18-548/15-548
Memory System Architecture

Test #3

December 2, 1998

SOLUTION

Instructions: ***DO NOT OPEN TEST UNTIL TOLD TO START***
YOU HAVE UNTIL 10:20 AM TO COMPLETE THIS TEST

The test is composed of three problems containing a total of 11 sub-problems, adding up to 100 points overall. The point value is indicated for each sub-problem. Attempt all problems and budget your time according to the problem's difficulty. Show all work in the space provided. If you have to make an assumption then state what it was. Answers unaccompanied by supporting work will not receive full credit. The exam is closed book, closed notes, and "closed neighbors." You are on your honor to have erased any course-relevant material from your calculator prior to the start of the test. Please print your initials at the top of each page in case the pages of your test get accidentally separated. You may separate the pages of the test if you like, and re-staple them when handing the test in.

Problem 3d is multiple choice – don't leave it blank!

Good luck!

1. Vector Processing

You have a chaining vector computer with the following bandwidths available for 8-byte data values:

- 7 memory banks at 90 Million B/sec each
- One bus at 720 Million B/sec
- A VDS with three concurrent inputs to arbitrary outputs at 360 Million B/sec each
- A VRF with three bidirectional ports at 360 Million B/sec each
- A pipelined vector multiplier at 45 MHz; 3 clock latency; 1 result per clock throughput
- A pipelined vector adder at 45 MFLOPS; 3 clock latency; 1 result per clock throughput
- Recall that DAXPY performs the vector operation: Y = aX + Y on double-precision (64-bit) floating point numbers.
- The vector unit has a latency of 35 clocks for the first result of a DAXPY, and completes further DAXPY results at the maximum speed possible consistent with bandwidth limitations. To simplify the problem:
  - Ignore bank conflicts and bandwidth for the scalar constant fetch.
  - Ignore any detailed timing effects that require drawing a spreadsheet timing chart.
- A scalar floating point coprocessor that can compute one DAXPY result every 9 clocks.

a) (12 points)
What is $R_{infinity}$ (infinite length vector speed) on this architecture for DAXPY operations? Show bandwidth for the following subsystems to receive full credit:

Total for Functional Units: ____1080_____ Million B/s (assumes single DAXPY instruction)

VDS: _____1080_____ Million B/s

VRF: _____1080_____ Million B/s

BUS: _____720_____ Million B/s

Memory: _____630_____ Million B/s

Compute $R_{infinity}$: _____52.5_____MFLOPS for DAXPY
DAXPY requires two loads and a store per vector element. At full speed the multiplier/adder pair produce one result per clock after startup latency.
Fn units: 2 inputs & 1 output per clock = 3 * 8 * 45 Million = 1080 Million B/s
VDS: 3 x 360 Million = 1080 Million B/s; VRF: 3 x 360 Million = 1080 Million B/s
Bus: 2 x 360 Million = 720 Million B/s (a bottleneck);
Memory: 7 x 90 Million = 630 Million B/s (a worse bottleneck!)
$R_{infinity}$ is limited by memory bandwidth to 630 Million B/s =>
630 Million/((3 operands/2 FLOPS) * 8bytes/operand) = 52.5 MFLOPS $R_{infinity}$

b) (10 points)
What is $N_{1/2}$ (**vector length** at which half of $R_{infinity}$ is attained) on this architecture for DAXPY operations?

35 clocks for first result + 45/(52.5/2) = 1.71 clocks per result after first; and 2 FLOPS per result;
clocks = 35 + 1.71(vector_length-1); rounded up
achieve MFLOPS = 2* 45 Mil. * #flops / #clocks

$N_{1/2}$ occurs when achieved MFLOPS = 52.5 / 2 = 26.25

Vector length	Clocks	Achieved MFLOPS
1	35	2.57
2	37	4.90
3	38	7.03
4	40	8.97
5	42	10.75
…		
19	66	25.97
20	68	26.65

Interpolating: (x – 19) / (26.25 – 25.97) = (20-19) / (26.65 – 25.97)
$N_{1/2}$ = 19.41

c) (10 points)
What is N_V (**vector length** at which vector is equal to scalar speed) on this architecture for DAXPY operations (interpolate as necessary; full credit result is not an integer value)?

