https://doi.org/10.2352/EI.2023.35.16.AVM-112
© 2023, Society for Imaging Science and Technology

Orchestration of Co-operative and Adaptive Multi-core Deep Learning Engines

Mihir Mody, Kumar Desappan, Pramod Swami, David Smith*, Shyam Jagannathan, Kevin Lavery*, Greg Shultz*, Jason Jones*
Embedded Processors Business, Texas Instruments (India and *USA)

Abstract

Automated driving functions, like highway driving and parking assist, are increasingly getting deployed in high-end cars with the goal of realizing self-driving car using Deep learning (DL) techniques like convolution neural network (CNN), Transformers etc. Deep learning (DL)-based algorithms are used in many integral modules of Advanced driver Assistance systems (ADAS) and Automated Driving Systems. Camera based perception, Driver Monitoring, Driving Policy, Radar and Lidar perception are few of the examples built using DL algorithms in such systems. These real-time DL applications requires huge compute requires up to 250 TOPs to realize them on an edge device. To meet the needs of such SoCs efficiently in-terms of Cost and Power silicon vendor provide a complex SoC with multiple DL engines. These SoCs also comes with all the system resources like L2/L3 on-chip memory, high speed DDR interface, PMIC etc to feed the data and power to utilize these DL engines compute efficiently. These system resource would scale linearly with number of DL engines in the system. This paper proposes solutions to optimizes these system resource to provide cost and Power efficient solution. (1) Co-operative and Adaptive asynchronous DL engines scheduling to optimize the peak resources usage in multiple vectors like memory size, throughput, Power/ Current. (2) Orchestration of Co-operative and Adaptive Multi-core DL Engines to achieve synchronous execution to achieve maximum utilization of all the resources.

Introduction

The Automated driving (AD) systems implement highway driving and value parking functions at multiple automation levels (L2-L5) [1-2]. The figure 1 shows the functional block diagram to achieve these functions. The key blocks for automated driving are namely Perception, Localization, Fusion, Driving Policy, Motion Planning and Control. The multi-modality perception (Camera, radar and Lidar) is used to gather environment information around car [3]. 'Fusion' module is used to give most reliable environment using Bayesian filtering among all modalities [4]. The 'localization' module is used to find exact position of vehicle in real world co-ordinates using High definition-HD Maps and perception data. The resulting environment model is used by 'Driving Policy' module to take decision e.g. stay in lane, lane change, yield, merge etc. The decision of Driving policy module is translated in actual car movement with 'Motion planning' module accounting kinematics and passenger's comfort. Lastly, typical 'control' module is used to track actual trajectory with respect to reference by controlling actuators.

As shown in figure 1, many of these modules are built with Deep Learning (DL) based approaches either completely or partially. With the pervasive used of DL, realization of autonomous systems includes one or more purpose-built processors for DL acceleration along with general purpose processors like Cortex ARM. As these DL engines (Hardware accelerators or DSPs) are purpose built, the silicon vender would provide custom software APIs to deploy DL model on these engines. DL algorithms in ADAS or Autonomous driving system are used for various kind of modalities (Camera, RADAR, LiDAR) and algorithms (Perceptions, Planning, Fusion), so a wide range of DL model architectures needs to be deployed/evaluated on these DL engines. These purpose-built HW engines would come with fixed set of functionally (DL Layer/Operator types) and software support. With rapidly evolving DL model architectures, development of these Automated Driving Systems would need to evaluate new models which may not be supported by exiting DL offering.

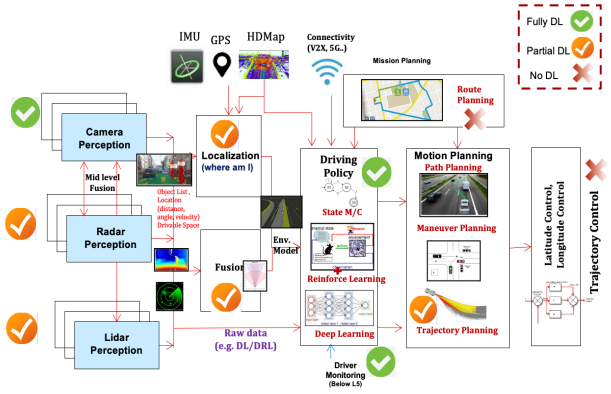

Figure 1. Computation blocks of Autonomous driving system.

