Accelerate Deep Learning Inference with MCTS in the game of Go on the Intel Xeon Phi

Ching-Nung Lin[1,a)] Shi-Jim Yen[1,b)]

Abstract: The performance of Deep Learning Inference is a serious issue when combining with speed delicate Monte Carlo Tree Search. Traditional hybrid CPU and Graphics processing unit solution is bounded because of frequently heavy data transferring. This paper proposes a method making Deep Convolution Neural Network prediction and MCTS execution simultaneously at Intel Xeon Phi. This outperforms all present solutions. With our methodology, high quality simulation with pure DCNN can be done in a reasonable time.

1. Introduction

Combining Deep Convolutional Neural Network(DCNN) and Monte Carlo Tree Search(MCTS) becomes a hot topic after AlphaGo beats a Korea professional go player. However, DCNN evaluation is 1000 time slower than the traditional pattern matching method. So, asynchronous combination and batch inference are used as a compromise. In fact, synchronized DCNN with MCTS performs better[1]. This paper focuses on how to accelerate it. Although Graphics processing unit(GPU) is good at DCNN training, Xeon Phi outperforms GPU at DCNN prediction with the batch size equal to 1.

Efficiently inference in GPU requires batch execution, which is able to use maximal threads in the hardware to archive good parallel performance. However, according to this research[2], it is possible to increase batch size equal to 1 prediction speed by modifying algorithms. But, MCTS requires sufficiently large simulations to show its potential. Batch inference effects MCTS quality badly because of the difficulty to estimate future node selections.

The contribution in this research describes how Xeon Phi can efficiently execute DCNN prediction and MCTS on itself: There are three steps: (1) explain how to use Single instruction, multiple data(SIMD) in Xeon Phi which is benefit for DCNN inference (2) invent Knowledge Plane to reduce DCNN size without losing good prediction accuracy (3) show Xeon Phi that is excellent standalone running parallel MCTS.

[1] Dept. of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan
[a)] 810221001@gms.ndhu.edu.tw
[b)] sjyen@mail.ndhu.edu.tw

	Xeon Phi(31SP1)	GTX 980 Ti
Cores	57	22 SMM
Threads	228	2816
L1 Data(KB)	32 + 32	64 + 48
L2 Data(KB)	512 x 57	3072
Independent core	228	22

Table 1 Table of Intel Xeon Phi and Nvidia GTX 980 Ti architecture

2. Architecture

Table 1 shows the main differences between Xeon Phi and GTX 980 Ti. On-board memory is excluded because of at least 10 times slowness. The GPU has more threads which is efficient for deep learning training. On the contrary, Xeon Phi has more independent cores which can run different processes simultaneously. In addition, the bigger cache size makes efficient data locality. However, For accelerating DCNN inference, fit as much as data in the local cache memory is an important key point. Xeon Phi has more local memory per independent core(4 threads share 512KB.) comparing to 22 SMM sharing with 3072KB memory(1SMM is about 140KB). In addition, the latency accessing the local L2 cache in Xeon Phi is 11 cycle[3] in contrast of GPU which is more than 100 cycle[4]. Moreover, there are two advantages:

- Half Precision Floating-point Format(fp16) or 8 Bit Integer Format(int8): store everything with 16 bit float or 8 bit integer type, which is half or

1/4 size of the Floating-point format without sacrificing any inference quality[5]. Although GPU starts to support fp16 or int8 recently, Xeon Phi remains better performance benefits because of its bigger cache size for each thread.

- Vectorization: Xeon Phi supports 512 bit SIMD instructions, Load, store and max(which is used in ReLU) 16 float or integer values simultaneously. In addition, SIMD can do 16 FMADD($A*B+C$) at one time which is important at matrix multiplication.

3. Method

Xeon Phi is similar to CPU and owns more cores. We will introduce how to accelerate performance from SIMD computing, network size reducing and MCTS paralleling.

