFO**		**RANDOM**	
	Comp	*Conflict*	*Comp*	*Conflict*	*Comp*	*Conflict*
4blocks 2sets						
8blocks 2sets						
16blocks 2sets						

(b) Define compulsory, capacity and conflict misses. Explain the difference between them.
(c) What is the best way to reduce conflict misses? Can it be used?

(d) List which set in the given cache will the following blocks be mapped

BLOCK	CACHE	#SET
0	*8block,2sets*	
9		
11		
4		
2	*8blocks,4sets*	
9		
10		
4		
7	*16blocks,2sets*	
1		
12		
3		

CHAPTER 4: CACHE TIME ANALYSIS

1) Consider a 4-way set associative cache of size 4KB and block size of 64bytes. Consider 25% writes, miss penalty of 40 cycles, hit time of 1 cycle and mem write timeof 6 cycles. How can the following happen under the condition that the *memory miss penalty*, *% write* and *hit time* remain the same for both the cases? Reason your findings in a few sentences.

(a) Avg. Memory access for Write Back cache > Write through cache - No write allocate
(b) Avg. Memory access for Write Back cache > Write through cache - Write allocate

2) Assume that the % of dirty blocks in the cache is 99%, if the total number of blocks in the cache is very high, the write back based cache should have a high memory access time. Is this case true for both small and large cache sizes from this animation program? Fill the given table and explain the situation for the 4 caches. The rest of the simulation term values are listed in the table.

Simulation terms	**Value**
Cache type	4-way set associative
% writes	25 %
Miss Penalty	40 cycle
Hit Time	1 cycle
Memory Write time	6 cycles

Cache size	**Block size**	**Hit rate**	**Memory access time**
1KB	256bytes		
	16bytes		
4KB	256bytes		
	16bytes		
32KB	256bytes		
	16bytes		
256KB	256bytes		
	16bytes		

CHAPTER 4: MULTICACHE CACHE DEMONSTRATOR

Note:

1) The trick to use the same set of memory references is to click on the "GENERATE MEMORY REFERENCES" only for the first time while setup. Do not click on this button until your simulation-based comparison is done.

2) Always choose to run the complete simulation

1) When multiple tasks are scheduled in a system, the number of cycles that one task is allowed to run before the CPU switches to another task is very important.
Assume the following setup for the animation.

Specification	**Value**
Replacement Policy	LRU
Scheduling Mechanism	Round Robin
Use Random Access Sequence	**NO**
Number of Tasks	4
Number of memory references for Task A,B,C,D	25 each

Consider the following sequences for the 4 tasks A, B, C and D

# 25	9	24	39	63
# **24**	9	28	10	14
# **23**	9	8	10	63
# **22**	57	46	46	63
# **21**	29	35	26	14
# **20**	15	28	63	8
# **19**	57	29	2	63
# **18**	57	56	13	53
# **17**	48	56	24	53
# **16**	60	22	32	53
# **15**	57	46	2	29
# **14**	48	52	6	62
# **13**	49	29	17	29
# **12**	49	22	10	8
# **11**	60	0	13	8
# **10**	24	0	62	8
# **9**	60	46	32	8
# **8**	15	52	10	5
# **7**	49	8	10	56
# **6**	40	29	28	29
# **5**	49	29	2	26
# **4**	49	52	34	14
# **3**	49	52	40	49
# **2**	9	39	10	62
# **1**	27	21	39	44
Task	**A**	**B**	**C**	**D**

Cache Size, # Sets	CPU Time Slice	Reload Transients
16, 2	1	
	3	
	4	
	5	
	8	
16,8	1	
	3	
	4	
	6	
	8	
32,2	1	
	3	
	4	
	6	
	8	

(a) Find the reload transient for the following caches with different CPU slice times.
(b) What is the CPU time slice number (in cycles) that reduces the number of reload transients in each of the following caches?

(c) In a few sentences, explain Reload Transient and Task Footprint. How are they related?
(d) Explain why it is important to find the ideal time for a CPU slice for a particular cache.

2) Use the following settings as simulation input

Specification	**Value**
Replacement Policy	LRU
Use Random Access Sequence	**NO**
Number of Tasks	5
Number of memory references for Task A,B,C,D,E	5 each
CPU Time slice	1
Priority A,B,C,D,E	3, 1, 0, 2, 4

Memory references of each task

# 5	52	93	113	28	9
# 4	29	93	113	28	9
# 3	19	93	104	125	9
# 2	93	19	104	89	31
# 1	19	76	113	125	9
Task	**A**	**B**	**C**	**D**	**E**

(a) Does the task scheduling technique affect the hit rate of the overall system? Explain.

(b) Find the hit rate in each of the following caches after 12 simulation cycles.

Cache Hit Rate % Cache Size, # Sets	FIFO	Round Robin	Priority Based 3-1-0-2-4 priority
16, 4			
16,8			
32,2			

(c) When does the round robin technique behave like the FIFO technique?
(d) When can the priority-based technique behave like the round robin technique?

CHAPTER 4: CACHE SIMULATOR

1) Consider the input sequence of length 120 given below.