Deep Learning application Development Workflow

A typical SoC targeted for high throughput DL applications like Automated driving would have multiple DL engines. The Figure-2 here describes example block diagram of such a SoC. Open-source DL inference frameworks (Tensorflow Lite, ONNX runtime, TVM, NEO-AI-DLR etc.) [5-8] have been developed to improve the ease of DL model deployment with offload mechanisms to DL accelerators [9-11]. The memory throughput requirements of DL engines are very high as these engines are designed to perform huge number of compute operations per cycles. To meet these requirements, each CNN/DL core has dedicated on chip level 2 memory and system resources like DMA engines statically linked to it. This would make the software execution on these cores would be independent and run totally asynchronously. The peak system resource requirement (like system resources like L2/L3 on-chip memory, high speed DDR interface, PMIC etc.) of this SoCs would traditionally scale linearly. For example, the L2 memory requirements would be 4x if the SoCs has 4 DL engines compared to one. The Figure -3

shows the software usage models of such linearly scaling SoCs design.

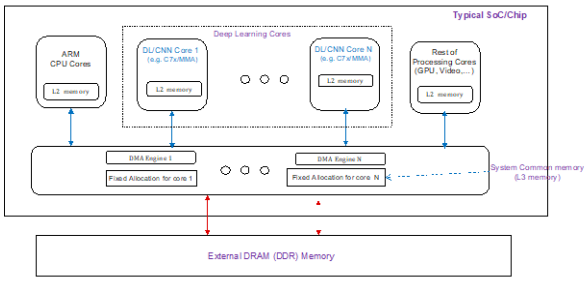

Figure 2. Typical system with Multi-Core DL/CNN Cores.

Scaling a such design for SoC with large number of DL engines would be practically not possible (peak power and Area requirements etc) to meet the compute requirements of systems like automated driving. An optimal software and SoC co-design would be needed to meet these requirements.

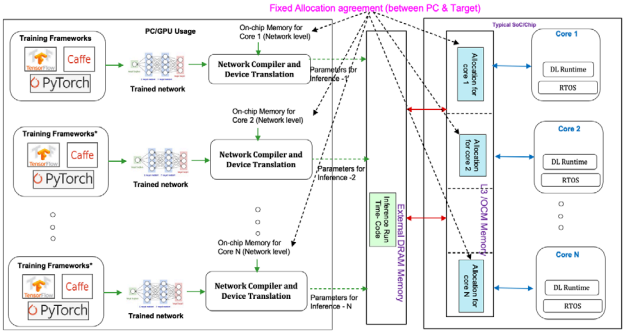

Figure 3. Software usage model for DL inference

Parallel execution flow of typical system described in the above section would be almost independent of other cores. The Figure 4 illustrates the execution flow of this system in a given time interval.

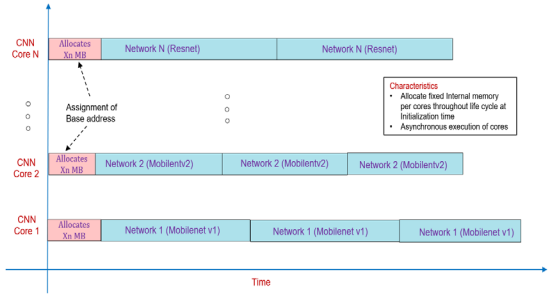

Figure 4. Typical Parallel CNN engines execution flow

Orchestration of Multi-core DL Engines

We propose below solutions to address the limitations described in the previous section.

1. Co-operative and Adaptive asynchronous DL engines scheduling to optimize the peak resources usage in multiple vectors like memory size, throughput, Power/ Current.
2. Orchestration of Co-operative and Adaptive Multi-core DL Engines to achieve synchronous execution to achieve maximum utilization of all the resources.