3.1 Accelerate Inference Computation

Table 2 A 7x7 Input Volume

x11	x21	x31	x41	x51	x61	x71
x12	x22	x32	x42	x52	x62	x72
x13	x23	x33	x43	x53	x63	x73
x14	x24	x34	x44	x54	x64	x74
x15	x25	x35	x45	x55	x65	x75
x16	x26	x36	x46	x56	x66	x76
x17	x27	x37	x47	x57	x67	x77

Table 3 A 3x3 Filter Weight

y11	y21	y31
y12	y22	y32
y13	y23	y33

Table 2 and 3 is a good example. As the filter size is 3x3, first evaluate the convolution value as $x11*y11+x21*y21+x31*y31+x12*y12+...$; Then, stride one cell right, calculating $x21*y11+x31*y21+x41*y31+x22*y12+....$ All elements in the input volume mostly are required to multiply all elements in the filter weight.

Our method is to calculate the multiplication of each element in the input volume and every element in the filter weight. By the benefit of vectorization, sixteen data can be processed at the same time. For example, In one execution, $x11, x21, x31, x41, ..., x13, x23$ can multiply $y11$ synchronously. Furthermore, at the same time, the previous result can be added such as $x11*y12+(x11*y11)$. After all result value is calculated, the next input volume can be merged by adding different value together, this can be done with SIMD swizzle and shuffle instruction.

3.2 Train a high quality DCNN with small size

As previous mention, if DCNN can fit into the thread local cache, the performance increases dramatically. For GPU, The latency of L2 cache is slow, so the performance is limited by the speed of memory access. But, in Xeon Phi, the speed is fast if the network fits into the L2 cache. Considering 256KB space to store net weight values(The others are for MCTS), There are 256*1024/(2(fp16) or 1(int 8)) = 131061 or 262122 weights stored as register like execution speed.

Next, how to train a high quality DCNN with small size of weights is the key. A new *Knowledge Plane* method is provided. Instead of using simple features such as liberty, history, distance to the center, ... as input planes, The knowledge from search, heuristic, ... can be merged as DCNN planes before training. This archives good prediction accuracy with quite small DCNN size. So other algorithms can be combined with DCNN by treating outcomes as planes. It might be a mechanism to mix different well developed algorithms and DCNN.

3.3 Parallel Monte Carlo Tree Search on Xeon Phi

GPU is not suitable for MCTS because of the dependency diversity. When there are a lot of diverge, the performance drop significantly. This is the reason that MCTS is running on CPU in most cases. Xeon Phi is close to CPU, each thread is hardware independent. It can archive good parallel MCTS performance as same as the SMP CPU architecture, there are more threads(228 vs. 32) in Xeon Phi. Depending on the playout numbers, it performs best if the store size of playouts fits into the local cache.

4. Experimental Result

4.1 Synchronized Performance

The darkforest bot from Facebook with synchronized DCNN which tree is expanded after getting the result from DCNN evaluations is tested against Pachi. The simulation number of pachi(11.00) is fixed as 400000(about 2 dan level, no DCNN) without loading any pattern. The pattern is not used because huge number of random simulations sometimes benefits for Semeai. It beats the darkforest with DCNN rollouts equal to 1024 by killing it. Table 4 shows that the winning rates increase significantly when the number of DCNN rollouts is big enough.

4.2 Inference Speed

The darkforest has total 12 layers with 25 feature planes; The alphago has 13 layers with 48 feature

Rollouts	1	256	1024
Pachi	44.1% ± 4.9%	81.4% ± 3.9%	88.2% ± 3.2%

Table 4 Results (Winning rates) of darkforest various DCNN rollouts against Pachi

planes. A small DCNN is defined as 11 layers with 7 feature planes. Each layer has 16 channels of kernel size 3x3 and the last layer convolves 1 channel of kernel size 1x1. Table 5 describes the evaluating time for different architectures. It is tested with a single thread on Intel Xeon Phi 31S1P card. The speed of batch size equal to 1 on GPU for deep learning inference in Go is around 0.2 second on darkforest[1] and 0.15 second on alphago[6]. The evaluation time on Xeon Phi depends on the DCNN size. When the DCNN fits the cache size, it outperforms the GPU which is bounded to 0.15 seconds because of the transferring, initializing, ...

