SCREENING DEEP LEARNING INFERENCE ACCELERATORS AT THE PRODUCTION LINES

Ashish Sharma, Puneesh Khanna, Jaimin Maniyar

AI Group, Intel, Bangalore, India

ABSTRACT

Artificial Intelligence (AI) accelerators can be divided into two main buckets, one for training and another for inference over the trained models. Computation results of AI inference chipsets are expected to be deterministic for a given input. There are different compute engines on the Inference chip which help in acceleration of the Arithmetic operations. The Inference output results are compared with a golden reference output for the accuracy measurements. There can be many errors which can occur during the Inference execution. These errors could be due to the faulty hardware units and these units should be thoroughly screened in the assembly line before they are deployed by the customers in the data centre.

This paper talks about a generic Inference application that has been developed to execute inferences over multiple inputs for various real inference models and stress all the compute engines of the Inference chip. Inference outputs from a specific inference unit are stored and are assumed to be golden and further confirmed as golden statistically. Once the golden reference outputs are established, Inference application is deployed in the pre- and post-production environments to screen out defective units whose actual output do not match the reference. Strategy to compare against itself at mass scale resulted in achieving the Defects Per Million target for the customers

KEYWORDS

Artificial Intelligence, Deep Learning, Inference, Neural Network Processor for Inference (NNP-I), ICE, DELPHI, DSP, SRAM, ICEBO, IFMs, OFMs, DPMO.

1. INTRODUCTION

Artificial Intelligence (AI) is growing in recent years and is expected to be re-shaping industries. Deep Neural networks are in the heart of the AI revolution. Deep Neural Network is composed of layers of simulated neurons with different connectivity schemes. The new computation model is based on massive parallel execution of linear algebra operations. New dedicated architectures that are optimized for Deep Learning execution is required to achieve high efficiency and to meet the market requirements. Deep learning inference accelerators are designed specifically to deliver superior performance, low latency, power efficiency and cost savings for cloud, data centres and other emerging applications.

SpringHill (NNP-I) is an Inference Chip from Intel which is used to accelerate execution of the arithmetic operations. It consists of 12 Inference Compute Engines (ICE) as described in Figure 1. Each ICE contains hardware accelerator IPs DELPHI and DSP. The operations that are supported by DELPHI includes direct CNNs (convolution Neural Network) as well as GEMM (General Matrix Multiplication), nonlinear activation, quantization, and pooling operations. The ICE core is highly programmable and integrates a strong VLIW vector Tensilica DSP as

David C. Wyld et al. (Eds): CST, NLMLT, DMS, CLBD, ITCS, VLSIE - 2022
pp. 119-126, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.121911

described in Figure 2. This allows a variety of operators that are not accelerated by the DL accelerator to be mapped to the DSP and executed in high efficiency. In addition, the ICE includes dedicated memory access blocks: a dedicated Deep Learning DMA (DSE) with dedicated features such as 4D walks, padding and stride; Compression/de-compression engine and MMU (Memory Management Unit). Each ICE has a large local SRAM of size 4MB to store the persistent data. A pair of ICE units are connected via ICEBO which allows the ICEs to share the data with each other. All the ICEs share LLC cache of size 24 MB.

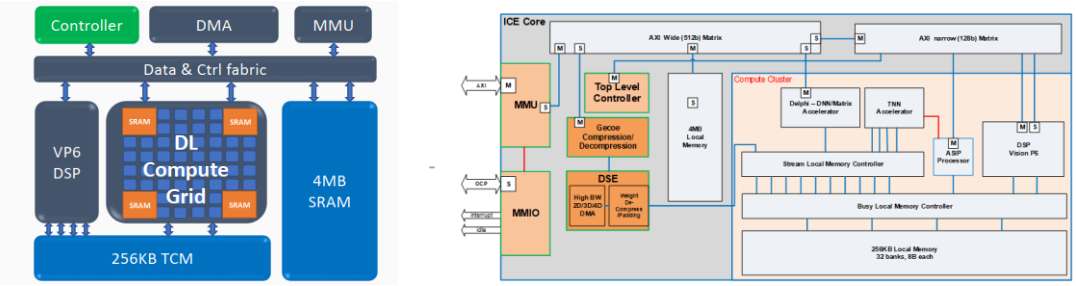

Figure 1. Inference Compute Engine (ICE)