220 66 131 35 94 172 126 217 73 176 250 84 114 187 201 116 4 102 84 22 44 87 114 82 144 28 211 131 25 192 12 134 176 157 197 211 223 67 199 203 30 154 51 123 140 172 218 249 27 91 5 51 202 59 196 240 238 71 100 217 49 231 226 12 118 233 204 222 220 31 220 66 173 5 6 94 62 126 124 250 21 81 74 116 233 9 167 62 20 4 161 35 152 102 79 73 86 84 182 22 92 44 66 159 187 240 167 100 169 201 174 114 232 82 187 87 175 131 156 301

	2- way			4 - way			8 - way		
	LRU	**FIFO**	**RAND**	**LRU**	**FIFO**	**RAND**	**LRU**	**FIFO**	**RAND**
8 blocks	98.33	98.33	95.83	97.5	97.5	95.83	97.5	97.5	97.5
16 blocks	91.67	91.67	94.17	94.17	94.17	92.5	93.3	93.3	95
32 blocks	89.17	89.17	87.5	89.17	89.17	85	88.33	88.33	85.83
64 blocks	85	80.3	74.17	82.5	79.17	72.5	84.17	83.3	80

(a) Analyze the effectiveness of different block replacement techniques by listing down the miss rate in each case.
FIFO technique has fewer misses than the LRU approach. The random technique seems to work better than LRU and FIFO for this sequence. But it cannot be relied upon for efficient block replacement.

(b) What other block replacement technique can be used and is proved to be the ideal? Explain.
Another block replacement technique that can be used is the optimal technique. The optimal technique predicts the frequency of different blocks based on the history of the block's usage. It is the ideal technique, but cannot be implemented easily due to its complexity.

2) In a N-way set-associative cache, blocks are mapped to different sets when N changes. Also, for a particular sequence, the number of compulsory and conflict misses change with the cache type. Consider the following sequence 4 0 9 7 8 11 7 5 2 1 12 6 8.

(a) List the compulsory and conflict misses for different replacement techniques for the caches below.

MISSES	**LRU**		**FIFO**		**RANDOM**	
	Comp	*Conflict*	*Comp*	*Conflict*	*Comp*	*Conflict*
4blocks 2sets	4	8	4	8	4	9
8blocks 2sets	8	3	8	3	8	4
16blocks 2sets	11	0	11	0	11	0

(b) Define compulsory, capacity and conflict misses. Explain the difference between them.

Compulsory Misses: The very first references to a block in the cache cannot be a hit, so the block has to be bought to the cache from a lower level memory. These are called compulsory or cold misses.
Capacity Misses: If the cache cannot contain all the blocks needed to execute a task, this leads to some blocks being replaced and have to be retrieved later. Such misses are called capacity misses. Such misses are common for smaller caches and also for tasks that require a large amount memory to execute.

Conflict misses: When the cache is a direct mapped or a set associative, conflict misses occur when blocks have to be replaced as many blocks can map to the same set.

(c) What is the best way to reduce conflict misses? Can it be used?
The best way to eliminate conflict misses is to use a fully associative cache. To accommodate all the blocks without any conflict for a task that requires large memory, the fully associative cache is the best. But this type of cache cannot is not generally used in practice as it is very expensive to have one.

(d) List which set in the given cache will the following blocks be mapped

BLOCK	**CACHE**	**#SET**
0	*8block,2sets*	0
9		1
11		1
4		0
2	*8blocks,4sets*	2
9		1
10		2
4		3
7	*16blocks,2sets*	1
1		1
12		0
3		1

CHAPTER 4: CACHE TIME ANALYSIS

1) Consider a 4-way set associative cache of size 4KB and block size of 64bytes. Consider 25% writes, miss penalty of 40 cycles, hit time of 1 cycle and mem write timeof 6 cycles. How can the following happen under the condition that the *memory miss penalty*, *% write* and *hit time* remain the same for both the cases? Reason your findings in a few sentences.
For both the cases below, when dirty data goes above a certain %, the WB cache hasmore memory access time than the WT cache.

(a) Avg. Memory access for Write Back cache > Write through cache - No write allocate

% dirty data	**WB cache**	**WT cache**
0	**3.59**	**4.20**
32	**4.22**	**4.20**

(b) Avg. Memory access for Write Back cache > Write through cache - Write allocate

% dirty data	**WB cache**	**WT cache**
0	**3.62**	**4.86**
50	**4.91**	**4.86**

Reason:
(i) WT cache writes all blocks in both the cache and the lower memory. WB cache writes only dirty blocks to the lower memory. Hence if the number of dirty blocks is very high, the WB cache on an average has more access time than the WT cache.

(ii) The dirty % to satisfy the condition is less for part (a) than (b) as the write misses do not constitute a write to the lower memory even if there are dirty blocks.

2) Assume that the % of dirty blocks in the cache is 99%, if the total number of blocks in the cache is very high, the write back based cache should have a high memory access time. Is this case true for both small and large cache sizes from this animation program? Fill the given table and explain the situation for the 4 caches. The rest of the simulation term values are listed in the table.

Simulation terms	**Value**
Cache type	4-way set associative
% writes	25 %
Miss Penalty	40 cycle
Hit Time	1 cycle
Memory Write time	6 cycles

Cache size	**Block size**	**Hit rate**	**Memory access time**
1KB	256bytes	0.8428	11.91
	16bytes	0.8925	8.46
4KB	256bytes	0.9357	5.465
	16bytes	0.942	5.028
32KB	256bytes	0.9858	1.986
	16bytes	0.9805	2.354
256KB	256bytes	0.9972	1.194
	16bytes	0.9938	1.430

The memory access time not only depends on the % dirty data, but also on the miss rate. If the miss rate remains constant for increase in cache size and number of blocks,the WB cache will have more access time for 100% dirty data.

