Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 5, May 2015, pg.825 – 833

RESEARCH ARTICLE

Parallelization of Node Based Game Tree Search Algorithm on GPU

Ms. Rutuja U. Gosavi[1], Mrs. Archana S. Vaidya[2]

[1]Department of Computer Engineering, GES's R. H. Sapat College of Engineering Management Studies & Research, Savitribai Phule Pune University, India

[2]Department of Computer Engineering, GES's R. H. Sapat College of Engineering Management Studies & Research, Savitribai Phule Pune University, India

[1] rutu.gosavi@gmail.com; [2] archana.s.vaidya.@gmail.com

Abstract- GTS is directed graph whose nodes indicates position and edges indicates moves in a game theory. Game tree is the hierarchical tree structure begins at the initial position as a node and contains all possible moves from each position. Game tree Search algorithm is used to search the best move in game tree. GTS is a combinatorial problem in which it is hard to find optimal solution from huge possible solutions. In literature, finding a better GTS algorithm to obtain best solution and use of advanced computing architectures to speed up the GTS computation are mostly studied. Focus of the system is to take advantage of GPU's massive parallelism capability to accelerate the speed of game tree algorithm and propose a concise and general parallel game tree algorithm on GPUs. GPU computing is getting popular among scientific community because of cheap and high performance computational power. Proposed system is implemented for Connect4 and Connect6 game using CUDA and MPI programming environment. It is found that parallelization tasks on SIMD processors of graphics cards perform better during searching and evaluating a GTS. In this CPU is responsible for maintain tree structure of game tree and GPU is responsible for evaluating node simultaneously. Thus choice is to use combination of CPU-GPU solution with DFS-BFS search respectively. Comparison is done with serial implementation for Connect4 and Connect 6 games.

Keywords- Connect4/Connect6, CUDA, GPU, Game Tree Search, Parallel Computing

I. INTRODUCTION

GPU is a powerful support for massive parallel computing for real time applications. The GPU is processor as a multiple core having support for many of threads running parallel.

GPUs are outcome of many processors with core aligned in a way that forms a unit of hardware. Cores may be a hundred or more as per specification. General purpose CPU tied applications which have important data in-dependency suits to the devices. So, parallel data or parallel codes perform efficiently since hardware can be classified as SIMD. Parallel work is implemented on GPU with the CUDA development environment.

CUDA stands for Compute Unified Device Architecture. It is framework or platform for parallel computation and also programming model created by NVIDIA and implemented by the GPUs. As CUDA is framework for parallel work it offers direct access of memory and virtual instruction set to the developers. Many applications are getting benefits from GPU massive parallelism capacity [2][3][4]. GPU used for solving Artificial Intelligence queries successfully [5]. GPU is another way of solving Artificial Intelligence queries or problems that are generally compute intensive because of its SIMD architecture specialized for parallel

computing. The SIMD is termed as Single Instruction Multiple Data architecture. It is used to performing same operation or instruction on multiple data simultaneously. Hence such devices perform concurrent computation but uses only one instruction, so these exploit a data level parallelism. GTS is key approach in AI as it used to choose nest move or best move in computer games or real time applications.

A. Game Tree Search on GPU

Game tree search is digraph, nodes in digraph indicate position and lines or edges denote moves in a game as it mentioned in a game theory. From game of point view, the complete game tree is the hierarchical game tree structure begins at the initial position as a node and containing all possible moves from each position. It is very hard to find optimal solution for taking best move for many computer games as it contains exponential time complexity, games like Connect4/Connect6 [6], Sim, Chess[7],Havannah etc. The focus on GTS algorithm to obtain near-optimal solutions using node based approach. It is used to speed up or accelerating the GTS algorithms for the computer.

B. Necessity of Parallelism

To satisfy a demand of reduction in a computational time for games, parallel computing needed to introduced an improve GTS algorithm performance. General purpose CPU-based parallelism approaches have been studied for many years [8][9]. In CUDA programming model parallelism is achieved through its set of parallel threads or multi-threaded architecture. These threads are organized into number of blocks and combination of blocks forms grids of thread blocks.

