GPU-powered MOLS Generation from Permutation Networks

Attenni Giulio, Baieri Daniele

Intensive Computation – Project presentation
MSc Computer Science
Università di Roma "La Sapienza"

August 2, 2021

Outline

Objective

- The purpose of this project is to find MOLS (Mutually Orthogonal Latin Squares) among Latin Squares composed by permutations obtained by performing routing on a set of MIN configurations.
- While a latin square is just an N-by-N table with entries in 0,. . . ,N-1 such that rows and columns have no repetitions, a MOLS is the result of the superimposition of two Latin Squares such that no pair resulting from the superimposition appears more than once. Generating MOLS is typically a hard problem.

MINs

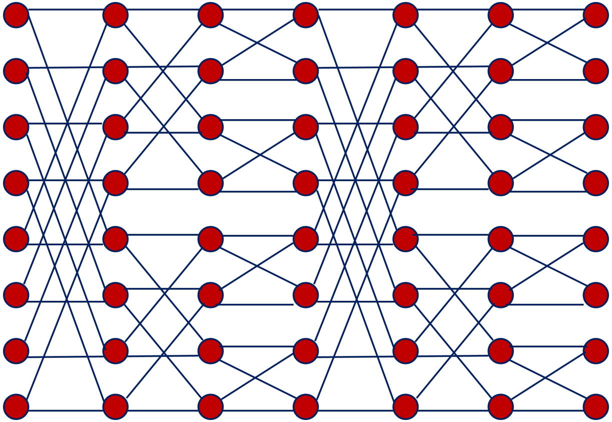


Figure: Inputs/Outputs: N = 16; Stages: 2logN-1 = 7; Topology: Butterfly-Butterfly;

Latin Square generation

- We generate a Latin Square considering $N = 16$ switch configurations. Each row of the Latin Square is a permutation obtained by performing the self-routing algorithm given a certain configuration.
- Let C be an initial random configuration, we obtain N new configurations performing the XOR of C with N different characteristic configurations.
- The first set of characteristic configurations is obtained starting from the configuration that outputs the identity permutation and then the ones that output the rotations of the identity.
- The second set of characteristic configurations is obtained considering a specific pattern. Given a configuration, all switches in the same layer are set equal and a row of switches follows the pattern 000**** where * can be either 0 or 1. Considering all the possible combinations, we obtain N configurations.

Outline

Implementation details

- To obtain a large number of Latin Squares, among pairs of which we look for MOLS, we use a parallel implementation in which each CUDA thread generates a different initial random configuration and outputs a Latin Square.
- We have organized all functions in several files so as to obtain a more readable and manageable code.
  - In Utils.h it is possible to find some functions that are used to provide the output to the user and other useful functions used elsewhere.
  - In MIN.h are grouped all the functions that concern generating MIN configurations and routing.
  - CudaMain.cu wraps the calls to the CUDA kernels.
  - LatinSquares.cuh defines the kernels used to compute and check Latin Squares and to search for MOLS.
  - Main.cpp is used just to call the host code.

CUDA threads organization

- The Latin Squares computation is performed in a very simple manner, using a 1D grid of size K and 1D blocks of size 1.
- The MOLS computation requires a little more structure, since it is more complex to organize computation across all possible pairs of Latin Squares, and if done naively it would result in excessive memory requirements or block sizes beyond our platform's threshold (1024), given that we have $O(K^2)$ pairs:
  - To reduce the number of threads in a block, we defined a 2D grid of size (K, S) and 1D blocks of size (T), where $K = S \times T$.
  - Alternatively, one could do a randomized search rather than a complete one and choose a 1D grid of size (K) and 1D blocks of size (L), where each thread in a block samples a random index to compare two latin squares.

Hardware limitations and Performance

- Running a complete search on all pairs imposed some limitations on search size: the largest test we could manage to run would generate 20.000 Latin Squares (as the GPUs we were using would not have enough memory for larger instances).
- Despite this, our program is quite fast, completing the MOLS search in around 0.01ms. The runtime can increase dramatically if output logs are requested, as writing the counterexample for each non-MOLS to file would require a very long time.