From the simulation we can see that the miss rate decreases for increase in number of blocks(decreasing block size) for smaller caches, this compensates for the increase in % dirty data. Hence the above case of WB cache having more access time is not seen for 99% dirty data.

In the case of larger caches (16KB and 256KB), the hit rate decreases for decrease in block size. Hence the access times are more for caches with smaller block size (more number of blocks).

PART 2 RESEARCH PROJECTS

A suggested project assignment form is the document **ProjectForm**, included in the folder **ResearchProjects**. The form includes guidelines for a final report and a set of slides. If the deliverable is slides, class time should be set aside for each team to present its results.

If projects are assigned to teams rather than individuals, problems can arise. A paper on this subject, in the document **Team.pdf** included in the folder **ResearchProjects**, might be useful to the instructor.

An interesting example of an implementation project is reported in:

Lee, V. et al, "Superscalar and Superpipelined Microprocessor Design and Simulation: A Senior Project." *IEEE Transactions on Education*, February 1997.

The ideas in this paper can be adapted for your own student projects. A copy of the paper is included in the folder **ResearchProjects**.

The research project will typically involve a web/library search and analysis. It could also involve some implementation or measurement. Typical questions that could be addressed with reference to technology X:

- What is X?
- What Standard governs X?
- How is X implemented?
- What is the cost of implementation?
- What companies are currently involved in this line of business?

Here are some ideas for research project topics:

- web PC, web TV
- Image Retrieval Systems
- SPEC benchmark
- cache coherence protocols
- network media technologies
- RAM bus
- flat panel displays
- active matrix LCD displays
- register allocation
- N-version programming/recovery blocks
- MPEG-2
- graph theory
- encryption hardware
- design for test
- adaptive control
- imaging systems
- image recognition
- quantum well transistors
- computing in space
- Linux
- SunOS
- Spring
- Windows NT/ Windows '95
- OSF/1
- Mach kernel
- taligent
- alpha
- ultraSPARC
- MIPS R10000
- Intel's IA-64 architecture
- PowerPC family
- AMD K5, other P5 clones

1. Introduction

SMPCache is a trace-driven simulator for cache memory systems on symmetric multiprocessors (SMPs) which use bus-based shared memory. This simulator operates on PC systems with Windows, and it offers a Windows typical graphic interface.

Some of the parameters you can study with the simulator are: program locality; influence of the number of processors, cache coherence protocols, schemes for bus arbitration,mapping, replacement policies, cache size (blocks in cache), number of cache sets (for set associative caches), number of words by block (block size), ...

Because of its easy and user-friendly interface, the simulator is recommended forteaching purposes; since it allows to observe, in a clear and graphic way, how the multiprocessor evolves as the execution of the programs goes forward (the memory traces are read).

Prerequisites

To make sense of the rest of *Getting Started with SMPCache*, you should be familiar with some theoretical considerations concerning cache memory systems, and particularly regarding their use in multiprocessor environments. These concepts are widely discussed in many computer architecture texts (like the William Stallings' *Computer Organization and Architecture*), and we will not mention them here. All operations and algorithms we use are similar to those found in these computer architecture texts. As a consequence, the results obtained with the simulator are very close to the real world.

Suggestions?

If you have comments about this guide to SMPCache or about the simulator, please contact Miguel A. Vega at (Fax: +34-927-257202) or at the following address:

Miguel A. Vega-Rodríguez

Dept. de Informática, Univ. de Extremadura, Escuela Politécnica

Campus Universitario s/n. 10071. Cáceres. España (Spain)

2. Installation

In order to begin the installation you must execute the Setup.exe program included in your copy of SMPCache. Then, follow the directions below and on the screen during the installation process:

1. Exit *all* Windows applications prior to continuing with the installation.
2. When the "Welcome" screen appears, read it. Click **Next** to continue.

3. Enter your name and company. Click **Next** to continue.
4. Select the location in which you want to install SMPCache. Choose the default folder or click **Browse** to select a different location or to enter your own folder name. Click **Next** to continue.
5. Click the type of setup you prefer, then click **Next** to continue.
   a) Typical — installs all program files to the location you selected in step 4. Recommended for most users.
   b) Compact — reduces the disk space required for the installation because it does not install the sample files.
   c) Custom — provides you with options on installing the sample files or help files.
6. If you select the Custom setup, then you must choose the exact components you want to install. Click **Next** to continue.
7. Select the program folder for SMPCache. Choose the default program folder or type a new folder name. You can also select one from the Existing Folders list. Click **Next** to continue.
8. Before starting copying files, the installation process shows you the current settings. If you want to change any settings click **Back**, otherwise click **Next** to begin copying files.
9. SMPCache will now finish being installed. Click **Finish** to complete the installation.

Once installation is complete, a SMPCache group, which includes the application icon, is created. You can then create a shortcut to SMPCache on your desktop.

Remember: If for any reason you wish to stop the installation, click **Cancel** and the installation of SMPCache will be terminated.

Uninstalling SMPCache

To uninstall the simulator:

1. Click the Windows **Start** button.
2. Click **Settings** and **Control Panel**.
3. Click **Add/Remove Programs**. The Add/Remove Programs Properties screen appears.
4. From the **Install/Uninstall** list, select **SMPCache** and select **Add/Remove**. After your confirmation, SMPCache will be removed from your computer.

