Eric Watson

ECE335 Homework 4

Two versions of code that copies the contents of one matrix to another were provided to us for this assignment. Our task was to implement both of these routines and then compare their execution times. The very same source matrix, populated with random values, was passed to each copy function and the execution times were compared for several different matrix sizes. To simplify the process of testing multiple matrix sizes, I wrote my program to accept user-inputted matrix dimensions and then randomly populate the source matrix accordingly. Once the dimensions have been specified, and the source matrix created, two destination matrices of the same size were created for the copied values to be copied to upon passing the source array to the two copy functions.

Test Results

Matrix Size		**Execution Time**	**(Microseconds)**
# of Rows	# of Columns	Copy IJ	Copy JI
4	4	19	14
4	64	21	15
4	256	24	18
4	1024	69	61
4	1536	77	75
4	2048	93	98
64	4	142	160
64	64	209	280
64	256	176	566
64	1024	549	1,844
64	1536	615	2,466
64	2048	669	2,661
256	4	460	566
256	64	591	1,013
256	256	827	1,954
256	1024	1,681	5,325
256	1536	1,897	9,342
256	2048	2,074	13,120
1024	4	1972	1,947
1024	64	2,080	3,340
1024	256	2,633	10,344
1024	1024	6,506	32,432
1024	1536	8,869	49,290
1024	2048	9,367	60,344
1536	4	2,880	2,709
1536	64	3,102	4,889
1536	256	3,568	14,466
1536	1024	9,714	48,927
1536	1536	11,494	71,834
1536	2048	13,710	92,850

2048	4	3,626	3518
2048	64	4,199	6,751
2048	256	6,559	17,702
2048	1024	12,539	66,590
2048	1536	15,670	95,954
2048	2048	18,933	119,237

<u>Table Coloring Key</u>
Green: (# of rows) < (# of columns)
Blue: (# of rows) = (# of columns)
Red: (# of rows) > (# of columns)

Analyzing the results in the table above, one can see that the execution time of the function Copy IJ is significantly faster than Copy JI in nearly all cases. This is due to the fact that arrays are stored in row major order. Thus, two array elements are stored adjacent in memory if their second indices are consecutive numbers. In the Copy IJ routine all words in the cache line are used by successive loop iterations, thereby ensuring more cache hits and faster execution time. However, when the number of columns in the matrix is very small, the execution times do become quite similar. Copy JI accesses the array elements in a column-wise manner. Consequently, the preloaded data in the cache line will not be reused if the array is too large to fit entirely in the cache. But, in the situation where the number of columns is very small, and a large portion of the array (if not all of it) can be loaded into cache at once, this column-wise copy method will perform similarly to Copy IJ.

The terminal output below demonstrates the full functionality of my program for a small matrix size, allowing me to print the contents of both the source and destination arrays without a lengthy output. For this reason, I have chosen to print out only the first few rows of the source and destination matrices for the rest of my tests to show that the copy is working correctly, without having a huge output for larger matrix sizes

```
Erics-MacBook-Pro-2:HW04 Watson$ gcc HW04.c
Erics-MacBook-Pro-2:HW04 Watson$ ./a.out

Input 1 = [rows] Input 2 = [columns]
EX: "10 10" creates Matrix[10][10]

Input Two Dimensional Array Size: 5 5
src[0][0] = 865
src[0][1] = 626
src[0][2] = 867
src[0][3] = 943
src[0][4] = 658

src[1][0] = 795
src[1][1] = 375
src[1][2] = 275
src[1][3] = 90
src[1][4] = 817
```

```
src[2][0] = 290
src[2][1] = 317
src[2][2] = 154
src[2][3] = 444
src[2][4] = 527

src[3][0] = 748
src[3][1] = 657
src[3][2] = 412
src[3][3] = 268
src[3][4] = 773

src[4][0] = 447
src[4][1] = 139
src[4][2] = 668
src[4][3] = 122
src[4][4] = 364

CopyIJ Execution Time: 0.000020 seconds
Copied Values for CopyIJ:

dst1 [0][0] = 865
dst1 [0][1] = 626
dst1 [0][2] = 867
dst1 [0][3] = 943
dst1 [0][4] = 658

dst1 [1][0] = 795
dst1 [1][1] = 375
dst1 [1][2] = 275
dst1 [1][3] = 90
dst1 [1][4] = 817

dst1 [2][0] = 290
dst1 [2][1] = 317
dst1 [2][2] = 154
dst1 [2][3] = 444
dst1 [2][4] = 527

dst1 [3][0] = 748
dst1 [3][1] = 657
dst1 [3][2] = 412
dst1 [3][3] = 268
dst1 [3][4] = 773

dst1 [4][0] = 447
dst1 [4][1] = 139
dst1 [4][2] = 668
dst1 [4][3] = 122
dst1 [4][4] = 364

CopyJI Execution Time: 0.000036 seconds
Copied Values for CopyJI:

dst2 [0][0] = 865
dst2 [0][1] = 626
dst2 [0][2] = 867
dst2 [0][3] = 943
dst2 [0][4] = 658

dst2 [1][0] = 795
dst2 [1][1] = 375
dst2 [1][2] = 275
dst2 [1][3] = 90
dst2 [1][4] = 817

dst2 [2][0] = 290
dst2 [2][1] = 317
dst2 [2][2] = 154
```

```
dst2 [2][3] = 444
dst2 [2][4] = 527

dst2 [3][0] = 748
dst2 [3][1] = 657
dst2 [3][2] = 412
dst2 [3][3] = 268
dst2 [3][4] = 773

dst2 [4][0] = 447
dst2 [4][1] = 139
dst2 [4][2] = 668
dst2 [4][3] = 122
dst2 [4][4] = 364
```