To each grid the kernel executes concurrently. Kernel is a user defined C function which is executed in GPU. On GPU parallel computation is performed by executing thread blocks concurrently. These are arranged into a Dimensional structure like 1D, 2D or 3D manner as shown in figure 1. The CUDA threads are organized into a two-level hierarchy using unique coordinates called block ID and thread ID. Hence each thread can be uniquely identified by its ID which is represented by the built in variable blockldx and threadIdx. A group of 32 threads with consecutive thread IDs is called a Warp, which is the unit of thread scheduling. Computation on graphics cards is taking complete advantage of usage of GPU which performs operations which are related to the computer graphics, which was performed by CPU traditionally. GPU achieves a higher performance parallelism in some specific tasks as compare to the conventional processor. Thus use of multiple graphics cards in single computer, even in large numbers of graphics chips helps for parallelizing the already parallel nature of graphics processing. Thus basic tasks are decomposed into small one that can be further evaluated or processed concurrently by GPU. It improves the computational time to find answer which is the feasible optimal solution. For getting right solution we are taking advantage of SIMD nature of GPU that allocating instruction to the number of threads and they works on finding next move of the game.

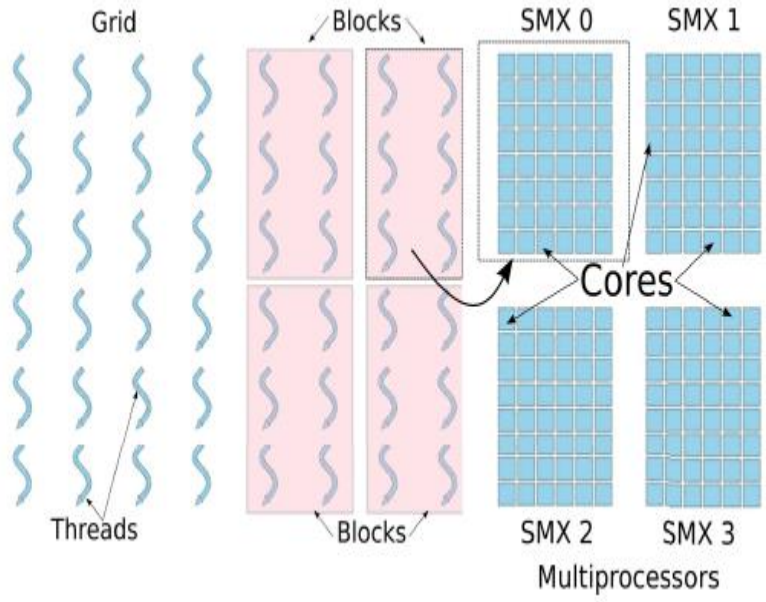

Fig. 1. The CUDA thread block structure

II. LITERATURE SURVEY

A work has been done on a game tree search. It is a search function that explores all possible moves of games and it results into giving optimal move to the player or node in terms of directed graph. Basically system was designed using classical tree-based algorithms on CPU. According to study some problems or challenges appeared:

- Designing of algorithm complexity due to SIMD structure.
- GPU gives performance degradation of divergence.
- Pruning efficiency is low in parallel GTS algorithm.

Exploitation of parallelism on GPU can gives solution to these above problems [1].
Algorithms used for getting solution in computer games includes negamax, minimax with alphabeta pruning [10][11]. These algorithms are useful for searching game trees and these are very widely used. The system was implemented using three algorithms Principal Variation Splitting, Enhanced Principal Variation Splitting, Dynamic Tree Splitting respectively.

A. Principal Variation Splitting

Principal Variation Splitting (PVS) is a straightforward and efficient parallel GTS algorithm on CPU. Two processors were used in that P1, P0 used to search game tree serially, marked by principal node. Once the search of principal nodes finished, according the PVS algorithm, all processors take efforts to begin parallel search by taking unvisited nodes. But this algorithm has pitfalls: there is no synchronization among processors, the one who finished its task need to wait till other one complete and it increases time of calculation because of serial approach.

B. Enhanced Principal Variation

To overcome drawback of PVS algorithm therefore, Enhanced PVS (EPVS) is introduced in which subtrees are assigned to idle processors from other busy processors. The performance improvement is significant for an unbalance tree and efficiency is improved. But the enhanced method will bring extra communication overhead.

