ITCS 5145 Parallel Computing
Test 2 CUDA program question

Qu. 21 Write a CUDA program that transposes an N x N matrix. Transposition of a matrix means swapping values in the columns and rows across the diagonal. For example: If the original matrix contains:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

the transposed matrix contains:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

The initial matrix is held in the integer array **A[N][N]** and the result is to be held in the integer array **B[N][N]**. You do not need to copy the result back to **A**.

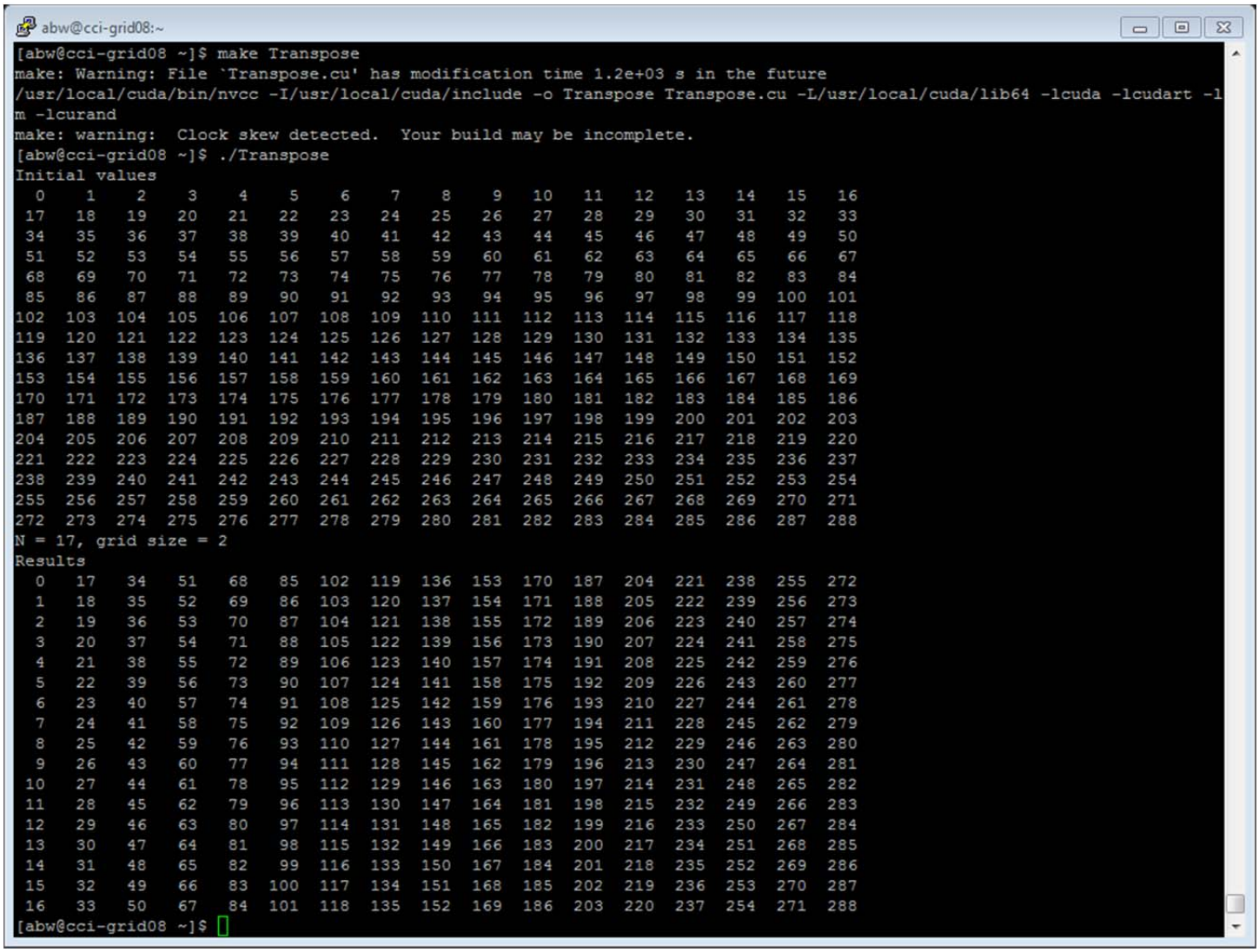
Use one CUDA thread to copy one element from array **A** to array **B**. Organize the kernel structure to a square 2-D grid of square 2-D blocks. Each block is organized as 16 x 16 threads. With these constraints, use the minimum number of blocks necessary, given **N** as a defined constant. Your code must take into account any value for **N**.

