Cache Lab Report Analysis

Computer System Organization

Lecturer: Professor Mohamed Zahran TA: Crystal Butler Student: Jason Yao

Level 1

```
- Cache unfriendly pattern: Accessing memory column by column in an array of array
    - [Description] By accessing column by column, instead of row by row, the cache
    does not contain the values that are next to the first value found.
    - [Fix] This was fixed by accessing the memory in the array of arrays
    consecutively by iterating through each row first before continuing to
    the next row.
```

Level 2

```
- Cache unfriendly pattern: Accessing memory column by column for specific
locations in memory (diagonals of the array of arrays)
    - [Description] This is similar to above, except that iterating through
    the whole array, we only target specific values, namely along the
    diagonal.
    - [Fix] By seeing that the array given is a mirrored array, and as such
    A[i][j] == A[j][i] for all j,i. This means that we can abuse this
    property, and sum up along the rows to the same effect as summing
    up the columns, which is faster due to the cache already having those
    row values.
```

Level 3

```
- Cache unfriendly pattern: Swapping positions column by column entry by column en
try
    - [Description] Instead of utilizing spatial locality, the unoptimized
    algorithm would end up shifting by columns.
    - [Fix] We fix this by making the swaps occur within the same row before
```

```
moving to the next row. This causes the swaps to actually utilize cache
memory, instead of going to main memory every time it iterates through
an array.
```

Level 4 (Honours)

```
- Cache unfriendly pattern: Inefficient sorting algorithm
    - [Description] The algorithm sorts the array utilizing bubble sort,
    which is a sorting algorithm that is alright on small arrays, but
    becomes very inefficient on large arrays, taking O(n^2) time
    complexity.
    - [Fix] We fix this by implementing another sorting algorithm that
    has the capabilities of sorting through a large array faster.
    For the purposes of this array, we used mergesort to implement
    the cache optimization. Time complexity is O(n*log(n)), which is
    much better than before.
```