3. Configuration Files

The simulator allows you to select the different choices for configuring a given architecture (see Table 1). The different choices selected may be stored on an ASCII data file

Student Projects using SMPCache 2.0

1. Introduction

This document contains some ideas for student projects using SMPCache. These idea descriptions are intended as starting point from which other many project assignments could be designed. Students should be familiar with the simulator to carry out any of the projects. We have developed the *"Getting Started with SMPCache"* manual with this aim.

If you have comments about this document or the simulator, please contact Miguel A. Vega at (Fax: +34-927-257202) or at the following address:

Miguel A. Vega-Rodríguez

Dept. de Informática, Univ. de Extremadura, Escuela Politécnica

Campus Universitario s/n. 10071. Cáceres. España (Spain)

2. Uniprocessor Traces

We will first study the basic algorithms and concepts that are present in every cache memory system, uniprocessor or multiprocessor. We will consequently configure the SMPCache simulator with a single processor, and we will use uniprocessor traces. For this first set of projects we will consider traces of some SPEC'92 benchmarks (*Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*), according to real tests performed on a

MIPS R2000 system. The traces used represent a wide variety of "real" application programs. These traces come from the Parallel Architecture Research Laboratory (PARL), New Mexico State University (NMSU), and they are available by anonymous ftp to *tracebase.nmsu.edu*. The traces had different formats, like Dinero or PDATS, and they have been changed to the SMPCache trace format (see *Getting Started with SMPCache 2.0*, section 4). These traces, with the correct format for SMPCache, are included in your copy of the simulator. A summary of the traces is given in Table 1.

Name	Classification	Language	Comments
Hydro	Floating point	---	Astrophysics: Hydrodynamic Naiver Stokes equations
Nasa7	Floating point	Fortran	A collection of 7 kernels. For each kernel, the program generates its own input data, performs the kernel and compares the result against an expected result
Cexp	Integer	C	Portion of a Gnu C compiler that exhibits strong random behaviour
Mdljd	Floating point	Fortran	Solves the equations of motion for a model of 500 atoms interacting through the idealized Lennard-Jones potential. It is a numerical program that exhibits mixed looping and random behaviour
Ear	Floating point	---	This trace, the same as the rest, was provided by Nadeem Malik of IBM
Comp	Integer	C	Uses Lempel-Ziv coding for data compression. Compresses an 1 MB file 20 times
Wave	Floating point	Fortran	Solves Maxwell's equations and electromagnetic particle equations of motion
Swm	Floating point	Fortran	Solves a system of shallow water equations using finite difference approximations on a 256*256 grid
UComp	Integer	C	The uncompress version of *Comp*

Table 1: Uniprocessor traces.

Remember: All these uniprocessor projects can be performed in a similar way with multiprocessor traces.

Project 1: Locality of Different Programs

Purpose

Show that the programs have different locality, and there are programs with "good" or "bad" locality.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Blocks in cache = 128 (cache size = 4 KB).

- Mapping = Fully-Associative.
- Replacement policy = LRU.

Obtain the miss rate using the memory traces: *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp* (trace files with the same name and extension ".prg").

Do all the programs have the same locality grade? Which is the program with the best locality? And which does it have the worst? Do you think that the design of memory systems that exploit the locality of certain kind of programs (which will be the most common in a system) can increase the system performance? Why?

During the development of the experiments, you can observe graphically how, in general, the miss rate decreases as the execution of the program goes forward. Why? Which is the reason?

Project 2: Influence of the Cache Size

Purpose

Show the influence of the cache size on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the blocks in cache using the following configurations: 1 (cache size = 0,03 KB), 2, 4, 8, 16, 32, 64, 128, 256, and 512 (cache size = 16 KB). For each of the configurations, obtain the miss rate using the trace files (extension ".prg"): *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

Does the miss rate increase or decrease as the cache size increases? Why? Does this increment or decrement happen for all the benchmarks or does it depend on the different locality grades? What does it happen with the capacity and conflict (collision) misses when you enlarge the cache? Are there conflict misses in these experiments? Why?

In these experiments, it may be observed that for great cache sizes, the miss rate is stabilized. Why? We can also see great differences of miss rate for a concrete increment of cache size. What do these great differences indicate? Do these great differences of miss rate appear at the same point for all the programs? Why?

In conclusion, does the increase of cache size improve the system performance?

Project 3: Influence of the Block Size

Purpose

Study the influence of the block size on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Main memory size = 256 KB (the number of blocks in main memory will vary).
- Cache size = 4 KB (the number of blocks in cache will vary).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the words by block using the following configurations: 4 (block size = 8 bytes), 8, 16, 32, 64, 128, 256, 512, and 1024 (block size = 2048 bytes). For each of the configurations, obtain the miss rate using the trace files: *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

Does the miss rate increase or decrease as the block size increases? Why? Does this increment or decrement happen for all the benchmarks or does it depend on the different locality grades? What does it happen with the compulsory misses when you enlarge the block size? What is the pollution point? Does it appear in these experiments?

In conclusion, does the increase of block size improve the system performance?

Project 4: Influence of the Block Size for Different Cache Sizes

Purpose

Show the influence of the block size on the miss rate, but in this case, for several cache sizes.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 32.
- Main memory size = 1024 KB (the number of blocks in main memory will vary).
- Mapping = Fully-Associative.
- Replacement policy = LRU.

Configure the words by block using the following configurations: 8 (block size = 32 bytes), 16, 32, 64, 128, 256, 512, and 1024 (block size = 4096 bytes). For each of the configurations of words by block, configure the number of blocks in cache in order to get the following cache sizes: 4 KB, 8 KB, 16 KB, and 32 KB. For each configuration obtain the miss rate using the memory trace: *Ear*.

