Randomized Parallel Proof-Number Search[1]

Jahn-Takeshi Saito[a] Mark H.M. Winands [a] H. Jaap van den Herik [b]

[a] *Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University*
[b] *Tilburg centre for Creative Computing (TiCC), Faculty of Humanities, Tilburg University*

Most computer programs for board games successfully employ $\alpha\beta$ search. For some games, however, $\alpha\beta$ search displays a weakness in the endgame that can currently neither be overcome by endgame databases nor by other $\alpha\beta$ extensions. To remedy the deficit, mate searches may be applied. One such alternative to $\alpha\beta$ search is Proof-Number Search (PNS). PNS enjoys popularity as a powerful method for solving endgame positions and complete games. Since its introduction by Allis *et al.* [2] in 1994, PNS has developed into a whole family of search algorithms (e.g., PN^2 [1], PDS [6], and df-pn [7]) with applications to many games, such as Shogi [9], the one-eye problem in Go [5], Checkers [8], and Lines of Action [11].

A variety of parallel $\alpha\beta$ algorithms have been proposed in the past [3], but so far not much research has been conducted on parallelizing PNS. With multi-core processors becoming established as standard equipment, parallelizing PNS has become an important topic. Pioneering research has been conducted by Kishimoto [4], who parallelized the depth-first PNS variant PDS. His algorithm is called ParaPDS and is designed for distributed memory systems. In this abstract we address the problem of parallelizing PNS and PN^2 for shared memory systems.

We solved this problem by introducing a new parallel Proof-Number Search algorithm for shared memory systems, called Randomized Parallel Proof-Number Search (RP–PNS). It adheres to the principle of *randomized parallelization* [10]. The method relies on a heuristic which may seem counterintuitive at first. Instead of selecting the child with the best heuristic evaluation, a probability distribution of the children determines which node is selected. This is called the *randomization* of the move selection. The specific probability distribution is based on the selection heuristic. The parallelization is now achieved by threads that select moves close to the principal variation based on the probability distribution. The same principle can be applied to the two-level variant of PNS, also known as PN^2. The resulting algorithm is then called RP–PN^2.

We evaluated RP–PNS and RP–PN^{2}on a test set consisting of 143 complex Lines-of-Action endgame positions. Two series of experiments were conducted. The first series tested the efficiency of RP–PNS; the second tested the efficiency of RP–PN^2.

In the first series of experiments we tested the performance of RP–PNS for solving the positions. The results regarding time, nodes evaluated, and nodes in memory for 1, 2, 4, and 8 threads are given in the upper part of Table 1. We observe that the scaling factor for 2, 4, and 8 threads is 1.6, 2.5, and 3.5, respectively. Based on the results we computed that the search overhead expressed by the number of nodes evaluated is only ca. 33% for 8 threads. It means that the synchronization overhead is responsible for the largest part of the total overhead. Finally, we see that RP–PNS$_8$ uses 50% more memory than PNS.

In the second series of experiments we tested the performance of RP–PN^2. The results regarding time, nodes evaluated, and nodes in memory for 1, 2, 4, and 8 threads are given in the lower part of Table 1. We observe that the scaling factor for 2, 4, and 8 threads is 1.9, 3.4, and 4.7, respectively. Compared to RP–PNS the relative scaling factor of RP–PN^2 is better for all configurations. The search overhead of RP–PN$^2_8$ is 27% which is comparable to the search overhead of RP–PNS$_8$. At the same time the total overhead of RP–PN$^2_8$ is smaller. This means that the synchronization overhead is smaller for RP–PN$^2_8$ than for RP–PNS$_8$. The reason is that more time is spent in the second-level trees. Therefore, the probability that two threads simultaneously try to lock the same node of the first-level tree is reduced. Finally, we

[1] The full version of this paper will be published in: *Advances in Computer Games (ACG 2009)*, Lecture Notes in Computer Science, Springer, Heidelberg, Germany.

Table 1: Experimental results for RP–PNS and RP–PN^2.

	PNS	RP–PNS$_2$	RP–PNS$_4$	RP–PNS$_8$
Total Time (sec.)	1,679	1,072	682	478
Total scaling factor	1	1.6	2.5	3.5
Total nodes evaluated (million)	612	673	745	815
Total nodes in memory (million)	367	423	494	550

	PN^2	RP–PN$_2^2$	RP–PN$_4^2$	RP–PN$_8^2$
Total Time (sec.)	6,735	3,275	1,966	1,419
Total scaling factor PN^2	1	1.9	3.4	4.7
Total scaling factor compared to PNS	0.25	0.52	0.85	1.18
Total nodes evaluated (million)	2,271	2,426	2,534	2,883
Total nodes in memory (million)	68	68	70	73

remark that in absolute terms, RP–PN$_8^2$ is slightly faster than PNS. Despite the fact that RP–PN^2 has a better scaling factor than RP–PNS, RP–PNS is still faster than RP–PN^2 when the same number of threads is used. However, RP–PN^2 consumes less memory than RP–PNS.

Based on these results we may conclude that RP–PNS and RP–PN^2 are viable for parallelizing PNS and PN^2, respectively.

Acknowledgments

This work is financed by the Dutch Organisation for Scientific Research in the framework of the project GO FOR GO, grant number 612.066.409.

References

[1] L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands, 1994.

[2] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–123, 1994.

[3] M. Brockington. A taxonomy of parallel game-tree search algorithms. *ICCA Journal*, (19):162–174, 1996.

[4] A. Kishimoto. Parallel AND/OR tree search based on proof and disproof numbers. In *5th Games Programming Workshop*, pages 24–30, 1999.

[5] A. Kishimoto and M. Müller. Df-pn in Go: An application to the one-eye problem. In H.J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games 10: Many Games, Many Challenges*, pages 125–141. Kluwer Academic Publishers, Boston, MA, USA, 2003.

[6] A. Nagai. A new AND/OR tree search algorithm using proof number and disproof number. In *Proceedings of Complex Games Lab Workshop*, pages 40–45. ETL, Tsukuba, Japan, 1998.

[7] A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and its Applications*. PhD thesis, The University of Tokyo, Tokyo, Japan, 2002.

[8] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[9] M. Seo, H. Iida, and J.W.H.M. Uiterwijk. The PN*-search algorithm: Application to Tsume-Shogi. *Artificial Intelligence*, 129(1-2):253–277, 2001.

[10] Y. Shoham and S. Toledo. Parallel randomized best-first minimax search. *Artificial Intelligence*, 137(1-2):165–196, 2002.

[11] M.H.M. Winands, J.W.H.M. Uiterwijk, and H.J. van den Herik. PDS-PN: A new proof-number search algorithm: Application to Lines of Action. In J. Schaeffer, M. Müller, and Y. Björnsson, editors, *Computers and Games (CG 2002)*, volume 2883 of *Lecture Notes in Computer Science*, pages 170–185, Berlin, Germany, 2003. Springer-Verlag.