Benchmarking Parallel Chess Search in Stockfish on Intel Xeon and Intel Xeon Phi Processors[⋆]

Pawel Czarnul[0000−0002−4918−9196]

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, Narutowicza 11/12, 80-233 Poland
pczarnul@eti.pg.edu.pl

Abstract. The paper presents results from benchmarking the parallel multithreaded Stockfish chess engine on selected multi- and many-core processors. It is shown how the strength of play for an n-thread version compares to 1-thread version on both Intel Xeon and latest Intel Xeon Phi x200 processors. Results such as the number of wins, losses and draws are presented and how these change for growing numbers of threads. Impact of using particular cores on Intel Xeon Phi is shown. Finally, strengths of play for the tested computing devices are compared.

Keywords: parallel chess engine, Stockfish, Intel Xeon, Intel Xeon Phi

1 Introduction

For the past several years, growth in performance of computing devices has been possible mainly through increasing the number of cores, apart from other improvements such as cache organization and size, much less through increase in processor clock speed. This is especially visible in top HPC systems on the TOP500 list [14]. The top system is based on Sunway manycore processors, the second is a hybrid multicore Intel Xeon + Intel Xeon Phi coprocessor based system and the third a hybrid multicore Intel Xeon + NVIDIA P100 GPUs.

It is becoming very important to assess which computing devices perform best for particular classes of applications, especially when gains from increasing the number of threads are not obvious. We investigate performance of parallel chess game playing in the strong Stockfish engine [13], especially on the latest Intel Xeon Phi x200 processor which features upgraded internal mesh based architecture, MCDRAM memory and out of order execution. This is compared to both scalability and playing strength on server type Intel Xeon CPUs.

2 Related work

Performance of chess players, both human and engines, that play against each other is typically assessed using the Elo rating system [6, 11]. Some approaches

⋆ partially supported by the Polish Ministry of Science and Higher Education, part of tests performed at Academic Computer Center, Gdansk, Poland, part of tests performed on hardware donated by Intel Technology Poland

have been proposed such as the Markovian interpretation for assessment of play by various players from various eras to be able to compare strengths of play [1].

The typical algorithm for tree search in chess has been alpha-beta search. Several algorithms and approaches[1] regarding parallelization of chess playing have been proposed. In Young Brothers Wait Concept [7] the algorithm first searches the oldest brother in a search tree to obtain cutoff values and search other branches in parallel. In Lazy SMP many threads or processes search the same tree but with various depths and move orderings. It is used in many engines today such as Stockfish. Asynchronous Parallel Hierarchical Iterative Deepening (APHID) is an algorithm that is asynchronous and divides the search tree among the master (top level) which makes passes over its part of the tree and slaves which search deeper parts of the tree. Paper [3] presents speed-ups from 14.35 up to around 37.44 on 64 processors for various programs including Chinook, TheTurk, Crafty and Keyano. Monte-Carlo Tree Search, while successful for Go, suffers from issues such as difficulty to identify search traps in chess [2]. Despite optimizations, the testbed implementation could not match the strength of alpha-beta search. On the other hand, the very recent paper [12] presents AlphaZero – a program that defeated Stockfish using alpha-beta search. AlphaZero uses MCTS combined with incorporation of a non-linear function approximation based on a deep neural network. It searches fewer positions focusing on selected variations. In paper [16] the authors proposed a method called P-GPP that aimed at improving the Game Position Parallelization (GPP) that allows parallel analysis of game subtrees by various workers. P-GPP extends GPP with assignment of workers to nodes using realization probability. Implementation was tested using Stockfish for workers with communication between the master and workers using TCP sockets for up to 64 cores using two computers. The authors have demonstrated increased playing strength up to sixty workers at the win rate of 0.646.