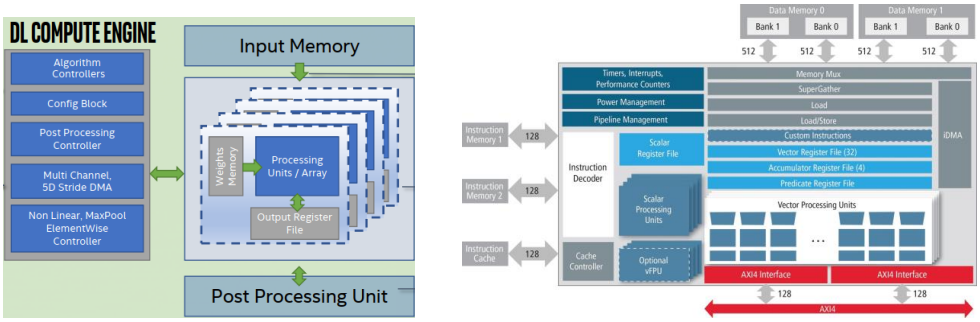

Figure 2. DL Compute Engine and Vector Processing Unit (DSP)

The NNP-I card is connected to the host processor using the PCI. The Inference applications run on the host. The model is loaded onto the card memory along with the inputs and weights. The inference outputs are transferred back to the host and subsequently to the Inference application. The inference accuracy for the same inputs for different inferences must remain the same. The inference application also collects the error statistics from the card.

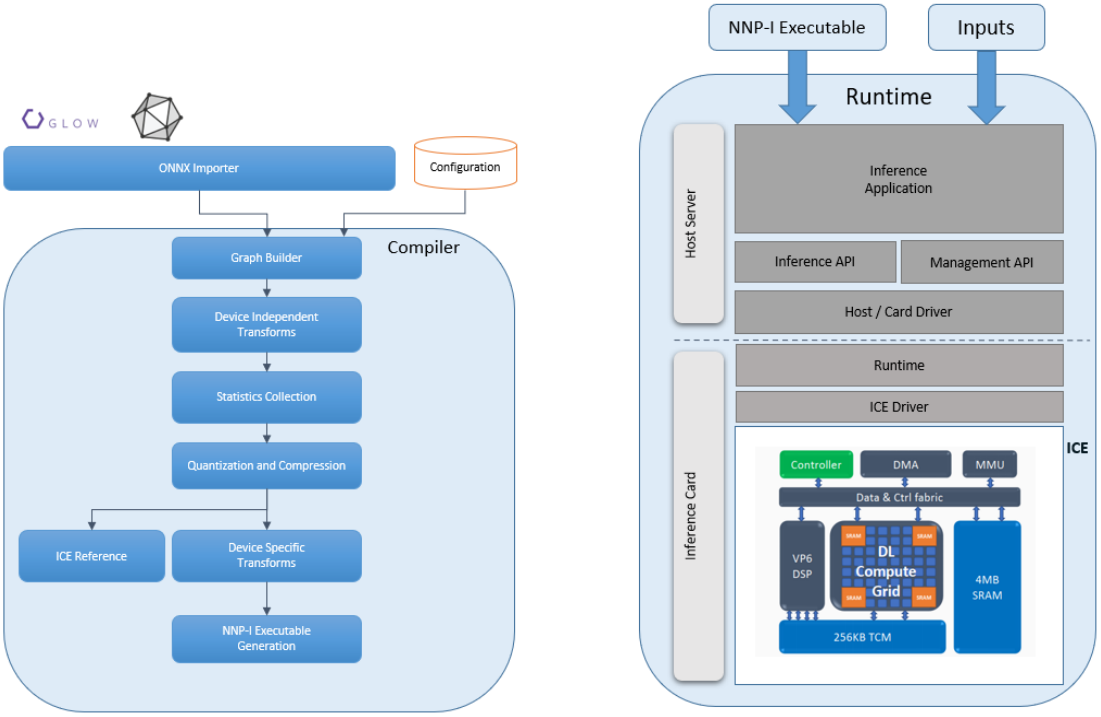

Figure 3. NNPI Software Stack

The NNPI Software stack described in Figure 3 includes the Graph Compiler, Platform/Drivers. The models are compiled using the Graph Compiler which generates the recipe. The recipe is passed to the Inference Application. The Inference Application runs on the host server and interfaces with the card using the platform drivers. The recipe gets executed on the ICEs and the final output is shared back to the Inference Application. The outputs received after the Inference are compared with the expected outputs (software reference) and the Inference efficiency/accuracy is measured.