	Darkforest	AlphaGo	Small
Weight number	13m	4m	0.04m
Time in seconds	5.21s	1.41s	0.0076s

Table 5 Results of evaluating time

In addition, a knowledge plane is added in addition to the darkforest 25 feature planes, which is a maximal 6 move semeai for 2 liberty group search with long ladder checking. For example, in Figure 1, the three stones killed is found in one move. Figure 2 shows a common two liberty semeai problems found in five moves. In addition, it can solve more complex problems such as Figure 3 to kill that four stones. In Figure 4, it is a common situation in many games. This killed stone is recognized instead of treating as one normal two liberty stone. It requires average one second to make this plane on a E5-2670 CPU. For practical reason to combine with DCNN inference, using Field-programmable gate array(FPGA) to accelerate this process is a good choice.

The Network Architecture is first 5x5 convolution and 18 3x3 convolution layers, each convolution layer follows a ReLU nonlinearity. Table 6 describes the channel sizes. There are total 20 layers, each layer has the same channel size but last one. The last layer has 3x3 kernel with 1 channel. The training data is 10 millions board positions from 9D player games on the Tygen Go server. The testing data is another 0.3 millions board positions from the same source. The input planes is the same as the darkforest one adding one knowledge plane, The knowledge plane is one extra search plane explained previously. The training batch size is set to 10; the initial learning rate is 0.05; the learning policy is Inverse Decay with Gamma equal to 0.0001 and Power equal to 0.75.

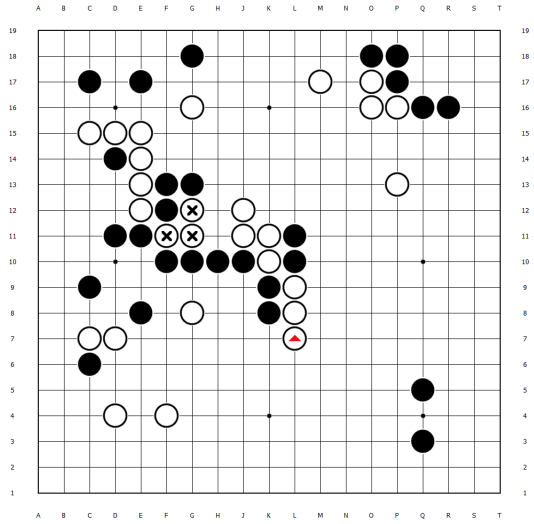

Fig. 1 Long ladder

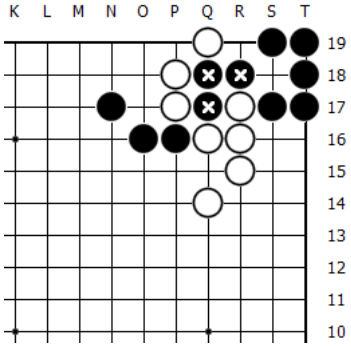

Fig. 2 Semeai 1

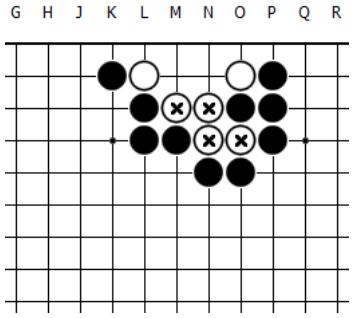

Fig. 3 Semeai 2

There are 55 epochs for each training. First, the accuracy is better than small DCNNs published lately which is less than 40%[7]. In addition, it outperforms any patterns trained from Minorization-Maximization Algorithm[8]. Second, when the channel size is reduced, the benefit from the extra knowledge plane increases. With decreasing network size, the predicting time is reduced from 0.24 second to 0.03 seconds. Int8 is slightly faster than fp16. In our experience, Int8 is faster significantly than fp16 when the network size is increased such as the channel size equal to 384. It is

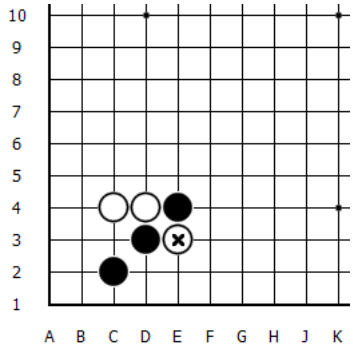

Fig. 4 Semeai 3

possible to get the same quality network with smaller size by adding more knowledge planes.