We are first going to ask you the same questions as in the previous project: Does the miss rate increase or decrease as the block size increases? Why? What does it happen with the compulsory misses when you enlarge the block size? Does the pollution point appear in these experiments?

Does the influence of the pollution point increase or decrease as the cache size increases? Why?

Project 5: Influence of the Mapping for Different Cache Sizes

Purpose

Analyse the influence of the mapping on the miss rate for several cache sizes.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 32.
- Words by block = 64 (block size = 256 bytes).
- Blocks in main memory = 4096 (main memory size = 1024 KB).
- Replacement policy = LRU.

Configure the mapping using the following configurations: Direct, two-way set associative, four-way set associative, eight-way set associative, and fully-associative (remember: Number_of_ways = Number_of_blocks_in_cache / Number_of_cache_sets). For each of the configurations of mapping, configure the number of blocks in cache in order to get the following cache sizes: 4 KB (16 blocks in cache), 8 KB, 16 KB, and 32 KB (128 blocks in cache). For each configuration obtain the miss rate using the memory trace: *Ear*.

Does the miss rate increase or decrease as the associativity increases? Why? What does it happen with the conflict misses when you enlarge the associativity grade?

Does the influence of the associativity grade increase or decrease as the cache size increases? Why?

In conclusion, does the increase of associativity improve the system performance? If the answer is yes, in general, which is the step with more benefits: from direct to 2-way, from 2-way to 4-way, from 4-way to 8-way, or from 8-way to fully-associative?

Project 6: Influence of the Replacement Policy

Purpose

Show the influence of the replacement policy on the miss rate.

Development

Configure a system with the following architectural characteristics:

- Processors in SMP = 1.
- Cache coherence protocol = MESI.
- Scheme for bus arbitration = Random.
- Word wide (bits) = 16.
- Words by block = 16 (block size = 32 bytes).
- Blocks in main memory = 8192 (main memory size = 256 KB).
- Blocks in cache = 128 (cache size = 4 KB).
- Mapping = 8-way set-associative (cache sets = 16).

Configure the replacement policy using the following configurations: Random, LRU, LFU, and FIFO. For each of the configurations, obtain the miss rate using the trace files (extension ".prg"): *Hydro*, *Nasa7*, *Cexp*, *Mdljd*, *Ear*, *Comp*, *Wave*, *Swm* and *UComp*.

In general, which is the replacement policy with the best miss rate? And which does it have the worst? Do the benefits of LFU and FIFO policies happen for all the benchmarks or do they depend on the different locality grades?

For a direct-mapped cache, would you expect the results for the different replacement policies to be different? Why or why not?

In conclusion, does the use of a concrete replacement policy improve the system performance? If the answer is yes, in general, which is the step with more benefits: from Random to LRU, from Random to LFU, or from Random to FIFO? Why (consider the cost/performance aspect)?

3. Multiprocessor Traces

After analysing the basic algorithms and concepts that are present in every cache memory system (uniprocessor or multiprocessor), we study some theoretical issues related with multiprocessor systems. In these projects, we will consequently configure the SMPCache simulator with more than one processor, and we will use multiprocessor traces with tens of millions of memory accesses (references) for four benchmarks (*FFT*, *Simple*, *Speech* and *Weather*). These traces were provided by David Chaiken (then of MIT) for NMSU PARL, and they are available by anonymous ftp to *tracebase.nmsu.edu*. The traces represent several real parallel applications (*FFT*, *Simple* and *Weather* traces were generated using the post- mortem scheme implemented by Mathews Cherian with Kimming So at IBM). The traces had different formats, like the canonical format for multiprocessor traces developed by Anant Agarwal, and they have been changed to the SMPCache trace format (see *Getting Started withSMPCache 2.0*, section 4). These traces, with the correct format for SMPCache, can be obtained at the address A summary of the traces is shown in Table 2.

A PROFESSIONAL PRACTICE COMPONENT IN WRITING: A SIMPLE WAY TO ENHANCE AN EXISTING COURSE[*]

Karen Anewalt
Department of Computer Science
Mary Washington College
Fredericksburg, VA 22401
anewalt@mwc.edu

ABSTRACT

The annual survey conducted in 2001 by the National Association of Colleges and Employers showed that employers rank good communication skills (both written and oral) as the most desirable quality in applicants seeking employment [7]. The recently published 2001 ACM/IEEE Computing Curriculum guidelines respond to industry demands by stressing the importance of incorporating "professional practice" components, including coursework focusing on written communication skills, in the undergraduate curriculum [1]. Despite evidence that communication skills are highly valued and professional recommendations to include writing in the computer science curriculum, many computer science faculty members are reluctant to add written components to their courses. This paper describes simple, practical ways of incorporating writing into existing computer science courses. Examples of writing assignments used in a sophomore-level data structures course are provided.

INTRODUCTION

The importance of developing good communication skills as part of the undergraduate curriculum has received recent attention in the ACM/IEEE Computing Curriculum 2001 guidelines [1]. The guidelines stress the importance of developing professional skills, including effective written communication skills, as part of the undergraduate curriculum. Motivation to enhance student communications skills also comes from the corporate world. In the annual surveyconducted in 2001 by the National Association of Colleges and Employers, employers ranked good communication skills (both written and oral) as the most desirable quality in applicants seeking employment [7]. It is not surprising that employers and professional organizations stress the importance of good communicationskills. Regardless of post-graduate choices, students will undoubtedly use their written communication skills in their future lives.

