

Homework Assignment #1

Due: Friday, 04/23/2010, 5pm @ Gates Hall 305

Please work in groups of 3 students

Instructions: Submit to the box outside Gates 305 by the due date above. Show your work, state your assumptions, and justify your answers to receive full credit; we prefer concise, correct explanations. Check the Bulletin Board for corrections and updates.

Homework Policies: Collaboration on homework assignments is encouraged subject to the Honor Code and the following guidelines:

No more than three (3) people may collaborate on a homework solution.

- A group should submit a single solution.
- Any assistance received should be acknowledged in writing on the assignment.
- All students should work on all problems.

SCPD students will receive graded homework assignments back through SCPD. There will be no extensions or exceptions to the submission deadline. Solutions to the HW will be posted on the class webpage a few hours after the submission deadline.

Problem 1 (10 points) – Benchmarks

Chip	# of cores	Clock Frequency (MHz)	Memory performance (MIPS)	Processor performance (MIPS)
Athlon 64 X2 4800+	2	2,400	3,423	10,359
Pentium EE 840	2	2,200	3,228	9,496
Pentium D 820	2	3,000	3,000	7,610
Athlon 64 X2 3800+	2	3,200	2,941	8,565
Pentium 4	1	2,800	2,731	7,621
Athlon 64 3000+	1	1,800	2,953	7,628
Pentium 4 570	1	2,800	3,501	11,210

Imagine that your company is trying to decide between a single-processor system and a dual-processor system. The figure above gives the performance on two sets of benchmarks – a memory benchmark and a processor benchmark. Both of the performances are inversely proportional to the execution time. (MIPS = million instructions per second) Assume that the performance benchmark is designed to use only

a *single* core. (Note: These performance numbers are not real values. Please refer to them only for this problem.)

- a) How much speedup do you anticipate getting if you move from using a Pentium 4 to an Athlon 64 X2 3800+ on a CPU-intensive application?
- b) Your target application will spend 20% of its time on memory-centric computations, and 80% of its time on processor-centric computations. How much speedup do you anticipate getting if you move from using a Athlon 64 X2 3800+ to a Pentium 4 570 on your target application?
- c) One day, you have found that the *processor-centric part* of your target application of b) can be fully parallelized and you can run the parallelized program on as many cores as you want. Which processor would you choose, Athlon 64 X2 3800+ or Pentium4 570? How much faster is the processor you've chosen than the other one on your parallelized target application?

Problem 2 (15 points) – Basic Cache Review

You are building a system around a uniprocessor with in-order execution that runs at 2.8 GHz and has a CPI of 0.8, excluding memory stalls. The only instructions that access memory are load (25% of all instructions) and store (4% of all instructions). The CPI of 0.8 includes the time that load and store instructions spend in the processor pipeline (in the decode stage, etc.) but not the extra time to fetch data from the memory subsystem. This system has a split L1 caches (separate I-cache and D-cache), and a unified L2 cache.

L1 cache The L1 cache has a hit time of zero; that is, a hit is serviced at the same cycle and it is already incorporated into the CPI above. Each of the I-cache and D-cache is 32KB direct-mapped cache. The I-cache has 3% miss rate and 32-byte blocks. The D-cache is write-through, no write-allocate with 10% miss rate and 16-byte blocks. The D-cache has a write buffer that eliminates stalls for 94% of all writes.

L2 cache The L2 cache is 512 KB and write-back, fetch-on-miss, with 64-byte blocks. Its access latency is 15 ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266 MHz and can transfer one 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 82% are satisfied without going to main memory. Also, 47% of all blocks replaced are dirty and need to be written back

Main memory The 128-bit wide main memory has an access latency of 70 ns, after which any number of bus words may be transferred at the rate of one per bus cycle on the 128-bit, 133 MHz main memory bus.

Assume there is no bus contention, miss penalties are cumulative, delivery is in address order, and the stated bus latency includes all possible extra overheads before the

access can be made. You don't need to think about other levels of memory hierarchy, such as disks.

