Dan Kondratyuk

Data Structures I

Charles University

10 December 2017

**Assignment 3**

This assignment analyzes the performance of two matrix transposition algorithms as pertaining to processor cache: a simple nested loop and a recursive cache-oblivious algorithm. Each matrix is in a row-structure of size , where each slot is bytes.

1. **Hardware Test**

The hardware test plots the amount of time (in seconds) that a matrix transposition algorithm will run for a given matrix of size . This test was benchmarked on an Intel Core i7 7700k (4.2 GHz) processor with 8 MiB cache.

**Figure 1: Matrix Size vs. Processor Time**

As expected, the amount of processor time is dependent on the number of swaps, which is . And the recursive (cache-oblivious) matrix transposition algorithm runs faster due to its ability to use the processor cache effectively. As the matrix size increases, the chance that the elements the simple algorithm needs will be in the cache decreases substantially due to the row-based layout of the matrix. The recursive algorithm transposes elements that are close together, increasing the likelihood that the next swapped elements will also be in the cache.

1. **Cache Simulator**

The cache simulator simulates the number of page faults that would occur in an LRU fully associative cache. The following plots display the number of cache misses that would occur for both algorithms as a function of the matrix width . Multiple plots are shown to examine the behavior of changing cache parameters and , where is the size of one page (in s) and is the number of pages.

To improve readability, all graphs are shown using a log-log scale.

**Figure 2:**

**Figure 3:**

**Figure 4:**

**Figure 5:**

**Figure 6:**

It can be seen from the plots that as the number of pages increase, the more a matrix can be fully stored in cache; the number of cache misses of the simple algorithm decreases up to the point at which the matrix can no longer be fully stored in the cache. Up until that point, the simple algorithm behaves identically to the recursive algorithm. The cache misses of the recursive algorithm decreases slightly as increases, but there is not much of an impact.

By increasing the size of each page (and holding the total cache size constant), it is apparent that both the simple and recursive algorithms benefit significantly. This is because there are fewer cache boundaries in which a cache miss can occur, i.e., more of each row in the matrix is loaded after a cache miss, reducing the likelihood of a subsequent cache miss. In the simple algorithm, the whole matrix is fully stored in the cache initially, but as the matrix size exceeds the cache size the number of cache misses returns to its base rate (as if there was little to no cache at all). On the other hand, the recursive algorithm shows improvement across the entire spectrum of matrix sizes, indicating that it is able to utilize the cache even when the entire matrix does not fit into cache.