Matrix Size 2048 x 2048

Printing only a 3x3 Block of each matrix to demonstrate that the copy functions work correctly, without producing a massive terminal output.

```
Input Two Dimensional Array Size: 2048 2048
3x3 Block of Source Values:

src[0][0] = 193
src[0][1] = 38
src[0][2] = 117

src[1][0] = 526
src[1][1] = 69
src[1][2] = 537

src[2][0] = 38
src[2][1] = 429
src[2][2] = 193

CopyIJ Execution Time: 0.019033 seconds
3x3 Block of Copied Values for CopyIJ:

dst1 [0][0] = 193
dst1 [0][1] = 38
dst1 [0][2] = 117

dst1 [1][0] = 526
dst1 [1][1] = 69
dst1 [1][2] = 537

dst1 [2][0] = 38
dst1 [2][1] = 429
dst1 [2][2] = 193

CopyJI Execution Time: 0.116661 seconds
3x3 Block of Copied Values for CopyJI:

dst2 [0][0] = 193
dst2 [0][1] = 38
dst2 [0][2] = 117

dst2 [1][0] = 526
dst2 [1][1] = 69
dst2 [1][2] = 537

dst2 [2][0] = 38
dst2 [2][1] = 429
dst2 [2][2] = 193
```

Eric Watson

ECE 335 Homework 4 Source Code (HW04.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

void print_src_values(int src[2048][2048], int num_row, int num_col){
  int i,j;
    for (i = 0; i < num_row; i++){
      for(j = 0; j < num_col; j++){
        printf("src[%d][%d] = %d\n", i, j, src[i][j]);
      }
        printf("\n");
    }
}

void print_dst_values(int dst[2048][2048], int num_row, int num_col, int dst_num){
  int i,j;
    for (i = 0; i < num_row; i++){
      for(j = 0; j < num_col; j++){
        printf("dst%d [%d][%d] = %d\n", dst_num, i, j, dst[i][j]);
      }
        printf("\n");
    }
}

void copyij(int src[2048][2048], int dst1[2048][2048], int num_row, int num_col){
  int i, j;
    for (i = 0; i < num_row; i++){
      for (j = 0; j < num_col; j++){
        dst1[i][j] = src[i][j];
      }
    }
}

 void copyji(int src[2048][2048], int dst2[2048][2048], int num_row, int num_col){
  int i, j;
  for (j=0; j < num_col; j++){
    for (i=0; i < num_row; i++){
      dst2[i][j] = src[i][j];
    }
  }
}

int main(){

  clock_t begin_ij, end_ij, begin_ji, end_ji;
  double ij_exec_time;
  double ji_exec_time;
  int num_row,num_col;
  int i= 0, j = 0;
  time_t seconds;
  time(&seconds);
  srand((unsigned int) seconds);

  printf("\nInput 1 = [rows] Input 2 = [columns]\n");
  printf("EX: \"10 10\" creates Matrix[10][10]\n");
  printf("\nInput Two Dimensional Array Size: ");
  scanf("%d %d", &num_row, &num_col);
//        printf("number of rows: %d\n", num_row);
//        printf("number of columns: %d\n", num_col);

// Allocating Space for max matrix size of [2048][2048]
  static int src[2048][2048];   // randomly generated source array
  static int dst1[2048][2048];   // destination array for copyij
  static int dst2[2048][2048];  // destination array fro copyji
// POPULATE SOURCE ARRAY WITH RANDOM NUMBERS
  for (i = 0; i < num_row; i++){
    for(j = 0; j < num_col; j++){
      src[i][j] = rand() % 1000;
    }
  }
// Print a block of 3 rows and 3 columns to show its working
  printf("3x3 Block of Source Values:\n\n");
  print_src_values(src, 3, 3);

  begin_ij = clock();                     // start clock
  copyij(src, dst1, num_row, num_col);    // copy matrix
  end_ij = clock();                       // end clock
  ij_exec_time = (double)(end_ij-begin_ij)/ CLOCKS_PER_SEC;  //calculate time (sec)

  printf("CopyIJ Execution Time: %f seconds\n", ij_exec_time);

// CHECK THAT VALUES WERE COPIED CORRECTLY
  printf("3x3 Block of Copied Values for CopyIJ:\n\n");
  print_dst_values(dst1, 3, 3, 1);

  begin_ji = clock();                      //start clock
  copyji(src, dst2, num_row, num_col);     // copy matrix
  end_ji = clock();                        // end clock
  ji_exec_time = (double)(end_ji-begin_ji)/ CLOCKS_PER_SEC;

  printf("CopyJI Execution Time: %f seconds\n", ji_exec_time);

// CHECK THAT VALUES WERE COPIED CORRECTLY
  printf("3x3 Block of Copied Values for CopyJI:\n\n");
// Print a block of 3 x 3 to show that it worked
  print_dst_values(dst2,3, 3, 2);

        return(0);
}
```