Memory subsystem performance analysis for CNN workloads

Dmytro V. Rahozin
Institute of Software Systems, NAS of Ukraine
Kyiv, Ukraine
dmytro.rahozin@gmail.com

Anatoliy Yu. Doroshenko
Igor Sikorsky Kyiv Polytechnic Institute
Kyiv, Ukraine
doroshenkoanatoliy2@gmail.com

***Abstract*—Details of modern convolution neural networks performance for parallel computer systems are considered. Memory subsystem performance analysis method, based on statistical elaboration of memory access patterns is proposed.**

***Keywords*—convolutional neural networks, parallel computers, memory performance.**

Introduction

Nowadays practically all the new object recognition and classification research is based on Convolutional Neural Networks [1] (CNN) technology, which looks to be the most profitable recognition technology for industrial implementation. Automotive industry and driverless cars are major CNN technology customers and require real-time object recognition. The CNN performance improvement is the computer science cutting edge task now.

Computational complexity problem for CNN

Here we are considering CNN implementations, which are targeted for nowadays industrial applications, mostly for object detection/recognition in real time. Two types of detectors are considered: two-stage, such as Faster-RCNN [2], where the region-of-interest is found first and recognition in the region is provided at the second stage; and single-stage such as YOLO (You Only Look Once) [3] and SSD (Single Shot Detector) that treat detection as a simple regression problem. The two-stage algorithms have slightly improved quality if compared to single-stage, but consume much more computing power. Single stage detectors have enough quality to satisfy the driverless cars technology. Car information systems need to process inputs form many cameras, lidars, ultrasonic sensors, and the essential requirement is to improve object detection quality and decrease power envelope. A power commercial direction for hardware object detection accelerators development was risen, for example ADAS cameras by Mobileye or Tesla car autopilot. The most popular YoloV3 implementation [3] uses specific method to improve system performance: it divides the source into smaller images: 13x13, 26x26, 52x52 pixels. These images are passed to so called "backbone" – a relatively small CNN which is able to classify the image even if it represents only a part of some object and able to predict the position of the object center point. The so called "neck" of YoloV3 gathers the prediction info from a grid of backbones and looks for objects, which are predicted simultaneously by multiple backbones. Grid image structure may skip some detections (for less than 5% of objects) but it is fast and uniform, as the backbone architecture is the same for all its runs and share the same data – see fig. 1.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 1. YoloV4 backbone sample architecture

Now there is no much sense to research performance of non-backbone parts of Yolo architecture, because the line of "Yolo-based" architectures constantly extends. For example, YoloV4 [3] is very different from YoloV3 and includes a big bucket of heuristics which improve overall recognition, and these heuristics are individual for each task and dataset. The research for "neck" performance should be done after deep research which heuristics are the most profitable for the task and the dataset. For 2019-2020 years the YoloV3 detector default resolution 416x416 looks as not acceptable for industry, as far away objects which are detectable at HD-ready resolution (720p) cannot be detected if down sampled 2 or 3 times – so these small objects contain only or less a dozen of pixels. It's highly important if a car moves with high speed any obstacle should be detected early. Moving from 416x416 pixels YoloV3 detector to 608x608 pixels increases performance requirements twice, the direct use of HD camera (1080p) requires 10x computational capability. Our investigations on a subset of CityScapes [4] dataset

```
Analysis                                  Ir          D1M          D2M  CODE
                                          15            2            1  for(b = 0; b < batch; ++b)
1.6KB read chunk lock in cache            50            1            1     for(i = 0; i < n; ++i)
                                      39,980          105            1        for(j = 0; j < size; ++j)
12.5MB R+modify chunk, non-cached 51,118,080 15,974,442 15,974,440           output[(b*n + i)*size + j] += biases[i];
```

Fig. 2. Tool report for “add bias” function

shows that the detection average precision raises with detector resolution (fig. 3), so that nowadays detector resolution should start from 832x832 pixels.