Benchmarking performance and speed-ups was performed in several works and for engines and algorithms playing various games as well as for various computing devices – CPUs, GPUs, coprocessors such as Intel Xeon Phi. For instance, in [9] Monte Carlo Tree Search (MCTS) was benchmarked on Intel Xeon Phi and Intel Xeon processors. Speed-ups up to around 47 were achieved for Intel Xeon Phi and up to around 18 for Intel Xeon across all tested implementations including C++ 11, Cilk Plus, TBB and TPFIFO with a queue implementing work sharing through a thread pool. Furthermore, paper [10] contains data and comparison of performance of Intel Xeon CPU to Intel Xeon Phi for the MCTS algorithm useful in games such as Hex and Go. The same authors tested performance and speed-ups on both 2x Intel Xeon E5-2596v2 for a total of 24 physical cores and 48 logical processor as well as Intel Xeon Phi 7120P with 61 cores and 244 logical processors. The authors, having benchmarked n-thread versions against n/2-thread versions, have determined that almost perfect speed-ups could be observed up to 16 and 64 cores for the two platforms respectively. Furthermore, they determined that the Intel Xeon Phi coprocessor offered visibly worse total

[1] https://chessprogramming.wikispaces.com/Parallel+Search

performance than CPUs due to relatively higher communication/compute ratio. In paper [8] the authors proposed a general parallel game tree search algorithm on a GPU and benchmarked its performance compared to a CPU-based platform for two games: Connect6 and chess. The speed-up compared to the latter platform with pruning turned out to be 10.58x and 7.26x for the aforementioned games respectively. In terms of large scale parallelization on a cluster, paper [15] presents results obtained on a cluster for for Shogi chess. The authors have investigated effects of dynamic updates in parallel searching of the alpha-beta tree which proved to offer significant improvements in performance. Speed-ups up to around 250 for branching factor 5 and depth 24 on 1536 cores of a cluster with dynamic updates and without sharing transposition tables were measured. The authors have shown that using Negascout in the master and proper windows in workers decreases the number of nodes visited and generates speed-up of up to 346 for the aforementioned configuration. Several benchmarks have been conducted for Stockfish that demonstrate visible speed-up versus the number of threads on multi-core CPUs[2] [3].

The contribution of this work is as follows:

1. benchmarking the reference Stockfish chess engine on the state-of-the-art Intel Xeon Phi x200 manycore processor,
2. testing on how selection of cores (thread affinity) affects performance,
3. comparison of Intel Xeon and Intel Xeon Phi performance for Stockfish.

3 Methodology and experiments

Similarly to the tests already performed on multicore CPUs[2] [3], in this work we benchmark the Stockfish engine running with a particular number of threads against its 1 thread version. Gains allow to assess how much better a multithreaded version is and to what number of threads (and cores on which the threads run) it scales. This is especially interesting in view of the recent Intel Xeon x200 processors with up to 72 physical cores and 288 logical processors.

For the experiments performed in this work the following computing devices were used: 2 Intel Xeon E5-2680 v2 at 2.80GHz CPUs with a total of 20 physical cores and 40 logical processors as multi-core processors and 1 Intel Xeon Phi CPU 7210 at 1.30GHz with a total of 64 physical cores and 256 logical processors.

Each configuration included 1000 games played by an n thread version against the 1 thread version. Games were played with white and black pieces by the versions with switching colors of the pieces after every game. For the 256 thread configuration, the Stockfish code was slightly updated to allow such a configuration (the standard version allowed up to 128 threads). Time controls were 60 seconds for first 40 moves.

As a reference, Figure 1 presents how the results changed for a configuration of an n-thread Stockfish against the 1 thread version over successive games played

[2] http://www.fastgm.de/schach/SMP-scaling.pdf
[3] http://www.fastgm.de/schach/SMP-scaling-SF8-C10.pdf

on the Intel Xeon Phi. The score is computed as follows from the point of view of the n-thread version:

$$s = \frac{1 \cdot n_{\text{wins}} + \frac{1}{2} \cdot n_{\text{draws}}}{n_{\text{wins}} + n_{\text{draws}} + n_{\text{losses}}} \tag{1}$$

where: n_{wins} – number of wins, n_{draws} – number of draws, n_{losses} – number of losses for the n-thread version.

