Exploring the Vision Processing Unit as Co-processor for Inference

Sergio Rivas-Gomez[1], Antonio J. Peña[2], David Moloney[3], Erwin Laure[1], and Stefano Markidis[1]
[1]*KTH Royal Institute of Technology*
[2]*Barcelona Supercomputing Center (BSC)*
[3]*Intel Ireland Ltd.*

***Abstract*—The success of the exascale supercomputer is largely debated to remain dependent on novel breakthroughs in technology that effectively reduce the power consumption and thermal dissipation requirements. In this work, we consider the integration of co-processors in high-performance computing (HPC) to enable low-power, seamless computation offloading of certain operations. In particular, we explore the so-called Vision Processing Unit (VPU), a highly-parallel vector processor with a power envelope of less than 1W. We evaluate this chip during inference using a pre-trained GoogLeNet convolutional network model and a large image dataset from the ImageNet ILSVRC challenge. Preliminary results indicate that a multi-VPU configuration provides similar performance compared to reference CPU and GPU implementations, while reducing the thermal-design power (TDP) up to 8× in comparison.**

***Keywords*-Vision Processing Unit; High-Performance Computing; Machine Learning**

I. Introduction

The recent advances in deep learning and convolutional networks, have dramatically influenced the role of machine learning on a wide-range of scientific applications [1], [2]. This fact has been motivated by an increase in object classification and detection accuracy [3], [4], alongside with better tools for data mining that allow us to understand large datasets of unstructured information [5], [6]. The inference error rate of machine learning algorithms has become remarkably low as well, reaching a state where the capacity of humans has been already surpassed in certain scenarios [7].

As a consequence, there is an existing trend that proposes the integration of data-centric models on HPC that combines specialized hardware with the aim of fulfilling this need [8]. Upcoming major supercomputers are expected to feature new hardware architectures that provide high-performance 16-bit / 32-bit mixed arithmetic support for machine learning [9], both during training and inference. In addition, innovation at software level is also observed with the appearance of novel data formats that use tensors with a shared exponent [10], [11], maximizing the dynamic range of the traditional 16-bit floating point data format. These breakthroughs provide multiple advantages in terms of performance and power consumption. Specifically, some of the aforementioned architectural changes are expected to increase the performance 5–10× in comparison with current large-scale HPC clusters, using just twice the power [12]. Hence, it will be of paramount importance for the success of the exascale supercomputer that we consider the embracement of these developments in the near-term future.

In this work, we set the initial steps towards the integration of low-power co-processors on HPC. In particular, we analyze the so-called *Vision Processing Unit* (VPU). This type of processor emerges as a category of chips that aim to provide ultra-low power capabilities, without compromising performance. For this purpose, we explore the possibilities of the Movidius Myriad 2 VPU [13], [14] during inference in convolutional networks, over a large image dataset from the ImageNet ILSVRC 2012 challenge [15]. In our evaluations, we use a pre-trained network from the Berkeley Vision and Learning Center (BVLC), which follows the *GoogLeNet* work by Szegedy et al. [3]. Preliminary results indicate that a combination of several of these chips can potentially provide equivalent performance compared to a reference CPU and GPU implementation, while reducing the thermal-design power (TDP) up to 8×. The observed throughput, measured as number of inferences per Watt, is over 3× higher in comparison. The estimated top-1 error rate is 32% on average, with a confidence error difference of 0.5%.

The contributions of this work are the following:

- We provide a comprehensive technical overview of the Myriad 2 VPU in the context of the Intel Neural Compute Stick (NCS) platform [16].
- We design and implement a small inference framework based on Caffe [17] and the Neural Compute API [18] to support our experiments on the VPU.
- We illustrate that VPUs feature an excellent ratio between throughput and power consumption compared to reference CPU and GPU implementations, including in multi-VPU configurations.
- We compare the top-1 error rate [3] with a reference CPU implementation to understand the implications of using FP16 on the VPU.

The paper is organized as follows. We provide a high-level overview of the VPU in Section II. We describe the implementation considerations of a small inference framework in Section III. The experimental setup and performance evaluation is presented in Section IV. We extend the discussion of the results and provide further insights in Section V. Related work is reported in Section VI. A summary of our conclusions and future work is outlined in Section VII.

II. Background

The emergence of machine learning and data-centric applications on HPC poses several constraints on general-purpose processors, mainly due to the irregularity of the memory accesses that they feature [19], [20]. These accesses have reduced temporal or spatial locality, incurring in long memory stalls and large bandwidth requirements. As a side effect, the power consumption and thermal dissipation requirements considerably increase as well [21]. Thus, during the last decade, scientists have experimented with the integration of novel algorithms that perform dynamic, in-memory data rearrangements of irregular structures [22], [23]. The aim is to overcome (or partially hide) some of the aforementioned limitations.

