An Efficient Parallel Algorithm for Latin Square Design: A Multi Core CPU Approach

Abhay B. Rathod
Department of Electronics and Telecomm. Engg
Jawaharlal Darda Institute of Engg. and Tech.
Yavatmal, India.

Sanjay M. Gulhane
Department of Electronics and Telecomm. Engg
Jawaharlal Darda Institute of Engg. and Tech.
Yavatmal, India.

***Abstract*— The theory of Latin squares is very important tool in design theory. Like much of design theory, Latin squares have various applications in statistics, finite geometries and experimental design, to name a few. In this paper, we proposed an efficient parallel algorithm for Latin square design which have desirable properties for parallel array access. These squares provide conflict free access to various subsets of an *n x n* array using *n* memory modules. A transversal of such a square is a set of *n* entries such that no two entries share the same row, column or symbol. We present a general construction method for building parallel Latin square of order n^2 for all *n*.**
The proposed algorithm presents a quick parallel method to produce a Latin square design and a parallel conflict access of data in SIMD system. The simulation results of the proposed parallel algorithm for Latin square design were compared with the traditional sequential algorithm Latin square design in terms of speedup and efficiency. The results of parallel Latin Square design were very promising and showed a potential that this design could successfully be applied to the parallel routing problems for conflict free data access. At last, the results show that the parallel versions of former sequential algorithm with simple modifications achieve the super linear speedup up to 200 times for matrix size of 256.

***Index Terms*: Latin square, multi core processor, parallel processing, simulation, parallel memory system, skewing scheme, multistage interconnection network.**

I.INTRODUCTION

A Latin square of order *n* is an *n x n* matrix A= (a_{ij}) with entries $a_{ij} \in \{1,2,\ldots,n\}$ containing *n* distinct symbols such that each symbol appears in each row and column exactly once. It is called a Latin square because its numbers could be Latin letters in what might have a counterpart with Greek letters, satisfying the all pairs property! [1]. Euler was the first person to define Latin squares and investigate its properties mathematically. In this paper we present a parallel Latin square design algorithm and compare it with sequential one on multi core processor.

In memory module of SIMD processors, conflict free parallel access of shared data plays vital role in the overall system performance [2]. The parallel array access problem is to store an *n x n* array into *m* memory modules such that no memory conflict occurs when various subsets of size *n* of the array (rows, columns, diagonals and $n^{1/2}$ x $n^{1/2}$ sub arrays etc.) are accessed [3]. The main focus of this paper is the storing of a matrix of data in memory modules so that various portions of it can be accessed with high parallelism.

To achieve conflict free parallel access of shared data we use perfect Latin square which have very useful properties for parallel array access. Using perfect Latin squares as skewing schemes, many interesting subsets of an array (rows, columns, diagonals and sub squares) can be accessed without memory conflicts [3].

A skewing scheme is a mapping of elements of an *n x n* array into memory modules so as to provide conflict free access to various subsets of the array. Subsets of particular importance are rows, columns, diagonals and sub arrays since they are frequently used in many scientific computations [3].

A good data skewing scheme should have the following features [2]:

1. Given the row and column indices of a data element, the computation of its address the memory module in which the element is stored and location within the module should be fast and efficient with very little hardware requirements.
2. It should provide simultaneous data paths between processors and memory modules, during the parallel access of data.
3. The skewing scheme should be represented compactly so that overheads are minimized.
4. The skewing scheme should allow efficient re-skewing of data between phases of computation when necessary.

Lots of analytical work is done on Latin square design and skewing scheme. However, to our knowledge, no one has presented simulation results of Latin square design using multi-core processors, which is the main focus of this paper.

The paper is organized as follows: Section 2 gives literature survey of previous work. Section 3 describes the Latin square design and skewing scheme to achieve parallel conflict free access of data. Simulation results will be presented in Section 4. Finally, Section 5 concludes the paper with a discussion and directions for further research.

II. RELATED WORKS

Y. Yuanyuan, and J. Wang in [8] presented a method to build a Latin square matrix for a given multistage network and on this network, data can be routed on using the tag information represented by the elements in the Latin square matrix.

Boppana, Rajendra V., and C. S. Raghavendra in their technical report [2] showed a generalized solution to the problem of conflict free access of various templates of data of a matrix in an SIMD multiprocessor system. For this the authors used scrambled skewing schemes. Further, the authors report

IJSMS

that to access parallel data namely, row, column, main diagonal, and square block of matrix, it is possible by realizing linear complement permutation. However, simulation result of accessing conflict free parallel data from matrix is not presented.