Adaptive Resource allocation for DL engines

The Figure - 3 describes the design of proposed SoC. Here all the system resources are not completely allocated statically at SoC design level. A portion of resources are statically allocated for each DL engine and a common portion is dynamically allocated across all the DL engines during runtime on as per the requirements of a layer being processed in given point of time. The SoC modules would be designed / enhanced to address such dynamic allocation. For example, to make the software view the dynamically allocated memory as linear extended memory a memory management unit (MMU) would be needed in DMA engines of each DL. The Figure - 4 shows the software usage models of SoCs design with dynamic resource allocation. A common resource manager software module with would be designed to service the request of dynamic resource allocation from each DL engine. The common resource manager would be registered with the layer level resource requirements from each DL engine at boot of the system to avoid any delays in allocation during the runtime.

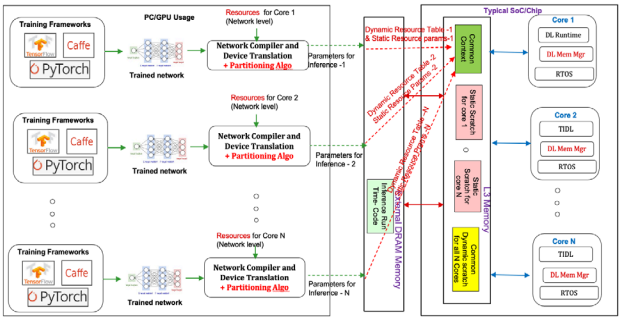

Figure 5. Software usage model for dynamic resource allocation and Adaption policies

Co-operative DL engines scheduling

When a resource manager runs out of resource of a given DL engine execution, we have two options to handle this based on user preference.

a) Adapt the layer execution policy to choose different strategy (either execute slowly or with reduce accuracy) with the limited system resource.

b) Stall the layer execution till all the required system resources are fully available.

The Figure 8. Illustrates the execution flow DL core with dynamic resource allocation and Adaption policies. During initialization of all the core, the minimum required resources are statically allocated. Also, layer level dynamic resource requirements are registered with the resource allocator. The applicable adaption polices for required accuracy and latency

requirements are configured as per the needs of application. The following steps describes sequence of resource allocation/execution at layer level for each DL core.

(1) If the statically allocated resources are enough to for the layer computation – Completes the execution flayer
(2) Request additional resources if needed, if they are available complete – Completes the execution flayer.
(3) Request the applicable adaption policy to compute the layer with available limited resources - if they are available complete – Completes the execution flayer.
(4) Stall the execution till one of request from steps (2) or steps (3) is serviced.

The Figure – 6 shows time-line view of such dynamically resource allocating scheduling.

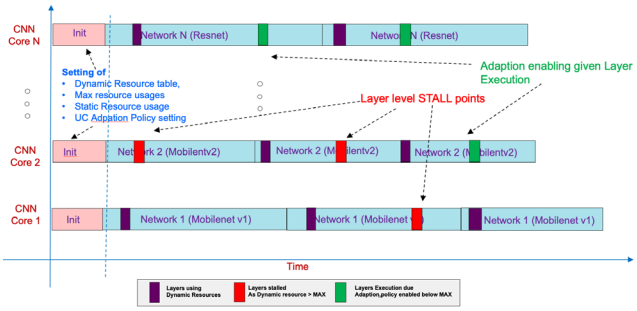

Figure 6. Execution flow with dynamic resource allocation and Adaption policies

Adaption policies to optimize system utilization

Adaption of resource requirements maximizes the system utilization without stalling the execution of any DL cores. This paper proposes various adoption configuration/policies for each critical system resources like memory size, throughput, Power/ Current. Some of these the polices also explained with details in the following sections. The Table 1 lists some of the example adaption configuration to reduce peak requirement of internal memory size, DRAM bandwidth, Internal memory throughput and Power.