Nonetheless, the inherent complexity of such techniques, coupled with the adoption of the "CPU + Accelerator" model to enhance the performance of scientific applications [24], makes programming general-purpose processors another key-factor to consider. In addition, transferring data among these different hardware layers can also become costly [25]. As a consequence, the industry is shifting towards designing processors where cost, power, and thermal dissipation are key concerns [14]. Specialized co-processors have recently emerged with the purpose of reducing the power envelope constraints, while improving the overall performance on scenarios such as machine learning [26]. In this regard, we observe that other scientific fields can benefit from this trend by adopting part of these technologies. In fact, energy consumption in HPC is considered one of the main limiting factors towards the exascale supercomputer [27].

In this section, we briefly describe the most relevant technical aspects of the Movidius Myriad 2 VPU [13], [14], in the context of the Intel Neural Compute Stick (NCS) platform [16]. Our goal is to understand how this type of low-power co-processors could potentially be integrated for computation offloading on HPC.

A. Vision Processing Unit

The Myriad 2 VPU is designed as a 28-nm co-processor that provides high-performance tensor acceleration. The chip dissipates less than 1W [13]. High-level APIs allow application programmers to easily take advantage of its features and, thus, enhance programming productivity. In addition, the software-controlled memory subsystem enables fine-grained control on different workloads, if required. The term "vision" is employed due to the original purpose of the VPU, which was meant to accelerate computer vision applications on the "edge" [28].

The architecture of this chip is inspired by Agarwal's observation, which states that beyond a certain frequency limit for any particular design and target process technology, the cost is quadratic in power for linear increases in operating frequency [14]. Following this statement, the Myriad 2 VPU

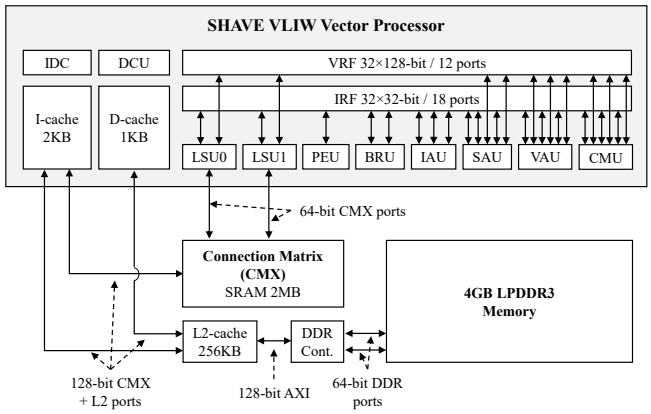


Figure 1: High-level representation of one of the SHAVE vector processors featured on the Myriad 2 VPU [14]. The Connection Matrix (CMX) enables seamless interaction between the vector processors and other hardware components.

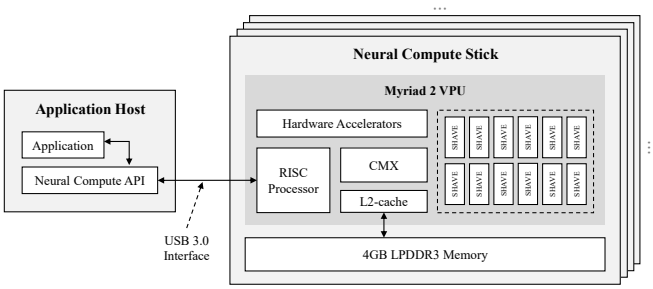


Figure 2: Approximate implementation of the Myriad 2 VPU used within the Neural Compute Stick (NCS) platform [16]. The Neural Compute API allows us to coordinate the execution on the VPU of one or more NCS devices [18].

is designed featuring 12 highly-parallelizable vector processors, named Streaming Hybrid Architecture Vector Engines (SHAVE). Each SHAVE processor contains wide register files and several functional units. These are controlled by Variable-Length Long Instruction Word (VLLIW) packets. Hence, enabling seamless SIMD operations on the chip. The nominal frequency is 600MHz.