K. Heinrich, K. Kim, and V.K. Prasanna Kumar in their paper introduced novel Latin squares called perfect Latin squares which have desirable properties for parallel array access. The authors used perfect Latin squares as skewing schemes. In addition, these squares provide conflict free access to various subsets of an n^2 x n^2 array using n^2memory modules.

K. Kim, and V. K. Prasanna Kumar proposed an efficient memory system for parallel array access. A novel Latin square (perfect Latin square) is introduced to be used as the nonlinear skewing scheme of the memory system. Using perfect Latin squares of order n^2 as the skewing scheme, many subsets of an n^2 *x* n^2 array (rows, columns, and diagonals) are accessible without any memory conflict using n^2 memory. However, comparison of parallel array access and serial array access time is not demonstrated.

III. PRELIMINARIES

A. Mapping Matrix

A mapping matrix is obtained from $A = (a_{i,j})$, by replacing each $a_{i,j}$ by the number of memory unit in which it is stored it is also called as buffer matrix[8].

$$A=\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}_{4\times4} \quad A=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix}_{4\times4} \quad A=\begin{bmatrix} 0 & 2 & 1 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 1 & 2 & 0 \end{bmatrix}_{4\times4}$$

a) Data Matrix A b) Mapping Matrix 1 c) Mapping Matrix 2

Fig. 1: Two Method of Storing 4X4 Matrix.

In first storage scheme given in Figure 1 (b) in one pass any row can be accessed without memory conflict, but not the case with column accesses.

In second method as shown in Figure 1 (c) in one pass any row as well as any column can be accessed conflict free. This second method of storing the matrix elements is called skewed storage method.

B. Latin Square and Transversal

A Latin square of order *n* is an *n x n* array of *n* symbols in which each symbol occurs exactly once in each row and column [4].

Example of Latin Square is given in Eq. (1).

$$A=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}_{4\times4} \tag{1}$$

A transversal of such a square is a set of *n* entries such that no two entries share the same row, column or symbol [1].
Example of transversal Latin Square is given in Eq. (2).

$$A=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}_{4\times4} \tag{2}$$

Then, T={ $a_{0,0}$=0, $a_{1,2}$=3, $a_{2,3}$=1, $a_{3,1}$=2} is a transversal in *A*.

In this paper we use parallel transversal Latin squares design to achieve conflict free access to various subsets of data. Latin squares are well known combinatorial objects for centuries [4]. The key idea of the method in [8] is to build a parallel Latin square matrix for a given multistage network and it can be routed using the tag information represented by the elements in the Latin square matrix [2].

A Latin square of order n is an *n x n* square composed with symbols from 0 to n - 1 such that no symbol appears more than once in any rows or in any columns [4] as given in Eq. (3).

$$A=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}_{4\times4} \tag{3}$$

Although in general there are many ways to construct a Latin square [9][3][4], in this section we describe one of the method of constructing the Latin square of an array *2nx 2n*.

Theorem: If we take Latin square matrix *A* of order *n x n* and add n in each number row wise and column wise we get Latin square matrix *B*.

Proof: Take the matrix of order *n x n* as show in Eq. (4).

$$A=\begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{bmatrix}_{n\times n} \tag{4}$$

Add *n* in each row we get a new matrix *B* of order *n x n* and repeat this process. Latin square matrix *B* is given in Eq. (5).

$$B=\begin{bmatrix} n+a_{0,1} & n+a_{0,2} & \dots & n+a_{0,n} \\ n+a_{1,1} & n+a_{1,2} & \dots & n+a_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ n+a_{n,1} & n+a_{n,2} & \dots & n+a_{n-1,n-1} \end{bmatrix}_{n\times n} \tag{5}$$

It is easy to construct Latin square matrix *B* from *A*. We give an algorithm to generate Latin square matrix *B* from *A* in Table 1. Its correctness can be seen from (7).The algorithm to generate Latin square is given in Table 1.