C. Dynamic Tree Splitting

In this a peer-to-peer model is used on multi-processor systems. In which global list of active split-points are stored and are shared by all processor. This list called as SP-LIST is used to store nodes. This list is used to find uncalculated nodes which are still not processed. Initially SP-LIST is empty or made cleared. After that one processor will take the main root node of the game tree. During that period other nodes remain in the idle state. When first one finishes its tasks, an idle processor will look up SP-LIST to find a node to traverse. If SP-LIST contains no points or nodes, then idle processor will broadcast a HELP message to all processors. Busy processors that receive the message and saves copy of the state of the subtree to SP-LIST. SP-LIST is again consult by the idle processor then consults SP-LIST again and obtains a split point. The advantages of DTS algorithm are its usability and scalability over PVS and EPVS. DTS does not split the tree by specific nodes. This DTS algorithm is able to search nodes simultaneously, efficiently because addition of processor is oversimplified as it uses point-to-point protocol, which results in high scalability in the algorithm. Still these three algorithms are tree-based.

III. IMPLEMENTATION STRATEGY

GPU consist of capacity of processing thousands of nodes simultaneously. GPU is subtle to the instruction divergence because of warp. In GPU-based GTS algorithm, the kernel of GPU is a user defined C function used to calculate the node and also involves lots of control flow instructions because of game rules.
PVS, EPVS, DTS algorithms are implemented using tree based approach consist of drawbacks like searching is done using DFS manner, it requires more time for computation. Game is splitted into number of possible choices that are considered as possible moves which is next best move for player. Using a tree-based approach many choices of games are computed serially by processor in DFS manner. Due to the SIMD feature of GPU, the tree based approach cannot be easily adopted in GPU. Different from tree-based approach, node-based approach is advantageous in which CPU generates number of possible trees contains the nodes as well as leaf. Number of possible moves in form of tree is created on CPU. CPU is responsible for execution control as well as it is responsible for maintaining the game tree structure.

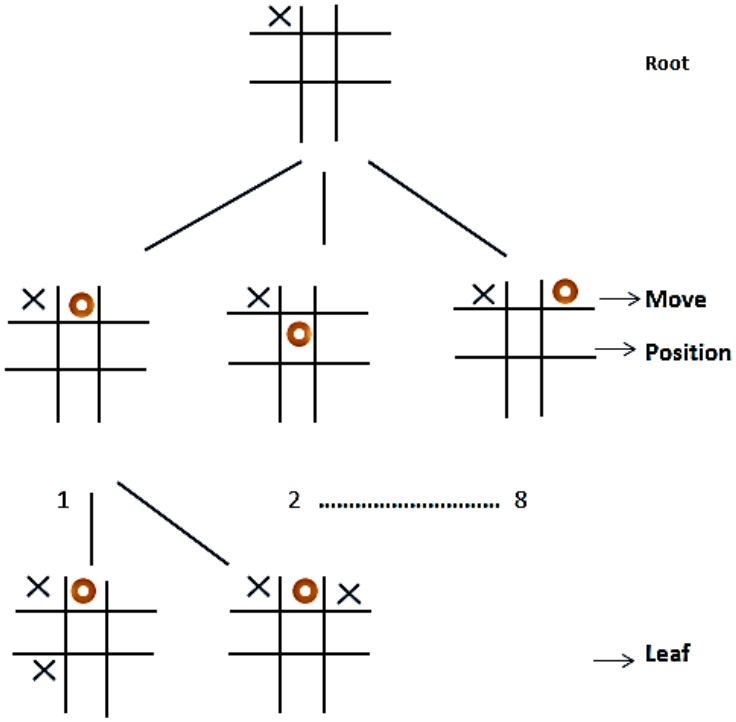

Fig. 2. The Two-Ply Game Tree of Tic-tac-toe

Representation of the game is shown as tree in figure 2. The tree starts at the initial position of the game as its root. Moves of the players are represented by "X" and "O". The player represented by the "X", plays game first and each node in this tree is a possible situation of the game. Possible moves by another player for "O" are generated on the CPU which is tree structure. Evaluation of all nodes, leafs are done by number of threads. Calculation of many tree nodes is done in the same depth in the current game tree, which is the BFS search. Further each cycle in the search process will take in the deepest nodes of the current game tree, which is the DFS search. That means on DFS approach CPU works to calculate nodes, since CPU will execute faster than GPU in this situation and on BFS approach GPU used for calculating the branch and the leaf nodes in parallel. By this hybrid manner, our algorithm is fully utilizes both architectures.

