Noname manuscript No.
(will be inserted by the editor)

A Parallel General Game Player

Jean Méhat · Tristan Cazenave

Received: date / Accepted: date

Abstract We have parallelized our general game player Ary on a cluster of computers. We propose multiple parallelization algorithms. For the sake of simplicity all our algorithms have processes that run independently and that join their results at the end of the thinking time in order to choose a move. Parallelization works very well for checkers, quite well for other two player sequential move games and not at all for a few other games.

1 Introduction

In this paper we evaluate the interest of parallelization for General Game Playing (GGP). Monte-Carlo Tree Search (MCTS) is currently the algorithm that gives the best results for GGP: using MCTS, our program Ary won the 2009 GGP competition. It has been established in games such as Go or Hex that a parallel implementation of MCTS gives good results [2,3,12,10, 8]. In this paper we investigate simple algorithms that parallelize MCTS for the games of the GGP competitions.

Section 2 explains the basics of Monte-Carlo Tree Search. Section 3 presents General Game Playing. Section 4 describes various parallel Monte-Carlo Tree Search algorithms. Section 5 details experimental results. Section 6 concludes.

Jean Méhat
LIASD, Université Paris 8, 2 rue de la Liberté, 93526 Saint-Denis Cedex, France
E-mail: jm@ai.univ-paris8.fr

T. Cazenave
LAMSADE, Université Paris-Dauphine, Place du Maréchal du Lattre de Tassigny, 75775 Paris Cedex 16, France
E-mail: cazenave@lamsade.dauphine.fr

2 Monte-Carlo Tree Search

Monte-Carlo tree search algorithms play random games (i.e. playouts) so as to select moves that are good on average. The most popular Monte-Carlo tree search algorithm is UCT [13]. It consists in building a move tree that is used to choose the first moves of playouts. The move tree is constructed incrementally, with a new node added for each random game. On the next exploration, a path is chosen in the already built move tree by choosing the branch whose estimated gain is maximum. This gain is estimated by the mean of the previous playouts plus confidence in the estimation. The confidence is calculated by a function of the number of explorations of the node t and of the number of exploration of the branch s as $\sqrt{log(t)/s}$. When arriving at a leaf node of the move tree, if it is not a terminal situation, a new node is added to the tree and a Monte-Carlo simulation is started to obtain an evaluation of this node and update the evaluation of its parent nodes.

UCT and its refinements are applied with success to Monte-Carlo Go [7,11] and to many other games [9,1, 4].

When there are some unexplored moves, UCT will choose to explore them. When all the branches from a node have been explored, UCT will tend to re-explore the most promising ones: this tendency is controlled by a constant C, that is used to multiply the confidence upper bound $\sqrt{log(t)/s}$. The higher the constant C, the more UCT will explore unpromising nodes. At the limit, when the whole game tree has been explored, UCT is favoring the better branches and will converge to the same choice as a Minimax exploration.

3 General Game Playing

3.1 Previous work

Lot of previous work on computers and games was to design algorithms tailored to a specific game. However a possible criticism of these works is that a specialized program lacks of general intelligence. Hence the field of General Game Playing has emerged with the goal of having computer programs play a large variety of games. Early work on General Game Playing can be traced back to Jacques Pitrat [16]. It was followed by the work of Barney Pell [15]. The General Game Playing community has been growing since the establishment of an annual competition since 2005 and the standardization of the Game Description Language [14].

3.2 Communication between the Game Master and the players

The communication between the Game Server and the players in the GGP competition takes the form of a client-servers interaction, with the players acting as servers and the Game Master acting as a client.

Every interaction starts with a new network connection established at the request of the Game Master with a player, through which it sends a request or notification: description of the game, timing constraints and role of the player, notification of the moves played by all players in the preceding step, notification of the end of the game. The message contents is prepended with a header.

The players have to answer on the same network connection in a delay specified by the initial message.