Figure 1 illustrates a high-level diagram of one of the SHAVE processors and the interactions with other components of the Myriad 2 VPU. The main vector register file (VRF) has 128-bit $\times$ 32 entries and 12 ports. A general register file (IRF) is also available with 32-bit $\times$ 32 entries and 18 ports. Among the functional units of each SHAVE processor, we highlight the 128-bit Vector Arithmetic Unit (VAU), the 128-bit Compare-and-Move Unit (CMU), the 32-bit Scalar Arithmetic Unit (SAU), and the 32-bit Integer Arithmetic Unit (IAU). The chip supports 8, 16, 32, and partially 64-bit integer operations, as well as native FP16

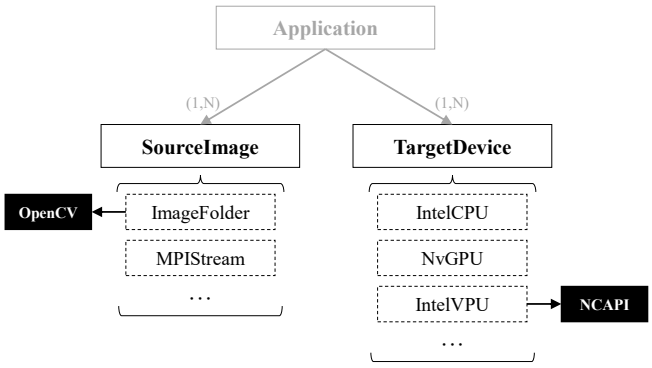


Figure 3: Class diagram specification of the NCSw framework. The simple modular design allows us to provide implementations for new kind of devices (e.g., FPGA).

```
1  ...
2  // Load the graph with the input image
3  mvncLoadTensor(graph, (half *)img, size, NULL);
4
5  /****************************************
6   * Perform other overlapping computations *
7   ****************************************/
8
9  // Retrieve the inference result from the NCS
10 mvncGetResult(graph, (half **)&result,
11                &result_size, &userParam);
12 ...
```

Listing 1: Source code example in C that illustrates how to conduct inference on the NCS using the NCAPI [18].

and FP32 arithmetic[1]. Each of these functional units can be operated independently through the VVLIW instruction packets. In addition, two 64-bit Load-and-Store Units (LSU) enable data transferring among the SHAVE processors through a shared, multi-ported 2MB memory block, named Connection Matrix (CMX). The CMX features 16 blocks of 128KB, comprising four 32KB RAM instances organized as 4096 words of 64-bits each, independently arbitrated. The variant used in our tests (MA2450) features a global stacked memory of 4GB LPDDR3. The memory fabric of the Myriad 2 VPU is designed for low-latency by endorsing data locality. It is also mostly software-controlled for flexibility purposes, as previously stated. This allows the VPU to support different kinds of application workloads.

Alongside the SHAVE vector processors, the chip features a Streaming Image Processing Pipeline (SIPP), which contains fully programmable hardware-accelerated kernels of common image processing operations [13]. For instance, some of the kernels include tone-mapping, Harris Corner detector, Histogram of Oriented Gradients (HoG) edge operator, luminance / chrominance denoising, and others. The typical configuration for the kernels is 5×5 per target output pixel. Each hardware-accelerated kernel is connected to the CMX memory block using a crossbar. A local controller on each SIPP filter manages the read / writeback of the results to the CMX. Thus, combining operations on the SHAVE vector processors and the hardware-accelerated kernels is feasible. The filters can output completely computed pixels individually per cycle.

B. Neural Compute Stick Platform

The Intel Neural Compute Stick (NCS) platform [16] is a System-on-Chip (SoC) implementation of the Myriad 2 VPU. A high-level overview of the device is illustrated in Figure 2. The diagram depicts the approximate implementation used in the NCS platform (variant MA2450). The NCS employs a total of 20 power islands, including one for each of the 12 integrated SHAVE processors. This is critical to effectively manage the power consumption of the SoC. Two RISC processors manage the communication with the host and the execution on the VPU (i.e., runtime scheduler). They are also in charge of the peripherals in other implementations (e.g., MIPI D-PHY) and running a Unix-based real-time OS (RTOS). In the diagram, applications communicate with the VPU using a USB 3.0 interface and the so-called Neural Compute API (NCAPI) [18]. The main purpose of this API is to enable the deployment of convolutional networks for inference on the NCS[2]. When the NCAPI initializes and opens a device, a firmware is loaded onto the NCS. At this point, the device is ready to accept the network graph files and execute commands to conduct inference on the VPU.

The NCAPI comprehends a set of operations that allow applications to connect to the NCS, deploy a pre-trained convolutional network model, obtain performance metrics per layer, and more. The programming interface is available in C/C++ and Python. For instance, in order to perform inference on the device, the API follows a set of operations that resemble the MPI non-blocking interface [30]. In this case, instead of having a single, blocking "`inference()`" function, the step is divided in two separate operations. First, a *load* operation transfers the input and prepares the NCS for execution. Thereafter, a *wait* operation blocks the process on the host until the execution on the NCS has finished. Hence, this model enables the design of decoupled strategies that overlap computations while inference has been offloaded to the NCS. In most cases, by the time that the host process has to wait, the inference is already completed and the result can be retrieved.

