CS704 – Advanced Computer Architecture-II

Solution to Assignment 4

The purpose of assignments is to give you hands on practice. It is expected that students will solve the assignments themselves. Following rules will apply during the evaluation of assignment.

- Cheating from any source will result in zero marks in the assignment.
- Any student found cheating in any two of the assignments submitted will be awarded "F" grade in the course.
- No assignment after due date will be accepted.

Question 1: Total Points (5+10=15)

The Alpha 21264 uses a virtually addressed instruction cache, removing the TLB from the critical path on instruction fetches. The use of virtually addressed caches can reduce time to fetch data from the cache, but can lead to problems (as discussed in lectures).

(a) The 21264 eliminated aliases in the virtually addressed cache by checking eight different locations on each access. Other systems use page coloring to do the same thing. Is this really necessary for instruction caches? Explain.

(b) Virtually addressed caches often need to have more tag bits than physically addressed caches, both because virtual addresses are often longer than physical addresses and because virtually addressed caches need to store additional tag bits to distinguish cache blocks from different processes. How much added overhead does this contribute? Assume 64-bit virtual addresses, 8-bit process identifiers, and a physical memory that can hold up to 64 GB of main memory (i.e., physical tags need only be large enough to handle 36-bit physical addresses). How does overhead vary for different cache block sizes?

Solution:

(a)

The 21264 does not allow writes into the executable segment of the program. The instruction cache is read only. However multiple virtual pages might refer to the same physical page as is the case with shared libraries. Even if multiple aliases are present in the instruction cache, they would all return the same value since no process can write to the cached values.

(b)

The problem states that we have 64-bit virtual addresses, 8-bit process identifiers, and 64GB of main memory. It takes 36-bits to address a location in main memory. Since we do not have any information about cache sizes, we can only say that a virtual address tag would require 28-bits more in space than a physical address tag. In addition, we now have to store an 8-bit process tag since programs can have virtual addresses in common. This gives a total of 36-bits in overhead. If we keep the page size constant, changing the cache block size does not have an effect on the overhead. It would only serve to decrease the number of bits used for the cache index.

Question 2: Total Points (20+5=25)

Prefetching is a technique that allows the "consumer" of data to request the data to its cache before it needs them. With multiprocessors, we can think of the "dual" of

this technique where the "producer" of the data "evicts" the data from its cache after it is done with them. An extension of such "postflushes" could be to send the data to the next processor that needs the data, in cases where that can be determined.

(a) Assume a shared-memory multiprocessor system that takes 100 cycles for a memory access and 300 cycles for a cache-to-cache transfer. A program running on this machine spends 60% of its time stalled on memory accesses and 20% of its time stalled on synchronization. Of these memory stalls, 20% are due to producer-consumer data access patterns where the writer of data can identify the processor that will read the value next. In these cases, producer-initiated communication can directly transfer data to the cache of the next processor needing the data. This will reduce the latency of these memory accesses from 300 cycles for a cache-to-cache transfer to 1 cycle for a cache hit. Another 30% of the memory stalls are due to migratory data patterns where data move from one processor to another, but the migration path is unclear to the source processor. In this case, a producer-initiated communication primitive, such as "postflush," can reduce the latency of the memory access from 300 cycles to 100 cycles. Further assume that all the synchronization is due to tree barriers and that the tree barrier overhead can be reduced by 40% with producer-initiated communication. Assuming no other overheads, what is the reduction in execution time with producer-initiated communication?

(b) What memory system and program code changes are required for implementing producer-initiated communication?

