Abstract—We address the parallelization of a Monte-Carlo search algorithm. On a cluster of 64 cores we obtain a speedup of 56 for the parallelization of Morpion Solitaire. An algorithm that behaves better than a naive one on heterogeneous clusters is also detailed.

Index Terms—Monte-Carlo, Search, Parallelization, Morpion Solitaire

I. Introduction

Monte-Carlo methods can be used to solve problems that have a large state space and no good heuristics. Nested Monte-Carlo search [7] improves Monte-Carlo search using a lower level Monte-Carlo search to choose moves at the upper level. For problems that do not have good heuristics to guide the search, the use of nested levels of Monte-Carlo search amplifies the results of the search and makes it better than a simple Monte-Carlo search. A similar algorithm has already been applied with success to Morpion Solitaire [6]. We add a parallelization of the Nested Monte-Carlo search algorithm on a cluster.

Section 2 describes related work, section 3 explains the sequential Nested Monte-Carlo Search algorithm, section 4 presents two parallel algorithms: the Round-Robin algorithm and the Last-Minute algorithm, section 5 details experimental results.

II. Related works

Parallel algorithms have been developed for many different metaheuristics [1], [19], [2]. Rollout algorithms were first proposed by Tesauro and Galperin for improving a Backgammon program [20]. The idea of a rollout is to improve a heuristic playing games that follow the heuristic and using the result of these games to evaluate moves. Rollouts are applied to different optimization problems [4], [3], [12], [18]. Nested rollouts were found effective for the game of Klondike solitaire [21]. The base level uses a domain-specific heuristic to guide the samples. Nested rollouts have also been found effective for Thoughtful Solitaire, a version of Klondike Solitaire in which the locations of all cards is known. In this case the base level uses heuristics that change with the stage of the game [5].

A strategy that has been found effective for the game of Morpion Solitaire is in which the locations of all cards is known. In this case the base level uses heuristics that change with the stage of the game [5].

Finally, the paper describes two parallel algorithms for the Traveling Salesman Problem (TSP) and the Sequential Ordering Problem (SOP). With their speculative strategy they obtained a modest speedup of 3.89 to 6.64 on 8 processors depending on the size of the neighborhoods for SOP.

In this paper we propose the parallelization of a nested rollout algorithm on a cluster.

III. The sequential algorithm

The basic sample function just plays a random game from a given position:

\[
\text{circle} \quad \text{can} \quad \text{be} \quad \text{drawn.} \quad \text{Line} \quad \text{can} \quad \text{be} \quad \text{horizontal,} \quad \text{vertical} \quad \text{or} \quad \text{diagonal.} \quad \text{The} \quad \text{starting position already contains circles disposed as in figure 1. In the disjoint version a circle cannot be a part of two lines that have the same direction. The best human score at Morpion Solitaire disjoint version is 68 moves [11]. The previous best computer score was 79 obtained with Simulated Annealing [16]. A reflexive Monte-Carlo algorithm was shown to be effective for Morpion Solitaire [6].}
\]
int sample (position)
1 while not end of game
2 play random move
3 return score

The score is the score of the game in the terminal position. For example at Morpion Solitaire the goal is to play as many moves as possible, so the score is the number of moves played in the game. In other games where the algorithm is of interest, the score can be computed completely differently. The idea of the score function is that the algorithm tries to find the sequence of moves that maximizes it.

The nested rollout function plays a game, choosing at each step of the game the move that has the highest score of the lower level nested rollout. A level 1 rollout uses the sample function to choose its moves. The \texttt{argmax} command sends back the move that returns the best score of a lower level search, over all possible moves:

\begin{verbatim}
int nested (position, level)
1 best score = -1
2 while not end of game
3 if level is 1
4 move = argmax_m (sample (play (position, m)))
5 else
6 move = argmax_m (nested (play (position, m), level - 1))
7 if score of move > best score
8 best score = score of move
9 best sequence = seq. after move
10 bestMove = move of best sequence
11 position = play (position, bestMove)
12 return score
\end{verbatim}

\section{Parallel Algorithms}

In order to parallelize nested rollouts we define four types of processes: the root process, the median process, the dispatcher process and the client processes. These processes work at different levels of nesting: the root process at the first level (the highest level of nesting), the median process at the second level and the client processes at the third level. The root process plays a game at the first level and calls the median processes to play games at the second level. The median processes ask the client processes to play games at the third level in parallel. The dispatcher process is used to tell median nodes which clients to use.

