Parallel and Distributed Search

In game search trees

Sahil Manchanda
CSE Dept. IIT Guwahati
sahil.manchanda@iitg.ernet.in

Manasi Sant
CSE Dept. IIT Guwahati
manasi@iitg.ernet.in

Vianyak Jadhav
CSE Dept. IIT Guwahati
vinayak.jadhav@iitg.ernet.in

Abstract— Game Trees represent position in game and possible moves in the form of a directed graph. Game trees are useful in finding the best move from the current position . Increasing the search depth generally increases the chances of winning but requires more computation (time / resources). Algorithms can be parallelized to some extend.

Keywords—parallel computing in AI; distributed computing in AI; Game search trees

I. INTRODUCTION

Many search algorithms perform search serially. Although to some extent it can be parallelized.
We present the following search algorithms which we have executed in Serial , Parallel and Distributed fashion:
1.State Space Search (SSS*)
2.Negamax with Alpha Beta Pruning
3.Scout Algorithm

II. SSS* ALGORITHM

SSS* is a search algorithm, introduced by George Stockman in 1979, that conducts a state space search traversing a game tree in a best-first fashion similar to that of the A* search algorithm.
It is non-directorial search algorithm.
The algorithm expands into multiple paths at the same time to get global-information of the search tree. However, it searches fewer nodes than fixed depth-first algorithms like Alpha-Beta algorithm.
The algorithm stores information for all active nodes which didn't solve yet in a list in decreasing order depends on their importance. The information consists of three parts:

N: a unique identifier for each node.

S: a status of each node whatever it's live or has been solved.

H: an important value for the node.

The core of the SSS* algorithm depends on two phases:

Node Expansion : Top-down expansion of a Min strategy.

Solution : Bottom-up search for the best Max strategy.

SSS* never examines a node that alpha-beta pruning would prune, and may prune some branches that alpha-beta would not.

A. *Pseudocode* [3]

```
int SSS* (node n; int bound)
{
    push (n, LIVE, bound);
    while ( true ) {
        pop (node);
        switch ( node.status ) {
            case LIVE:
                if (node ## LEAF)
                    insert (node, SOLVED, min(eval(node),h));
                if (node ## MIN_NODE)
                    push (node.1, LIVE, h);
                if (node ## MAX_NODE)
                    for (j#w; j; j--)
                        push (node.j, LIVE, h);
                break;
            case SOLVED:
                if (node ## ROOT_NODE)
                    return (h);
                if (node ## MIN_NODE) {
                    purge (parent(node));
                    push (parent(node), SOLVED, h);
                }
                if (node ## MAX_NODE) {
                    if (node has an unexamined brother)
                        push (brother(node), LIVE, h);
                    else
                        push (parent(node), SOLVED, h);
                }
                break;
        }
    }
}
```

B. Serial Computing

Serial algorithm basically iterates over all the chances/Childs from current state and calculates score for each chance. The move for which the value returned is maximum is considered as next move.

C. Parallel Computing

- As stated above the serial version of the algorithm perform pruning better than alpha-beta search.
- But size of the open list that is priority queue becomes too large. Since priority queue sorts nodes, this degrades the performance of SSS*.
- In parallel version of the algorithm, we tried to split the entire tree into number sub trees depending upon number of available chances/Childs.
- For each tree we run SSS*() and tried to find the optimal value.
- Each process will return the optimal value.
- The move for which the value is maximum is selected by the program

D. Distributed Computing

In distributed version of the algorithm all the computation for each of the search tree is performed on different CPUs.
Each of the next chance is represented as job and each job is assigned to a CPU.
We have tested the program with two machine and three machine in distributed environment.

E. Results

TABLE I. TIME COMPUTATION FOR SSS* ALGORITHM

Height	Serial	Parallel	Cluser size 2	Cluster Size 3
	Time	*Time*	*Wall Time*	*Wall Time*
8	2.357284	1.227879	2.092	2.092
6	0.054065	0.038985	0.138	0.196
4	0.004975	0.016311	0.1	0.132
2	0.000678	0.012078	0.092	0.258

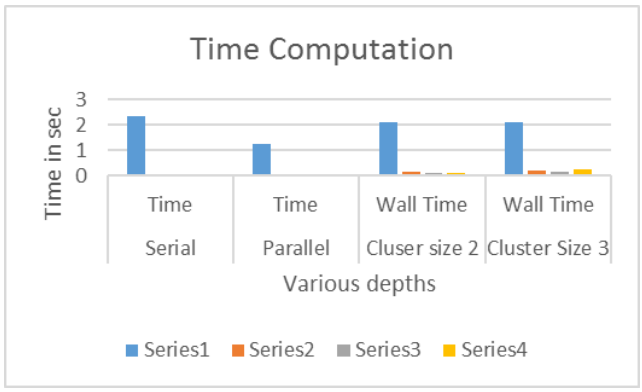

Fig. 1. Time Conputation (SSS* Algorithm)