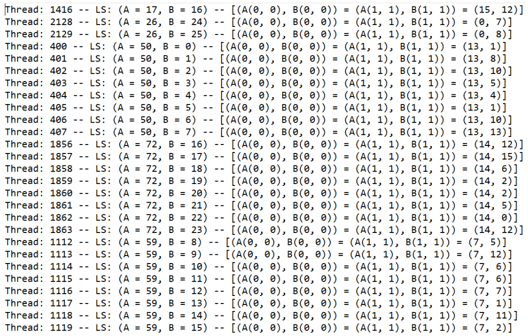
Outline

SAPIENZA
UNIVERSITÀ DI ROMA

Experimental Results

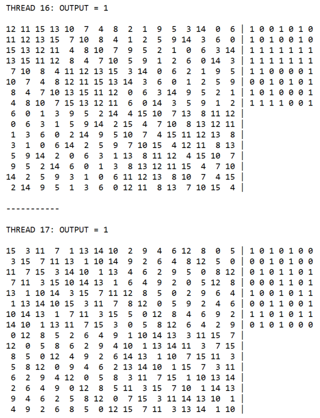
- Settings:
  - Checking MOLS using pairs of Latin Squares obtained with rotations.
  - Checking MOLS using pairs Latin Squares obtained with 000**** pattern.
  - Checking MOLS using pairs of Latin Squares obtained one with rotations and the other with 000**** pattern.
- Results:
  - No MOLS have been found. In particular, the results for 000**** matrices show an extremely clear pattern in counterexamples which made us suspect generating MOLS this way to be provably impossible. The other results were less regular, but still unsuccessful.

Test I: 000**** Matrices (a)



```
Thread: 1416 -- LS: (A = 17, B = 16) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (15, 12)]
Thread: 2128 -- LS: (A = 26, B = 24) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (0, 7)]
Thread: 2129 -- LS: (A = 26, B = 25) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (0, 8)]
Thread: 400 -- LS: (A = 50, B = 0) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 1)]
Thread: 401 -- LS: (A = 50, B = 1) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 8)]
Thread: 402 -- LS: (A = 50, B = 2) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 10)]
Thread: 403 -- LS: (A = 50, B = 3) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 5)]
Thread: 404 -- LS: (A = 50, B = 4) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 4)]
Thread: 405 -- LS: (A = 50, B = 5) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 1)]
Thread: 406 -- LS: (A = 50, B = 6) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 10)]
Thread: 407 -- LS: (A = 50, B = 7) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (13, 13)]
Thread: 1856 -- LS: (A = 72, B = 16) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 12)]
Thread: 1857 -- LS: (A = 72, B = 17) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 15)]
Thread: 1858 -- LS: (A = 72, B = 18) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 6)]
Thread: 1859 -- LS: (A = 72, B = 19) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 2)]
Thread: 1860 -- LS: (A = 72, B = 20) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 2)]
Thread: 1861 -- LS: (A = 72, B = 21) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 5)]
Thread: 1862 -- LS: (A = 72, B = 22) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 0)]
Thread: 1863 -- LS: (A = 72, B = 23) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (14, 12)]
Thread: 1112 -- LS: (A = 59, B = 8) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 5)]
Thread: 1113 -- LS: (A = 59, B = 9) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 12)]
Thread: 1114 -- LS: (A = 59, B = 10) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 6)]
Thread: 1115 -- LS: (A = 59, B = 11) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 6)]
Thread: 1116 -- LS: (A = 59, B = 12) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 7)]
Thread: 1117 -- LS: (A = 59, B = 13) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 1)]
Thread: 1118 -- LS: (A = 59, B = 14) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 11)]
Thread: 1119 -- LS: (A = 59, B = 15) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (7, 2)]
```

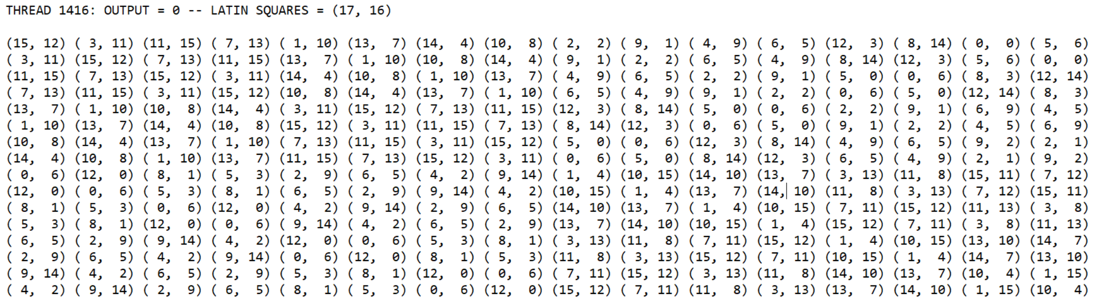
Figure: Debug output for some threads out of a 2000 latin squares test. The lines show the two latin squares considered (A, B) and the two pairs of indices containing the same pair.



```
THREAD 16: OUTPUT = 1

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

-----------

THREAD 17: OUTPUT = 1

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

Figure: Latin squares output. On the left, the permutation matrix generated from the network routing. On the right, the random configuration used to XOR with the 16 000**** matrices.

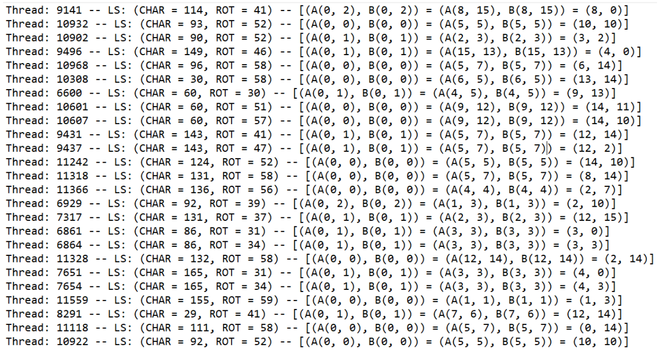
Test I: 000**** Matrices (b)



