Preview:
Problems 5 and 6 examine the effects of temporal locality on cache memory performance. Problem 5 should help you understand the output generated in Problem 6 by developing an equation for RWT (Read-Write Time) of an integer which is the approximate time it takes to read an integer, do a simple arithmetic operation on the integer, and then write the new integer back to memory. It also examines the effect of architectural features (e.g., cache line size) on the EMAT.

In Problem 6, you compile and run a program that attempts to measure the RWT of an integer. The results should reinforce some of the theoretical analysis done in Problem 5 and begin to give you a sense of typical memory access times and how they are affected by memory reference patterns.

Problem 7 should begin to develop your sense of the time required to perform different types of operations (e.g., system calls). It should also develop your skill in using the gettimeofday function to measure the passage of time.

Problem 1
Continue to work on Homework 1, Problem 4 if you were unable to complete it on time.

Problem 2 (0 Points)
[This is an expanded form from Homework 1. ] 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)
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 the 5-5-5-5 DRAM? Explain.

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 µsec (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, and so on for each group of 32 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 5 (6 Points)**

Consider the following code fragment:

```c
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 effectively 64-bits wide and runs at 200 MHz. Sequential memory accesses can be burst out such that after a 4 cycle latency 64 bits are available in each of the next 4 cycles; i.e., 5-1-1-1 memory. 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 \( T1 = 1 \) 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) Suppose that \( x[i] = x[i]+1 \) is implemented as:

```
Register <- Memory   // read from \( x[i] \)
Increment Register   // + 1
Register -> Memory   // write to \( x[i] \)
```

If the \textit{stride} is 2, describe the memory reference string (sequence) and cache hits and misses for the inner loop. For example, if the \textit{stride} is 1 and we ignore memory writes, the sequence is:

\[
M \ H \ldots \ H \ M \ H \ldots
\]

\[
x[0], x[1], \ldots, x[7], x[8], x[9], \ldots
\]

That is, the inner loop references consecutive elements of \( x[] \) with every eighth reference causing a cache miss. But note that for this part of the problem, you need to consider the case of \textit{stride} 2 instead of 1 and reads and writes, not just memory reads.

b) Let \textit{RWT} (Read-Write Time) be the average time to do the memory operations in one iteration of the inner loop; i.e., read \( x[i] \) followed by write \( x[i] \). What is the \textit{RWT} for the program fragment when the \textit{stride} is 2 (i.e., the \textit{stride} is two integers)?

c) Consider for the moment only small \textit{stride} values of \( S = 1, 2, 4, 8 \). A student was asked to derive a general equation for the \textit{RWT} when \( N\) and \( K\) are very large. Their solution is shown below:

I will use the general \textit{EMAT} equation because the \textit{RWT} is just a read and a write:

\[
\text{RWT} = 2 \times \text{EMAT}
\]

\[
= 2 \times \left[ H \times T1 + M \times (T1 + T2) \right] / (H+M)
\]

\( T1 = 1 \) nsec, \( T2 = 40 \) nsec:

\[
\text{RWT} = 2 \times \left[ 1 \times H + 41 \times M \right] / (H+M)
\]

\[
\begin{array}{ccc}
S & H & M \\
--- & --- & --- \\
1 & 7N/8 & N/8 \\
2 & 3N/4 & N/4 \\
4 & N/2 & N/2 \\
8 & 0 & N \\
\end{array}
\]

\[
\text{RWT}(S=1) = 2 \times \left[ 1 \times 7N/8 + 41 \times N/8 \right] / N
\]

\[
= 12/N
\]

\[
\text{RWT}(S=2) = 2 \times \left[ 1 \times 3N/4 + 41 \times N/4 \right] / N
\]

\[
= 22/N
\]

\[
\text{RWT} = 2 \times (1 + 5S), \quad S = 1, 2, 4, 8
\]
This solution is unclear because there is almost no explanation of how the student got from one step to the next. Give a better solution and briefly explain how your solution is an improvement.

**Problem 6 (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), a Pentium III (Redhat Linux 7.0), Pentium M (Linux 2.4), AMD Opteron (Linux 2.6), and many other machines.

a) List the type of machine and OS you used to run the `membench` code.

b) The `membench.c` source code includes the following comment:

```c
/* repeat empty loop to subtract loop overhead */
```

In one or two sentences, explain how the code accomplishes that.

c) Suppose that $m_1, m_2, \ldots$ is a sequence of memory references (i.e., addresses). A measure of temporal locality associated with a memory address $m_i$ is the *temporal distance* $D(m_i) = k-i$ where $k$ is the minimum positive integer such that $m_k = m_i$; i.e., how many references into the future the memory address reappears. After reading the source code for the `membench` program, discuss whether $D()$ is a good indicator of temporal locality. In particular, consider the two cases: 1) $csize = 128$ and $stride = 1$; and 2) $csize = 2^{20}$ and $stride = csize/2$.

d) Run `membench` on the machine, and plot the measurement results. The y-axis should show the read+write time (RWT), 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.

e) The previous problem showed the dependence of RWT on the cache line size. From your plots in Part d, what is the cache line size of the L2 cache? Explain.

f) It is worthwhile trying to relate the results from the preceding problem to the plots in Part d 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.

g) Why are the RWTs so small for the four largest strides?
**Problem 7** (6 Points)
The purpose of this problem is to get rough estimates for the time of a function call (stub), a system call (gettimeofday) and process creation (fork). 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.

The course web page contains a partial C/C++ program called `timeit.c` that does most of what you need for this problem. **You should only need to insert a few lines of code to complete it.** Try to understand what the code does before spending any time changing it. Also, read the man pages on the system calls gettimeofday(2), fork(2), wait(2), system(3) and uptime(1).

a) Complete the `timeit.c` code so that it can determine the time required to do the following operations:

- A call of a stub function (i.e., the function has an empty body)
- The system call gettimeofday(2)
- The system call fork(2) (followed by a call to wait(2))

Note that we will use the system call gettimeofday(2) to measure the elapsed time (even of the system call gettimeofday(2)! 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 $k = 1, 2, 3,$ and $4$. Submit your source listing and output.

b) Run the program three (3) times, and provide output and a table showing the time it took to do the three operations to call the stub function from the three runs. How consistent are the results for the different values of $N$? From one run to another?

c) How much slower is the fork call as compared to the gettimeofday call? How much slower is the gettimeofday call as compared to the function call? Explain why the results make sense.

d) A student has written the following code to display "hello" 100 times every 50 usec (microseconds):

```c
struct timeval oldt, t;
int doit = 1;
gettimeofday( oldt, 0 );
for (int i=0; i<100; i++) {
    printf("hello\n");
    while (doit) {
        gettimeofday( t, 0 );
        if ( (t.tv_usec - oldt.tv_usec) >= 50000 ) doit = 0;
    }
    oldt.tv_usec = t.tv_usec;
}
```

Discuss how likely (or unlikely) this code will do what the student expects.