Listing 1 provides a source code example in C where the NCAPI is utilized to perform inference on the VPU. Error-checking is excluded for illustration purposes. In this

[1]The maximum theoretical performance claimed by the manufacturer is 1000 Gflops using FP16 arithmetic [14].

[2]Training these networks, however, is accomplished outside the scope of the NCS using the regular Caffe [17] or Tensorflow [29] frameworks (i.e., the device is only used for inference).

example, the `mvncLoadTensor()` transfers a certain input image to the NCS device and loads the pre-compiled graph for execution. This will automatically coordinate the data transfer with one of the RISC processors into the NCS. It will also immediately queue the execution of the graph on the SHAVE processors through the runtime scheduler. The operation will return as soon as the data is transferred and the execution is scheduled, without blocking the host process. At this point, the application is able to overlap additional computations while the inference has been offloaded to the NCS (e.g., decode the next frame). Multi-device is also supported, meaning that we could easily offload more inference operations to other devices. When the result is required, a call to `mvncGetResult()` will guarantee that the host process is blocked until the inference has finished and the result is ready. The output result is a list of labels with the correspondent confidence, based on the input.

Note that fine-grained general-purpose computing using C/C++ is also possible through the Movidus Development Kit (MDK) [26], [31]. The MDK enables OpenCL support and provides several optimized libraries designed for the Myriad 2 VPU chip (e.g., LAMA, a linear algebra library). Tools for debugging and profiling are also available. We consider exploring the possibilities of using this development kit for general-purpose computing in future work.

III. Inference Framework

We design and implement a very simple inference framework using C/C++ that supports diverse types of target devices. The framework, named Neural Compute Stick Wrapper (NCSw), is mostly based on the use of Caffe [17] in the context of the NCS platform. In addition, we integrate the specific Caffe project forks optimized for Intel processors and NVIDIA graphics cards to conduct our experiments. This allows us to compare the inference performance with the VPU chip. The source code is available on a public Git repository[3].

The NCSw framework is divided in several abstract classes that represent the *source* of the input datasets and the *target* (or where) to conduct inference (Figure 3). The aim is to provide an easy-to-use implementation that could enable the integration of new kinds of input sources (e.g., MPI streams [32]) or target devices (e.g., FPGA) in the future. The VPU implementation is based on the use of the NCAPI. We use OpenEXR [33] half-precision class for converting the pixel data from FP32 to FP16. This is the compatible format for the Myriad 2 VPU [14].

Batch-processing is supported by defining a parallel, multi-VPU implementation. This approach differs from the traditional Caffe batched execution, which resizes the input blob layer of the convolutional network to achieve better data communication throughput (e.g., on GPUs). In this case, we schedule simultaneous inferences using the same graph on multiple NCS devices. The main host process is responsible for connecting to each device and offloading the execution. By default, if the NCSw framework is compiled with OpenMP support, the multi-VPU implementation will become multi-threaded. Hence, the host process will spawn multiple threads to handle the execution on each NCS device available. The threads will concurrently transfer the source input and retrieve the output, thus, effectively overlapping the communication with the RISC processor on the SoC and maximizing the bandwidth utilization.

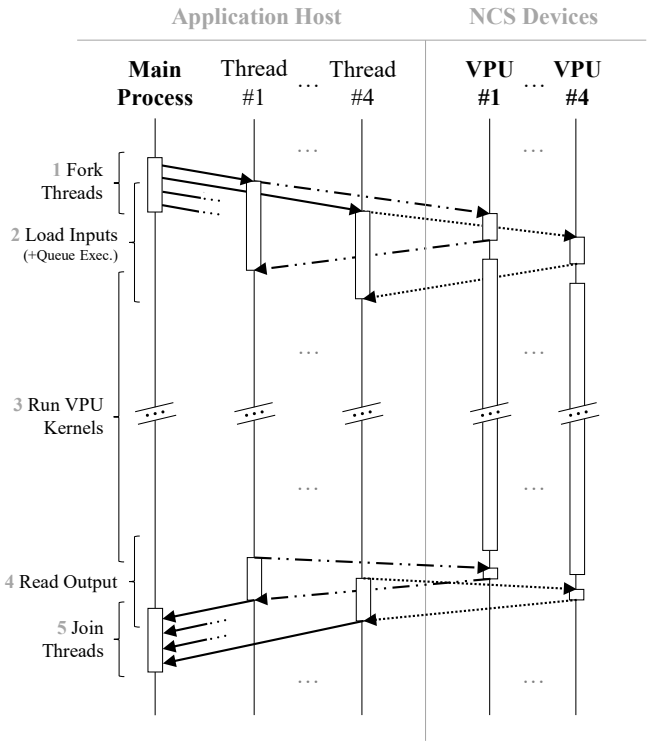


Figure 4: Execution timeline of the parallel, multi-VPU implementation of NCSw. In this example, the host process offloads the execution to four threads, one per NCS device.