```
THREAD 1416: OUTPUT = 0 -- LATIN SQUARES = (17, 16)

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

Figure: Superimposition of previously shown latin squares. As you can see, indices (0, 0) and (1, 1) contain the same pair (15, 12), as previously shown. Actually, every pair of latin squares generated using 000**** matrices ended up with this counterexample.

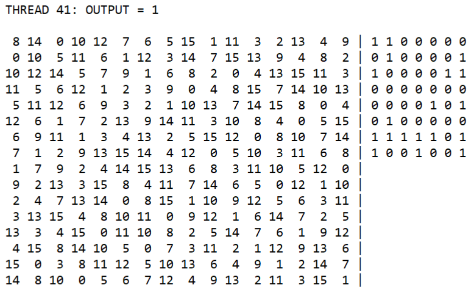
Test II: Both matrix types (a)



```
Thread: 9141 -- LS: (CHAR = 114, ROT = 41) -- [(A(0, 2), B(0, 2)) = (A(8, 15), B(8, 15)) = (8, 0)]
Thread: 10932 -- LS: (CHAR = 93, ROT = 52) -- [(A(0, 0), B(0, 0)) = (A(5, 5), B(5, 5)) = (10, 10)]
Thread: 10902 -- LS: (CHAR = 90, ROT = 52) -- [(A(0, 1), B(0, 1)) = (A(2, 3), B(2, 3)) = (3, 2)]
Thread: 9496 -- LS: (CHAR = 149, ROT = 46) -- [(A(0, 1), B(0, 1)) = (A(15, 13), B(15, 13)) = (4, 0)]
Thread: 10968 -- LS: (CHAR = 96, ROT = 58) -- [(A(0, 0), B(0, 0)) = (A(5, 7), B(5, 7)) = (6, 14)]
Thread: 10308 -- LS: (CHAR = 30, ROT = 58) -- [(A(0, 0), B(0, 0)) = (A(6, 5), B(6, 5)) = (13, 14)]
Thread: 6600 -- LS: (CHAR = 60, ROT = 30) -- [(A(0, 1), B(0, 1)) = (A(4, 5), B(4, 5)) = (9, 13)]
Thread: 10601 -- LS: (CHAR = 60, ROT = 51) -- [(A(0, 0), B(0, 0)) = (A(9, 12), B(9, 12)) = (14, 11)]
Thread: 10607 -- LS: (CHAR = 60, ROT = 57) -- [(A(0, 0), B(0, 0)) = (A(9, 12), B(9, 12)) = (14, 10)]
Thread: 9431 -- LS: (CHAR = 143, ROT = 41) -- [(A(0, 1), B(0, 1)) = (A(5, 7), B(5, 7)) = (12, 14)]
Thread: 9437 -- LS: (CHAR = 143, ROT = 47) -- [(A(0, 1), B(0, 1)) = (A(5, 7), B(5, 7)) = (12, 2)]
Thread: 11242 -- LS: (CHAR = 124, ROT = 52) -- [(A(0, 0), B(0, 0)) = (A(5, 5), B(5, 5)) = (14, 10)]
Thread: 11318 -- LS: (CHAR = 131, ROT = 58) -- [(A(0, 0), B(0, 0)) = (A(5, 7), B(5, 7)) = (8, 14)]
Thread: 11366 -- LS: (CHAR = 136, ROT = 56) -- [(A(0, 0), B(0, 0)) = (A(4, 4), B(4, 4)) = (2, 7)]
Thread: 6929 -- LS: (CHAR = 92, ROT = 39) -- [(A(0, 2), B(0, 2)) = (A(1, 3), B(1, 3)) = (2, 10)]
Thread: 7317 -- LS: (CHAR = 131, ROT = 37) -- [(A(0, 1), B(0, 1)) = (A(2, 3), B(2, 3)) = (12, 15)]
Thread: 6861 -- LS: (CHAR = 86, ROT = 31) -- [(A(0, 1), B(0, 1)) = (A(3, 3), B(3, 3)) = (3, 0)]
Thread: 6864 -- LS: (CHAR = 86, ROT = 34) -- [(A(0, 1), B(0, 1)) = (A(3, 3), B(3, 3)) = (3, 3)]
Thread: 11328 -- LS: (CHAR = 132, ROT = 58) -- [(A(0, 0), B(0, 0)) = (A(12, 14), B(12, 14)) = (2, 14)]
Thread: 7651 -- LS: (CHAR = 165, ROT = 31) -- [(A(0, 1), B(0, 1)) = (A(3, 3), B(3, 3)) = (4, 0)]
Thread: 7654 -- LS: (CHAR = 165, ROT = 34) -- [(A(0, 1), B(0, 1)) = (A(3, 3), B(3, 3)) = (4, 3)]
Thread: 11559 -- LS: (CHAR = 155, ROT = 59) -- [(A(0, 0), B(0, 0)) = (A(1, 1), B(1, 1)) = (1, 3)]
Thread: 8291 -- LS: (CHAR = 29, ROT = 41) -- [(A(0, 1), B(0, 1)) = (A(7, 6), B(7, 6)) = (12, 14)]
Thread: 11118 -- LS: (CHAR = 111, ROT = 58) -- [(A(0, 0), B(0, 0)) = (A(5, 7), B(5, 7)) = (0, 14)]
Thread: 10922 -- LS: (CHAR = 92, ROT = 52) -- [(A(0, 0), B(0, 0)) = (A(5, 5), B(5, 5)) = (10, 10)]
```