Channel size	64	32	24
Accuracy for 25 planes	49.41%	47.27%	46.41%
Accuracy for 25+1 Plane	49.57%	48.01%	47.58%
Weight number	705152	186688	108912
Time in seconds(int8)	0.24	0.05	0.03
Time in seconds(fp16)	0.25	0.06	0.03

Table 6 Results of small DCNN

4.3 MCTS scalability on Xeon Phi

According to previous research[9], Xeon Phi is scaling up to 47 times fast on MCTS. Also, in Table 7, our Chinesedark chess program which is MCTS with domain knowledge can scale up to 112 threads. It executes standalone MCTS on itself. The number of playout is 22400 which is similar to the number as darkforest plays on the computer go tournaments. The Tree parallelization is used with the global lock. If each thread evaluates its own DCNN and then runs MCTS on its own, it outperforms GPU in synchronize mode in the same time constraint because each thread has its own computing and storing resource and no communication bottleneck.

Furthermore, in our Small network, the inference speed is more efficient than GPU, doing a DCNN only simulation(There are average 200 moves in a simulation.) costs 1.5 seconds averagely. According the previous paper[7], a two layer network with 0.02 million weights is used, it can perform 80 to 170 simulations on one GPU. The similar evaluation speed($112 threads/1.5 = 75$) can be archived with a bigger network(0.04 v.s. 0.02 million) on Xeon Phi.

Thread number	28	56	112	224
Time in seconds	1306	653	375	321

Table 7 Scalability of Xeon Phi on MCTS

5. Conclusions

Xeon Phi can outperforms deep learning Inference with GPU especially when the network fits the cache size. It is possible to merge playout policy with pure DCNN evaluation in MCTS. This will improve the strength of the simulation quality dramatically. In the future, adding more knowledge in the planes is a good direction to go.

6. Acknowledgement

The authors would like to thank anonymous referees for their valuable comments in improving the overall quality of this paper, and Ministry of Science and Technology of Taiwan for financial support of this research under the contract numbers 104-2221-E-259 -009 -MY2.

References

[1] Tian, Yuandong and Zhu, Yan, “Better Computer Go Player with Neural Network and Long-term Prediction”, ICLR, 2016.
[2] Nvidia, “GPU-Based Deep Learning Inference: A Performance and Power Analysis”, Nov., 2015.
[3] Jianbin Fang; Ana Lucia Varbanescu; Henk J. Sips; Lilun Zhang; Yonggang Che and Chuanfu Xu, “An Empirical Study of Intel Xeon Phi”, CoRR, 2013.
[4] X. Mei; X. Chu, “Dissecting GPU Memory Hierarchy through Microbenchmarking,” in IEEE Transactions on Parallel and Distributed Systems , vol.PP, no.99, pp.1-1
[5] Suyog Gupta; Ankur Agrawal; Kailash Gopalakrishnan and Pritish Narayanan, “Deep Learning with Limited Numerical Precision”, CoRR, 2015.
[6] David Silver; Aja Huang; Christopher J. Maddison; Arthur Guez; Laurent Sifre; George van den Driessche; Julian Schrittwieser; Ioannis Antonoglou; Veda Panneershelvam; Marc Lanctot; Sander Dieleman; Dominik Grewe; John Nham; Nal Kalchbrenner; Ilya Sutskever; Timothy Lillicrap; Madeleine Leach; Koray Kavukcuoglu; Thore Graepel and Demis Hassabis, “Mastering the game of Go with deep neural networks and tree search”, Nature, 2016, Pages 484-503, Volume 529.
[7] Peter H. Jin and Kurt Keutzer, “Convolutional Monte Carlo Rollouts in Go”, CoRR, 2016.
[8] Coulom, Rmi. “Computing elo ratings of move patterns in the game of go.” Computer games workshop. 2007.
[9] S. Ali Mirsoleimani; Aske Plaat; H. Jaap van den Herik and Jos Vermaseren, “Scaling Monte Carlo Tree Search on Intel Xeon Phi”, CoRR, 2015.