International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 06 | June 2019 www.irjet.net p-ISSN: 2395-0072

Latin Square Computation of Order-3 using Open CL

Avnish Kansal[1], Ashish Chaturwedi[2]

[1,2]*Department of Computer Science & Engineering, Carlox Teacher's University, Ahmedabad, Gujarat*

Abstract: Latin sqaure is widely used in steganography, cryptography, digital watermarks, computer games, sudoku, graph analysis, error correcting codes; generate magic squares, statistics and mathematical field. The Sudoku puzzles are a special case of Latin squares. When we have to make the latin computation using the sequential algorithm then it waste more clock time. By using parallel programming (OpenCL) the time taken is reduced and throughput is increased. Traditionally Latin square methodology is based on heuristic cell based technique and generates random Latin square using genetic algorithmic approach both consumes high processing time and decreases the throughput. Here we are presenting the algorithm by using parallel processing environment using OpenCL for computing latin square of order3.

Keywords: OpenCL, gnuplot, Sequential architecture, parallel architecture, GPU.

I. Introduction:

Latin square is an $n \times n$ array in which each cell is having at most one symbol, chosen form an n-set, such that every symbol occurs at most one time in each row and at- most one time in each column. The "Latin square" name was stimulated by mathematical papers by Leonhard Euler. Two latin squares are said to be orthogonal if both Latin squares of the same size such that when one latin square is superposed on the other latin, each letter of the one coincides once with each letter of the other. The two Latin squares are held to be conjugate if the rows of one are the columns of the other that is if the rows and columns of a square be interchanged then conjugate square is generated. An Adjugacy is a generality of the concept to conjugacy in which a permutation of the constraints of one generates another [12]. Each Latin square is defined as a triple (r,c,s), where r is the row, c is the column, and s is the symbol and from this triplet we attain a set of n^2 triples called the orthogonal array representation of the square. Latin square of n x n order, in which every row is derivative from any other in a cyclic permutation of degree n, or by a power of such a permutation, is a cyclic Latin square [9].

While the implementation done using sequential algorithms techniques but with help of high level languages parallel processing technique we are able to decrease the processing time for matrix processing application [3]. As the problem is divided into discrete set of instances which are solved concurrently [1]. With help of this parallel computation technique we are able to execute two or more instructions at a same time simultaneously. While executing the sequential algorithms on a CPU it runs slower. In the proposed system the sequential algorithms which have task parallelism or data parallelism those algorithms are implemented to OpenCL [14]. By the help of OpenCL we minimizes overhead on the CPU and makes matrix processing run faster and efficiently to get higher throughput.

Concurrency is the way to sharing of multiple resources in a software. For the applications that are naturally parallel the concurrency provides an abstraction [5].

When the execution of the multiple threads running in parallel, this means that the active thread running simultaneously on different hardware resources and processing elements [10]. The execution of the simultaneous threads is provided by the platforms, for achieving parallel computing. In latest computing machines we have SIMD or MIMD which have capabilities to exploit either data level or task level parallism[6].

Task level parallism, to handle different number of tasks, within a single problem at the same time. The efficiency of this model will depend on the independent operations of the task [6].

Data level parallelism, to handle the discrete chunks of the same task at the same time simultaneously. The efficiency of this model will depend on the independent operations of the task [6].

1.1 Existing System & Proposed System

Traditionally all the computations and execution of instructions are handled by the CPU in the computer. Architecturally, the CPU consists of very few cores with lots of cache memory which are able to handle a few software threads at a time concurrently. These few cores are used to optimize the query by sequential serial

processing architecture. So to alleviate the load of the CPU by handle all its advanced computations which are necessary to project the final display on the monitor a concept of GPU's being introduced. The capability of the GPUs with 100+ cores to process thousands of threads can accelerate software by 100x over a CPU alone. The GPUs have massively parallel architecture which consists of thousands of smaller and more efficient cores that are designed for handling multiple tasks simultaneously. This is the reason for the wide and mainstream acceptance of the GPU's now a day. The GPU-accelerated computing has now grown into a mainstream movement that is supported by the latest operating systems from Apple (with OpenCL) and Microsoft (using DirectCompute). This accelerated computing consist of graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications.

1.2 Defining Algorithm

The parallel processing algorithms are intended for Latin square. The representation of algorithms is presented in Figure 1.The first step is input matrix of order 3, we have input as a matrix in OpenCL programming code. The matrix have complex data in the form of array values when we applying sequential algorithm on an matrix it takes more of time for execution on CPU. But if we are applying the parallel processing concepts then we positively we reduce time taken by the matrix execution.