There are different kinds of errors which can occur during the inference execution due to the hardware issues. These include Byte Mismatch (Computational) errors, Deep SRAM errors, ECC errors, PCIe AER errors, parity errors. The cards should be thoroughly screened at the factory with the help of a dedicated test content so that the faulty cards are not delivered to the customers.

The silicon units are screened at the fabrication assembly line, followed by the screening at the card manufacturer before sending it to the customer ODM. The customer ODM runs extensive regression tests before deploying these cards in the data centre. The customer also keeps on running the periodic sanity tests on the cards to check the health of these cards.

2. SCREENING PROCESS

2.1. Hardware Screener Inference Application

The Hardware Screener Application has been developed based upon the Inference APIs and Management APIs exposed by the NNPI software. The Application creates an Inference context and for each such context a Runtime process instance gets created on the device to run inferences. The Inference library parses the recipe generated by the graph compiler and then allocates

memory buffers on the host/device(s) and pass this information to the Runtime/ICE driver framework. The ICE Driver framework schedules the different networks on the different ICE units. Multiple threads are created in each context to extract the maximum utilization of the ICEs.

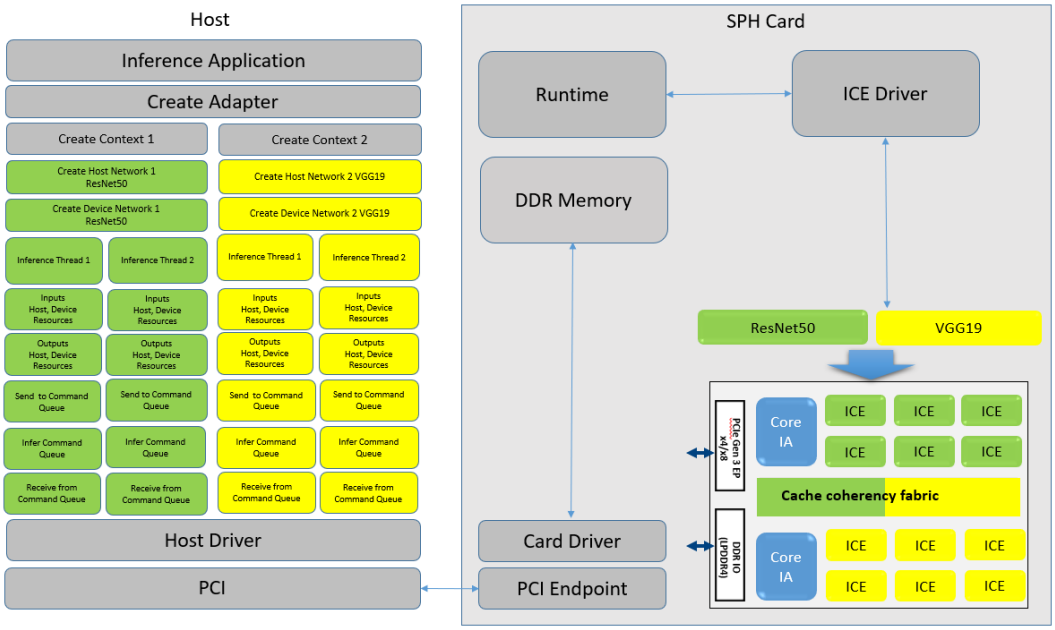

Figure 4. Hardware Screening Application

The Hardware Screening Application described in the Figure 4 is used to infer 50K images from the ImageNet dataset for different models/workloads. This application generates the IFMs (Input Feature Maps) and the OFMs (Output Feature Maps) on one hardware unit. The Inference is executed on multiple hardware units and the outputs are compared with the saved OFMs from the first hardware unit. If the outputs match, then these set of IFMs and OFMs would be used as reference and this package is deployed at the assembly line to screen the units.

The other errors like ECC, DSRAM, MCE, PCIe are identified using the Software counters incremented during the Inference execution. These software counters are being monitored by the Hardware Screener Application using the Management APIs.

The detailed screening flow is described in the Figure 5.

