

Homework 2*Reading: Textbook, Finish Chapter 1, Begin Chapter 2**Due: Feb. 8, 2006***Problem 1**

Continue to work on Homework 1, Problem 2 if you were unable to complete it on time.

Problem 2 (0 Points)

If you are unfamiliar with basic Unix, the course web page contains a link to *An Introduction to Linux* by Machtelt Garrels. I suggest reading Chapters 1, 2, 3 and 6 (if you would rather use the emacs editor instead of vim, use www.google.com to find an emacs tutorial). Chapters 4, 5 and 7 will also be eventually helpful.

Problem 3 (0 Points)

Consider a 1 GHz CPU that has one instruction pipeline with five (5) stages. Suppose that each stage can execute in one clock cycle when not accessing main memory.

- a) What is the maximum instruction rate of this machine expressed in MIPS (Millions of Instructions Per Second)? Explain.
- b) Consider a main memory that causes the CPU to enter the wait state for M clock cycles and a program that must access main memory for data every N instructions. Derive an expression for the MIP rating for the above CPU and memory system. Assume that all instructions following the memory access instruction must wait for the memory operation to complete. Explain how you arrived at your answer.
- c) Suppose $M = 50$. For what values of N will the MIP rate in Part b be atleast 50% of its maximum rate found in Part a?

Problem 4 (0 Points)

Consider a memory hierarchy and program with the following access times and hit ratios:

- Cache Memory: 20 nsec with $h = 0.9$
- Main Memory: 40 nsec with $h = 0.6$
- Disk: 12 msec with $h = 1.0$

- a) Derive an expression for the EMAT (effective memory access time) of a 3-level memory hierarchy.
- b) What is the EMAT (effective memory access time) of this memory hierarchy?

Problem 5 (0 Points)

Suppose that the processor is executing a program that is running at the speed of the main memory; i.e., its progress is indicated by the amount of memory that it has accessed. The main memory is a 100 MHz 5-5-5-5 DRAM that accesses memory in 64-bit quantities. The notation 5-5-5-5 here means that it takes 5 memory cycles to read 64 bits (8 bytes) from memory, and each consecutive 8 bytes requires another 5 memory cycles.

- a) How long will it take to access 32 consecutive bytes from memory?
- b) Suppose that this system is used primarily to forward incoming network packets; i.e., most of the work involves copying data to and from main memory and handling packet interrupts. Each packet is N bytes long. Packet forwarding involves each byte of each packet to be accessed four (4) times. In addition, $1 \mu\text{sec}$ (microsecond) of CPU overhead is required to process each packet. Derive an expression for the packet forwarding rate in Mbps (megabits per second) of the system.
- c) What is the memory speed-up offered by 5-1-1-1 memory for consecutive byte accesses? Note 1: The notation 5-1-1-1 here means that it takes 5 memory cycles to read the first 64 bits (8 bytes) from memory, and then only 1 memory cycle to read each of the next three consecutive 8 bytes. This access time of 5-1-1-1 repeats again for the next 32 consecutive bytes. Note 2: Ignore CPU overhead here.
- d) What effective speed-up can be attained by replacing the 5-5-5-5 memory with 5-1-1-1 memory? We define the effective speed-up here as the speed-up when all overheads are considered.
- e) Plot the forwarding rates as a function of packet length for packet lengths of 64, 128, 256, 512, 1024, and 2048 bytes for the two memory systems.

Problem 6 (6 Points)

Consider the following code fragment:

```
int    x[N];
for (register int k=0; k<K; k++) {
    for (register int i=0; i<N; i += stride)
        { x[i] = x[i] + 1; }
}
```

Assume the following:

- *Memory system:* The main memory bus is 64-bits wide and runs at 100 MHz. Sequential memory accesses occur in bursts of 5-1-1-1 bus cycles. There is an instruction cache that is large enough to hold a typical code loop and a 16 KB L1 data cache that can deliver *8-byte aligned* 64 bits to the CPU in 1 nsec. Suppose that the memory is read in 32-byte chunks (called a cache line) aligned on an integer multiple of 32. It takes $T_1 = 1$ nsec to read an integer from cache to a register.
- *Write-back (WB) cache.* The cache is a WB cache. A write to a WB cache is not pushed out to main memory until absolutely necessary; i.e., when there is no more room in the cache. So, if all of the data fits entirely in cache, there will be no write until the program is done.

