Homework 6
Sophia Sharif
105 956 925

[6.41]

Our cache size is 2^15 and we store 2^3 bytes per block, which means that we can store 2^12 blocks in the cache. We can store structs per block, so that means we can store 2^13 structs per cache. If this loop followed the principles of locality, we would access the column in the outer loop and the row in the inner loop, which would save us a memory access on every second loop iteration.

However, since our inner loop is accessing the row first instead of the column, we violate the principle of locality, so the first line within the two loops will miss every time. However, the next three lines read the same object which we already have loaded into the cache, so those lines will hit every time. So, we will have 3 hits and 1 miss each iteration, so the resulting hit rate is 75%.

[6.45]

```
void transpose(int *dst, int *src, int dim) {
    int i, j;
    int nTimesDim[dim];

    // optimization 1:
    //      compute all the multiplications so we don't have to repeat
them n^2 times
    //      exploit loop structure to replace multiplication with addition

    int counter = 0;
    for (i = 0; i < dim; i++) {
        nTimesDim[i] = counter;
        counter += dim;
    }


    // optimization 2: switch order of the loops so we can best use cache;
    // this way, we're reading from the array in sequence instead of
jumping around rows

    for (j = 0; j < dim; j++) {

        // optimization 3: save row so we don't have to repeatedly read
from memory within the inner loop
        int row = nTimesDim[j];

        // optimization 4: loop unroll inner loop so we can parallelize
our code
        for (i = 0; i < dim - 1; i+=2) {
            dst[row + i] = src[nTimesDim[i] + j];
            dst[row + (i + 1)] = src[nTimesDim[i] + (j + 1)]; //
optimization 5: reassociation
```

```
        }
        for (; i < dim; i++)
            dst[row + i] = src[nTimesDim[i] + j];

    }
}
```