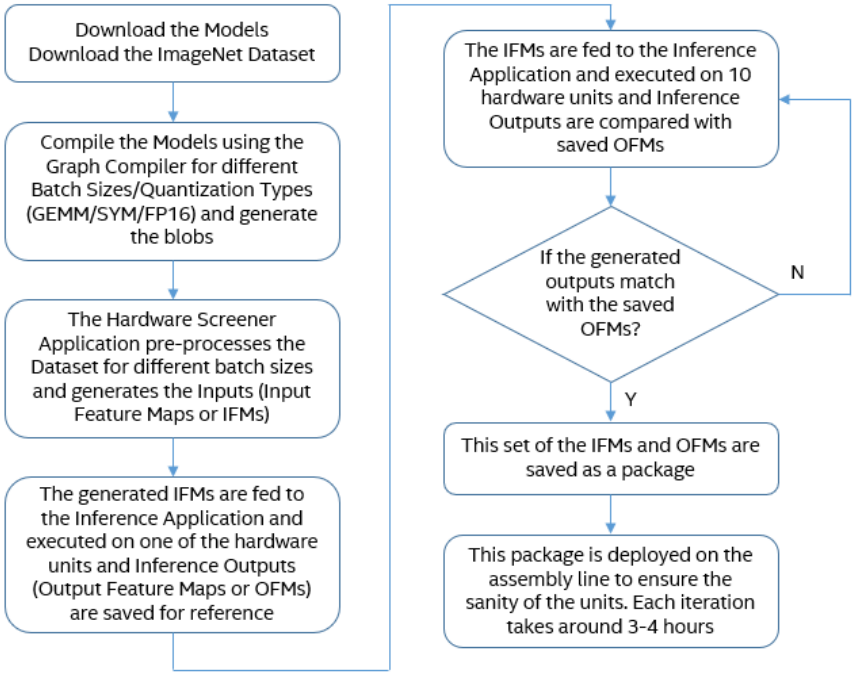

Figure 5. Flowchart of screening process

The AI workloads configurations are described in the Table 1. The workloads are generated for different Quantization Types, Batch Sizes, Number of ICEs. The other parameters include the Persistent Deep SRAM, Shared Read (ICEBO), Class of Service, ICEDC Sync (Barrier/Fence). This test content stresses upon all the different execution flows within the ICEs and helps in determining the errors.

Table 1. AI Workloads and Configurations

Workloads	Quantization Type	Batch Size	Number of ICEs	Persistent Deep SRAM	Shared Read (ICEBO)	Class of Service (CLOS)	ICEDC Sync (Barrier / Fence)	Number of Inputs	Execution Time
ResNet50	GEMMLOWP	12	12	N	Y	Y	Y	50K	30 secs
ResNet50	SYMLOWP	12	12	N	Y	Y	Y	50K	30 secs
ResNet50	GEMMLOWP	2	2x6	N	Y	Y	Y	50K	150 secs
ResNet50	SYMLOWP	2	2x6	N	Y	Y	Y	50K	150 secs
ResNet50	FP16	2	2x6	N	Y	Y	Y	50K	900 secs
ResNet50	GEMMLOWP	1	1x12	Y	N	N	N	50K	350 secs
ResNet50	SYMLOWP	1	1x12	Y	N	N	N	50K	350 secs
ResNet50	FP16	1	1x12	Y	N	Y	Y	50K	2000 secs
ResNext	GEMMLOWP	12	12	N	Y	Y	Y	50K	60 secs
ResNext	SYMLOWP	12	12	N	Y	Y	Y	50K	60 secs
ResNext	GEMMLOWP	2	2x6	N	Y	Y	Y	50K	300 secs
ResNext	SYMLOWP	2	2x6	N	Y	Y	Y	50K	300 secs
ResNext	GEMMLOWP	1	1x12	Y	N	N	N	50K	1000 secs
ResNext	SYMLOWP	1	1x12	Y	N	N	N	50K	1000 secs
ShuffleNet	GEMMLOWP	12	12	N	Y	Y	Y	50K	20 secs
ShuffleNet	SYMLOWP	12	12	N	Y	Y	Y	50K	20 secs
ShuffleNet	GEMMLOWP	2	2x6	N	Y	Y	Y	50K	120 secs
ShuffleNet	SYMLOWP	2	2x6	N	Y	Y	Y	50K	120 secs
ShuffleNet	GEMMLOWP	1	1x12	Y	N	N	N	50K	220 secs
ShuffleNet	SYMLOWP	1	1x12	Y	N	N	N	50K	220 secs
ShuffleNet	FP16	1	1x12	N	Y	Y	Y	50K	1350 secs
ResNet50	GEMMLOWP	1	1x12	Y	N	N	N	50K	350 secs
ResNet50	SYMLOWP	1	1x12	Y	N	N	N	50K	350 secs
ResNet50	SYMLOWP	12	12	Y	N	N	N	50K	30 secs
VGG19	GEMMLOWP	1	1x12	Y	N	N	N	50K	2400 secs
VGG19	SYMLOWP	12	12	Y	N	N	N	50K	180 secs