- K is very large; i.e., initial transients will not appear in your solution.
 - N is very large (e.g., 2^{20}).
- a) If an integer is 32 bits, how large is the x array.
 - b) What is the EMAT (Effective Memory Access Time) for the program fragment when the **stride** is 1; i.e., the stride is one integer?
 - c) Consider for the moment only small stride values of $S = 1, 2, 4, 8$. Construct a table showing the values of S , H (hits) and M (misses) for the code fragment above, and then derive a general equation for the EMAT when N and K are very large. Note that the simplified form of this equation should be a function of S .
 - d) Derive an equation for the EMAT for larger values of the stride S (e.g., 10, 12, 14, 16) when N and K are very large.
 - e) Derive a general equation for the EMAT for the stride values considered above when the time to get one integer from cache is $T1$ and the time to get eight integers from memory is $T2$. HINT: Your result should be piecewise linear.
 - f) What value of N would result in a perfectly flat curve? Explain.

Problem 7 (6 Points)

The CS422 Web page contains a link to the C program `membench.c`. Compile this program; run it on your favorite machine; and answer the questions below. Note: I have successfully compiled the program using `gcc` on a SPARC 5 (Solaris 2.7), a Pentium II (NetBSD 1.3.2), and a Pentium III (Redhat Linux 7.0).

- a) List the type of machine and OS you used.
- b) Describe the memory access pattern of the `membench.c` program when $csize = 128$ and $stride = 1$. Repeat for $csize = 2^{20}$ and $stride = csize/2$.
- c) Run `membench` on the machine, and plot the measurement results. The y-axis should show the read+write time, and the x-axis should show the memory stride (in a logarithmic scale). There should be one line for each cache size. You can use your own plotting tool or the script on the CS422 Web page. The script uses `gnuplot` to produce a Postscript file that can then be displayed using `ghostview`. Submit the plot.
- d) It is worthwhile trying to relate the results from the preceding problem to the plots in Part c even though there are differences in the memory configurations. Consider the top curve (i.e., the largest $csize$). Explain how the curve qualitatively matches the results of the preceding problem for the four smallest strides. Repeat for the four largest strides.
- e) Which array sizes and which stride values demonstrate good temporal locality? Explain.
- f) The plot appears as a family of concave curves. It would be interesting if we could determine some of the cache size(s) from the curves. What is the size of the smallest cache as shown by the curves? Explain.

Problem 8 (3 Points)

The purpose of this problem is to get a rough estimate of the time for a system call and a function call. Because this involves the measurement of a real system, the experiment will need to be repeated multiple times to have any hope for getting representative values. Note: You do not have to subtract out the time for loop overhead.

You will want to also read the man pages on the system calls `gettimeofday(2)` and `uname(2)`.

- a) The course web page contains a partial C and a partial C++ program for this problem. Complete the C or C++ program called `timeit` to determine the time required to make the system call `uname(2)` and the time required to call a stub function (i.e., the function has an empty body). Note that `uname(2)` is the system call (see `man 2 uname`), not the Unix command line `uname(1)`. Use the system call `gettimeofday(2)` to measure the elapsed time. Note that in each case, you will want to obtain the time $T(N)$ to make N calls and then compute the average time; i.e., $T(N)/N$. The program should do this for values of $N = 10^k$ for $k = 2, 4, \text{ and } 6$. Submit a source listing.
- b) Run the program three (3) times, and provide output and a table showing the time it takes to call `uname` and the time to call the stub function from the three runs. Which machine did you use to make this measurement? How consistent are the results for the different values of N ? From one run to another?
- c) How much slower is the system call as compared to the function call? Why is a system call substantially slower than a function call?