3.3 Ary

In this subsection we present Ary, the program used for the experiments.

Ary is written in C. It uses Monte-Carlo Tree Search (MCTS) and a Prolog interpreter as an inference engine. Ary won the 2009 GGP competition.

Its architecture is somewhat similar to the one of CadiaPlayer [9,1] since it also uses UCT and Prolog. It is different from the winners of the 2005 and 2006 competitions, ClunePlayer and FluxPlayer [6,17] that used evaluation functions.

4 Parallel Monte-Carlo Tree Search

Cazenave and Jouandeau have proposed three algorithms to parallelize UCT [2]. The most simple one is the single run algorithm renamed as the root parallel algorithm in [5]. Other algorithms that share the UCT tree and that perform playouts in parallel have been proposed either on a cluster [3,12,10] or on a multithreaded computer [8].

In this paper we focus on the most simple of these algorithms, namely the root parallel algorithm. The principle of the root parallel algorithm is to perform independent Monte-Carlo tree searches in parallel on different CPU. When the thinking time is elapsed the results of the different searches are joined in order to choose the move.

For each move at the root of a UCT tree, we have a mean and a number of playouts. Each CPU has its own UCT tree. Let $\mu_{i,m}$ be the mean of move m on CPU number i, and $p_{i,m}$ be the number of playouts associated to move m on CPU number i.

The evaluation of a move using the root parallel algorithm and n CPUs is: $evaluation_m = \frac{\sum_{i=1}^{n} \mu_{i,m} \times p_{i,m}}{\sum_{i=1}^{n} p_{i,m}}$

The root parallel algorithm plays the move with the greatest $evaluation_m$.

4.1 Structure of the root parallelisation

In this section we present the structure of the program when doing root parallelization.

The player is built with two components: subplayers that actually perform the search and a multiplexer that ensures coordination between the subplayers, as illustrated on figure figure 1.

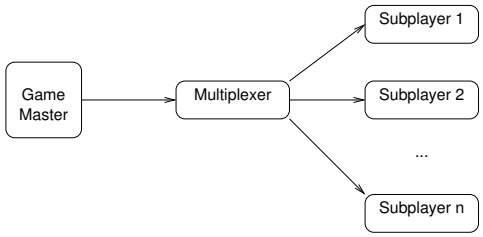

Fig. 1 The interactions between the Game Master the multiplexer and the subplayers

The multiplexer is the contact point for the game master. When it receives a message from the game master, it dispatches it to every subplayer. The message transmitted to the subplayers is an exact copy of the one received from the game master, except that the initial clock and play clock can be decreased to account for networks delays introduced by the multiplexer.

Subplayers are not aware of the presence of the multiplexer and respond to its request as they would do to a request from the Game Master. We only modified the

player to let it add an evaluation of the moves it explored in new fields of the header of its answer. The current Dresden Game Master accepts these supplementary headers fields without errors, so a single subplayer can be used to interact directly with the Game Master, bypassing the multiplexer.

After all the players responded or the thinking time is nearly exhausted, the multiplexer combines the evaluation of moves explored received from the subplayers, selects accordingly one of these moves and transmit it to the Game Master.

The principal advantage of this scheme is its simplicity: subplayers are nearly exact copies of the usual player; the interaction between the processus takes place through standard network connections, not requiring a middleware; the only place where races are possible is in the multiplexer between the threads used to communicate with each subplayers. Its principal disadvantage is that during the search phase of one step, the search in one subplayer has no influence on the other subplayers.

4.2 Different ways to combine sub-players evaluations

There are different ways to enrich the explorations of the moves in one subplayer with the information acquired in the other subplayers [2,10]. Given the model of interactions between the subplayers and the multiplexer, we can only use the combination in the multiplexer of the evaluation of moves in the subplayers. We explored four different combination strategies: *Best, Sum, Sum10 and Raw*.