The pruning procedure is another key component of our node-based algorithm which is used to prune redundant nodes. Finally solution is returned to the root node. Such a hybrid approach takes an advantage of computation on CPU in DFS manner and evaluation of nodes by GPU in BFS manner. This approach can be applied for HEX, CHESS, Connect4/Connect6 games. The game like Connect4/Connect6 is compared with serial implementation and single GPU implementation. Approach is made for improving the GTS algorithm and extends it to the large-scale clusters with GPU environments. In this work a well-established HPC framework, Message Passing Interface is used to minimize amount of latency and handle the communication between the devices. The work is responsible for dividing node generation task on multiple CPU using MPI. Each CPU is uses GPU for node evaluation task.

A. System Architecture

The system architecture of parallel evaluation with combination of CPU-GPU is shown in the fig 3. The system works with combination of both, where CPU does calculation and GPU evaluates nodes parallely.

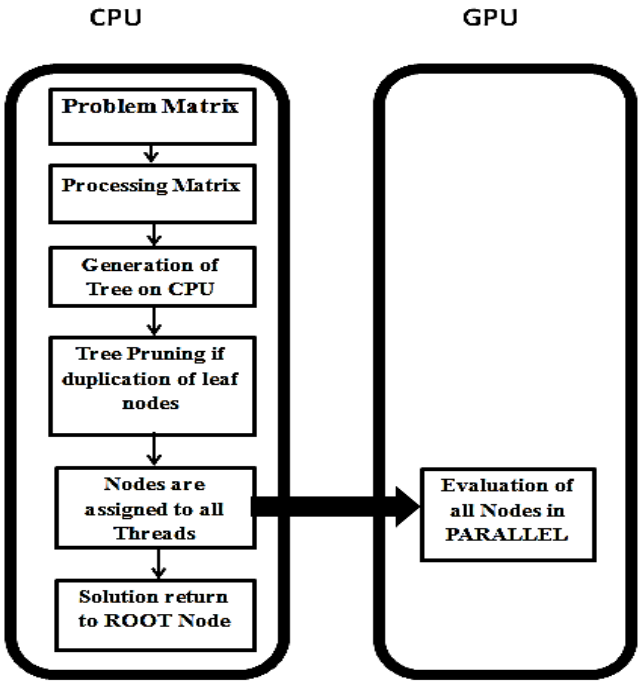

Fig.3. System Architecture

B. Methodology

The most common goal of game tree search is finding of players move that maximizes his chance of winning. To achieve this goal it is necessary to create game tree as much as possible generated by legal moves of both players. Consequently the game tree is calculated and it is chosen the move that leads to highest price. Node-based GTS algorithm does functionality as discussed for Tic-tac-toe.
From figure 3, CPU and GPU are shown differently. Problem data set is nothing but matrix which is provided as an input. CPU performs operation like maintaining tree structure, processing of data, generation of all nodes, tree pruning, checking of leaf nodes, solution returned to the root node. Where number of tree nodes are evaluated by threads in GPU section.

C. Tree Generation on CPU

The use of DFS and BFS approach will provide not only calculate nodes on GPU, but also avoid the exponential growth of the space complexity through the parallel search process and prune nodes after calculation on GPU. When matrix is provided as an input to the system, CPU generates number of possible trees contains the nodes as well as leaf. That means generation of possible move of the player is created on the CPU. CPU is responsible for maintaining the game tree structure.

D. Parallel Evaluation on GPU

Calculation of many tree nodes is done in the same depth in the current game tree, which is the breadth-first search. In addition, each cycle in the search process will take in the deepest nodes of the current game tree, which is the depth-first search. That means on DFS approach CPU works to calculate nodes, since CPU will execute faster than GPU in this situation and on BFS approach GPU used for calculating the branch and the leaf nodes in parallel.