TABLE II. SPEEDUP FOR SSS* ALGORITHM

Height	Parallel	Cluster SIze 2	Cluster Size 3
8	1.9198025	1.126808942	1.126808942
6	1.3868109	0.391771828	0.275839348
4	0.3049814	0.049746037	0.037686391
2	0.0561017	0.007365061	0.002626301

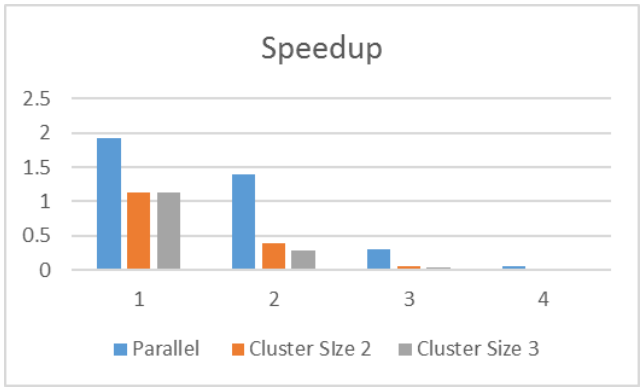

Fig. 2. Speedup (SSS* Algorithm)

F. Observation

In parallel version of the algorithm we tried to lower down the initial movement time.From the result we can see that the time required for the initial moves is less as compare to time we required in serial implementation. So we have gained in terms of time for this initial moves where search space is high.
But as game moves ahead search space decreases. In this case time required for parallel is increases as compare to serial.
Here search space is less and number of resources/process are more as compare to search space. So here we are over utilizing the resources as compare to search space. Since as number of process increases, we have to do special handling to

synchronize their output this adds to total computation time which we are saving in serial.
The result of distributed implementation are not as expected. Here time required for initial moves is more as compare to parallel version. The time required with three machine environment is more as compare to two machine environment. Here search space of the game is relatively small as compare to number of resources(over utilization of the resources).
As number of machines increases network overhead increases, this adds to total computation time.

G. Conclusion

Since size of the search space for this game is not so large. So in case of distributed environment performance decreases since it includes some overhead in splitting the work to different machines (Network overhead) but we can expect good result in case of distributed environment as search space increases.

III. NEGAMAX ALGORITHM [8]

Negamax is a game tree search algorithm for a two player game. To choose the best move for a player , its score is negation of score of the other player.
The root node receives scores from its child nodes , according to which it retains the best score and the best possible move.
Negamax can be improved by using Alpha Beta pruning.

A. Pseudocode

```
function Negamax(CurrentNode ,Alpha , Beta, typeofNode )
    if CurrentNode has no children
      return typeofNode * cost( CurrentNode )

    Set childNodes = getChildrenStates (CurrentNode)
    Set bestScore  -INFINITY
    for all child in childNodes:
       set score = −Negamax(child, −Beta, − Alpha , -1* typeofNode )
          // assuming 2 types of nodes .
       set bestScore   = max( bestScore , score)
       set Alpha         = max( Alpha, score )

          //Cutoff
          if Alpha≥ Beta
                Break

  return bestScore
```

Whenever algorithm encounters a child node value outside an alpha beta range , it cuts off (stops checking the remaining tree) and returns.

B. Parallel Approach

Since all the children provide a result to the parent node, This can be done in parallel.
The top of the tree creates a process for each of its child nodes.
The child nodes run parallel (individually perform Negamax with alpha beta pruning) and return the result back to the root of the tree.

C. Distributed Approach

Every child node of the root is made a Job.
The Jobs(child nodes) are distributed on various machines(nodes) on the cluster created.
The child nodes individually run on a node in a cluster and return the computed results to the central server. The server computes the best value from the received results and takes the best decision.

D. Observations

- Parallel Algorithm:

Speed up for parallel algorithm is shown in table. It can be seen that speedup is greater when there are more nodes to explore(i.e at height 8). The processes utilise the CPU's effectively and instead of serial computation , parallel computation is seen to be advantageous.
When the number of nodes or height of tree is less , the amount of computation involved to create process ,retrieve results etc. is quite significant compared to the computation time for the algorithm(as this is less for less number of nodes).