VGG19	GEMMLOWP	12	12	Y	N	N	N	50K	180 secs
ResNet50	GEMMLOWP	12	12	N	Y	Y	Y	50K	300 secs
ShuffleNet	GEMMLOWP			Y	N	N	N		
ResNet50	GEMMLOWP	2	2x3	N	Y	Y	Y	50K	300 secs
ShuffleNet	GEMMLOWP			N	Y	Y	Y		
ResNet50	GEMMLOWP	2	2x2	N	Y	Y	Y	50K	500 secs
ResNext	GEMMLOWP			N	Y	Y	Y		
ShuffleNet	GEMMLOWP			N	Y	Y	Y		
Total Time									4 hours

2.2. Hardware Screening at Assembly Lines

The hardware units are screened at different stages using the Hardware Screener Application as shown in the Figure 6.

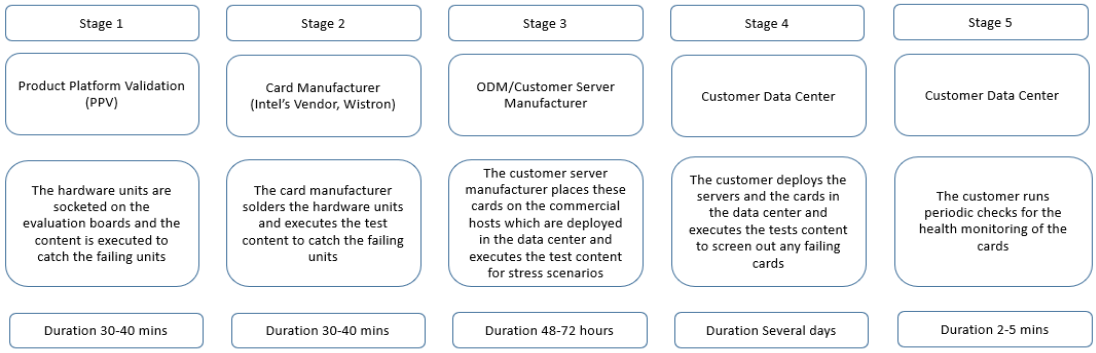

Figure 6. Stages of Deployment

The different error types which lead to failing cards are described in the Table 2.

Table 2. Error Types and their description

Error Type	Description
Byte Mismatch (Computational Errors)	The Inference output should match byte by byte with the golden reference output. If these outputs differ, then it leads to the byte mismatch errors.
ECC Errors	Error correction code memory (ECC memory) uses an error correction code (ECC) to detect and correct n-bit data corruption which occurs in memory. The ECC errors are also divided into the correctable and non-correctable errors.
Deep SRAM Errors	DSRAM arrays are the densest circuitry on a chip. The error correction logic is also added with the SRAMs to detect/correct the DSRAM errors. If the DSRAM errors are not corrected, then it leads to DSRAM single/multi bit errors.
PCI Express Advanced Error Reporting (AER)	PCI Express errors can occur on the PCI Express link itself or on behalf of transactions initiated on the link. The PCI AER errors are divided into the correctable errors and uncorrectable errors. Uncorrectable errors can cause a particular transaction or a particular PCI Express link to be unreliable.
Parity Errors / Memory Check Errors (MCE)	Parity error is an error that results from irregular changes to data, as it is recorded when it is entered in memory. Different types of parity errors can require the retransmission of data or cause serious system errors, such as system crashes.
VMin Errors	VMin errors are seen due to the voltage sensitivity at various operating frequencies of the chip. This also leads to the byte mismatch or the computational errors.

The following table shows the data for the 20 bad units which have been screened using the Hardware Screener application. The errors have been detected in the first iteration using the respective workloads.