Solution:

(a)

The system spends its time as follows: 60% stalled on memory access, 20% stalled for synchronization, and 20% doing useful work. Producer-initiated communication will not reduce the time spent doing useful work. For the memory stall time 20%, or 12% of the total time, is wasted during producerconsumer access patterns where the writer can identify the next reader. These stalls can be improved from 300 cycles to 1 cycle. Another 30% of the memory stall time, or 18% of the total time, involves producer-consumer access patterns where the writer cannot identify the next reader. Whichever processor is the next reader, the stall time can be reduced from 300 cycles to 100 cycles if the writer postflushes the data from its cache to memory, making the data more quickly accessible for the reader. The remaining 50% of memory access stalls, or 30% of the total time are not helped by producer-initiated communication. Synchronization stalls comprise 20% of the total time, and producer-initiated communication can reduce that by 40%.

Thus,

$$\text{Execution time}_{\text{original}} = \sum_i \text{fraction}_i$$

= Memory stalls + Synchronization stalls + Useful work

= (12% + 18% + 30%) + 20% + 20%
= 1

Execution time $_{\text{enhanced}} = \sum_i \frac{fraction_i}{speedup_i}$

= Memory stalls + Synchronization stalls + Useful work
= (12%/300 + 18% (100/300) + 30%) + 20% (1 - 40%)+ 20%
= 0.68

The total execution time reduction is 32% for a speedup of 47%.

(b)

The cache coherence protocol needs to be changed to include support for producer-initiated communication. However, these changes need to be done with care to avoid the possibility of deadlocks. The program software also needs to be modified to include user-specified hints for producer-initiated communication, just as with prefetching.

Question 3: Total Points (5+5+5+5=20)

The memory consistency model provides a specification of how the memory system will appear to the programmer. Consider the following code segment, where the initial values are A = flag = C = 0.

P1	P2
A = 2000	while (flag == 1) {;}
flag = 1	C = A

(a) At the end of the code segment, what is the value you would expect for C?

(b) A system with a general-purpose interconnection network, a directory-based cache coherence protocol, and support for nonblocking loads generates a result where C is 0. Describe a scenario where this result is possible.

(c) If you wanted to make the system sequentially consistent, what are the key constraints you need to impose?

(d) Assume a processor supports a relaxed memory consistency model. A relaxed consistency model requires synchronization to be explicitly identified. Assume that the processor supports a "barrier" instruction (e.g., the SPARC instruction set), which ensures that all the memory operations preceding the barrier instruction complete before any memory operation following the barrier are allowed to begin. Where would you include barrier instructions in the above code segment to ensure that you get the "intuitive results" of sequential consistency?

Solution:

(a)

Because flag is written only after A is written, we would expect C to be 2000, the value of A.

(b)

Case 1: If the write to flag reached P2 faster than the write to A.
Case 2: If the read to A was faster than the read to flag.

(c)

Ensure that writes by P1 are carried out in program order and that memory operations execute atomically with respect to other memory operations.

(d)

Here C is guarded by the flag variable. We need extra synchronization between the two variables.

P1	P2
A = 2000	while (flag == 1) {;}
Barrier	Barrier
flag = 1	C = A

Question 4: Total Points (5+5+5+5+5+5=30)

In this case study, you will design an I/O subsystem, given a monetary budget.
Your system will have a minimum required capacity and you will optimize for performance, reliability, or both. You are free to use as many disks and controllers as fit within your budget.
Here are your building blocks:

- A 10,000 MIPS CPU costing $1000. Its MTTF is 1,000,000 hours.
- A 1000 MB/sec I/O bus with room for 20 Ultra320 SCSI buses and controllers.
- Ultra320 SCSI buses that can transfer 320 MB/sec and support up to 15 disks per bus (these are also called SCSI strings). The SCSI cable MTTF is 1,000,000 hours.
- An Ultra320 SCSI controller that is capable of 50,000 IOPS, costs $250, and has an MTTF of 500,000 hours.
- A $2000 enclosure supplying power and cooling to up to eight disks. The enclosure MTTF is 1,000,000 hours, the fan MTTF is 200,000 hours, and the power supply MTTF is 200,000 hours.
- The SCSI disks described in Figure 1.1.
- Replacing any failed component requires 24 hours.

