International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075, Volume-9 Issue-4S2, March 2020

Parallel Computing of PVS Node based GTS Algorithm on GPU

T. Ramathulasi, M. Rajasekhara Babu

***Abstract:** The developments in the field of computer architecture, it allows humans to play games like Chess, Tic-tac-toe, Go, etc. with computer machines using AI technology. In AI, Game tree search (GTS) is an important approach and is directed toward finding the finest choice of move for computer games. Using the traditional GTS algorithm the computer could not win a human. So there is a need for enhancing algorithms using dynamic parallelism of GPU. The block of thread set is organized on GPU is either one or two or three dimensional for parallel computations. On this, each thread designated by an inimitable mixture of indices. In this paper, parallel computing of node-based Principal Variation Search (PVS) GTS algorithm presented, which runs on GPU using libraries of CUDA. this experiment tested on chess games with different depths and results are compared with the threads on CPU and GPU. The results proved that GPU improves the performance of speedup up to 80 percent on the checkers game. Parallel computing greatly increasing the efficient use of CPU and improves the performances of the PVS-GTS algorithm on GPU to search deeper layers and find the optimal moves for the current players for two-player computer games.*

***Keywords:** PVS algorithm, parallel computing, Artificial Intelligence, CUDA.*

I. INTRODUCTION

The solution for to find the optimal and finest move strategy for games played by computers using the method of AI (Artificial Intelligence) is called Game tree-Search(GTS). In-game theory, a game tree is a directed graph [1] in which posi-tions are nodes represents different states, edges are choices of moves players' current game choices. The principle of the GTS method is, aggressively stimulate the structure of tree from the current existing node called position, traversing the GST, assess all feasible edges called moves and encounter the finest edge [2]. GTS performs two primary functions. One, the search function of GTS mainly targets on traversing the game tree to asset the finest choice way to move. In this case, using historic knowledge and pruning, we can avoid irrelevant visiting on convinced positions. The second, node function consists of the evaluation function and moves generation function. It calculates scores and assigns for each node or position, the position with a high score value will be selected for the finest choice of the move by player. This computation of scores depends on rules for the respective game and may differ for different games [3].

In general complexity of the game can be measured by quite a few factors such as depth average, branching factor, time complexity and computation time.

Revised Manuscript Received on March 18, 2020.

T. Ramathulasi , MTech, computer science engineering from JNTUA, Anantapuram, India.

Dr. M. Rajasekhara Babu, Associate Professor in School of Computer Science and Engineering, Vellore Institute of Technology, Vellore.

Different rules and implementations are differed from game to game but even have some level of complexity games i.e., calculation of essential nodes and moves.

Table 1 list outs the depth, width and it's time complexities of different two-player computer games. Solution for the current challenging problem, adopt gauge of parallel node based GST algorithm parallelly on GPU. In this, our innovation will assign a collection of different sets of nodes from one as a choice of multiple sub-trees to different processors.

Table 1. Game tree Depths, Widths for some different Games

Name of the Game	Depth	Width	Tree Complexity
Tic-tac-toe	9	4	10^5
Sim	14	3.7	10^8
Connect Four	36	4	10^{21}
Chess	80	35	10^{123}
Connect6	30	46,000	10^{140}
Havannah	66	240	10^{157}
Arima	92	17,281	10^{402}
G0(19X19)	150	250	10^{360}

The algorithms like Negamax [4], PVS [5], YBWC [6] or DTS where numerous nodes run parallelly on both CPU along with GPU for two player computer games like tic-tac-toe, etc.

Launching and implementing of the design of gauging parallelly for this purpose on processor (CPU) using the "Programming Language C" and execute correctly and it is converted to code of GPU using "Compute Unified Device Architecture (CUDA)" [2]. As a first step 'C' processor process the code, sends the signal the kernel with current position of board which used for the current game, then the kernel call contemporary kernel with a dynamic number of threads and these are handled all at once in the GPU using dynamic parallelism. And also its eliminates renege to CPU and call contemporary kernel on GPU shown in Figure 1 Algorithm execution on GPU [7].

