CE 660– Parallel Computer Architecture

Homework 1

Due on Thursday, October 14

1. Amdahl's law. Assume that we accelerate portions of some application running in a computer by a factor of 3. The accelerated portion of the code corresponds to 60% of the time, measured as a percentage of the execution time *when the accelerated version is in use.*

a) What is the speed up we have obtained from accelerated mode?

b) What percentage of the original execution time has been converted to fast mode?

2. Branch prediction. Suppose we have a deeply pipelined processor, for which we implement a branch target buffer (BTB) for the conditional branches only. Assume that the misprediction penalty is always 4 cycles and the buffer miss penalty is always 3 cycles. Assume 90% hit rate and 90% accuracy (in case of a BTB hit), and 15% branch frequency. How much faster is the processor with the branch target buffer versus a processor that has a fixed 2-cycle branch penalty? Assume a base CPI without branch stalls of 1. Note: study example of page 123 in H&P, v. 4 before you attempt this problem.

3. Cache functionality. The following data memory addresses are produced by a CPU during the execution of an application: 2, 3, 11, 16, 21, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11. Assume a 16-byte L1 Data Cache, with block size 4 bytes. For each data access, you have to resolve whether it is a hit or miss and show the final contents of the Data Cache at the end of the sequence, when

a) the cache is direct mapped

b) the cache is 2-way set associative

4-5. Cache performance optimization. Exercises 5.6, 5.7 of H&P, version 4. Assume that a double precision array element is 8 bytes. Note: Study examples at page 307 and 308 (or 440-442 in version 3 of the book) before you attempt this exercise. Below, I have cut and paste the exercises 5.6 and 5.7 from version 4.

Concepts illustrated by this case study

m Nonblocking Caches

- Compiler Optimizations for Caches
- Software and Hardware Prefetching
- Calculating Impact of Cache Performance on More Complex Processors

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

$$\begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \\ A41 & A42 & A43 & A44 \end{bmatrix} \Rightarrow \begin{bmatrix} A11 & A21 & A31 & A41 \\ A12 & A22 & A32 & A42 \\ A13 & A23 & A33 & A43 \\ A14 & A24 & A34 & A44 \end{bmatrix}$$

Here is a simple C loop to show the transpose:

```
for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
        output[j][i] = input[i]j];
    }
}
```

Assume both the input and output matrices are stored in the row major order (row major order means row index changes fastest). Assume you are executing a 256 x 256 double-precision transpose on a processor with a 16 KB fully associative (so you don't have to worry about cache conflicts) LRU replacement level 1 data cache with 64-byte blocks. Assume level 1 cache misses or prefetches require 16 cycles, always hit in the level 2 cache, and the level 2 cache can process a request every 2 processor cycles. Assume each iteration of the inner loop above requires 4 cycles if the data is present in the level 1 cache. Assume the cache has a write-allocate fetch-on-write policy for write misses. Unrealistically assume writing back dirty cache blocks require 0 cycles.

5.6 [10/15/15] <5.2> For the simple implementation given above, this execution order would be nonideal for the input matrix. However, applying a loop interchange optimization would create a nonideal order for the output matrix. Because loop interchange is not sufficient to improve its performance, it must be blocked instead.

a. [10] <5.2> What block size should be used to completely fill the data cache with one input and output block?

b. [15] <5.2> How do the relative number of misses of the blocked and unblocked versions compare if the level 1 cache is direct mapped?

c. [15] <5.2> Write code to perform a transpose with a block size parameter *B* that uses *BxB* blocks.

5.7 [12] <5.2> Assume you are redesigning a hardware prefetcher for the *unblocked* matrix transposition code above. The simplest type of hardware prefetcher only prefetches sequential cache blocks after a miss. More complicated "nonunit stride" hardware prefetchers can analyze a miss reference stream, and detect and prefetch nonunit strides. In contrast, software prefetching can determine nonunit strides as easily as it can determine unit strides. Assume prefetches write directly into the cache and no *pollution* (overwriting data that needs to be used before the data that is prefetched). In the steady state of the inner loop, what is the performance (in cycles per iteration) when using an ideal nonunit stride prefetcher?