\subsection{The Round-Robin algorithm}

Figure 2 details the Round-Robin algorithm. There are four possible communications. The first one (fig. 2(a)) consists in a message from the root node to a median node that asks the median node to perform a nested search at the lowest level.

During the second communication (fig. 2(b)), the median node asks the dispatcher which clients it should use; one of them has a client that asks the client to perform a nested search at the lowest level.

The third communication (fig. 2(c)) occurs when a client of a median node has finished its search. It sends back the result to the median node that has asked for this search. Once the median node has all its results, it can choose the best move, play it and continue its game going back to the second communication. When the game is over, it uses the
fourth communication (fig. 2(d)) which consists in sending the result of the game to the root node.

Fig. 3. Parallel communications that can occur during the Round-Robin algorithm

Figure 3 shows that the second, third and fourth communications can occur in parallel.

The root process plays a game until the end at the highest level. The number of median nodes is greater than the number of possible moves. Median nodes are mainly used to dispatch the computation at a lower level on the child nodes. They are not used for long computation. Starting from the current position of the game, the root process tries each possible move and sends the resulting position to a different median node. Then it waits for all answers from the median nodes previously chosen. Once it has all answers, it can choose the move that has the highest score, play it and loop until the end of the game.

The pseudo code for the root process is:

1. while not end of game
2. node = first median node
3. for m in all possible moves
4. p = play (position, m)
5. send p to node
6. node = next median node
7. for m in all possible moves
8. receive score from node
9. position = play (position, move with best score)
10. return score

The Round-Robin dispatcher consists in choosing the next client in the list of clients for each request. It receives messages from median nodes that ask it on which client to send their computation. It simply sends back clients one after another, always in the same order.

The code for the Round-Robin dispatcher is:

1. client = first client
2. while true
3. receive position from root process
4. while not end of game
5. m = play (position, m)
6. send self id and number of moves played in m to dispatcher
7. receive client from dispatcher
8. send p to client
9. for m in all possible moves
10. receive score from client
11. position = play (position, move with best score)
12. send score to root

A client process waits for a position from a median node. When it receives the position, it plays a nested rollout at a predefined level, and sends back the resulting position to the median node. In the case of the Last-Minute algorithm, it also sends back the score of the game to the root process.

The code for a median process is:

1. while true
2. receive position from root process
3. while not end of game
4. for m in all possible moves
5. p = play (position, m)
6. send self id and number of moves played in p to dispatcher
7. receive client from dispatcher
8. send p to client
9. for m in all possible moves
10. receive score from client
11. position = play (position, move with best score)
12. send score to root

A client process waits for a position from a median node. When it receives the position, it sends back the result of a nested rollout at a predefined level, and sends back the score of the game to the median node.
or if no available jobs it adds it to the list of free clients. The other communications are similar to the Round-Robin algorithm.

Figure 5(e') shows that communications can occur in parallel.

The Last-Minute dispatcher waits for a node from any client. If it is a client node it means that this client node is waiting for a new job, thus it sends the job with the smallest number of moves to this client. If it is a no pending job, it simply adds the client to the list of free clients. If it is a median node, it also sends the number of moves of the position to analyze. If it is a no free clients, it adds the job to the list of pending jobs. If it is a median client it sends the client to the median node and the moves to the client from the list of free clients.