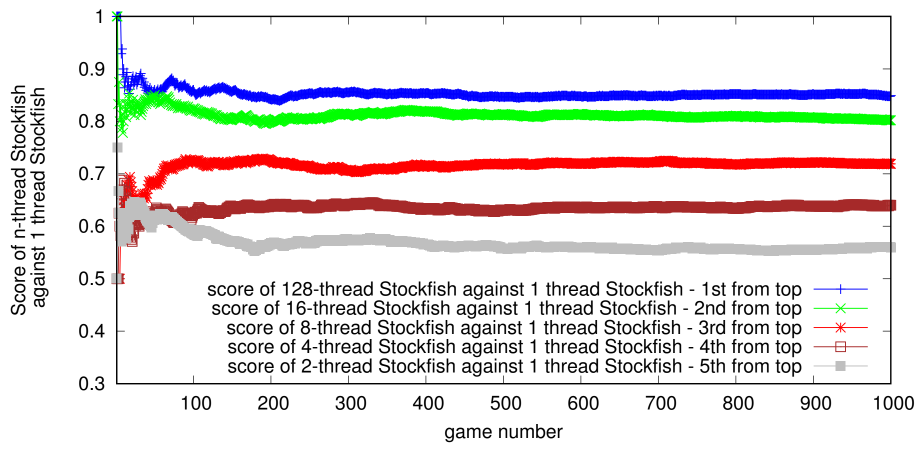

Fig. 1. Score of n-thread Stockfish against 1 thread Stockfish on Intel Xeon Phi x200

Tests were performed using tool `cutechess-cli` [4]. Firstly, tests were performed on the Intel Xeon Phi on how using particular cores affects performance. In one version, the application was instructed to use physical cores first. This was achieved with command `taskset` according to placement and identification of cores provided in file `/proc/cpuinfo`. In the other version, threads could use all available logical processors, no `taskset` command was used. Comparison of results for the Intel Xeon Phi and the two versions is shown in Figure 2.

Following tests were performed with using physical cores first on the Intel Xeon Phi x200. Figure 3 presents numbers of games with a given result: win, loss or draw out of 1000 games played by each n thread version against the 1 thread version and the final scores for each version computed using Equation 1. It can be seen that gain is visible up to and including 128 threads with a slight drop for 256 threads. The Stockfish code was modified (changed limits) to allow running on 256 threads as the original version limited the number to 128 threads.

Furthermore, analogous tests were performed for a workstation with 2 Intel Xeon E5-2680 v2 2.80GHz CPUs with a total of 20 physical cores and 40 logical processors, available to the author. Numbers of particular results and final scores are shown in Figure 4. For this configuration, best results were obtained for 16 threads with a slight drop for 40 threads.

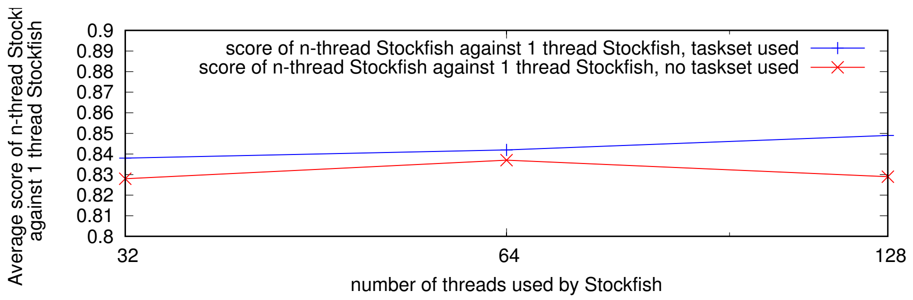

Fig. 2. How using `taskset` affects performance on Intel Xeon Phi x200

Apparently, better scalability for the Intel Xeon Phi stems from relatively lower performance of a single core and consequently better potential for improvement of the playing strength. This can be observed also for parallel computation of similarity measures between large vectors for which better speed-up compared to 1 core was observed for an Intel Xeon Phi coprocessor but final performance appeared to be similar for Intel Xeon Phi and two Intel Xeon CPUs [5].

Additionally, Stockfish was benchmarked for nodes/second processed for various numbers of threads involved. Results are shown in Figure 5. The performance of a single Intel Xeon Phi core is lower than that of a single Intel Xeon processor core. For the maximum number of threads equal to the number of logical processors on both platforms, the theoretical performance of the Intel Xeon Phi is slightly larger. It should be noted though that this is more of a theoretical benchmark for performance of processing of this particular code.
Finally, the best version on the Intel Xeon E5 CPUs using 16 threads was tested against the best version on the Intel Xeon Phi processor using 128 threads. Out of 1050 games, 38 were won on Intel Xeon Phi, 37 were lost and 975 draws were observed for a final score of 0.500047619 computed using Equation 1.