As a player does, a subplayer selects a move and transmits it in his answer to the Multiplexer or the Game Master. Moreover, it adds in the header its evaluation of this selected move. In one-player games, this evaluation is the best reward obtained in a playout starting with this move; in multiplayer games, this evaluation is the mean of the rewards obtained in playout starting with this move, i.e. the left part of the UCT formula used to select a node to explore, excluding the confidence part. In multiplayer games, the subplayer transmit also this evaluation and the number of explorations for all the moves available to the player.

The *Best* strategy consists simply in selecting in the multiplexer the move with the best evaluation found by one subplayer. In MCTS explorations for one player games, each subplayer stores the playout with the best result, and uses it through steps to ensure that the reward obtained can only grow. This strategy works well for one player games, as it is the only one that ensures the preservation of the best path found, and we tried to observe how it performs in multiplayer games.

The *Sum* strategy uses the evaluation of all the explored moves transmitted by the subplayers: for all the subplayers s, the evaluation e_s and the number of playouts p_s are used to compute a new evaluation with $\sum e_s \times p_s / \sum p_s$. *Sum* combines the top of the UCT trees built into the subplayers.

The *Sum10* strategy is the same as the *Sum* strategy, but uses only the ten best evaluated moves of each subplayer. For the games like *Connect Four* where a player has less than ten moves available, it amounts to the same strategy as *Sum* In games with more than ten moves, we expect that the refutation of the false good move discovered in one subplayer has no effect on the evaluation of this move, as this move will not appear in the best ten of the subplayer that has discovered the refutation.

On the contrary, the *Raw* strategy combines the evaluations of the subplayers without considering the number of explorations in the subplayers, simply as $\sum e_s / n$ where n is the number of subplayers having evaluated this move. Once a refutation of a false good move is found, UCT will tend to avoid exploration of the subtree starting with this false good move, while it will tend to explore it in the subplayers that have not found the refutation: ponderating the evaluation of a subplayer by its number of explorations could lead to a bias favorizing the false good move in the *Sum* strategy. Ignoring the number of explorations, as done in *Raw*, appears as a simple way to avoid this bias.

5 Experimental Results

In this section we have run the parallel algorithms on various games and with 1 to 16 subplayers. Each parallel algorithm played between 91 and 230 games against the sequential algorithm which used the same time (10 seconds per move).

The results for Best are given in table 1. We can observe moderate improvements for games such as breakthrouh, checkers and othello. The improvements are either small or inexistent for the other games. Blocker and skirmish are simultaneous moves games.

Table 1 Results for Best

Best with parallel as first player						
games	1	2	4	8	16	#experiments
blocker	35	46	42	42	35	/100-100
breakthrough	44	55	58	56	58	/100-100
checkers	44	58	70	66	72	/99-100
connect4	60	67	69	75	65	/100-100
othello	39	54	55	61	58	/100-100
pawn_whopping	60	65	65	70	69	/100-100
pentago	54	66	68	64	62	/100-100
skirmish	75	78	74	76	77	/99-100

The results for Sum are given in 2. The results are better than with the Best algorithm for sequential moves games. For the two simultaneous moves games the parallelization does not work. The results are especially good at checkers, othello and pentago. However at breakthrough it appears useless to have more than two subplayers. A reason why parallelization does not work well at breakthrough is that the game is not well suited for Monte-Carlo Tree Search and UCT: UCT frequently does not find the refutation of moves that appear good but are not.

Table 2 Results for Sum

Sum with parallel as second player						
games	1	2	4	8	16	#experiments
blocker	63	61	57	71	67	/93-100
breakthrough	44	65	60	67	65	/97-100
checkers	47	65	83	86	94	/96-99
connect4	28	44	63	66	75	/100-100
othello	59	60	72	84	83	/99-100
pawn_whopping	44	45	43	46	35	/94-100
pentago	35	54	68	64	68	/98-100
skirmish	71	71	74	76	71	/97-100

Table 3 gives the results for the Sum10 algorithm. The results are slightly better but the small margin is not significant enough to draw a firm conclusion.