- a) Calculate the average memory access time for instruction accesses.
- b) Calculate the average memory access time for data reads.
- c) Calculate the average memory access time for data writes.
- d) Calculate the overall CPI, **including** memory accesses.
- e) If you upgrade the CPU to a new model with a 3 GHz processor that is otherwise identical, how much faster would the new system be?

Problem 3 (15 points) – Optimization Techniques

The transpose of a matrix interchanges its rows and columns and is illustrated below:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \rightarrow \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{41} \\ A_{12} & A_{22} & A_{32} & A_{42} \\ A_{13} & A_{23} & A_{33} & A_{43} \\ A_{14} & A_{24} & A_{34} & A_{44} \end{bmatrix}$$

You are executing a 256 x 256 double-precision (64-bit) matrix transpose, using the following code:

```
for (i = 0; i < 256; i++) {
    for (j = 0; j < 256; j++) {
        output[j][i] = input[i][j];
    }
}
```

Both the input and output matrices are stored in the row major order; a multi-dimensional array in linear memory is accessed such that rows are stored one after the other. Thus, accessing `input[i][j]` is the same as accessing memory at the address of 'address(input) + 256×i + j'.

You are running this program on a processor with a 16 KB fully associative (so you don't have to worry about cache conflicts) L1 data cache with perfect LRU replacement. The L1 block size is 64 bytes. L1 is a write-back, write-allocate, fetch-on-miss cache. L1 misses or prefetches always hit in the L2 cache, and require 16 cycles to refill a cache lines. The L2 cache can process a request every 2 processor cycles. Unrealistically assume that writing back a dirty cache block requires 0 cycles, and there is no limit on the number of pending L2 write requests allowed, and no latency required between write requests.

When the output and input data are in cache, each iteration of the inner loop (the element transpose, evaluating the branch condition, and branching/not branching) takes 4

cycles. Each iteration of the outer loop takes 3 cycles in addition to the execution time of the inner loop.

- a) With the code above, what is the execution time of the first iteration of the outer loop?

Your system has no automatic hardware prefetcher, but it allows software prefetching. The processor can issue up to 2 prefetch instructions per cycle; assume, for the sake of simplicity, that it can't issue any other instructions during a cycle when it issues a prefetch. Up to 32 prefetch requests can be outstanding; outstanding requests to the same cache block will be merged. You may assume a nonfaulting prefetch, which means that you can issue prefetch instructions to load data from out-of-bound addresses without causing any exception.

Prefetches of the forms,

```
prefetch (input[0][0])
prefetch (input[i][j])
prefetch (input[0][i+const])
```

are all allowed and all take the same amount of time.

- b) Modify the code above to perform software prefetching. Try to improve performance as much as possible. With your new code, what is the execution time of the first iteration of the outer loop?

You are now asked to optimize this code on a machine that does not support prefetching. You decided to cut the input and output matrices into small square submatrices or blocks, in order to reduce the cache miss rate. (It is called 'blocking', which you can find more details at H&P, chapter 5.2)

- c) Modify the original code above to compute the transpose using blocks of $B \times B$ elements. What is the range of the block size, B , which gives you the maximum speedup? You can assume that B is a power of 2.

Problem 4 (20 points) – Unknown Memory System

You have a task to optimize a program on a certain machine, but you had no idea what kinds of techniques are used in your computer system. Thus, you decided to read the hardware specification of the computer so that you can apply sophisticated techniques. However, the most of the parts in the memory system chapter have been erased! Here are the facts that you could read on the memory system chapter:

- It has a split 16KB/16KB L1 I-cache and D-cache.
- Both L1 caches are direct-mapped.
- L1 cache block size is 64 bytes.

- There are no L2 or L3 caches.
- L1 cache hit time is t_{hit} , and the main memory access time is t_{mem} . As usual, t_{mem} is larger than t_{hit} in the order of magnitude.

You are eager to know about the other details regarding to the memory system, and optimize your program even better. Describe a short program you could run on the machine in order to answer the following questions. You are required to write a short and simple pseudocode for each question and also required to explain how it helps to answer the question as well as any assumptions you made.

- Is the L1 cache write-through or write-back?
- Does the L1 cache have a victim cache?
- Is the memory system blocking or non-blocking?
- Suppose the memory system is non-blocking. How many outstanding memory references can be supported? You can assume that the number of outstanding memory references is less than 32.