- Distributed Algorithm:
    - Cluster of size 2

Speed up for Distributed algorithm with cluster of size 2 is shown in table. It can be seen that speedup is greater when there are more nodes to explore(i.e at height 8).
For less number of nodes , the distributed algorithm performs worse than the serial one . This is possibly due to the network delay . The amount of time required to perform computation was less than the network delay.

    - Cluster of size 3:

Speed up for Distributed algorithm with cluster of size 3 is shown in table. It can be seen that speedup is greater when there are more nodes to explore(i.e at height 8).
The same issues occur as with cluster of size 2 due to the network delay. But with introduction of a newer node in the cluster , performance has slightly improved over the size 2 cluster. This is possibly because relatively less computation was done in more nodes and all nodes started sending results to server early.

TABLE III. COMPUTATION TIME FOR NEGAMAX

Height	Serial	Parallel	Distributed (Cluster size 2)	Distributed (Cluster size 3)
	Time	*Time*	*Wall Time*	*Wall Time*

8	0.605	0.096	0.227	0.203
6	0.0584	0.0085	0.133	0.067
4	0.0011	0.00054	0.089	0.119
2	0.00014	0.00013	0.083	0.084

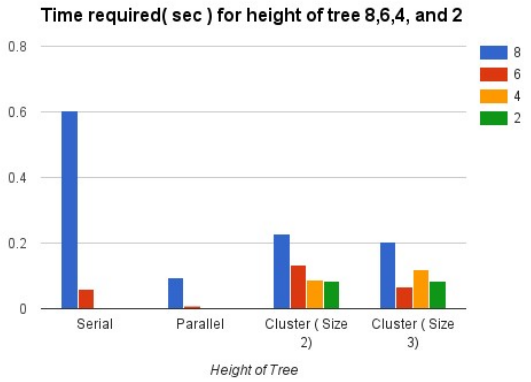

Fig. 3. Time Conputation (NegaMax)

TABLE IV. SPEEDUP FOR NEGAMAX

Height	Parallel	Distributed (2 Cluster)	Distributed (3 cluster)
8	6.302	2.66519	2.9706
6	6.8788	0.4 39	0.8716
4	2.0777	0.01269	0.00924
2	1.1127	0.00178	0.001667

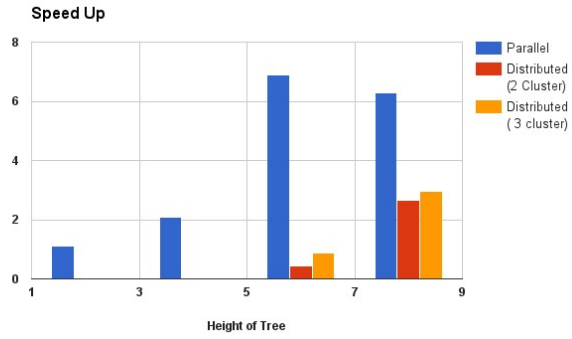

Fig. 4. Speedup (NegaMax)

IV. SCOUT ALGORITHM

The basic idea of scout algorithm was to dominate alpha-beta algorithm.

Scout algorithm searches first son of every node. So obtains lower bound on v. And then it tests that if any of the sibling of first son is able to give better value than v or not. For those siblings only it goes and searches for new v value. Maximum v value is returned to the parent node.

A. Pseudocode[4]

```
procedure TEST(position p, value v, condition >)
        Determine the successor positions p1.....pd
        if d = 0, then // terminal
                if f(p) >v // f is eval function
                        return TRUE
                Else
                        return FALSE
        for i := 1 to d do
                if p is a MAX node and TEST(pi, v,>) is
                TRUE, then
                        return TRUE
                if p is a MIN node and TEST(pi, v,>) is
        FALSE, then
                        return FALSE
        if p is a MAX node, then
                return FALSE
        if p is a MIN node, then
                return TRUE

procedure Scout(p)
        Determine the successor positions p1...pd
        if d = 0, then
                return f(p)
        else
                v = SCOUT(p1)
        for i = 2 to d do
                if p is a MAX node and TEST(pi, v, >) is
                TRUE then
                        v = SCOUT(pi)
                if p is a MIN node and TEST(pi, v, >=) is
                FALSE then
                        v = SCOUT(pi)
        return v
```

Behavior of MAX node is like OR node while MIN node is like AND node.[5]