Table I: Example adaption configuration

Adaption Type	Adaption Cofig-1		Adaption Config-2	
	Algorithm	Tradeoffs	Algorithm	Tradeoffs
DRAM BW	Parameter Precision	Quality Degrade	Feature Precision	Quality Degrade
Internal Memory Size	Use external DRAM Memory	Lesser FPS	Parameter and Feature Precision	Quality Degrade
Internal Memory Throughput	Parameter and Feature Precision	Quality Degrade	Use external DRAM Memory	Lesser FPS
Power	Run given Layer at lower speed	Lesser FPS	Run given layer on power efficient core	Lesser FPS

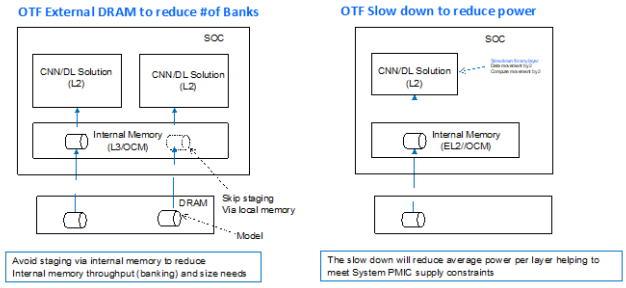

Figure 7. On-the-fly adaption policy to reduce peak internal memory throughput and peak power requirement

As described in the introduction section memory throughput is very critical contributor in efficiently utilizing the compute capabilities of a DL engines. Increasing external memory throughput to meets needs of multi core DL SoC design would need have impact cost and power requirements. The common Resource allocation manager would allocate these internal memory resources on demand. When multiple DL engines requests the internal memories at the same time and it would not allocate for all of them. In this case, the resource allocator would choose a applicable adaption policy, for example allocate DRAM for one of the DL engine instead of allocating in internal memory. This would impact the utilization for one of the DL engines and resulting higher latency. The adaption polices are decided at runtime to avoid any underutilization of internal memory in terms of both size and throughput. Another example of such adaption policy for internal memory would be to choosing lower bit depth for either parameters or feature vectors of layer. This would result in accuracy degradation, software would be configured to choose this based on acceptable accuracy targets by the application requirements. Figure 7 and 8 illustrates the selection of few adaption polices for memory and power optimization requirements.

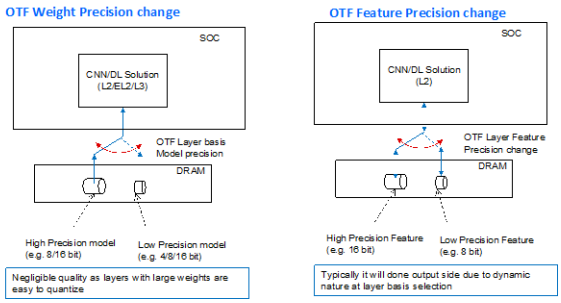

Figure 8. On-the-fly adaption policy to select weights and feature tensors Precision

Synchronous Execution of Multiple Cores

The dynamic resource allocator with adaption policy would proposed in the previous section would try to maximize the utilization of resource. As showcased in Figure - 6 time-line of such dynamically resource allocator, there will be scenarios were some of the DL engine execution would be stalled when a required resource could be allocated. When a SoC has multiple such DL engine, then occurrence and frequency of such stalls would non-

deterministic. Safety critcle systems have hard requirement to be deterministic interns both resource allocation and accuracy of the application. To address these requirements, we propose a multi core orchestrator on top of the dynamic resource allocation and Adaption policy software model described in previous sections. The multi core orchestrator prepares the dynamic resource requirements by analyzing the models running in all DL engine instead of analysis one model/network individually. The Figure -9 describes the software usage model of this orchestrator for Synchronous Execution of DL engine.