Problem 5 (20 points) – The Sum of All Beers (15 points)

You work for a brewery that keeps meticulous inventory. In particular, you have an array representing the beer stock as it is distributed around the country, and an application that sums up the elements in that array:

```
accum=0;
for (i=0; i<8192; i++) {
    for (j=0; j<8192; j++) {
        accum += input[j][i];
    }
}
```

`input` is a double-precision (64bit) array of size 8192×8192 ; `accum` is a double-precision scalar variable. The array is laid out in row-major order. You can find the meaning of ‘row-major order’ at the Problem 3.

You also have a main memory system (no caches). It has 64 64-Mbit DRAM chips. Each chip has an 8-bit interface.

The address for each DRAM chip is 23 bits long. 13 bits are used for the row access (RAS) and 10 bits are used for the column address (CAS). Thus, each DRAM chip can provide up to 1024 distinct 8-bit values from each row when in fast page mode.

The latency for a row access to a DRAM chip is 20 ns (RAS time). The start of a row access to a different row must be spaced apart by 30 ns (row cycle time) from the start of the most recent access to the current row. A column access for an 8-bit word in a row opened by a row access can start immediately after the row access.

The latency for an 8-bit column access to an open row is 4 ns (CAS time). At the end of these 4 ns, the data are available at the pins of the DRAM chip. Back-to-back column

accesses to the same row must be spaced apart by 6 ns; again, this is the spacing between the initiations of accesses.

If a row is accessed within a DRAM chip, that row will remain open until a new row is accessed. Thus, any subsequent accesses to that row require only column accesses.

The processor produces a 32-bit address for each array element. Each address is sent to the memory controller, where it is translated to the proper number of DRAM chip accesses. The memory controller does not reorder accesses.

- a) Assume each DRAM chip consists of one bank, and that each DRAM chip is a rank. There is a single 8-bit data bus that connects all DRAM chips to the memory controller. In addition, the array has been laid out across chips so that there is no element interleaving; each DRAM chip contains a contiguous set of array elements.

In this arrangement, the memory address consists of

- Bits [31:26] chip select
- Bits [25:13] row select
- Bits [12:3] column select
- Bits [2:0] unused

Note that bits [2:0] are literally not used. You can read each byte by alternating column bits, [12:3]

For the code above, what is the total DRAM latency? You want to know immediately if any beers have been lost, so rewrite the code to minimize DRAM latency for this configuration, and calculate the new latency.

- b) You have rewritten your application to access the array by rows rather than by columns:

```
accum=0;
for (i=0; i<8192; i++) {
    for (j=0; j<8192; j++) {
        accum += input[i][j];
    }
}
```

Assume each DRAM chip consists of 4 banks, and that each DRAM chip is a rank. There is an 8-bit data bus that connects all DRAM chips to the memory controller. Again, the array has been laid out across chips so that there is no element interleaving; each DRAM chip contains a contiguous set of array elements. Within a chip, however, rows of elements are interleaved across banks.

In this arrangement, the memory address consists of

- Bits [31:26] chip select
- Bits [25:15] row select
- Bits [14:13] bank select
- Bits [12:3] column select

- Bits [2:0] unused

Note that we now have two fewer bits used for row select.

Assume that all banks are precharged and that each can have one row open at any given time. Bank accesses are not independent; we have to open the same row across all banks on a chip. However, we don't incur the row cycle time between accesses to the same row of different banks on a chip.

For the new code, what is the total DRAM latency? Rewrite the code to minimize DRAM latency for this configuration, and calculate the new latency. Which version of the application would you use to track your precious brew?

Problem 6 (20 points)—Memory Hierarchy

Do problem 5.18 from the text, using the attached graph instead of figure 5.33. **Explain your answers fully**, or you will receive no points. You need to read H&P p.351-352 carefully before starting this problem. Additionally, here are some assumptions you might need.

- Part a. The machine has one or more levels of caches and a main memory. You may ignore 1MB plots for part a.
- Part e. Assume that the size of L2 cache is 1MB.

Memory access latencies for arrays of various sizes