You may make the following assumptions about your workload:

- The operating system requires 70,000 CPU instructions for each disk I/O.
- The workload consists of many concurrent, random I/Os, with an average size of 16 KB.

All of your constructed systems must have the following properties:

- You have a monetary budget of $28,000.
- You must provide at least 1 TB of capacity.

	Capacity (GB)	Price	Platters	RPM	Diameter (inches)	Average seek (ms)	Power (watts)	I/O/sec	Disk BW (MB/sec)	Buffer BW (MB/sec)	Buffer size (MB)	MTTF (hrs)
SATA	500	$375	4 or 5	7,200	3.7	8–9	12	117	31–65	300	16	0.6M
SAS	37	$150	1	15,000	2.6	3–4	25	285	85–142	300	8	1.2M

Figure 1.1 Serial ATA (SATA) versus Serial Attach SCSI (SAS) drives in 3.5-inch form factor in 2006. The I/Os per second (IOPS) are calculated using the average seek plus the time for one-half rotation plus the time to transfer one sector of 512 KB.

(a) You will begin by designing an I/O subsystem that is optimized only for capacity and performance (and not reliability), specifically IOPS. Discuss the RAID level and block size that will deliver the best performance.

(b) What configuration of SCSI disks, controllers, and enclosures results in the best performance given your monetary and capacity constraints?

i. How many IOPS do you expect to deliver with your system?
ii. How much does your system cost?
iii. What is the capacity of your system?
iv. What is the MTTF of your system?

Solution:

(a)

If reliability is not a concern, then RAID 0 gives the best capacity and performance; with RAID 0, we waste no space for redundancy to recover from failures and each independent disk can be used to handle a random I/O request. Larger block sizes amortize the positioning costs, while smaller block sizes ensure that only needed data is actually transferred; therefore, the block size should roughly match the request size of 16 KB.

(b)

In our system, we want to purchase as many disks as possible. To best use our budget of $28,000, we therefore maximize the number of disks, given the following constraints:

$$System_{Cost} = CPU_{Cost} + Cntrler_{Cost} \times Cntrler_{Num} + Encl_{Cost} \times Encl_{Num} + Disk_{Cost} \times Disk_{Num}$$

$$Disk_{Num} \leq Encl_{Num} \times 8$$

$$Disk_{Num} \leq Cntrler_{Num} \times 15$$

$$28{,}000 = 1000 + 250 \times Disk_{Num}/15 + 2000 \times Disk_{Num}/8 + 150 \times Disk_{Num}$$

$$Disk_{Num} = 27{,}000/417 = 64.8$$

Since the number of each component must be an integer, we choose

$Disk_{Num} = 64$, $Cntrler_{Num} = 5$ and $Encl_{Num} = 8$

i. Each disk can deliver 285 IOPS. The CPU is not the bottleneck and the 50,000 IOPS controller is not the bottleneck of the system (assuming 15 disks or fewer per string). If all disks are operating concurrently on the random requests, and the disks are the bottleneck of the system, then the storage system can deliver 18,240 IOPS.
ii. $1000 + 250 \times 5 + 2000 \times 8 + 150 \times 64 = 27{,}850$
Note that even though we have $150 remaining, it cannot be used for another disk, because we do not have space in any of the enclosures.
iii. 64×37 GB = 2.31 TB
iv.

Failure rate =

$$(1/1{,}000{,}000) + (5/1{,}000{,}000) + (5/500{,}000) + (8/1{,}000{,}000) + (8/200{,}000) + (8/200{,}000) + (64/1{,}200{,}000)$$

$$= 0.0001573$$

MTTF = 1/(Failure rate) = 6356 hours

Question 5: Total Points (10)

Read the paper **"Temporal Distribution Based Software Cache Partition To Reduce I-cache Misses"** and explain the proposed cache partitioning algorithm.

Solution:

Research-paper based question.