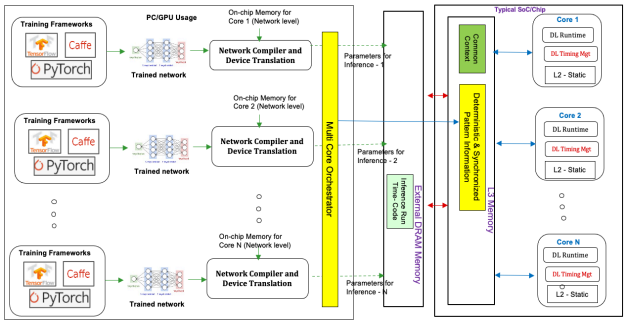

Figure 9. Software usage Model for Synchronous Execution

The common context table prepared by offline software would determine the stalls in the execution of each DL engines. These sync points are decided based on the resource requirements of the al the DL engines for a given time instance. This decision is made in offline software by analyzing adaption policies of all the model in a given time instance, compared one model's policy in asynchronous execution. The Figure -10 captures the time-line view of Synchronous Execution flow with Multi Core Orchestrator

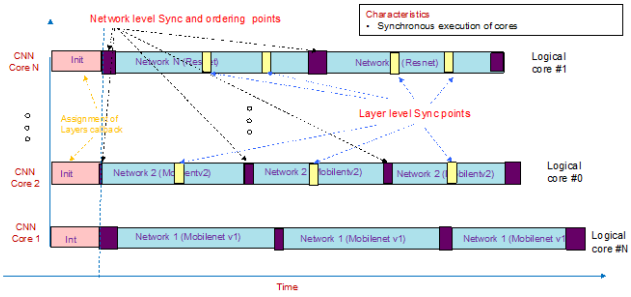

Figure 10. Synchronous Execution flow with Multi Core Orchestrator

Results

The graph in Figure 11 shows a SoC design decision for choice of number of internal memory ports while scaling the scaling numbers of DL engines with proposed solutions in this paper. The adoption configuration could achieve up to 25% reduction in number required ports, which would result in cost and power efficient SoC design. The graph in Figure 12 shows compute utilization with adaptive configuration in given SoC with multi core Orchestrator proposed for synchronous execution which results in up to 20% higher utilization.

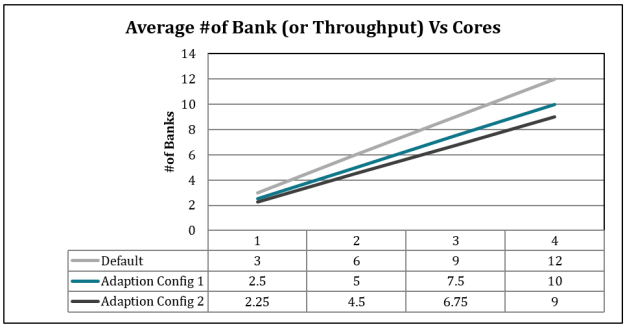

Figure 11. Peak Internal memory throughput requirements comparison

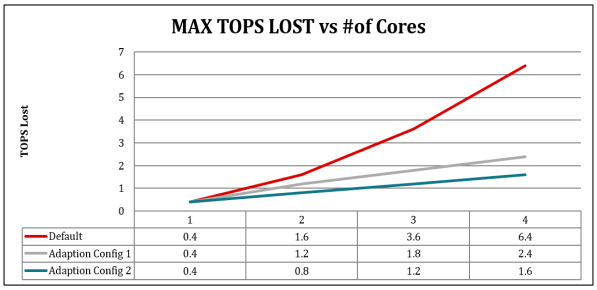

Figure 12. Peak TOPs utilization/Loss comparison

Conclusion

This paper proposed solutions to optimizes the system resource to provide cost and Power efficient solution of real-time DL applications requiring huge compute capabilities. The Co-operative and Adaptive resource allocator proposed here make it possible to design a SoC with multi-core DL engines without linearly scaling system resources. The multi core Orchestrator proposed for synchronous execution of core to meets the deterministic system requirement of safety critical applications.

References

[1] Mody, Mihir, Jason Jones, Kedar Chitnis, Rajat Sagar, Gregory Shurtz, Yashwant Dutt, Manoj Koul, M. G. Biju, and Aish Dubey. "Understanding vehicle e/e architecture topologies for automated driving: System partitioning and tradeoff parameters." *Electronic Imaging* 2018, no. 17 (2018): 358-1

[2] SAE J3016, "Taxonomy and Definitions for terms related to on-road automated motor vehicles"

[3] Mihir Mody, Niraj Nandan, Shashank Dabral, Hetul Sanghvi, et.al, "Image Signal Processing for Front Camera based Automated Driver Assistance System", IEEE International Conference on Consumer Electronics, (ICCE) , Berlin, 2015

[4] Shyam Jagannathan, Mihir Mody, Jason Jones, Pramod Swami and Deepak Poddar, "Multi-sensor fusion for Automated Driving: Selecting Model and Optimizing on Embedded Platform", AVM track, Electronic Imaging, 2018

[5] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin et al. "Tensorflow: A system for large-scale machine learning." In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265-283. 2016.