E. Algorithm

Node-based GTS Algorithm
Input: Initial position for GTS P0
Output: Best Move M_{Best}

Begin:
Step1: Set P0 as the root of the GTS
Step2: if Tree T Null then
Step3: return
Step4: End if
Step5: else
Step6: T formation on CPU
Step7: Node formation and structure maintain on CPU
Step8: Depth First Search to process tree and prunes redundant nodes
Step9: Leaves, branch nodes are assigned to GPU to calculate concurrently
Step10: Calculates branches and leafs nodes in parallel as Breadth first search
Step11: Updates parent node P0
Step12: Returns result M_{Best}
End

Algorithmic strategy mentions tree generation and parallel evaluation. The pruning procedure is another key component of Node-based algorithm. That is after calculating scores for all leaves, algorithm update the parent node and check its brother nodes to cut off some nodes as per the pruning procedure. This algorithm is used for achieving good pruning efficiency.

IV. PROBLEM MODELLING

As SIMD feature GPU, thread works on the same instruction but processes multiple data. Let Sf is solution that operates number of threads by the system.
Input is matrix of game applied to the system.
Processing is done by both tree generation and node evaluation CPU and GPU respectively.
Output is next move with minimum time returned to the root.

Sf = { P0, T, T',D, S, V, Fi }

Where,
P0: is a problem set contains a1, a2, a3, an numbers.

Fi: Functions,
T: Trees built on the CPU
D: Threads which parallel evaluates nodes on GPU.
V: stores value 0 or 1.
T: Thread processing leaf nodes
Sf :Final Solution

Let,
P0=a1,a2,a3,a4,….an;
T=P1,P2,P3,P4,.Pn
T'=P1',P2',P3',P4',.Pn'
D=D1,D2,D3,D4,..Dn
V=0,1
Sf=T
Fi=F1,F2,F2,F3,F4

A. *Input to the system:*

P0=a1,a2,a3,a4,….an;
a1,a2,a3,a4,….an are numbers which is given as input in this case it is input board matrix. Numbers are possible values differ as per games.

Function F1:
It reads all generated tree on CPU. Trees are all possible moves for the players.
F1:P0 {T}

T= { P1,P2,P3,P4,.Pn};
T: Tree generates on CPU, where P1,P2,P3,P4,.Pn are the nodes of the tree.

Function F2: It assigns generated tree to the threads.
F2:(T,D)->S
Here T is made up of number of nodes which are assigned to the D that is threads.
S is a solution in which performs node computation. Parallel evaluation is performing.

Function F3:
F3:S->T'
In function F2 parallel evaluation is performed. T' represents a parallel evaluation on GPU
Where T'={P1',P2',P3'…Pn} m<= n parallel evaluation of each nodes till leaf node get. m<=n is evaluation of nodes until leaf node is found.

Function F4:
F4:Pi->V
V{0,1}
This is a choice of move to be made or not depends on setting value of 0, 1.

B. Output from the system

Pi represents output function to read all nodes generated by tree. Depending on V{0, 1} is optimized next move for a player or system.
Pi takes all evaluates node by the threads. It returns a node which contains an optimized next move.
Pi(ti')-> Sf is a final solution generated for the player by parallel evaluation.

V. EXPERIMENT AND RESULTS

Connect 4 and connect 6 game is performed on a machine contains CPU configuration Intel ® Core i3-4150 CPU @ 3.50GHz x 4 with 7.7 GiB memory. Operating system is UBUTNTU 14.04 LTS of 64 bit. GPU configuration consist of Quadro 6000 Graphics Processor, 448 CUDA cores, Total memory 6144 MB. We use CUDA toolkit 4.0 version to implement algorithms.
Table consist of whole value of results seconds. Example
1.(4260.42) seconds is equal to (71.007) min or (4260.42)sec is equal to (1.19)hr.
2. (0.266) seconds is equal to (0.0045) min
Following Table shows results of Connect4/Connect6 games.