II. PARALLEL NODE BASED GST ALGORITHM

We present the parallel computing of nodes based PVS game tree searching algorithm, advantages of our approach. Principal Variation Search (PVS) [8] is a truthful adequate parallel GTS algorithm on GPU. The basic idea of our work intend take enormous nodes through GPU and calculates leaves and branches concurrently and explained the process of these two GPU based functions in the Algorithm 1 and Algorithm 2. By making a list of all positions of a game (pl) as input to GPU from CPU and returns candidates move for the branch calculation, returns an evaluated value (i.e. parameter V) for leaf calculation.

GPU starts execution to calculate fine and optimal moves for the current player of the computer game by taking the current position from the list of pl shown in Algorithm 3.

Algorithm 1. Leaf calculation

```
Input: List of node positions in game tree( $pl$ )
Output: Leaf(node) list VL
1: Obtain current thread $id$
2: According to the current thread $id$ , position list (pl) revise the position of $P_{id}$
3: for every set of point $p$ in position $P_{id}$ do
4:     Observe the shapes about $p$ as stated in the game rule
5:     Follow through the output for the outline
6:     Increase the weigh for $p$ ;
7: end-for
8: summation of assessed values, assign assessed value to list $VL$ ;
9: return value-list
```

Algorithm 2. Branch calculation

```
Input: List of positions in game tree( $pl$ )
Output: Moves(edges) list ML
1: Obtain current thread $id$
2: According to the current thread $id$ , position list (pl) revise the position of $P_{id}$
3: for every set of point $p$ in position $P_{id}$ do
4:     Observe the shapes about $p$ as stated in the game rule
5:     Follow through the output for the outline
6:     Increase the weigh for $p$ ;
7: end-for
8: pick up the respected changes as move-list( $ML$ )
9: return move-list
```

The specific implementation of parallel computing PVS node based GST algorithm is as follows

Algorithm 3. Parallel computing of PVS node based GST

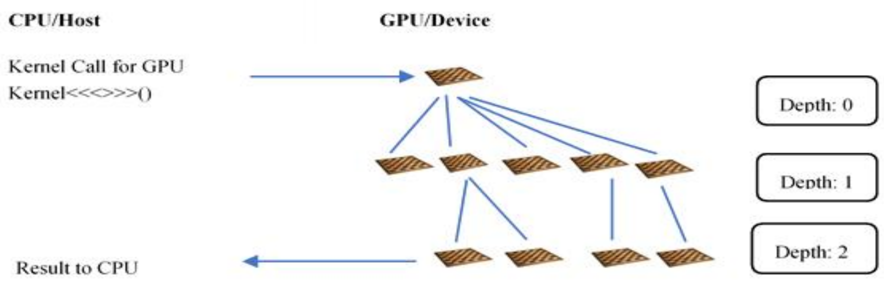

Figure 1 Algorithm execution on GPU

Algorithm

```
Input: position of current node $P_0$
Output: Best move $(MoveBest)$
1: Set up root node $p_0$ for $T$ (Game-tree)as $p_0$
2: If $T$ is free then
3: return move as $MoveBest$
4: end-if
5: if estimation value of rest leaves is more than the value of $LMin$ then
6: Obtain $l$ leaves from tree T, $l \leq LMax$
7: Call Leaf calculation function on GPU
8: Obtain evaluated value list (VL) by GPU
9: for each in $VL$ do
10:        Revise its parent node by each in $T$
11:   if parent node is root then
12:    $MB$ = each
13:   end-if
14:   Exclude $T$ from the revised nodes
15:   end-for
16: else
    Go to Step-2
17: end-if
18:    if the estimation of rest leaves is more than $0$ then
19:  Obtain one leaf node from tree $T$
20:  Call leaf Calculation Function on CPU
21:  Revise the tree $T$ by the amount of leaf node
22:  if $parentnode \equiv root$ then
23:      $MB$ = each
24:  end-if
25: Exclude $T$ from revised nodes;
26: Go to Step-2
27: end-if
28: if the resting branches is more than $B_{\min}$ then
29: Obtain b branches from tree $T, b \leq B_{\max}$
30: Call Branch calculation function on GPU;
31: Obtain $childnodelist$ by GPU;
32: for each in $childnodelist$ do
33:    Revise T by generated child nodes from each;
34:    end-for
35:    Go to Step-2
36: end-if
37: if remain branches is more than 0 then
38:    Obtain one leaf $node$ from tree $T$
39:    Call branch Calculation Function on CPU
40:    Revise $T$ by contemporary child nodes from $node$
41:    Go to Step-2
42: end-if
```