[6] ONNX Runtime - https://onnxruntime.ai/docs/execution-providers/

[7] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI), Carlsbad, CA, USA, 2018, pp. 578–594.

[8] Neo-AI DLR - https://github.com/neo-ai/neo-ai-dlr

[9] K Desappan, et.al, "CNN Inference: Dynamic and Predictive Quantization", IEEE International Conference on Consumer Electronics, (ICCE) , Berlin, 2018.

[10] M. Mathew, K. Desappan, P. K. Swami, S. Nagori, and B. M. Gopinath, "Embedded low-power deep learning with tidl," Texas Instrum., Dallas, TX, USA, Tech. Rep. SPRY314, 2018.

[11] Kumar Desappan, Anand Pathak, Pramod Swami, Mihir Mody, Yuan Zhao, Paula Carrillo, Praveen Eppa, Jianzhong Xu, "Open source deep learning inference libraries for autonomous driving systems" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Autonomous Vehicles and Machines, 2022, pp 118-1 - 118-5, https://doi.org/10.2352/EI.2022.34.16.AVM-118

Author Biography

Mihir Mody is SoC Architect lead (DMTS) responsible for roadmap and chip definition for Sitara MCU business in Texas Instrument (TI). His domains of interest are real time control, image processing, computer vision, deep learning and Video coding. He received his master's in electrical engineering from Indian Institute of Science (IISc) in 2000

Kumar Desappan is Senior Member of Technical Staff (SMTS) at Texas Instruments (TI) Incorporated. His domains of interest are Machine/Deep learning, image processing and computer vision algorithms with a focus on software solution for edge devices. He received Bachelor of Engineering (BE) from Anna University - Chennai in 2005

Pramod Swami is Distinguish Member of Technical Staff (DMTS) at Processors Business in Texas Instruments (TI) leading the software development for EdgeAI processing. His domains of interest are Embedded systems, Digital Signal Processors, Deep Learning, Computer Vision, Image Processing, and Video coding. He received his Bachelor's degree in Electronics and communication engineering from Malaviya National Institute of Technology (MNIT) Jaipur in 2001.

David Smith is a Design Engineer for Processors business at Texas Instruments (TI) Incorporated, responsible designing IPs Concentrating on optimal data movement and sub-system optimization. He received his Master of Science (MS) degree from Harvey Mudd College in 1996.

Shyam Jagannathan is an EdgeAI architect and Senior Member of Technical Staff at Embedded Processors Group, Texas Instruments. He received a master's degree from IIT Chicago in the field of Signal Processing and Communications and has been with Texas Instruments since 2005. His areas of interest include DSP architecture, SoC architecture, hardware accelerators, deep learning, perception, sensor fusion localization, path planning and overall system optimization

Kevin Lavery is an application's engineer, working in device characterization. His interests lie on the analog / digital boundary. He has a master's degree in physics from the University of Houston (1994).

Gregory Shurtz is Principal Systems Architect for Processors business at Texas Instruments (TI) Incorporated responsible for SoC platform definition for Infotainment & ADAS solutions. Former positions include FPGA Logic Designer at Lockheed Martin and Designer for mass storage systems at Symbios Logic. He received a Bachelor of Science in Electrical Engineering (BSEE) degree, summa cum laude, from Wichita State University in 1995.

Jason Jones is the Principal Systems Architect for Processors business at Texas Instruments (TI) Incorporated, responsible for current and future roadmap solutions encompassing Digital Cockpit, ADAS, and Vehicle Gateway applications. He holds Bachelor of Science (BS) degree from Texas A&M University in 1993.