Table 3 Results for Sum10 as second player

Sum10 with parallel as second player						
games	1	2	4	8	16	#experiments
blocker	69	59	66	63	73	/100-100
breakthrough	55	54	65	73	72	/97-100
checkers	54	66	88	91	95	/96-100
connect4	28	44	63	66	75	/100-100
othello	45	54	72	77	79	/100-100
pawn_whopping	34	43	53	33	46	/100-100
pentago	30	42	35	45	59	/100-100
skirmish	72	74	74	76	74	/100-100

Table 4 gives the results for the same games and the same algorithm but this time as the first player in all the games. Again the parallelization works well at checkers, connect4, othello and pentago but does not work for blocker, breakthrough, pawn_whopping and skirmish. This table show that the parallelization works in a similar way whether the parallel algorithm is the first player or the second player.

Table 4 Results for Sum10

Sum10 with parallel as first player						
games	1	2	4	8	16	#experiments
blocker	36	35	40	38	36	/167-172
breakthrough	55	56	56	61	61	/98-100
checkers	50	67	80	89	92	/87-100
connect4	60	71	75	79	86	/100-158
othello	48	50	67	66	76	/150-151
pawn_whopping	55	69	77	75	81	/154-230
pentago	60	53	66	65	78	/150-221
skirmish	77	79	79	78	76	/150-150

Table 5 gives the results for the Raw algorithm. The parallelization is worse for all the games than with the Sum algorithm. For some games such as connect4 Raw does not improve on a single subplayer. Even at breakthrough were refutations are difficult to find for UCT, the Raw algorithm is worse than the Sum algorithm.

Table 5 Results for Raw as second player

Raw, with parallel as second player						
games	1	2	4	8	16	#experiments
blocker	61	62	60	61	63	/109-110
breakthrough	51	41	52	57	58	/97-99
checkers	50	59	64	73	84	/94-99
connect4	44	41	42	40	44	/91-100
othello	47	53	62	66	79	/96-100
pawn_whopping	44	45	43	46	35	/94-100
pentago	45	37	42	50	59	/92-100
skirmish	75	75	74	75	72	/97-100

We performed similar experiments with the sequential algorithm. In table 6 the sequential algorithm plays against the same sequential algorithm with twice more time, four times more times, ... until 16 times more time.

Comparing table 6 with table 4 we can observe the influence of having a single sequential UCT tree versus having parallel separate trees in the subplayers. We can see that for breakthrough having more time for the sequential version of UCT improves the results a lot while the parallelization does not improve as much.

Concerning blocker, even with the sequential algorithm there is no improvement with additional time. Results at skirmish are slightly improved with the sequential algorithm while there is no improvement with the parallel algorithm. Similarly, for pawn_whopping the parallelization does not work when the sequential algorithm benefits from additional time.

Concerning checkers, the Sum algorithm works almost as well as the sequential algorithm. On the other games (connect4, othello and pentago) the results are worse than the sequential algorithm but are still quite beneficial.

Table 6 Results for the sequential algorithm

Sequential game with asymetrical time as a second player						
game	t	$2t$	$4t$	$8t$	$16t$	
blocker	51	68	71	55	54	/80-96
breakthrough	58	72	81	88	95	/63-85
checkers	46	79	94	93	100	/58-81
connect4	43	56	65	80	89	/74-89
othello	50	70	84	91	93	/72-82
pawn_whopping	39	66	74	74	79	/79-90
pentago	35	62	80	93	91	/83-99
skirmish	71	82	86	83	86	/79-90

6 Conclusion

We have presented four simple parallel algorithms for MCTS: Best, Sum, Sum10 and Raw. These algorithms wer compared for various games of the GGP competitions. They were also compared to the sequential algorithm that uses as much time as the sum of the times of all the subplayers. Results are very good for checkers that parallelize well. They are quite good for connect4, othello and pentago yelding substantial improvements even if the behaviour is not as good as giving more time to the sequential algorithm. Parallelization does not help for breakthrough, blocker, skirmish and pawn_whopping. Overall parallelization is worthy since it improves much the results in half of the games we have tested. The best algorithms are Sum and Sum10 and their results are very close.