III. PERFORMANCE ANALYSIS

In our algorithm the depth and width of Game tree is and , by considering all the iterations one by one individually and it is necessary to calculate l and b (leaves and branches)'s cooperatively in parallel assets the moderate execution time of l and b's respectively. The total time taken by our algorithm

$$T_{GPU} = t_0 + (m_l \times t_{GPU}(l)) + (m_b \times t_{GPU}(b)) \quad (1)$$

here t_0 represents the beyond of the parallel search algorithm including maintenance of 'Game tree' T, m_l as well as m_b acts variables represents the number of times of loops calculating leaves and branches separately and having relationship among them is

$$m_l \times l + m_b \times b \approx N_{GPU} \quad (2)$$

here N_{GPU} represents the overall estimate of leaves calculated by the proposed algorithm. As GST T ,we have

$$m_l \times l \approx m_b \times b \times w \quad (3)$$

which suggests the furthermost of nodes calculated in the proposed algorithm are leaves. Taken away the above specified equations we Obtain

$$m_b \approx \frac{N_{GPU}}{(1+w)b} \quad (4)$$

$$m_l \approx \frac{N_{GPU}}{(1+\frac{1}{w})l} \quad (5)$$

Substituting (4) & (5) equations in to (1) we have

$$T_{GPU} = t_0 + \frac{N_{GPU}}{(1+\frac{1}{w})l} \times t_{GPU}(l) + \frac{N_{GPU}}{(1+w)b} \times t_{GPU}(b) \quad (6)$$

The time taken by GPU for executing our algorithm can be expressed in equation (6). In fact N_{GPU} holds the value

$$N_{GPU} \le N_{total} \quad (7)$$

where the overall number of nodes in game tree T is N_{total}. That is

$$N_{total} = \sum_{i=0}^{d} w^i = \frac{w^{d+1}-1}{w-1} \quad (8)$$

Now need to analyze different cases of N_{total} to check the performance of our proposed algorithm. At the same time it is difficult to achieve the correct and formal equation of N_{GPU} .

IV. EXPERIMENT ANALYSIS

The above experiment implemented and executed on a two-player computer chess game, results are attained. Besides the comparison of our proposed algorithm with all other algorithms by using the same evaluation function. Even our algorithm also gives the worst results when compare with other algorithms when the search depth is too small and also tests were made for large depth search. The experiment was done on the computer which has the configuration setup as follows:

- Intel(R) Core™ i7-8750H CPU @ 2.20GHz 2.21GHz with six 'physical cores' and twelve 'logical cores'.
- 16GB Random Access Memory(RAM).
- "NVIDIA GeForce GTX" 1050 Ti.

The earlier implementation of 'Negamax' algorithm[9] on GPU with single-core and twelve-core CPU threads results were compared with our PVS-GTS algorithm and results are verified with different depth size searches[10]. This was compared and shown the results in the following figures. Figure 2 Results of chess game on GPU shows the results of the chess game speedup vs level of depth, where the level of depth values on the x-axis, and the y-axis denotes the speedup and Figure 3 computation/ communication ratio on GPU shows the computation and communication percentage on GPU versus increases in speedup.