In scalar mode the processor achives (2 Flops/result) * 45 MHz / (9 clocks/result) = 10 MFLOPS

Interpolating from the table for part (b) (although it can be done other ways):

(x – 4) / (10-8..97) = (5-4) / (10.75 – 8.97)
$N_{1/2}$ = 4.58

2. Error Correcting Codes & Buses

You are doing a bus tradeoff study for a noisy system environment in which you expect there to be frequent single-bit errors in transmission on the bus. The system characteristics are:

- 64-bit data bus used to fetch 256-bit cache blocks from memory (we're going to ignore writes)
- 40-bit address bus
- 25 control signals, power, ground, etc. (i.e., 25 "pins" in addition to data and address)
- Cost per card for a bus connection = $3.50 + $.01/"pin"
- Bus uses burst transfers (and is *not* a split-transaction bus, since people will ask…)
- Latency is 6 clocks to read the first 64-bit datum from memory; one 64-bit datum per clock for additional transfers in burst mode after the first datum is returned

a) (10 points)
Your boss tells you to implement the system by putting one bit of parity for every 8 bits of data bus and address bus (i.e., "per-byte parity" on both address and data bits). How much will it **cost in dollars** per card to build this bus connection?

8 bytes on data bus + 5 bytes on address bus = 13 bits parity

64 + 40 + 13 + 25 = 142 pins on bus; $3.50 + ($.01 * 142) = $4.92 per card

b) (7 points)
Using per-byte parity as in part (a), it is possible to detect faulty data but not correct it. Thus, an entire 256-bit cache block must sometimes be fetched a second time to get its correct value when the first attempt fails due to a parity error. What is the **average memory access latency** for this system (**in clocks**)? Assume:

- 1% of reads suffer a parity error (ignore the possibility of multiple bit errors per message)
- A 2 clock penalty after receiving faulty data before the address is re-sent for another try
- Retries always succeed (not realistic, but simplifies the math for exam purposes)
- Reads always transfer all 256 bits of data, even if an error is detected partway through

6-1-1-1 timing = 9 bus clocks/result

9 + (1/100 * (2 + 9)) = 9.11 clocks average latency

c) (8 points)
You decide to use a SECDED coding scheme (Single Error Correction, Double Error Detection) to eliminate the time delay associated with retries. How many **total bus pins** are required on the bus assuming that you get rid of the byte parity bits and that you re-use the same bus pins for both the address and data SECDED codes?

40 bit address bus takes 6 bits for SEC + 1 bit extra to attain SECDED = 7 bits
(log2 (40 + 6 + 1) = 5.55 < 6)

64 bit data bus takes 6 + 1 (because of overflow past 64 bits) for SEC; 8 bits for SECDED max(7,8)= 8 extra bits.

Total bus pins = 64 + 40 + 8 + 25 = 137 pins (a savings of 5 pins, and faster performance!)

d) (8 points)
The ancient Sumerian god of error correct codes appears in a rather surreal dream and suggests that you use the following scheme instead: Use a SECDED coding scheme on the entire 256 bit value returned for data and break it up into multiple parts, sending portions of the code on the bus while the data is returning. Thus, the SECDED code would be sent in 4 bus cycles, with approximately one fourth the bits being sent on each cycle. Now what is the minimum number of **total bus pins** required? (Ignore having to do error checking at all on the address bits; you can get them "for free" if you're clever but that isn't the point of this problem.)

256 bits requires 8 + 1 (because of overflow past 256 bits) = 9 bits for SEC; 10 for SECDED. Over 4 clocks that's 3 bits/clock (3 + 3 + 2 + 2 = 10 for 4 data clocks).

Total bus pins = 64 + 40 + 3 + 25 = 132 pins (an additional 5 pins of savings)

Not required, but you ended up saving $.11 per card over the baseline (assuming the error correcting logic doesn't increase component cost) and get a faster product as well.

