Class work: Matrix layouts and space-filling curves

Laura Toma, csci3225, Bowdoin College

1. Consider a matrix of size n by n layed out in row-major order in an array a. Highlight the part of the array that correspond to the first quadrant a_{11}.

   For the sake of this exercise, assume that $n = 8$.

2. Same setup as above: a matrix a of 8-by-8 elements, layed out in row-major order. Assume block size is $B = 3$ elements.

   How many blocks span a_{11} in this case? What cause this? Reflect on best and worst cases.

3. Consider in general a sub-matrix of size r-by-r insize a matrix a (a is laid out in row-major order). Give an upper bound on how many blocks span the sub-matrix as function of r, B.

   Draw examples of best-case and worst-case.

4. We want to layout the matrix a so that all elements of $a_{11}, a_{12}, a_{21}, a_{22}$ are contiguous, respectively (this is often refered as Morton layout). Sketch a function that accomplishes this.

```
b = calloc(sizeof(double), n*n);


//a is a matrix of size n by n in row-major order
//b should contain the elements in a in the new order
void mortonlayout( double* a, double* b, int n)  {









}
```

For example

- calling $mortonlayout(a, b, 2)$ with $a = [1, 2, 3, 4]$ should write b as $b = [1, 2, 3, 4]$.
- calling $mortonlayout(a, b, 4)$ with

$$a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$$

  should write b as

$$b = [1, 2, 5, 6, 3, 4, 7, 8, 9, 10, 13, 14, 11, 12, 15, 16]$$

5. The claim is that having a matrix in Morton layout simplifies both the algorithm for matrix multiplication, and also the cache-miss analysis.

   Show this by writing the code of matrix multiplication when a, b, c are given in Morton layout.

```
c = calloc(sizeof(double), n*n);


//a, b are matrices of size n by n in Morton layout
//c is produced in Morton layout as well
void mortonlayout( double* a, double* b, double* c, int n)  {











}
```

6. Cache-miss analysis for matrix multiplication with Morton layout:

   (a) How many cache misses to read a block of size r-by-r?

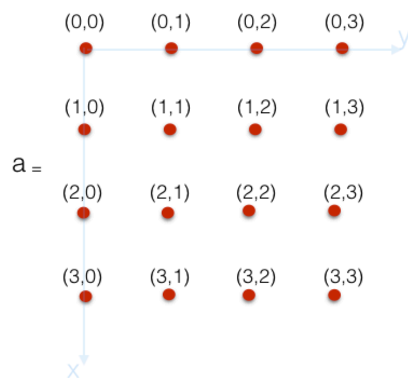
   (b) How does this compare to when the matrix is layed out in row-major order?

7. Compute the Zindices of 16 2D-points

$$\{(0,0),(0,1),...(3,3)\}$$

. Draw the Z-order of the points (the points ordered by their z-indices).

k=2 bits



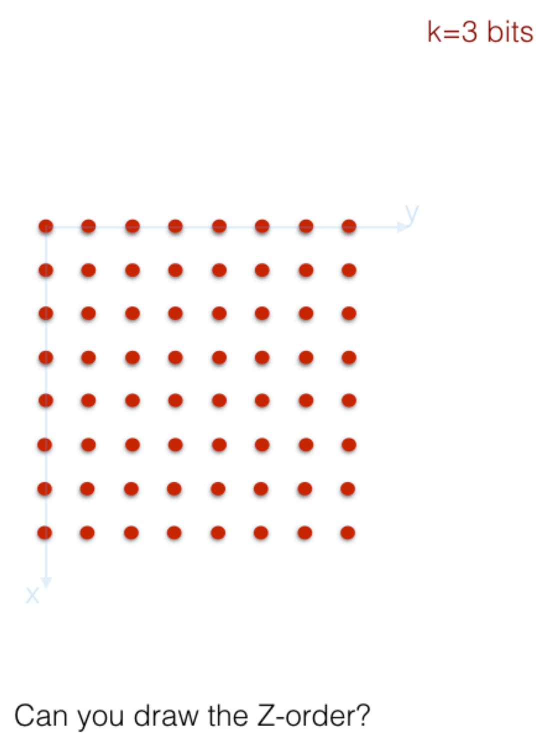
Can you draw the Z-order?

p	**Z_index(p)**
(00,00)	0000=0
(00,01)	0001=1
(00,10)	
(00,11)	
(01,00)	
(01,01)	
(01,10)	
(01,11)	
(10,00)	
(10,01)	
(10,10)	
(10,11)	
(11,00)	
(11,01)	
(11,10)	
(11,11)	

8. Compute the Zindices of 64 2D-points

$$\{(0,0),(0,1),...(7,7)\}$$

.



9. Sketch code to implement the zinxdex:

```
//x,y are 32-bit integers
//the result is a 64-bit integer
int64 zindex(int32 x,  int32 y)
```