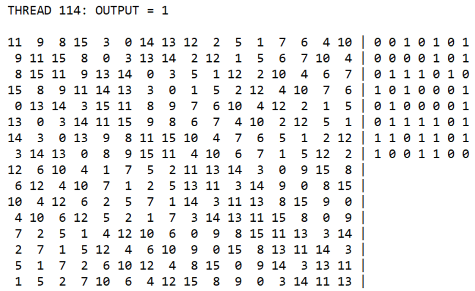
Figure: Debug output for some threads out of a 2000 latin squares test. The lines show the two latin squares considered (CHAR, ROT), telling which was generated with 000**** matrices and which with rotation matrices, and the two pairs of indices containing the same pair.

Test II: Both matrix types (b)



```
THREAD 41: OUTPUT = 1

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

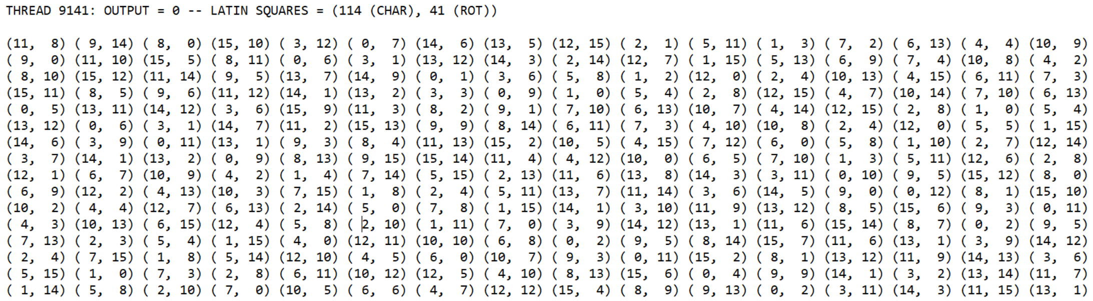
Figure: Latin squares output. On the left, the permutation matrix generated from the network routing. On the right, the random configuration used to XOR with the rotation matrices.



```
THREAD 114: OUTPUT = 1

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

Figure: Latin squares output. On the left, the permutation matrix generated from the network routing. On the right, the random configuration used to XOR with the 16 000**** matrices.

Test II: Both matrix types (c)



```
THREAD 9141: OUTPUT = 0 -- LATIN SQUARES = (114 (CHAR), 41 (ROT))

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

Figure: Superimposition of previously shown latin squares. As you can see, indices (0, 2) and (8, 15) contain the same pair (8, 0), as previously shown. You can see from the log that the indices are not as regular as in the 000**** matrices case.

Outline

Conclusions

- Despite no MOLS have been found, we have developed a useful tool to investigate whether a certain set of characteristic configurations is able to generate Latin Squares and if among the generated Latin Squares there exists a pair that is a MOLS.
- This tool is expandable. Indeed, it is possible to redefine the topology of the MINs allowing to repeat the same investigation over different kinds of MINs, or for other methods of generating permutation matrices.

Contributions

Here we describe how we split the work.
We had a couple of coding sessions together to design a first basic implementation, then we worked individually as follows:

- (Daniele) My main contributions to the project were the organization of the GPU computation (defining the CUDA grid), the kernel code for MOLS searching and the result gathering.
- (Giulio) My main contributions to the project were the code for Butterfly-Butterfly topology definition and for rotations-based characteristic configurations.

Thank you for your attention!