IJSMS

Table I: The Algorithm To Generate Latin Square B From A

Procedure: To generate Latin Square Matrix B from A
Input:
-A $=(a_{i,j})_{n\times n}$ // *Latin Square of order 2×2*
Output:
-B $=(b_{i,j})_{n\times n}$ // *Latin Square of order n×n*
-C $=(c_{i,j})_{2n\times 2n}$ // *Latin Square of order 2n×2n*
Main()
{
for(i=0; i<n; i++){ // *do parallel*
for(j=0; j<n; j++){
$B_{ij}=n+a_{i,j}$
}
}
$C=\begin{bmatrix} A & B \\ B & A \end{bmatrix}$
}

To construct Latin square C of order $2n\times 2n$ place the matrix A and B in such a way that A and B will appear more than once in any row or in any column. It is given in Eq. (6).

$$C=\begin{bmatrix} A & B \\ B & A \end{bmatrix}_{2n\times 2n} \quad (6)$$

From above described method we can construct Latin square C given in (7).

$$C=\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\ 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix} \quad (7)$$

he performance of parallel algorithm for Latin square is compared with conventional sequential Latin square algorithm.

IV. PERFORMANCE EVALUATION AND RESULTS

This section describes the computational experiments used to evaluate the performance of the proposed algorithm. The simulation is carried out on an Intel(R) Core(TM) i3 CPU with 2.53 GHz, 3.00 GB RAM memory. We have implemented improved firefly algorithm using Java 7 language under 32 bit Windows 7 operating system, and Eclipse 3.7 as an integrated development environment (IDE). Java support multithreading, however, we considered one thread per core for this computational experiment. The simulation environment and the performance results are described in next sub section.

A. Simulation Result

The simulation results obtained based on speedup of an algorithm. In parallel processing *speedup* emphasizes time reduction of a given problem and it is defined as sequential execution time over parallel execution time [10]. It is given in Eq. (8).

$$Speedup(S)=\frac{\text{Sequential Time of Solving Scaled Workload}}{\text{Parallel Time of Solving Scaled Workload}} \quad (8)$$

and Efficiency (η) of parallel algorithm is given as in Eq. (9).

$$\text{Efficiency}\,(\eta)=\frac{\text{Speedup(S)}}{\text{Number of Core(P)}} \quad (9)$$

The simulation result of Serial and Parallel Latin square algorithm is given in Table II and Table III.

Table II: Simulation result of Serial and Parallel Latin Square

Matrix Size	Serial Latin Square Algorithm Time in ms	Parallel Latin Square Algorithm with 2 Core			Parallel Latin Square Algorithm with 4 Core		
		Time (T)	*Speed Up (S)*	*Efficiency (n)*	*Time (T)*	*Speed Up (S)*	*Efficiency (n)*
4	3 ms	0 ms	0	0	0 ms	0	0
9	14 ms	0 ms	0	0	0 ms	0	0
16	3 ms	3 ms	1	0.5	1 ms	3	0.7
25	14 ms	0 ms	0	0	0 ms	0	0
36	18 ms	0 ms	0	0	0 ms	0	0
49	21 ms	1 ms	21	10	0 ms	0	0
64	26 ms	0 ms	0	0	0 ms	0	0
81	35 ms	0 ms	0	0	0 ms	0	0
100	51 ms	0 ms	0	0	0 ms	0	0
121	78 ms	0 ms	0	0	4 ms	19	5
144	123 ms	0 ms	0	0	0 ms	0	0
169	194 ms	1 ms	194	97	1 ms	194	48
196	314 ms	1 ms	314	157	2 ms	157	39
225	549 ms	2 ms	225	112	3 ms	183	45
256	1069 ms	2 ms	525	262	4 ms	267	67

Table III: Simulation result of Serial and Parallel Latin Square

Matrix Size	Serial Latin Square Algorithm Time in ms	Parallel Latin Square Algorithm with 8 Core		
		Time (T)	*Speed Up (S)*	*Efficiency (n)*
4	3 ms	0 ms	0	0
9	14 ms	0 ms	0	0
16	3 ms	0 ms	0	0
25	14 ms	0 ms	0	0
36	18 ms	0 ms	0	0
49	21 ms	0 ms	0	0
64	26 ms	0 ms	0	0
81	35 ms	4 ms	8	1
100	51 ms	0 ms	0	0
121	78 ms	21 ms	3	0
144	123 ms	0 ms	0	0
169	194 ms	2 ms	97	12
196	314 ms	2 ms	157	19
225	549 ms	4 ms	137	17
256	1069 ms	7 ms	152	19

The speedup of parallel Latin square algorithm is shown in Figure 2.

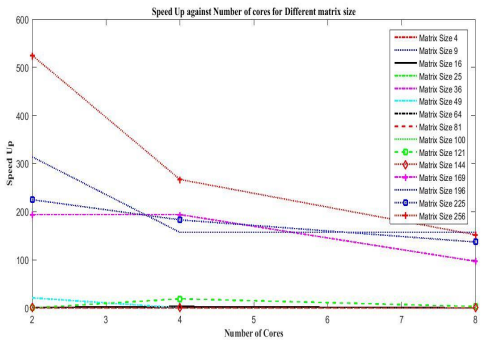


Fig. 2: Illustrates the speedup of Parallel Latin Square algorithm.