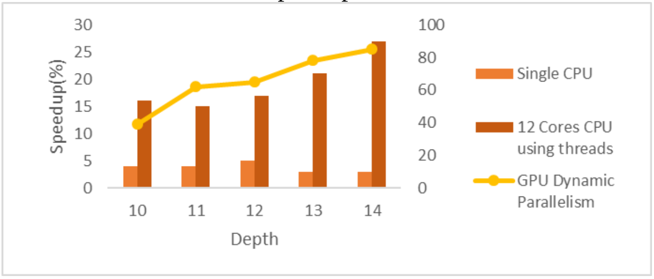

Figure 2 Results of chess game on GPU

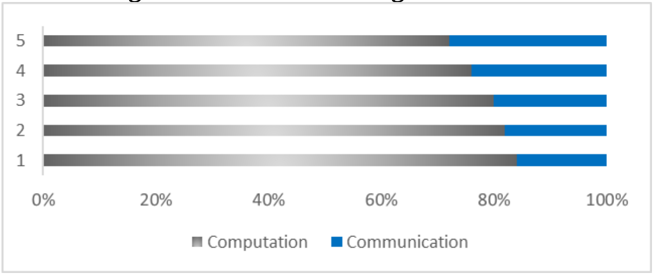

Figure 3 computation/ communication ratio on GPU

V. CONCLUSION

For the framework of the two-player computer game, this paper proposed the PVS algorithm, its optimization with parallel computing of the node-based PVS GTS algorithm. This method can be tested by applying it for the chess game. By using both CPU and GPU architecture, this algorithm takes advantage of dynamic parallelism of GPU to calculate best that gives the optimization solution for two-player computer games.

REFERENCES

1. L. Li, S. Member, H. Liu and H. Wang, "A Parallel Algorithm for Game Tree Search Using GPGPU," IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 8, pp. 2114-2127, 2015.
2. A. A., M. Abdel, M. Gadallah and H. El-Deeb, "A Comparative Study of Game Tree Searching Methods," International Journal of Advanced Computer Science and Applications, vol. 5, no. 5, pp. 68-77, 2014.
3. C. Johnson, L. Barford, S. M. Dascalu and F. C. Harris, "CUDA implementation of computer go game tree search," Advances in Intelligent Systems and Computing, vol. 448, pp. 339-350, 2016.
4. T. Heineman, Algorithms in Nutshells, vol. 18, 2012, pp. 995-996.
5. A. I. G. Seminar, "Enhanced Forward Pruning," no. 3529070, 2019.
6. A. Kishimoto and J. Schaeffer, "Distributed game-tree search using transposition table driven work scheduling," Proceedings of the International Conference on Parallel Processing, Vols. 2002-Janua, pp. 323-330, 2002.
7. A. A. Elnaggar, M. Gadallah, M. A. Aziem and H. El-Deeb, "Enhanced parallel NegaMax tree search algorithm on GPU," PIC 2014 - Proceedings of 2014 IEEE International Conference on Progress in Informatics and Computing, pp. 546-550, 2014.

8. B. Tong, H. Qiu, T. Guo and Y. Wang, "Research and Application of Parallel Computing of PVS Algorithm Based on Amazon Human-Machine Game," 2019 Chinese Control And Decision Conference (CCDC), pp. 6293-6298, 2019.
9. K. Rocki and R. Suda, "Parallel minimax tree searching on GPU," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6067 LNCS, no. PART 1, pp. 449-456, 2010.
10. D. Strnad and N. Guid, "Parallel alpha-beta algorithm on the GPU," Journal of Computing and Information Technology, vol. 19, no. 4, pp. 269-274, 2011.

AUTHORS PROFILE

T. Ramathulasi received the MTech degree in computer science engineering from JNTUA, Anantapuram, India. in 2014. She is currently doing a Ph.D. in the area of High-performance computing. Her research interests include service-oriented computing, software engineering, and data mining.

Dr M Rajasekhara Babu, received the Ph.D. degree from the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, in 2008. He is currently an Associate Professor in School of Computer Science and Engineering, Vellore Institute of Technology, Vellore. He has close to 18 years of experience in High Performance Computing, Internet of Things, Big Data, Cloud Computing, Mobile Applications and has more than 190 publications in these areas.