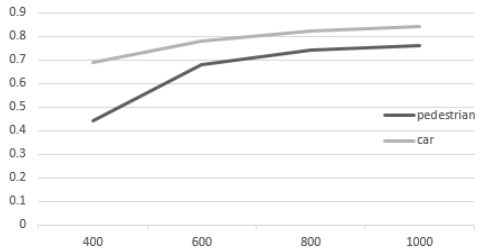

Fig. 3. Average precision rate vs detector resolution for Cityscapes subset

CNN PERFORMANCE IMPROVEMENT

So, the main target for CNN performance improvement is the backbone. There is no much attention for backbone optimization in research community, as it changes faster than an architecture to support this backbone can be manufactured. The common way for software performance improvement is the analysis of software execution and providing some common ways to eliminate performance bottlenecks [5]. For example: 1) introduce complex function accelerators into microprocessor; 2) extend cache memory size; 3) make SIMD vector twice wide; 4) place more execution kernels on chip – 128 instead 64 and so on. For example NVidia provides software development kits, which are used as a back-end for popular CNN engines line Tensorflow [6] or Caffee. These ways to improve software performance depend on hardware changes, but usually the hardware cannot be changed so basically two methods work: 1) optimize CNN for existing computing resources, use special function units; 2) control cache memory. For bigger detector resolutions, the number of backend runs increases, and it is a good chance to benefit from analysis of memory subsystem performance. Memory subsystem performance optimization is possible in hardware and software. Hardware includes: 1) use more cache memory; 2) use hardware prefetch - the cache controller analyses memory fetch addresses, try to predict a sequence of addresses and prefetch data ahead of execution; 3) prefetch co-processors, which should be programmed separately. The hardware prefetch is severely limited, as prefetch unit “sees” memory references locally. A goal of our research is a development of analysis tool which can provide a “1000 ft height” picture for the CNN execution, analyzing the memory and cache subsystem performance: 1) the improvement of hardware prefetches efficiency; 2) feasibility of hardware prefetch coprocessors utilization; 3) ability of cache control, i.e. locking data in cache for further reuse, forced write to memory with data invalidation, simple software prefetch use. An important question is if the multiple backbone runs can synchronize and share the data fetched from system memory in efficient way. We use Cachegrind [7] – cache memory simulator, it simulates instruction execution and memory state for different cache memory configurations. The Cachegrind tool can be used to analyze backbone performance for YoloV3/YoloV4. The tool statistics include: 1) identification of large data blocks, fetched to cache (see motivating example at fig. 2). Data blocks include strided arrays and linked lists used as a big data chunk. The sample annotation for better memory use is provided for code lines as in fig. 2. Big data structures should be checked for further data reuse and sharing to reserve cache memory for useful data. 2) Detecting points where read-only data are not used for processing any more allows to insert synchronization points into backbone code so that multiple running backbone code can share data from cache but not go to main memory.

For analysis YoloV4 code [3] is used, it is based on well-known Darknet infrastructure for CNNs inference.

CONCLUSION

We have proposed to acquire statistics from memory access traces for further optimization of CNN and other applications for different computer architectures. The memory access analysis is akin to common resource allocation tasks (e.g. register allocation in compilers) and provides improvements in memory performance. This allows to improve CNN code performance and use less computational resources.

REFERENCES

[1] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. In Proc. of IEEE CVPR ‘14, pp. 1725-1732.

[2] S.Ren, K.He, R.Girschick, J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In Proc. of the 28th Int. Conf. on Neural Inf. Proc. Systems, Vol. 1, Dec. 2015, pp. 91–99

[3] Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020.

[4] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, 2016.

[5] P.I. Andon, A.Yu. Doroshenko, K.A. Zhereb, O.A. Yatsenko. Algebra-algorithmic models and methods of parallel programming. Kyiv: ”Akademperiodika”, 2018, 192 p.

[6] M. Abadi, P. Barham, J. Chen, Zh. Chen, A. Davis, J. Dean et all. TensorFlow: A System for Large-Scale Machine Learning. In Proc. of the 12th USENIX SympI. on Operating Systems Design and Implementation (OSDI ’16).

[7] J. Weidendorfer. Sequential Performance Analysis with Callgrind and KCachegrind. In Proc. of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart, pp. 93-113