In spite of the obvious need for students to develop writing skills, many faculty members are reluctant to add writing assignmentsto their existing computer science courses. A common attitude is that instruction in writing is better left to humanities and social science instructors. This opinion is not a wise one. In [8], Pensente states that "isolated attempts to teach writing hinder the transfer of learning to other courses and, eventually, to the work place". The Computing Curriculum states that educators have a responsibility to provide exposure to professional practice and ease the transition from academia to the business world [1]. In addition to providing exposure to professionalpractice, teaching writing in a computer science context can have many benefits that would not be realized if students were only to receive writing practice in non-computer science courses.

! **When writing is taught in a computer science setting, the assignments and feedback can be relevant to the computer science discipline.**

Writing experiences in computer science courses can provide students with valuable practice communicating with assorted audiences in various formats that will be required inthe corporate or postgraduate world. Computer scientists do not operate in a vacuum. In the modern world, computer scientists and software engineers need to be able to effectively communicate, both orally and on paper. In industry, programmers are expected to communicate with groups of other programmers in order to design and complete software projects. Programmers may also be required to communicate with non-technical individuals about project goals and requirements. Thus it is important to teach computer science students how to communicate with a range of audiences. Because computer scientists are muchmore familiar thannon-computer scientists withthe types of documents typically required of computer scientists, it is appropriate that instruction in creating such written works come from computer scientists.

! **Parallels can be drawn between the software design process and the writing process.**

The similarity between the writing process and the software design process is something that few traditional English literature courses recognize, but can make computer science students feel more connected to the writing process. The software design process is iterative. Most programmers do not receive a project assignment and immediately sit down to write all of the code perfectly. The software design process should involve thought and planning prior to the implementation phase. Following the implementation phase, software design involves a cycle of testing and modifying the code. Finally, software enters the maintenance phase where it is modified to meet changing needs and goals. The writing process followsa verysimilarsequence ofphases [9]. When a written assignment is given, the writer should think about the goals of the final document and organize his or her thoughts about the assignment. After a clear plan is constructed, the process of selecting language to communicate the ideas begins and the document is created. There is a stage of reading and modifying the document to ensure that the final product has the desired qualities. Finally, manycomputer science-related documents, like user's manuals, enter a maintenance stage inwhichtheyare modifiedas the software that they refer to is modified. Even beginning computer science students are familiar with the software design process and can see the parallels. By presenting this type of analogy to students, the relationship between the writing process and the software design process is made clear, and computer science students willfeelmore familiarwiththe writingprocess.

! **Writing promotes active learning and can increase student understanding of course material.**

Inmyexperience, studentsoftenbelieve that theyunderstand complexconcepts presented in lecture without any additional exposure or hands-on practice with the concepts. However, when these students are asked to write a detailed explanation of the concept, they realize that their understanding is far from complete. Written assignments are one wayofencouraging students to interact withcourse content ona personallevel, whichcan in turn increase student understanding [2]. As faculty, we continually engage in active learning; we read textbooks, compare ideas fromdifferent sources, prepare lecture notes and create assignments. Students are often deprived of this type of active learning experience [5]. Using targeted written assignments as part of a course encourages students to participate in the same type of rich learning experience that we engage in as professionals. Writing activities can teach students to pose questions, develop hypotheses, collect and analyze data, and organize arguments. Thesecriticalthinkingskills are important to develop in order to be successful in the computer science field.

INCORPORATING WRITING INTO AN EXISTING COURSE

Committingto includingwriting assignments ina coursedoes not necessarily implythat the course format must be restructured. Assignments that could easily be modified to include a written component are already used in most computer science courses. Typical computer science courses do use programming projects and/or homework problem sets to reinforce lecture topics. Written components can be easily added to programming projects by requiring a short summaryofthe software'spurpose, a design summarydescribing design decisions made by the student, or a document describing the efficiency of an algorithm being used in the software. This type of assignment can be particularly useful because it provides practice in written communication and does so in a professionalcontext.

Another methodof incorporating additional writing into an existing course is byreplacing traditional problem sets with writing assignments that achieve the same learning goals. Alternately, a traditional homework assignment can be used along with a question to be answered in paragraph form. The focus ofthe assignment remains the same, but by adding the additional writing component to the assignment, students receive additional practice and exposure to writing. Writing can also be used to reinforce topics covered in lecture or reading assignments. For example, when reading is assigned, the instructor can provide a question to whichstudents should write a short one-page response. The questionwill focus studentson the key points of the reading, engage the students in actively thinking about issues discussed in the reading, and forces the students to be accountable for the reading by creating a tangible deliverable. This type ofwriting assignment can actuallyreduce the amount oftime that is spent coveringthe basics ofa concept during lecture, becausethestudentswillhave alreadyabsorbed the basics by completing the reading and writing assignment.

The key to enhancing a course through the use of writing assignments is to evaluate the learning objectives of the course and the existing assignments. Assignments can then be added or modified to increase the amount of writing required while maintaining the original learning goal. In this way, writing assignments can be designed to enhance course content rather than detract from course content.

WRITING ASSIGNMENT SUGGESTIONS

I was first exposed to teachingwritingwithinacomputer science context after being asked to teach a data structures course designated to fulfill part of the writing-across-the-curriculum requirement at my institution. The primary computer science goals of the course are to teach data structures and to introduce students to Java as a second programming language. The course is the third semester programming course for computer science majors and covers heaps, trees, and graphs while emphasizing efficiency analysis and software reusability. In the course, students learn to use Java as their second programming language, having already acquired skills in C++. Like most computer science courses, there are many opportunities to introduce writing assignments into this type of course without sacrificing course content.