Second step, is decomposing the input matrix according to task parallelism or data parallelism technique. Then this divided matrix is being used in third step for concurrent execution

Third step, is defining the individual sub matrices which are further being processed by various processing elements.

Fourth step, after the division of matrix into sub matrices each individual sub matrix is sent to number of processing elements simultaneously in GPU.

Fifth step, the sub matrices result is being calculated with the help of processing elements concurrently.

Sixth step is to combine the results of all matrices in single processing element so as to obtain the final output of input matrix in reduced time using parallel architecture.

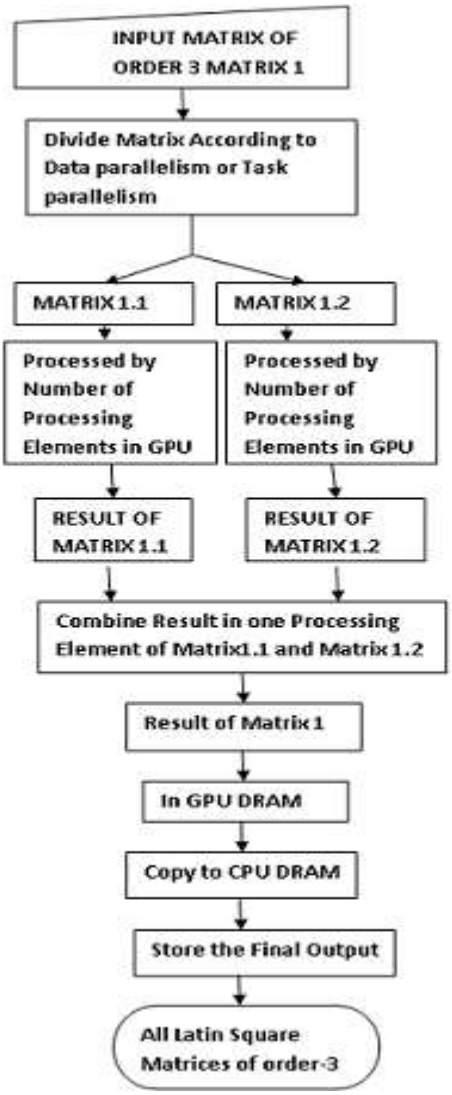

Figure 1 Flowchart of Proposed Work

Seventh step, is to store the final result of input matrix from various sub matrices which are further being saved in the dynamic random access memory (DRAM) of graphical processing unit.

Eighth step, after the final result is saved in the DRAM of GPU; that result is copy to the DRAM of CPU for displaying the final output to the user.

2. Implementing Latin Square using OpenCL:

The algorithm which we are chosen to parallel the matrix over a number of workgroups, the matrix is further divided into number of chunks as per the coarse grained division. The workgroup is the part of a matrix by which we make the sub-matrix stored in the on chip local

memory of the GPU. These partial latin square are further reduced into the single main latin square results.

2.1 Overview is as follows, when we begin to design a OpenCL kernel to the respective hardware we have to take care work-items and the size of the work_group. The local memory is resides in the work_group and we share the data inside the work_group. The local memory is not shared in the work_group only. If we want to share the results of the each work_group to another then we are not able to do this. The work_group in OpenCL consists of work_items which further share between the local memories within that work_group. We are performing all this to reduce the memory overheads by storing sub-matrix in local memory.

After that we have to define more work_groups for exchanging more local memory data into the global memory but we must assure that the number of work_groups for efficient use of local memory and reduce overhead on the global memory the number should be close to the number of compute units we have in our hardware.

To traverse the total input matrix we use here, global Ids, local Ids, group Ids, group size, etc.

Our OpenCL program consists of two parts: First is kernel part, the instances of kernel are copied to the different compute units and the kernel is executed on the each compute unit individually. Another is host code which we have to add to work for the different models of the OpenCL which is executed on the host or CPU.

The host program defines the context for the kernels and manages their execution. Each OpenCL device has a command queue, where the host programs are queued for kernel execution and memory transfer.

The core of the OpenCL execution model is defined by execution of kernels. When kernel is executed an index space is defined which is an instance of the kernel as in our problem. Each work_item executes the same code but the execution pathway and the data used will be different.

Work items are organized into workgroups. Each work_group is assigned a unique workgroup_Id and each work_item in the workgroup assigned the local_Id.

A single work_group can be identified by its global_Id or by its workgroup_Id or local_Id. The work_item in a single work_group executes concurrently on a processing elements of a single compute units. Work_item in a work_group can synchronize each other and share data through local memory in the compute unit. All the work_items has read and write access to any position in the global memory. The global and constant memory can also be accessed from the host processor before and after kernel execution.