4 Summary and future work

In the paper we have investigated speed-up potential of the Stockfish multithreaded chess engine on both multi- and many-core processors such as Intel Xeon and latest Intel Xeon Phi x200 processors. It was shown that using `taskset` to select cores on an Intel Xeon Phi improved performance. Performance of two tested Intel Xeon processors appeared to be practically the same as one tested Intel Xeon Phi processor for the chess engine. We plan to test more computing devices including latest Intel Xeon CPUs as well as to conduct tests for a wider range of time controls, especially larger ones that might turn out to be more beneficial for processors with more cores.

References

1. Alliot, J.M.: Who is the master? ICGA Journal **39**(1), 3–43 (May 2017)

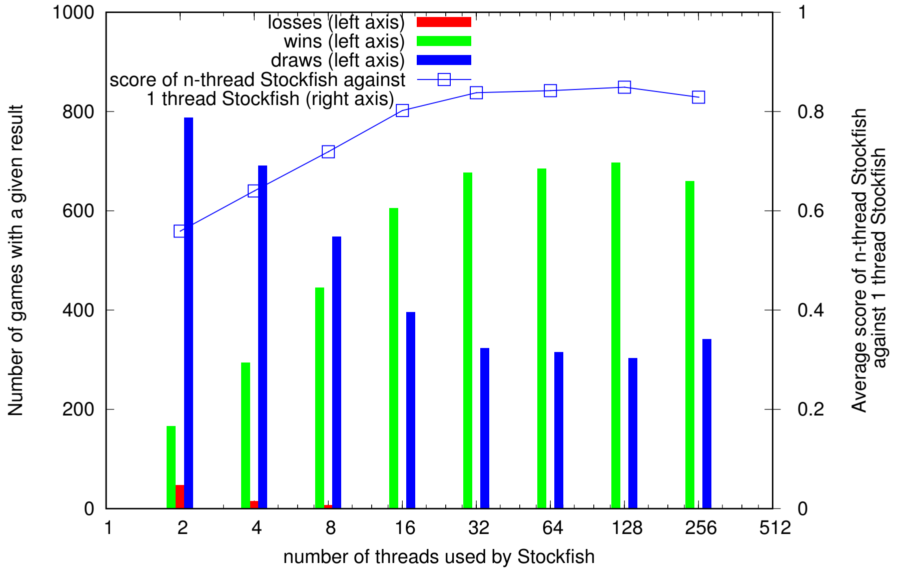

Fig. 3. n thread Stockfish against 1 thread Stockfish on Intel Xeon Phi x200

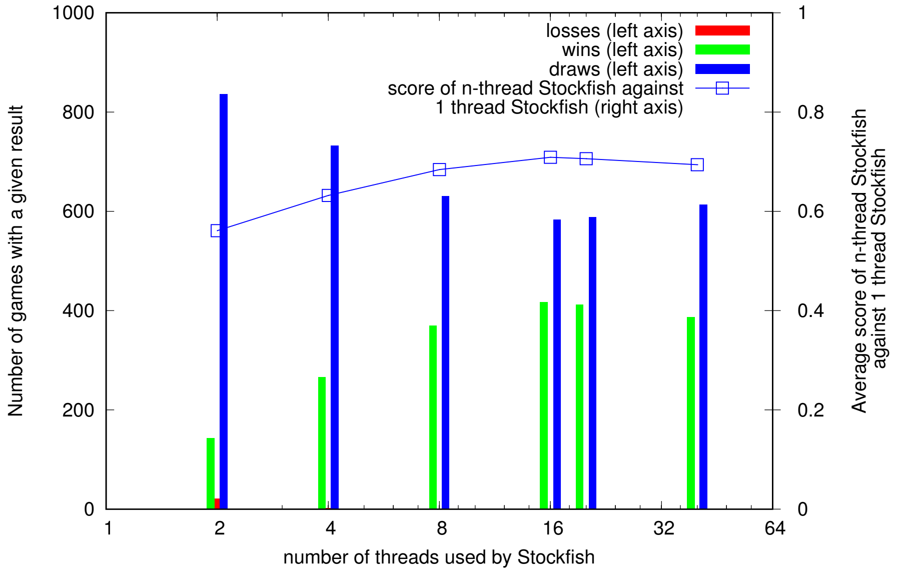

Fig. 4. n thread Stockfish against 1 thread Stockfish on 2 x Intel Xeon E5-2680 v2