I designed the writing component of the course with two main goals. The first goal is to provide students with significant practice in writing and communicating ideas. The second goal is to expose students to a variety of types of technical writing appropriate to the computer science discipline. These goals were achieved using a variety of writing assignments. I have found that using a variety of types of writing assignments benefits the students. The variety increases the chance that a student will be successful in at least one paper. Some students are better at writing for peers; others excel in creating documents for non-technical audiences. Some students enjoy creative papers; others feel more confident when the assignment is very structured. When a broad spectrum of writing styles is explored, students are likely to find something with which they feel comfortable. In addition, the students benefit by familiarizing themselves with the goals and challenges of a larger variety of document styles.

My course uses three or four formal writing assignments (3-5 page papers) during the semester. Examples of some writing assignments that I have used in this course are included in the Appendix. Other types of formal writing assignments that I've successfully used in the course include software design documents, descriptive essays about course topics and short research papers.

As discussed earlier, adding writing to a course does not necessarily mean that formal papers must be assigned. Shorter written assignments can be equally effective in providing additional practice in writing and focusing students' attention on key concepts discussed in a course. In addition to the formal papers described above, I assign short, informal writing assignments to encourage active learning. I typically assign an informal writing assignment once a week and grade the papers on a credit/no credit basis. Examples of informal writing assignments that I have used include assignments to summarize the reading assignment in a few paragraphs (depending on the length of the reading), answer a focus question (provided at the time the assignment is given) related to the reading, to find Web resources that contradict one another about a definition or concept described in class and argue that one definition is correct, to make connections between lecture topics discussed at different times in the course, and to compare and contrast two data structures. These informal writing assignments are designed to stimulate independent thought about course topics and to promote active learning. I have found that the informal assignments were very useful to me as an instructor because the student responses occasionally indicated weaknesses in student understanding and could be used to generate discussion in future class meetings.

STUDENT RESPONSE TO WRITING IN COMPUTER SCIENCE

Each semester that I have offered the data structures course with a strong writing component, several students have approached me at the end of the semester and mentioned that the writing assignments were very beneficial and helped them to draw connections about course material that they would have otherwise missed. They have said that by writing about computer science topics, they began to recognize their own weaknesses in understanding and were better able to formulate questions about the course topics. After having completed a writing assignment, they felt that they had a more complete understanding of the related course material.

In the most recent offering of the data structures course, I requested that students complete a short questionnaire about the writing portion of the course. I prepared a list of statements and the students were allowed to anonymously respond that they agreed or disagreed with each statement. Because the enrollment in the course was small, statistical analysis of the data is not meaningful, however I felt the information collected would be useful to get a feel for the student reactions to the effectiveness of the writing assignments used in the course. The responses show that most of the students felt that the writing assignments were beneficial both because they exposed them to the types of writing that might be required of them after graduation and because they felt that the emphasis on writing enhanced their written communication skills.

The responses collected from the students enrolled in the course along with the questions asked are indicated below:

		Agree	Disagree
1)	In general, I feel that my writing has improvedduring. the semester.	5	2
2)	After completing a computer science course withwriting assignments, I feel that I have a clearer picture of how writing is used in the computer science field.	6	1
3)	I feel that the documents assigned as formalwriting assignments in class were representative of the types of documents that I may be required to write as a professional in the computer science field.	6	1
4)	I think that I will benefit from being exposed to thesetypes of documents even if I will not be required to write any documents in my future career.	7	0
5)	In general, I felt that the writing in the course helpedme to develop understanding of course material.	7	0
6)	I felt that I would have learned as much in thecourse if it had not included writing assignments.	2	5

CONCLUSION

The ACM/IEEE Computing Curriculum guidelines emphasize the importance of good communication skills and encouraged institutions to include communication skill development in their curriculum. Including writing assignments in a computer science context exposes students to the types of writing that will be expected of them in the future and can also encourage studentsto be more personally involved in learning course material. I have found that includingwritingassignments ina computer science course can improve student communication skills, understanding of the professional practice and can enhance student understanding of course topics.

REFERENCES

CPSC 321: Data Structures -- Writing Assignment 1

Assignment Goals:

! Learn to verbalize technical concepts for a less-technical audience

! Familiarize yourself with Forte for Java and the Java programlifecycle

Part I

Task: In order to familiarize yourself with the Forte for Java Integrated Development Environment (IDE), create two programs that display a simple message (such as “Hello, world!”). You will create an applet as well as an application program with this functionality.

(The message printed by the applet should be displayed in the applet viewer; the application should print the message to standard output.) Very basic instructions on how to create, compile, and run Java programs withinthe Forte IDE are provided at the end of this document.

Deliverables: Turn in printouts of your source files (staple to the end of your paper for Part II).

Part II

Task: You are a member of the programming team responsible for developing Forte for Java at Sun Microsystems. Your team has recently completed the development and testing of the Forte software package. At the weekly team meeting, your supervisor assigns you the task of writinga user's guide for the IDE. She tells you that the guide must explain how to create a Java source file, compile it, and execute it using Forte for Java. This guide will be distributed along withthe software when it is sold commercially. She also tells you that your guide should focus on writing one type ofprogram. In other words, your guide should either explain to the reader the steps necessary to create an applet OR to create an application (BUT NOT BOTH). Someone else on the team will be responsible for the instructions for developingthe other type of program.