Table 3. Units detected using Hardware Screening Application

SNo	Workload	Quantization Type	Batch Size	No of ICEs	No of errors	Failed Iteration	Error Type
1	ResNext	GEMMLOWP	2	2	2	1	Bytes Mismatch
2	ResNext	SYMLOWP	1	1	1	1	DSRAM Error
3	ResNext	SYMLOWP	1	1	35	1	Bytes Mismatch
4	ResNext	GEMMLOWP	2	2	4	1	Bytes Mismatch
5	ResNext	SYMLOWP	1	1	6	1	Bytes Mismatch
6	ResNet50	SYMLOWP	1	1	22	1	Bytes Mismatch
7	ShuffleNet	SYMLOWP	1	1	11	1	Bytes Mismatch
8	ShuffleNet	GEMMLOWP	2	2	3	1	MCE Error
9	ResNet50	GEMMLOWP	2	2	16	1	Bytes Mismatch
10	ResNet50	FP16	1	1	29	1	Bytes Mismatch
11	ResNext	SYMLOWP	1	1	12	1	Bytes Mismatch
12	ResNext	SYMLOWP	2	2	7	1	Bytes Mismatch
13	ShuffleNet	GEMMLOWP	2	12	11	1	Bytes Mismatch
14	ResNext	SYMLOWP	1	1	4	1	Bytes Mismatch
15	ResNext	SYMLOWP	1	1	1	1	ECC Error
16	ShuffleNet	GEMMLOWP	2	2	3	1	Bytes Mismatch
17	ShuffleNet	GEMMLOWP	2	2	14	1	Bytes Mismatch
18	ResNet50	FP16	1	1	36	1	Bytes Mismatch
19	ResNext	SYMLOWP	1	1	22	1	Bytes Mismatch
20	ResNext	SYMLOWP	1	1	3	1	Bytes Mismatch

3. CONCLUSIONS

The Hardware Screener Application to screen out any bad/defective units is currently deployed at pre- and post-production environments within Intel, Intel Card Manufacturers, Customer ODM Manufacturers, and Customer Data Centres.

This work has helped significantly to screen out the cards due to the following errors:

- Byte Mismatch/Computational errors
- Vmin sensitive issues fixing the voltage sensitivity at various operating frequencies
- Power/ Frequency related issues
- Memory Related errors – Deep SRAM, MRC errors
- ECC Correctable/Uncorrectable errors
- Floating point accumulator errors

The execution of this flows has also helped in the achieving the following significant targets:

- Stability of Hardware/Software - Many significant software bugs, hangs were also revealed and fixed in pursuit of executing continuous inferencing over multiple days
- DPMO Target - Defects per million opportunities target has been achieved
- Deviations in expected performance over multiple hours/days of execution

REFERENCES

[1] Quantization Algorithms: https://intellabs.github.io/distiller/algo_quantization.html

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).

[5] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.
[6] SpringHill (NNPI-1000) Intel's Data Center Inference Chipset, HotChips Conference 2019.
[7] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.
[8] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5987-5995, doi: 10.1109/CVPR.2017.634.
[9] Spring Hill - Microarchitectures - Intel - WikiChip. (n.d.). from https://en.wikichip.org/wiki/intel/microarchitectures/spring_hill
[10] Glenn Henry, Parviz Palangpour, Michael Thomson, J Scott Gardner, Bryce Arden, Jim Donahue, Kimble Houck, Jonathan Johnson, Kyle O'Brien, Scott Petersen, Benjamin Seroussi, Tyler Walker, "High-Performance Deep-Learning Coprocessor Integrated into x86 SoC with Server-Class CPUs Industrial Product", 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp.15-26, 2020

AUTHORS

Ashish Sharma (ashish3.sharma@intel.com) is an AI Software Engineering Manager at Intel Technologies, Bangalore. He holds a Masters in Computer Systems from BITS Pilani. He has been in R&D for 23 years. He leads the SW Validation and Development activities primarily for Tensorflow, PyTorch, PyLightning frameworks, Security and Linux Kernel drivers on Intel Training/Inference Accelerators. He has been leading multiple work groups within the team focusing on Leveraging Industry Practices, Standardization, Competition in AI.

Puneesh Khanna (puneesh.khanna@intel.com) is a AI Frameworks Architect at Intel Technologies, Bangalore. He holds a Masters in Machine Learning and AI from LJMU University, UK. He has been in R&D for 17 years. He works primarily on Tensorflow and Pytorch frameworks and enabling state of art deep learning models on Intel AI accelerators. He has been also contributing to the opensource code of these AI frameworks. He is leading an AI cohort group of GAR region, solving enterprise specific problems by leveraging AI and ML at Intel.

Jaimin Maniyar (jaimin.maniyar@intel.com) is a AI Solutions Engineer at Intel Technologies, Bangalore. He holds a Masters in Machine Learning and AI from Vellore Institute of Technology. He has been in R&D for 5 years. He works primarily on Tensorflow and Pytorch frameworks and enabling state of art deep learning models on Intel AI accelerators. He has delivered many AI and ML Sessions at Intel collaborating with Dataa Centre Skills Academy.