3. Storage & Coherence

When you get to industry, you find out that your boss is Erik Riedel (just your luck…) Due to the Great Stock Crash of January 1st, 2000, there are no jobs and you have to do what he says or go sell apples on the street corner with all the unemployed Y2K consultants.

The problem Erik is working on is 3-D atmospheric modeling involving grotesquely large data sets. It turns out that Erik wants to do a computation on a single 3D data array 16K on a side (16K x 16K x 16K x 8 bytes). He's going to have you build a network of disks with local processors to store the data and compute results.

a) (8 points)
The disk drives spin at 7200 RPM. Given the way the program operates, on a typical disk access the heads must move 213 tracks at a rate of 5 microseconds/track plus 250 microseconds for settling (a very simple time model for exam purposes). What is the **average latency** for an access in milliseconds?

Average latency is ½ a disk rotation plus seek+settling time =
(0.5 rev * 60 sec / 7200 rev) + 250e-6 + (213 * 5e-6) = 5.48 msec

b) (7 points)
Assume 512 bytes/sector; 270 sectors/track; 4500 tracks/surface; 16 surfaces. What is the **minimum number of disk drives** required to store the array (ignore any file system overhead; just worry about storing the data itself)?

16K*16K*16K*8 = 32TB and the drives hold 512x270x4500x16 ~= 10 GB,
so array size / drive size => 3535 drives.

c) (4 points)
Because the network is a potential bottleneck, each disk will hold a perfectly square cubic sub-array of the main data array with an overlap of exactly one data element with each adjacent cube (remember this is a 3-D array). For simplicity, each of these cubes will be 1025x1025x1025 in size. This means that each disk holds 1023x1023x1023 of private data and a "skin" one cell deep on all six sides of data shared with adjacent cubes (don't bother making a special case of what happens at the very outside boundaries of the main data array). The importance of this design approach will become obvious in the next sub-problem.

Obviously, the disks won't be entirely full, but this will make the software job easier. Now how many disks are required? (It may help to draw a sketch… but that won't be graded.)

Each disk will effectively hold a 1K x 1K x 1K cubic sub-array of data. That gives a drive topology of 16 x 16 x 16 disks for the raw array = 4096 disks

d) (16 points – the other shoe drops -- but it's "multiple guess!")
Evil Erik tells you to build a coherence protocol for that distributed disk system that treats every sector of a disk drive as if it were a memory element subject to being shared the same as a coherent cache. That way he can do computions on data local to each disk and let the coherence mechanism take care of any shared data in the areas in which the subarray overlap (remember, they'll overlap only on the "skin" of each sub-array). In particular, you have to build a MESI protocol, where the four states correspond to:

M = modified (modified data; exclusive copy)
E = exclusive (unmodified data; exclusive copy)
S = shared (unmodified; more than one disk has a copy)
I = invalid

(Recall that in class we studied a 3-state "ESI" protocol – this protocol adds an extra state called "modified" compared with what we discussed in lecture.)

Since this machine is only going to run one program, we can customize the system to know that only the "skin" of each sub-array can possibly be shared. Thus, each disk sector is appropriately initialized to be "Shared" if on the skin of that disk's sub-array, or "Exclusive" if not on the skin.

Fill in the appropriate state transitions in the diagram below.

- Assume a snoopy bus implementation with write invalidation, but customize the protocol for this particular problem to minimize bus traffic.
- "Bus read" means some other disk reads the data for that particular disk sector; "processor read" means the processor local to the given disk reads that particular disk sector, and similarly for the writes.
- *Every blank box must have an **M**, **E**, **S**, or **I** in it.* – that's the multiple choice aspect of this problem. The box contains the "next state" information for the finite state machine given the current state and the activity occurring. Grading will be strictly based on the letter in each boxes (go ahead and guess if you feel a need to do so – this is multiple choice!).
- Probably you'll want to sketch a state transition diagram for your own understanding – but it won't be graded.

Current State	Bus Read	Bus Write	Processor Read	Processor Write
M odified	S	I	M	M
E xclusive	---	---	E	E
S hared	S	I	S	M
I nvalid	I	I	S	M

The two "---" boxes are freebies – they can never happen so they're "don't care", although the defensive way to design the system is to make them transitioned to "Shared".