Figure 4 illustrates an example timeline for the parallel, multi-VPU implementation using four different NCS devices. Here, the host process begins by spawning one thread per VPU. These threads will then load different inputs into the global LPDDR3 memory of each NCS. This fact will guarantee that, while the next input is being loaded on the succeeding device, the runtime scheduler in the preceding device has started the execution on the SHAVE processors and SIPP hardware-accelerated filters. Thereafter, the results are retrieved in the queueing order to guarantee an overlap with the rest of the NCS devices. We follow a simple static scheduling (i.e., round-robin) for this purpose.

Applications can decide whether to use one or more VPUs

[3]https://github.com/sergiorg-kth/ncs-wrapper

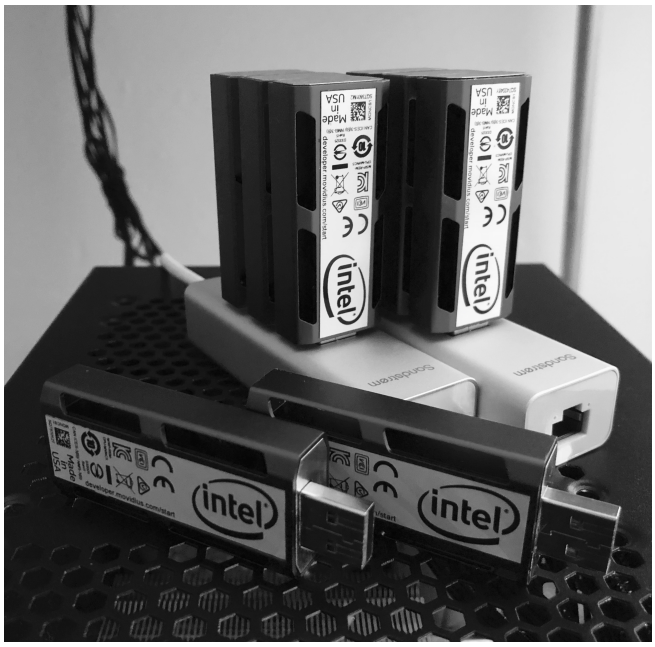


Figure 5: Our testbed contains 8 different NCS devices, where 6 devices are connected using two USB 3.0 HUBs and 2 devices are connected using the ports of the motherboard.

simultaneously, or to define groups of the same target type. In other words, different sources can be easily connected to the same or multiple targets. Therefore, some applications might choose to run a specific subset of inputs on a GPU, and at the same time another subset on two different groups that connect to several VPUs using the described approach.

IV. Experimental Results

In this section, we analyze three implementations inside the NCSw framework that target a CPU, a GPU, and a multi-VPU configuration, respectively. We evaluate these implementations in terms of inference performance and confidence error. For this purpose, we use the Intel-optimized Caffe-MKL fork (v1.0.7) for Intel processors, the NVIDIA-optimized Caffe-cuDNN fork (v0.16.4) for NVIDIA graphic cards, and the Neural Compute SDK (v1.12.00.01) for the Myriad 2 VPU on the NCS. Thus, we aim to take advantage of the compute capabilities of each of these devices using reference implementations provided by the manufacturers.

The simulations are conducted in a workstation with two four-core Intel Xeon E5-2609v2 processors running at 2.5GHz. The workstation is equipped with a total of 72GB DRAM. The graphics card is a Quadro K4000, with 3GB of GDDR5 and 768 CUDA cores. The NVIDIA driver version is v384.81. The storage consists of two 4TB HDD (WDC WD4000F9YZ / non-RAID) and a 250GB SSD (Samsung 850 EVO). The OS is Ubuntu Server 16.04.1 LTS with Kernel 4.4.0-62-generic. The NCSw framework is compiled with `gcc` v5.4.0 and linked with OpenCV v2.4.9.1 to decode the input images. We compile Caffe with Intel MKL v2018.1.163. For the GPU implementation, we use CUDA v9.0, cuDNN v7.0.5.15-1, and NCCL v1.3.4-1.

Note that all the figures reflect the standard deviation of the samples as error bars. In addition, we omit from our results the decoding time per image, but account for the data transferring time from the host to the target device. We also enable OpenMP to support multi-threading on the multi-VPU configuration. In this case, a maximum of 8 simultaneous NCS devices are employed, where 6 devices are connected using two USB 3.0 HUBs (Sandstrøm 164903) and 2 devices are connected using directly the USB 3.0 ports of the motherboard (Figure 5). Lastly, we use the traditional Caffe batch-based processing on the CPU and GPU tests.