B. Parallel and Distributed Approach

Pseudo code :

```
procedure Scout(p)
        Determine the successor positions p1...pd
        if d = 0, then
                return f(p)
        else
                v = SCOUT(p1)
```

```
for i = 2 to d do //Parallel/Distributed execution
        if p is a MAX node and TEST(pi, v, >) is
        TRUE then
                v = SCOUT(pi)
        if p is a MIN node and TEST(pi, v, >=) is
        FALSE then
                v = SCOUT(pi)
return v
```

C. Results

TABLE V. TIME COMPUTATION FOR SCOUT ALGORITHM

Height	Serial	Parallel	Cluser size 2	Cluster Size 3
	Time	*Time*	*Wall Time*	*Wall TIme*
8	2.7149	0.418	0.385	0.331
6	0.0495	0.193	0.133	0.123
4	0.002	0.1748	0.1	0.079
2	0.001	0.1747	0.113	0.126

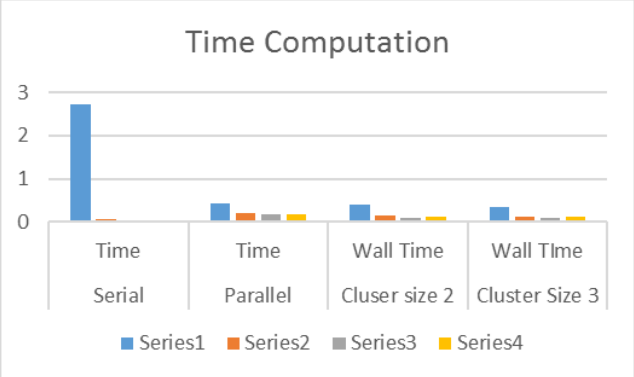

Fig. 5. Time Computation for Scout Algorithm

TABLE VI. SPEEDUP FOR SCOUT ALGORITHM

Height	Parallel	Cluser size 2	Cluster Size 3
8	6.482065513	7.051864847	8.202320139
6	0.256378726	0.372248485	0.402512589
4	0.011473798	0.020060539	0.025393088
2	0.005757275	0.008901664	0.007983238

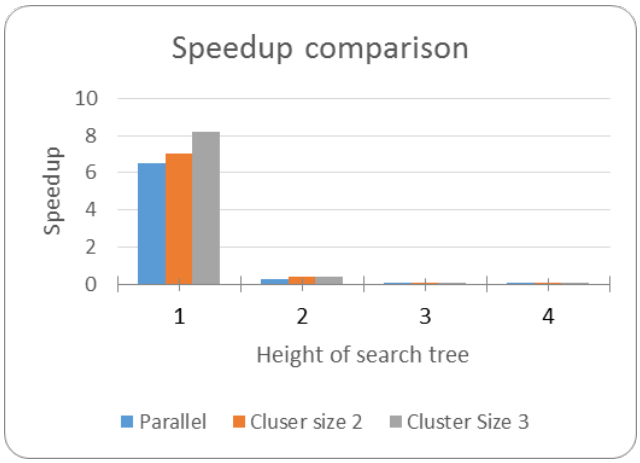

Fig. 6. Speedup for Scout Algorithm

D. Observation

- Parallel Computing

When search space was big parallel was performing better than serial. But when search space was small (height of the tree is less) then serial search was better than parallel one.
The scout algorithm prunes branches so results were varying according to player's moves.

- Distributed Computing

While running distributed one CPU from each cluster is used. For large search space distributed is showing better results than serial and parallel. But for small search space its performance is going down. When cluster size in increased from 2 to 3 then algorithm is performing better for large search space.
The performance is worse than the serial and parallel one because of smaller search space and pruning by the algorithm.

E. Conclusion

The scout algorithm was designed to improve upon alpha-beta pruning. Sometimes number of nodes explored by scout algorithm are less than alpha-beta algorithm. But counterexamples showed that scout cut offs some nodes which are explored by alpha beta. This is the major drawback of the algorithm. So in this strategy some of the times agent is pruning good moves and so human player can win.

REFERENCES

[1] https://en.wikipedia.org/wiki/SSS*
[2] http://www.ijcai.org/Proceedings/89-1/Papers/007.pdf
[3] http://www.cs.otago.ac.nz/cosc411/1980-Pearl-Scout.pdf
[4] http://kti.mff.cuni.cz/~bartak/ui_seminar/talks/2012/HraniHer.pdf
[5] .Dispy - Python Library for Distributed computing http://dispy.sourceforge.net/
[6] Parallel Search Algorithms- https://chessprogramming.wikispa