You may assume that the program has code to store initial values in the array **A**. Declare all variables and arrays needed

Note: The C library function **double ceil(double x)** returns the smallest integer value greater than or equal to **x**. Use this to round a number up.

Provide comments in your code to help the grader! If I do not understand the code, I will assume it is incorrect.

```
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>

#define N 17  // size of arrays

__global__ void transpose (int *a, int *b) {
        int col = blockIdx.x*blockDim.x+threadIdx.x;
        int row =blockIdx.y*blockDim.y+threadIdx.y;
        int index1 = col + row * N;
        int index2 = row + col * N;
        if ( col < N && row < N) b[index1]= a[index2];
}

int main (int argc, char **argv ) {
        int i,j;
        int size = N * N *sizeof( int);
        int  a[N][N], *devA, *devB;
        int gridsize = (int) ceil((double) N/16);

         for (i = 0; i < N; i++)     {               // put some numbers into array
                 for (j= 0; j < N; j++) {
                         a[i][j] = j + N * i;
                 }
         }

        printf("Initial values");

        for (i = 0; i < N; i++) {
                printf("\n");
                 for (j= 0; j < N; j++) {
                         printf("%3d  ",a[i][j]);
                 }
        }

        printf("\nN = %d, grid size = %d\n",N,gridsize);

        dim3 block (16,16);
        dim3 grid (gridsize, gridsize);

        cudaMalloc( (void**)&devA, size );
        cudaMalloc( (void**)&devB, size );

        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);

        transpose<<<grid, block>>>(devA, devB);

        cudaMemcpy( a, devB, size, cudaMemcpyDeviceToHost);

        printf("Results");
        for (i = 0; i < N; i++) {
                printf("\n");
                for (j= 0; j < N; j++) {
                        printf("%3d  ",a[i][j]);
                }
        }
        printf("\n");

        cudaFree( devA);
        cudaFree( devB);

        return (0);
}
```


abw@cci-grid08:~
[abw@cci-grid08 ~]$ make Transpose
make: Warning: File `Transpose.cu' has modification time 1.2e+03 s in the future
/usr/local/cuda/bin/nvcc -I/usr/local/cuda/include -o Transpose Transpose.cu -L/usr/local/cuda/lib64 -lcuda -lcudart -lm -lcurand
make: warning: Clock skew detected. Your build may be incomplete.
[abw@cci-grid08 ~]$ ./Transpose
Initial values
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101
102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152
153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169
170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186
187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203
204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237
238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
N = 17, grid size = 2
Results
0 17 34 51 68 85 102 119 136 153 170 187 204 221 238 255 272
1 18 35 52 69 86 103 120 137 154 171 188 205 222 239 256 273
2 19 36 53 70 87 104 121 138 155 172 189 206 223 240 257 274
3 20 37 54 71 88 105 122 139 156 173 190 207 224 241 258 275
4 21 38 55 72 89 106 123 140 157 174 191 208 225 242 259 276
5 22 39 56 73 90 107 124 141 158 175 192 209 226 243 260 277
6 23 40 57 74 91 108 125 142 159 176 193 210 227 244 261 278
7 24 41 58 75 92 109 126 143 160 177 194 211 228 245 262 279
8 25 42 59 76 93 110 127 144 161 178 195 212 229 246 263 280
9 26 43 60 77 94 111 128 145 162 179 196 213 230 247 264 281
10 27 44 61 78 95 112 129 146 163 180 197 214 231 248 265 282
11 28 45 62 79 96 113 130 147 164 181 198 215 232 249 266 283
12 29 46 63 80 97 114 131 148 165 182 199 216 233 250 267 284
13 30 47 64 81 98 115 132 149 166 183 200 217 234 251 268 285
14 31 48 65 82 99 116 133 150 167 184 201 218 235 252 269 286
15 32 49 66 83 100 117 134 151 168 185 202 219 236 253 270 287
16 33 50 67 84 101 118 135 152 169 186 203 220 237 254 271 288
[abw@cci-grid08 ~]$