A. Performance Evaluation

With the purpose of evaluating the image classification performance of the three aforementioned implementations, we use one of the reference datasets from the ImageNet database [15]. This project is an on-going research effort that aims to provide researchers around the world with an easily accessible, large image database organized according to the WordNet hierarchy [34]. Each meaningful concept in ImageNet is described by multiple word phrases (i.e., "synonym set" or "synset"), and contains on average 1000 images to illustrate its definition.

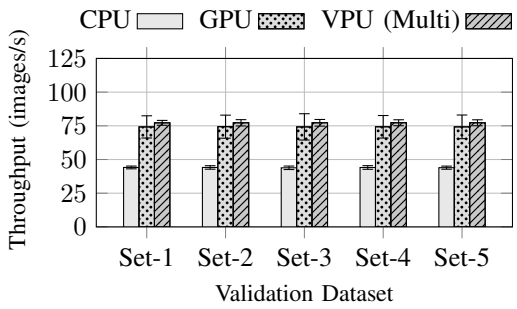
The success of ImageNet is largely due to the Large Scale Visual Recognition Challenge (ILSVRC). This challenge is a benchmark in object category classification and detection on hundreds of object categories and millions of images. Since its inception in 2010, ILSVRC has become the de-facto standard benchmark for large-scale object recognition [35]. The publicly released dataset contains a set of manually annotated training and test images.

In this regard, we use the *Validation* dataset from the ILSVRC 2012 challenge[4] to conduct our experiments. This dataset contains 50000 images in total. Each target device in our implementation uses the pre-trained BAIR GoogLeNet network[5] from the Berkeley Vision and Learning Center (BVLC). This network is trained specifically for the ILSVRC 2012 challenge, as described by Szegedy et al. [3]. The input geometry of the network is 224x224. The mean values are retrieved directly from the ILSVRC 2012 training dataset. Finally, the Caffe engine is set to "MKL2017" for the CPU-based implementation.

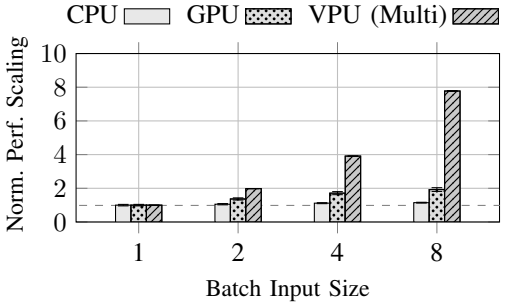
Using a multi-VPU configuration, we determine that the Myriad 2 VPU provides a very well-balanced ratio between performance and power consumption. Figure 6a reports the throughput in images per second ($img \cdot s^{-1}$) for the CPU, GPU, and multi-VPU configurations. We use batch-processing mode with 8 inputs to match the number of

[4]http://image-net.org/challenges/LSVRC/2012

[5]http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel

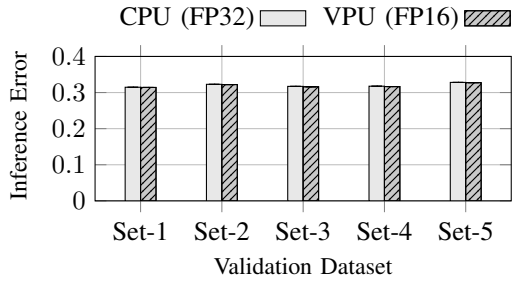


(a) **Inference Performance** per subset / 8×Input (batch)

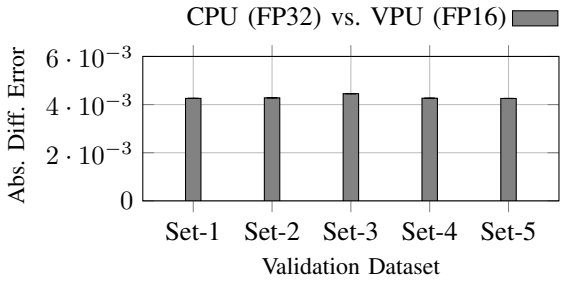


(b) **Relative Inference Performance** per batch size

Figure 6: (a) Inference performance (per subset) of the ILSVRC 2012 Validation dataset using batch-mode on the CPU, GPU, and multi-VPU configurations. (b) Normalized performance scaling per batch size relative to the baseline of a single input for each device type. Note that, in both figures, the number of active VPU chips is coupled with the input size (1-8).