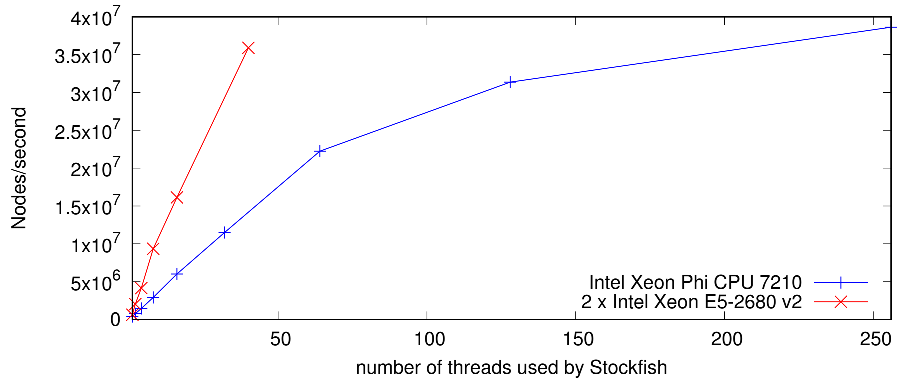

Fig. 5. Nodes/second processed on the testbed platforms

2. Arenz, O.: Monte carlo chess (April 2012), bachelor Thesis, Technische Universitat Darmstadt
3. Brockington, M.G., Schaeffer, J.: Aphid: Asynchronous parallel game-tree search. J. Parallel Distrib. Comput. **60**, 247–273 (2000)
4. Cute Chess Website: (2017), https://github.com/cutechess/cutechess
5. Czarnul, P.: Benchmarking performance of a hybrid intel xeon/xeon phi system for parallel computation of similarity measures between large vectors. International Journal of Parallel Programming **45**(5), 1091–1107 (Oct 2017)
6. Elo, A.: The Rating of Chess Players, Past and Present. Ishi Press (2008)
7. Feldmann, R., Monien, B., Mysliwietz, P., Vornberger, O.: Distributed Game Tree Search, pp. 66–101. Springer New York, New York, NY (1990)
8. Li, L., Liu, H., Wang, H., Liu, T., Li, W.: A parallel algorithm for game tree search using gpgpu. IEEE Trans. on Parall. & Distr. Systems **26**(8), 2114–2127 (2015)
9. Mirsoleimani, S.A., Plaat, A., Herik, J.V.D., Vermaseren, J.: Scaling monte carlo tree search on intel xeon phi. 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS) **00**, 666–673 (2016)
10. Mirsoleimani, S.A., Plaat, A., van den Herik, H.J., Vermaseren, J.: Parallel monte carlo tree search from multi-core to many-core processors. In: TrustCom/ BigDataSE/ISPA, Helsinki, Finland, Aug. 20-22, Vol. 3. pp. 77–83. IEEE (2015)
11. Rydzewski, A., Czarnul, P.: A distributed system for conducting chess games in parallel. In: 6th International Young Scientists Conference in HPC and Simulation. Procedia Computer Science, Kotka, Finland (November 2017)
12. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. ArXiv e-prints (Dec 2017)
13. Stockfish: (2017), https://stockfishchess.org/
14. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: Top500, www.top500.org/
15. Ura, A., Tsuruoka, Y., Chikayama, T.: Dynamic prediction of minimal trees in large-scale parallel game tree search. J. of Inform. Processing **23**(1), 9–19 (2015)
16. Yokoyama, S., Kaneko, T., Tanaka, T.: Parameter-Free Tree Style Pipeline in Asynchronous Parallel Game-Tree Search, pp. 210–222. Springer International Publishing, Cham (2015)