TABLE I: SHOWS RESULTS OF CONNECT4 GAME

Board Width	Board Height	Tree max depth	Average CPU time (seconds)	Average GPU time (seconds)
8	8	8	20.70	0.266
8	8	10	40.78	0.261
8	8	12	1356.60	0.230
8	8	14	3160.50	0.229
8	8	16	4260.42	0.254

TABLE II: SHOWS RESULTS OF CONNECT4 GAME

Board Width	Board Height	Tree max depth	Average CPU time (seconds)	Average GPU time (seconds)
10	10	8	101.94	2.705
10	10	10	317.30	2.703
10	10	12	4352.76	2.677
10	10	14	5008.94	2.718
10	10	16	7179.56	2.706

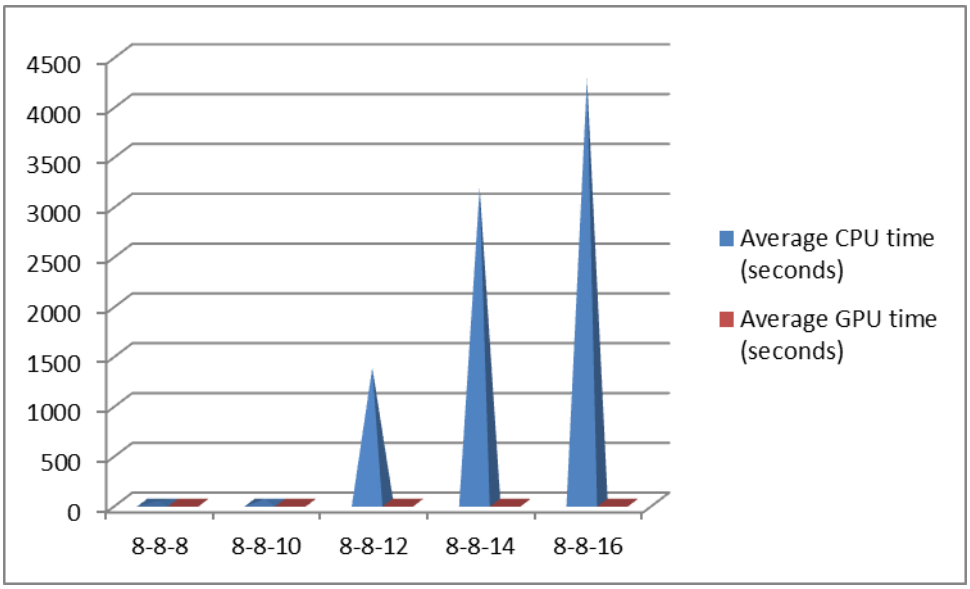

Fig. 4. Results of Connect4 Game

Above graph shows result of Table I consist of Board Width-Height-Depth at X-Axis

TABLE III: SHOWS RESULTS OF CONNECT6 GAME

Board Width	**Board Height**	**Tree max depth**	**Average CPU time (seconds)**	**Average GPU time (seconds)**
8	8	8	20.33	0.225
8	8	10	409.45	0.240
8	8	12	2593.02	0.278
8	8	14	2590.40	0.181
8	8	16	2700.52	0.333

TABLE IV: SHOWS RESULTS OF CONNECT6 GAME

Board Width	**Board Height**	**Tree max depth**	**Average CPU time (seconds)**	**Average GPU time (seconds)**
10	10	8	72.83	2.617
10	10	10	696.27	2.756
10	10	12	2720.80	2.652
10	10	14	2789.89	2.942
10	10	16	2856.78	2.950