The pseudo-code for the Last-Minute dispatcher:

1. listFreeClients = all Clients
2. jobs = empty list
3. while true
4. receive node from any node
5. if node is a client node
6. add node to listFreeClients
7. if jobs is not empty
8. find j in jobs with the smallest number of moves
9. send j.sender to the node
10. remove j from jobs
11. remove node from listFreeClients
12. else if node is a median node
13. receive number of moves from node
14. if listFreeClients is empty
15. add {node, number of moves} to jobs
16. else
17. client = first element of listFreeClients
18. send client to node
19. remove client from listFreeClients

V. EXPERIMENTAL RESULTS

Our cluster is composed of 20 1.86 GHz dual core PCs, 12 2.33 GHz dual core PCs and one quad core server connected with a Gigabit network. We used message passing with Open MPI [13] as parallel programming model. Open MPI is designed to achieve high performance computing on heterogeneous clusters. All communications are done with the global communicator MPI_COMM_WORLD. Each node runs two client processes. Processes are created at the beginning of the execution, via the use of the master-slave model. The server runs the root process as well as all the median processes and the dispatcher. Most of the computation is performed by the clients. We run the 40 median processes on the dual core PCs.

All experiments use Morpion Solitaire disjoint model. All the times results as a mean over multiple runs of each algorithm except for results in parenthesis which were run only once. The standard deviation is given in parenthesis to the times. The algorithms were tested on playing only the first move of a game, and on playing an entire game. All experiments consist in a single run of the algorithms at level 3 and 4 of nesting. Each rollout is a game that is slightly different from others in terms of random game's insides. Each rollout can have different lengths. Times taken by two rollouts can be different. Standard deviation shows these times variations.

The results for the sequential algorithm are given in Table I. We can observe that level 4 takes approximately 207 times more time than level 3. One rollout takes approximately 9 times more time than the first move.

<table>
<thead>
<tr>
<th>Level</th>
<th>First Move</th>
<th>One Rollout</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>08m03s (19s)</td>
<td>1h07m33s (42s)</td>
</tr>
<tr>
<td>4</td>
<td>28h00m06s (58m55s)</td>
<td>09h18m58s (428)</td>
</tr>
</tbody>
</table>

TABLE I

Times for the sequential algorithm
The speedup for level 4 is 28.50 for 32 clients. Table III gives the times for playing a rollout with the Round-Robin algorithm at level 3 and 4. The speedup of the algorithm for 64 clients is 44.

Table IV gives the results of the Last-Minute algorithm for the first move at levels 3 and 4. Results are similar to the Round-Robin algorithm at level 3. The speedup for level 4 is 30 which is a little higher than the Round-Robin algorithm.

Table V gives the results of the Last-Minute algorithm for rollouts at levels 3 and 4. Results are slightly better than the Round-Robin algorithm.

Table VI compares the two algorithms with an heterogeneous repartition (where 16x4+16x2 means that there are 16 PCs with 4 clients and 16 PCs with 2 clients and where 8x4+8x2 means there are 8 PCs with 4 clients and 8 PCs with 2 clients). At b-w-1 4 the Last-Minute algorithm (LM) has b-t-r s-suits than the Round-Robin algorithm (RR). The Last-Minute algorithm gives b-t-r s-suits than the Round-Robin algorithm in an b-s-rog: a-ous-environm-nt. The spa-dup of the Last-Minute algorithm for rollouts of b-w-1 4 with 64 clients is approxima-bly 56 which is qui-gd.

Running the algorithm at b-w-1 4 on our cluster, we have discov-er-d two b-w s-qu-a-s of 80 moves which is b-i-current world s-cord.

VI. Conclusion

We have ps-a-n-d two algorithms that par-al-l-i-ns-Mons-Carlo Search on a class-s-r. The spa-dup for 64 clients is approxima-bly 56 for Morpion Solitais- which is a prob-m with a large s-t-a-s and no good known b-uris-tic. The Last-Minute algorithm is mos-adap-ted to b-s-rog: a-ous class-ss. The par-al-l-i-ns algorithm run at b-w-1 4 has found s-qu-a-s of length 80 which is the b-i-current world s-cord at Morpion Solitais- disjoint v-ss (ss- for e-xamp-le figu-re 1).