In Figure 2, the horizontal x-axis represents the number of cores from 0 to 8. The vertical axis represents the speedup value. As shown clearly from Figure 2, we have achieved

better speed up with core 2 which is a wonderful result of a multi-core architecture.

The Efficiency*(n)* of parallel Latin square algorithm is shown in Figure 3.

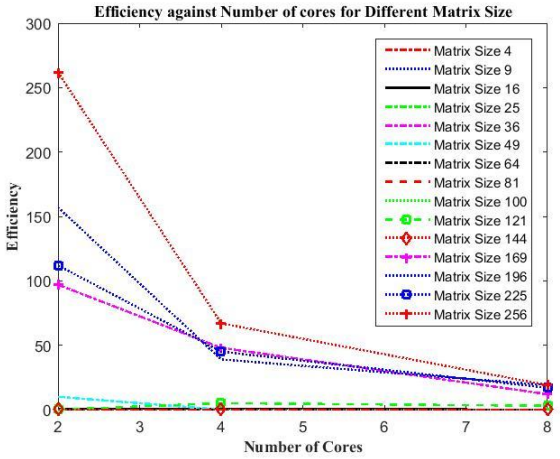


Fig. 3: Illustrates the Efficiency of Parallel Latin Square algorithm.

In Figure 3, the horizontal x-axis represents the number of cores from 0 to 8. The vertical y-axis represents the efficiency value. As shown clearly from Figure 3, we have achieved better efficiency with core 2 which is a wonderful result of a multi-core architecture. Further, result obtained by parallel approach is far better than results generated by traditional serial approach.

V. CONCLUSION

In this paper, we have presented an efficient parallel algorithm for Latin square design which has desirable properties for parallel array access. For this transversal skewing scheme is applied.

We compare the parallel Latin Square algorithm in this paper to the other previously proposed sequential algorithms of Latin Square design [1]. The results have demonstrated that parallel Latin Square algorithm could be a better choice for conflict free routing in multistage interconnection network due to its desirable properties of parallel array access, better efficiency and it achieve speedup up to 200 times for matrix size of 256.

Further work in this direction would be to apply this parallel Latin Square algorithm for conflict access of data for efficient parallel routing in various multistage interconnection networks and find the efficiency and speedup of parallel routing algorithm.

In addition to this, we can add new parameter such as quality of solution so that the performance of parallel algorithm is measured on the basis of speedup, efficiency and quality of solution.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers of this article for their comments and suggestions.

REFERENCES

[1] Andersen, Lars Døvling. Chapter on The history of Latin squares. Department of Mathematical Sciences, Aalborg University, 2007.

[2] Boppana, Rajendra V., and C. S. Raghavendra, " Self routing scheme in parallel memory access", Technical report no ceneg 89- 36, 12th December 1989.

[3] K. Heinrich, K. Kim, and V.K. Prasanna Kumar, "Perfect Latin squares," Discrete applied mathematics, pp.281-286, April 1990.

[4] K. Kim, and V. K. Prasanna Kumar, "Perfect Latin squares and parallel array access," ACM SIGARCH Computer Architecture pp.372-379,1989.

[5] Carla P. Gomesa, Rommel G. Regisb, and David B. Shmoys, "An improved approximation algorithm for the partial Latin square extension problem", Operations Research Letters, Press, 479-84, 2004.

[6] Bae, Yongeun, Chunkyun Youn, and Ilyong Chung. "Application of the hamiltonian circuit Latin square to the parallel routing algorithm on 2-circulant networks," International Conference on Computational and Information Science, Springer Berlin Heidelberg, 2004.

[7] Beyer, and Andrew James, "Enumeration of orthogonal Latin squares." PhD diss., San Francisco State University, 2010.

[8] Y. Yuanyuan, and J. Wang,"Routing permutations with link-disjoint and node-disjoint paths in a class of self-routable interconnects," IEEE Transactions on Parallel and Distributed Systems , vol. 14, no. 4, pp. 383-393, April 2003.

[9] Y. Yuanyuan, and J. Wang,"Optimal all-to-all personalized exchange in self-routable multistage networks," IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 3, pp.261-274, March 2000.

[10] X. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," Journal of Parallel and Distributed Computing, pp.183-188,May2009.