3 Experiments and results: For each latin square, both the GPU kernel code and CPU serial code are designed. The processing that takes place on the CPU; the kernel code in the algorithm as an instance copied to the GPU. The kernel is executed on compute unit and after that the results are copied to the CPU.

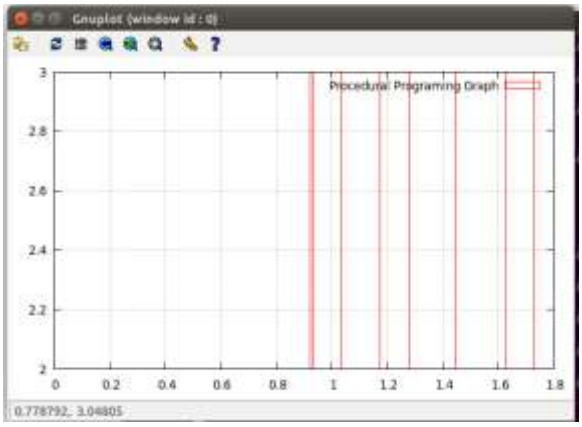

Figure 2: Latin Square Computation (Order-3) order v/s time graph using procedural programming code technique having more time complexity.

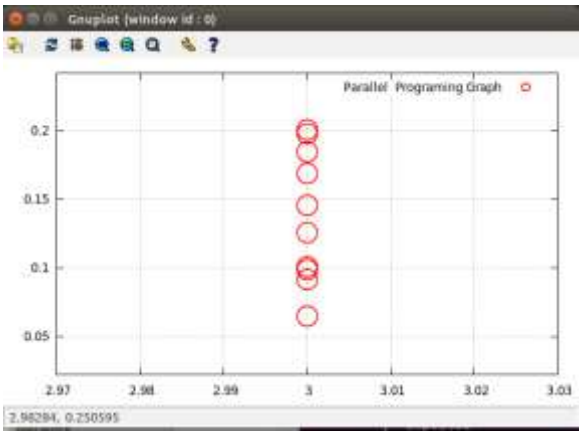

Figure 3: Latin Square Computation (order-3) time v/s order graph using OpenCL code having reduced time complexity.

The speed up of the latin square computation by the GPU is significantly improves the computing speed by reducing the time complexity and throughput. Here the graph depicts while using the procedural programming

technique the time complexity increases as the order for latin square increased. But when the same code is implemented using OpenCL the time complexity decreases tremendously.

4. Conclusions

In this paper a framework is tried to be developed an efficient algorithmic approach to find a solution to different latin square to enhance the existing approach. We implemented the Latin square problem definition of order-3 in GPGPU (GP2U), which providing heterogeneous environment to execute and capability to reduce time complexity which improves the performance of Latin square computation over intensive domains latin square We executed the latin square by using OpenCL environment and compared with the sequential implementations on CPU. On CPU algorithm takes very huge amount of time. Obviously, the time taken becomes low on GPU device. It provides novel and efficient acceleration technique for matrix calculation and is cheap in hardware implementation.

Future work is to gain deeper knowledge about the parallization techniques to make best use of GPU device and to work with other algorithms related to these project concepts.

5. **References**

[1] "*Heterogeneous computing with OpenCL*" by Benedict gaster,british libraries,printed USA

[2] http://www.khronos.org/OpenCL ,"Khronos Group".

[3] Roberto Fontana, Random Latin squares and Sudoku designs generation,2013

[4] Nan Zhang, Yun-shan Chen, Jian-li Wang. "*Image Parallel Processing Based on GPU*". International Conference on Advanced Computer Control, March 2010.

[5] Pardalos P.M., Xue, J, "*The maximum clique problem*", Journal of Global Optimization, 4, 1994, 301—328.

[6] Demetres Christofides, Klas Markstrom (2003) Random Latin square graphs

[7]http://developer.amd.com/pages/default.aspx
"University Kit 1.0".

[8] C.Colbourn(1984) The Complexity of completing partial Latin squares. Discrete Applied Mathematics8: 25-30. Doi:10.1016/0166-218X(84)90075-1

[9] Denes, J. and A. Keedwell. 1991. Latin Squares: New Developments in the Theory and applications. North-Holland.

[10] *AMD Accelerated Parallel Processing OpenCL Programming Guide1.pdf*

[11] Jacobson, M.T and P. Matthews (1996) Generating uniformly distributed random Latin squares. Journals of Combinatorial Designs 4(6), 405-406

[12] Brendan D. McKay and Ian M. Wanless(2000) On the number of Latin squares Australian National University, Canberra, ACT 0200, Australia

[13] J.A. Bate, G.H.J. van Rees, The Size of the Smallest Strong Critical Set in a Latin Square University of Manitoba, Winnipeg, Manitoba.