(a) **Top-1 Inference Error** per subset



(b) **Confidence Difference** per subset

Figure 7: (a) Top-1 prediction error (per subset) of the ILSVRC 2012 Validation dataset using the CPU (FP32) and VPU (FP16) implementations. (b) Absolute confidence difference error after filtering the top-1 miss-predictions between the CPU and VPU implementations.

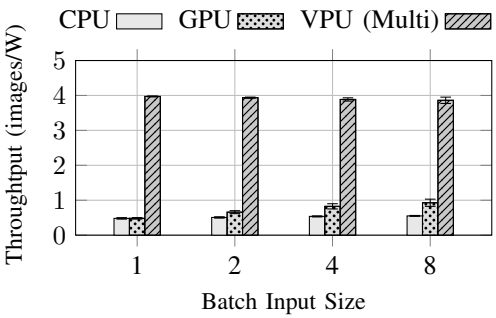
simultaneous VPUs available in our testbed (i.e., eight NCS devices). For evaluation purposes, we divide the complete validation dataset in groups of 10000 images, forming 5 subsets in total. From this figure, we can determine that the throughput using eight Myriad 2 VPU chips is approximately 77.2 img·s^{-1} (12.9ms per inference). The optimized Caffe framework on the CPU is 40.7% slower, with an average of 44.0 img·s^{-1} (22.7ms per inference). However, the GPU-based implementation produces similar results, with a throughput of 74.2 img·s^{-1} on average per subset (13.5ms per inference).

If we compare the performance scalability of each implementation, we observe an almost ideal scaling when increasing the number of active VPU chips. Figure 6b illustrates the relative performance scaling during inference by varying the batch input size on the CPU, GPU, and multi-VPU configurations. The figure reflects how well each implementation scales independently. Hence, the values are normalized per device type using their respective single-input test as reference for the normalization (i.e., 26.0ms for the CPU, 25.9ms for the GPU, and 100.7ms for the VPU). We use only one of the subsets of 10000 images from the validation dataset. In this case, we determine that the execution time required per inference is approximately reduced 50% when duplicating the number of active VPU chips, reaching a performance increase factor of close to 8× for the last case. This matches the number of NCS devices. Nonetheless, a small penalty is observed due to the thread-management overhead and the data transferring involved. On the other hand, the performance of the CPU implementation is barely affected, with an improvement of only 14.7% for the last case (1.1×). Similar results are observed for the GPU implementation, which improves only 92.5% for the last case (1.9×). Thus, both implementations reflect relatively poor scaling in comparison.

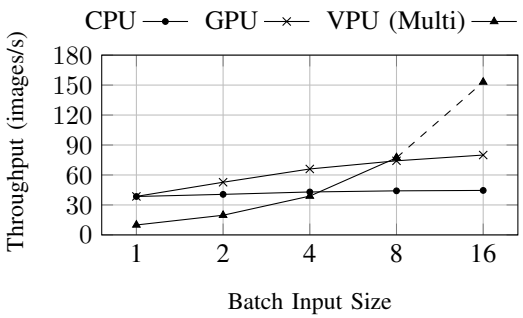
B. Error Rate Comparison

Using the same Validation dataset from the ILSVRC 2012 challenge, we evaluate the confidence accuracy of each implementation. Our goal is to understand how the differences in floating point precision can affect the predictions from the pre-trained BAIR GoogLeNet network model.

We estimate the miss-prediction rate by extracting the labels from the *Validation Bounding Box Annotations* dataset of the ILSVRC 2012 challenge. For the inference error rate, we use a *top-1 estimation*, as Szegedy et al. [3] describe.



(a) **Throughput-TDP Comparison** per batch size



(b) **Projected Inference Performance** per batch size

Figure 8: (a) Throughput performance comparison per Watt using the CPU, GPU, and multi-VPU configurations. (b) Inference performance per batch size on a subset of the ILSVRC 2012 Validation dataset using the CPU, GPU, and multi-VPU configurations. The dashed lines represent the projected value of the multi-VPU configuration if the scaling continues. Note that, in both figures, the number of active VPU chips is coupled with the input size (1-16).

This estimation implies to accept only those predictions whose correct label has the highest confidence. In addition, we compare the absolute confidence difference between the CPU implementation[6], which uses FP32 precision, and the VPU implementation, which uses FP16. This value reflects the average error after filtering the incorrect predictions according to the top-1 estimation.