Because the user's guide will be included in the commercial release package, the guide should be understandable to a novice computer scientist (think of someone taking computer science for the first time) who has never used Forte, but who is familiar with the concept of programming(in C++ or Ada), windows, menus, point and click, etc. As you write your user's guide, anticipate questions and problems that the reader might have as they use the IDE and provide the reader with guidance and explanations in your text.

The paper should contain a separate section for each of the following:

! An introduction section introducing the Forte application. (What is the purpose of this guide? What is Forte used for? Why will Forte make it easy/easier for the reader to develop Java code?)

! Directions on how to launch the Forte application. Assume that the reader already has the Forte program installed on their computer. (What does the environment look likewhen it comes up? What should the reader expect?)

! Separate sections instructing the user in each of the following tasks: opening a new project, creating a source (.java) file, saving a project, building a project, finding and correcting syntax errors, and executing a program. Use an example program (like the "Hello, world!" program that you wrote in Part I) to walk the user through the steps required to complete these tasks. Including an example within the text of a user's guide makes the guide more interesting and understandable to the reader.

! Because the guide should enlighten the reader about the nature of syntax errors and howa programmer identifies and corrects such errors, you should include at least one intentionalsyntax error in your example code. Explain to the user the nature of this error and how to correct the error. (Where do the compilation errors appear in the development environment? What does the error message given to the user mean? Is there a debugger that might help the userto correct errors? How is the debugger used?)

! The guide should explain how to identify correct output for the program. (Where is the output displayed? How should it appear?)

! A conclusion summarizing the guide and providing other appropriate information (such as where the reader can find more information about the topics covered).

The guideshould serve severalpurposes for the reader. It should familiarize the reader with the product, be useful as a hands-on exercise that illustrates the Java program development cycle from file creation to execution, and it should serve as a reference for the preparation of future programs.

Deliverables: The paper should be 3-4 typed double-spaced pages. Standard margins (1 inchat the top, 1 inchat the bottom, 1 ¼ inches on either side) and 12-point Times Romanfont should be used. Any example Java code provided with in the text of the guide should be obviouslydelimited fromthe rest ofthe text and spaced appropriately (as it would appear inthe editor, not necessarily double spaced).

SAMPLE 2:

As thissecond example assignment is given, students have just "reviewed"object-oriented concepts. Because the department is in a transitionalperiod, manyofthe students inthe course had not had significant exposure to object-oriented designprinciples. In a previous homework assignment, I had asked the students to do some research on the Web to find definitions for various object-oriented terms including "encapsulation" and "information hiding". Several students found resourcesthat claimedthat these terms have identicalmeanings. After reviewing the informal writing assignment, I corrected the misconception during lecture and created this formal writing assignment to reinforce the concepts.

Course Goals:

! This assignment encourages students to review object-oriented terminology.
! This assignment emphasizes that information hiding and encapsulation are in fact distinct concepts.

Writing Goals:

! The assignment emphasizesthat all informationfound onthe Web (or even in a textbook) is not factually correct and demonstrates the need to verify sources.
! The importance of appropriate documentation is discussed in class and students are expected to use documentation in the paper.
! In this assignment, fewer guidelines were given (no leading questions) and the students wrote to an audience of peer computer scientists.

CS 321 – Data Structures -- Writing Assignment #2

Project Goals:

! To explore some ideas of importance in object-oriented design and how the ideasrelate to one another
! To apply object-oriented design to a real-world object
! To practice verbalizing technical topics to peer computer scientists

Paper Description:

Object-oriented programming (OOP) differs fromproceduralprogramming. The principles of encapsulation and information hiding are cornerstones of object-oriented program design. Many people incorrectly believe that the terms "encapsulation" and "information hiding" refer to the same concept. Read at least three references on the topic of OOP. At least one of your references must be a source other thanthose givenin the "Possible References" list. After you have consulted your sources, write an essay in whichyou:

! Describe the concepts of a class, an object, encapsulation and information hiding
! Discuss the importance of encapsulation and information hiding to OOP
! Describe the difference between the concept of information hiding and the concept of encapsulation

To demonstrate the usefulness of the object-oriented design philosophy, apply the concepts of a class, an object, information hiding, and encapsulation to an example item. You may choose a radio, microwave oven, or television and describe that real-world object using object-oriented design. Include a discussion of the public interface, data members, and methods of each object. You should NOT write any Java code that would be associated with implementing your chosen object.

Example:
Suppose that we are describing a watch using object-oriented design.
A watch could be an object of the class Clock.
The public interface of the watch could include:
    the nob to set the correct time
    a button that controls the alarm
    a knob to set the alarm time
The data members of the watch object might include:
    hours, minutes, seconds (for the current time)
    hours, minutes and seconds (for the alarm time)
    a flag that indicates whether the alarm is set or not set
Some methods could include:
    changeAlarmStatus(), changeTime(), changeAlarmTime()

There is no correct or incorrect design for each item. Your design will depend on the radios, microwaves or TVs that you are familiar with. Unlike the example above, your paper should contain a narrative response and NOT a list of information.

Specifications:
! This paper should be 3-4 typed, double-spaced pages.
! Audience: Other computer science students who are studying the concepts of object-oriented programming (for example someone who has takencomputer science 2 or data structures).
! Your paper must have a title, introduction and conclusion!
! Your paper should include a list of references that you consulted and contain appropriate citations.