References

1. Björnsson, Y., Finnsson, H.: Cadiaplayer: A simulation-based general game player. IEEE Transactions on Computational Intelligence and AI in Games **1**(1), 4–15 (2009)
2. Cazenave, T., Jouandeau, N.: On the parallelization of UCT. In: Computer Games Workshop 2007, pp. 93–101. Amsterdam, The Netherlands (2007)
3. Cazenave, T., Jouandeau, N.: A parallel monte-carlo tree search algorithm. In: Computers and Games, *Lecture Notes in Computer Science*, vol. 5131, pp. 72–80. Springer (2008)
4. Cazenave, T., Saffidine, A.: Utilisation de la recherche arborescente monte-carlo au hex. Revue d'Intelligence Artificielle **23**(2-3), 183–202 (2009)
5. Chaslot, G., Winands, M.H.M., van den Herik, H.J.: Parallel monte-carlo tree search. In: Computers and Games, *Lecture Notes in Computer Science*, vol. 5131, pp. 60–71. Springer (2008)
6. Clune, J.: Heuristic evaluation functions for general game playing. In: AAAI, pp. 1134–1139 (2007)
7. Coulom, R.: Efficient selectivity and back-up operators in monte-carlo tree search. In: Computers and Games 2006, Volume 4630 of LNCS, pp. 72–83. Springer, Torino, Italy (2006)
8. Enzenberger, M., 0003, M.M.: A lock-free multithreaded monte-carlo tree search algorithm. In: ACG, *Lecture Notes in Computer Science*, vol. 6048, pp. 14–20. Springer (2009)
9. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: AAAI, pp. 259–264 (2008)
10. Gelly, S., Hoock, J.B., Rimmel, A., Teytaud, O., Kalemkarian, Y.: The parallelization of monte-carlo planning - parallelization of mc-planning. In: ICINCO-ICSO, pp. 244–249 (2008)
11. Gelly, S., Silver, D.: Achieving master level play in 9 x 9 computer go. In: AAAI, pp. 1537–1540 (2008)
12. Kato, H., Takeuchi, I.: Parallel monte-carlo tree search with simulation servers. In: 13th Game Programming Workshop (GPW-08) (2008). URL `http://www.gggo.jp/publications/gpw08-private.pdf`
13. Kocsis, L., Szepesvàri, C.: Bandit based monte-carlo planning. In: ECML, *Lecture Notes in Computer Science*, vol. 4212, pp. 282–293. Springer (2006)
14. Love, N., Hinrichs, T., Genesereth, M.: General game playing: Game description language specification. Tech. rep., Stanford University (2006)
15. Pell, B.: A strategic metagame player for general chess-like games. In: AAAI, pp. 1378–1385 (1994)
16. Pitrat, J.: Realization of a general game-playing program. In: IFIP Congress (2), pp. 1570–1574 (1968)
17. Schiffel, S., Thielscher, M.: Fluxplayer: A successful general game player. In: AAAI, pp. 1191–1196 (2007)

Jean Méhat is 51 years old. He completed his PhD on the control and programming of a massively parallel computer in 1989. He is associate professor at the university of Paris 8 and currently doing his reseach in the domain of General Game Playing.

Tristan Cazenave is professor of computer science at LAMSADE, Universite Paris-Dauphine. He holds a PhD from Universite Paris 6. His interests are in search algorithms and computer games. He has written programs for multiple board games and has authored more than one hundred scientific papers on artificial intelligence in games.