REFERENCES

<table>
<thead>
<tr>
<th>clients</th>
<th>b-w-1 3</th>
<th>b-w-1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>01m52s</td>
<td>5h09m16s</td>
</tr>
<tr>
<td>32</td>
<td>03m08s</td>
<td>(6h31m)</td>
</tr>
<tr>
<td>16</td>
<td>05m22s</td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>10m18s</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>21m41s</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>1h26m28s</td>
<td>—</td>
</tr>
</tbody>
</table>

TABLE III
Rollout times for the Round-Robin algorithm

<table>
<thead>
<tr>
<th>clients</th>
<th>b-w-1 3</th>
<th>b-w-1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>09s</td>
<td>27m20s</td>
</tr>
<tr>
<td>32</td>
<td>19s</td>
<td>59m44s</td>
</tr>
<tr>
<td>16</td>
<td>37s</td>
<td>(2h05m17s)</td>
</tr>
<tr>
<td>8</td>
<td>01m11s</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>02m23s</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>09m30s</td>
<td>(33h06m57s)</td>
</tr>
</tbody>
</table>

TABLE IV
First move times for the Last-Minute algorithm

<table>
<thead>
<tr>
<th>clients</th>
<th>b-w-1 3</th>
<th>b-w-1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>01m32s</td>
<td>4h10m09s</td>
</tr>
<tr>
<td>32</td>
<td>02m43s</td>
<td>6h58m21s</td>
</tr>
<tr>
<td>16</td>
<td>05m35s</td>
<td>(52m42s)</td>
</tr>
<tr>
<td>8</td>
<td>11m33s</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>19m51s</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>1h31m40s</td>
<td>—</td>
</tr>
</tbody>
</table>

TABLE V
Rollout times for the Last-Minute algorithm

<table>
<thead>
<tr>
<th>clients</th>
<th>alg</th>
<th>b-w-1 3</th>
<th>b-w-1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x4+16x2</td>
<td>LM</td>
<td>14s</td>
<td>28m37s</td>
</tr>
<tr>
<td>16x4+16x2</td>
<td>RR</td>
<td>16s</td>
<td>45m17s</td>
</tr>
<tr>
<td>8x4+8x2</td>
<td>LM</td>
<td>18s</td>
<td>58m21s</td>
</tr>
<tr>
<td>8x4+8x2</td>
<td>RR</td>
<td>25s</td>
<td>1h24m11s</td>
</tr>
</tbody>
</table>

TABLE VI
First move times on an heterogeneous cluster

th- singl- cl-nt fis-quency we- obtain r = ((20 × 1.86 + 12 × 2.33)/32)/1.86 = 1.09. So the spa-dup should rather be closed to 56/1.09 = 51. The s-suit for 32 cl-nts is obtained using only 1.86 GHz PCs, and this time the spa-dup is 29.8.

Concerning level 4 the speedup is 28.50 for 32 clients.

Table III gives the times for playing a rollout with the Round-Robin algorithm at b-w-1 3 and 4. The spa-dup of the algorithm for 64 cl-nts is 44.

Table IV gives the b-s-suits of the Last-Minute algorithm for b-w-1 3 and 4. The b-s-suits are similar to the Round-Robin algorithm at b-w-1 4.

The spa-dup of the Last-Minute algorithm for rollouts at b-w-1 3 and 4. The b-s-suits are slightly better than the Round-Robin algorithm.

Table V gives the b-s-suits of the Last-Minute algorithm for rollouts at b-w-1 4 which is a little higher than the Round-Robin algorithm.

Table VI compasses two algorithms with an b-s-rog: a-ous s-partition (wbs-16x4+16x2 ma-ans that the s-s is 16 PCs with 4 cl-nts and 16 PCs with 2 cl-nts and wbs-8x4+8x2 ma-ans that s-s is 8 PCs with 4 cl-nts and 8 PCs

<table>
<thead>
<tr>
<th>clients</th>
<th>b-w-1 3</th>
<th>b-w-1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>01m52s</td>
<td>5h09m16s</td>
</tr>
<tr>
<td>32</td>
<td>03m08s</td>
<td>(6h31m)</td>
</tr>
<tr>
<td>16</td>
<td>05m22s</td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>10m18s</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>21m41s</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>1h26m28s</td>
<td>—</td>
</tr>
</tbody>
</table>