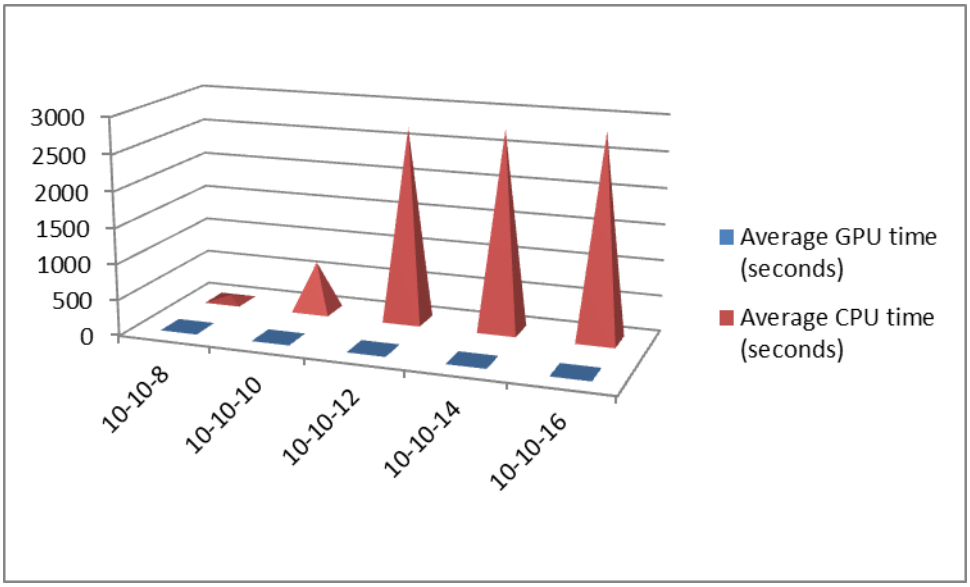

Fig.5. Results of Connect6 Game

Above graph shows result of Table IV consist of Board Width-Height-Depth at X-Axis

VI. CONCLUSION AND FUTURE SCOPE

System focuses on a Parallelization of Node based Game Tree Search Algorithm on GPU. Parallel GTS algorithm presented node based approach for obtaining speedy optimal solution of real time computer games on GPU using node-based parallel computing for GTS and combination of DFS-BFS on CPU-GPU respectively.

Connect4/Connect6 game is tested on Quadro 6000 Graphics Processor having 448 CUDA cores. From graphical representation it is seen that node based GTS algorithm gains the speed over traditional approach. Future scope can be extended for large scale clusters with GPU environments and applicable for other games.

ACKNOWLEDGEMENT

We are glad to express our sentiments of gratitude to all who rendered their valuable guidance to us. We would like to express our appreciation and thanks to Prof. Dr. P. C. Kulkarni, Principal, G. E. S's. R. H. Sapat College of Engg., Nashik. We are also thankful to Prof. N. V. Alone, Head of Department, Computer Engg., G. E. S. R. H. Sapat College of Engg., Nashik. We thank the anonymous reviewers for their comments.

REFERENCES

[1] Liang Li, Hong Liu, HaoWang, Taoying Liu, Wei Li, "*A Parallel algorithm for Game Tree Search using GPGPU"*, IEEE Transaction on Parallel and Distributed Systems,31 July,2014
[2] X. Huo, V. T. Ravi, W. Ma, and G. Agrawal, "*Approaches for parallelizing reductions on modern GPU*", International Conference on High Performance Computing (HiPC), 2010, pp. 1-10.
[3] W. Ma and G. Agrawal, "*An integer programming framework for optimizing shared memory use on GPU*", International Conference on High Performance Computing (HiPC), 2010, pp. 1-10.
[4] J. Soman, M. K. Kumar, K. Kothapalli, and P. J. Narayanan, "*Efficient Discrete Range Searching primitives on the GPU with applications*", International Conference on High Performance Computing (HiPC), 2010, pp. 1-10.
[5] A. Bleiweiss, "*GPU Accelerated Pathfinding*", The 23rd ACM Symposium on Graphics Hardware, 2008, pp. 65-74.
[6] H.van den Herik, S.-C. Hsu, T. Hsu, H. Donkers, I.C. Wu and D.Y. Huang, "*A New Family of k-in-a-Row Games*", Eds. Springer Berlin /Heidelberg, 4250:180-194, 2006
[7] C. E. Shannon, *"Programming a Computer for Playing Chess*", Philosophical Magazine Series 7, 41(314):256-275, 1950.
[8] M. G. Brockington, "*Taxonomy of Parallel Game-Tree Search Algorithms*", 1996.
[9] R. M. Karp and Y. Zhang, "*On parallel evaluation of game trees*", The first annual ACM symposium on Parallel algorithms and architectures, 1989, pp. 409-420.
[10] V. Manohararajah, *"Parallel alpha-beta search on shared memory multiprocessors*", 2001.
[11] P. Borovska and M. Lazarova, "*Efficiency of parallel minimax algorithm for GTS*", The international conference on Computer systems, 2007