With subtle inference error differences, we observe that the use of FP16 arithmetic on the Myriad 2 VPU does not have a major impact in the overall miss-prediction rate. Figure 7a illustrates the top-1 inference error per subset using the CPU and VPU implementations. Once again, we use the validation dataset with 50000 images, and divide it in groups of 10000 subsets. From the figure, we estimate that the top-1 inference error is 31.92% on average using the VPU. Surprisingly, the reference CPU implementation features a slightly worse error of 32.01%. As a result, given that the top-1 inference error using the VPU implementation with FP16 arithmetic only varies 0.09% in comparison, we confirm negligible differences due to arithmetic precision.

Looking at the absolute confidence error, we estimate once again that the use of FP16 arithmetic on the Myriad 2 VPU does not considerably affect the network output. Figure 7b depicts the absolute confidence difference per subset using the VPU implementation in comparison with the CPU implementation, after filtering the top-1 miss-predictions. In this case, the average difference per subset is estimated at 0.44% on average.

V. Discussion

The previous results indicate that the use of VPUs can be beneficial for certain operations, such as tensor processing. Even though we observe that the execution time per inference using one chip is 4× slower compared to a reference CPU / GPU implementation, we demonstrate equivalent performance results by using a parallel, multi-VPU configuration with eight NCS devices. Yet, we have not accounted for the power consumption required on each case. In fact, the estimated thermal-design power (TDP) for both the Intel Xeon E5-2609v2 and the NVIDIA Quadro K4000 GPU used in our experiments is 80W. In comparison, the TDP of the Myriad 2 VPU is 0.9W, with an overall estimated peak consumption of 2.5W for the NCS device [36], [37]. If we assume that the maximum power consumption was required[7], we can estimate a throughput function per Watt based on the number of inferences conducted per second:

$$Throughput_{Watt} = \frac{Images \cdot Second^{-1}}{TDP} \quad (1)$$

By following this metric, we confirm that, in theory, VPUs could provide a throughput per Watt of over 3× higher in comparison. Figure 8a reflects the performance measured as images per Watt (img·W^{-1}) for the CPU, GPU and multi-VPU configurations. From this figure, we observe that the throughput is 3.97 img·W^{-1} when using one VPU. Increasing the number of simultaneous VPU chips does not largely affect this ratio, except for a small performance penalty due to the required data transfers. The CPU features a theoretical throughput of 0.55 img·W^{-1} in the last case. The GPU shows similar results, with 0.93 img·W^{-1}. Nonetheless, actual power measurements would be required in future work to understand the practical differences (i.e., the TDP can be far from the real power draws per device).

On the other hand, if we assume that the ideal scaling is maintained as we increase the number of VPU chips, we could obtain power and thermal dissipation benefits while still improving the average execution time required

[6]Even though the GPU implementation is excluded from the comparison, we confirm that it provides equivalent confidence results.

[7]Technically, the CPU and other components are necessary to connect to the NCS (e.g., USB controller), which are not included in the estimation. Here, we only account for the operational TDP of each device.

per inference. Figure 8b reflects this comparison using the CPU, GPU, and multi-VPU configurations. We vary the batch size from 1 to 16 inputs on the CPU and GPU implementations. In the case of the multi-VPU, we show the projected execution time after the number of NCS devices available is exceeded (i.e., eight devices). From this figure, we determine that the CPU and GPU implementations do not illustrate relevant performance improvements, with a maximum of 44.5 img·s^{-1} and 79.9 img·s^{-1}, respectively. The Myriad 2 VPU, however, has a projected throughput of 153.0 img·s^{-1} using 16 VPU chips. This is a factor of 3.4× improvement over the CPU implementation, and a factor of 1.9× over the GPU version.

Despite these positive observations, we still consider that VPUs should complement the utilization of more powerful CPU and GPU architectures. For instance, it has been largely demonstrated that GPUs can be ideal for deep learning [38], [39]. Moreover, recent architectures, such as the NVIDIA Volta V100 [40] or the Intel Nervana Neural Network Processor [41], have been specifically designed for training and inference. Consequently, we foresee the potential of integrating the high-performance vector architecture featured on the Myriad 2 VPU in the form-factor of a co-processor to reduce the overall power consumption of future HPC clusters. Energy consumption is considered one of the main limiting factors towards the exascale supercomputer [27], as we have previously motivated. In such case, one or several of these co-processors could be included on each node. Scientific applications could then use the VPU chips to offload certain operations that involve tensor computation, avoiding the utilization of the CPU (or GPU) on less-critical tasks. We consider to explore this path in the future.

VI. Related Work

The adoption of power-efficient co-processors for computer vision and machine learning on the "edge" has been widely studied for robotics and the Internet-of-Things (IoT). For instance, Georgiev et al. [42] present an integrated sensing system that uses low-power DSP co-processors of commodity mobile devices to perform complex audio inferences. More specifically, Dexmont et al. [37] conduct a study of the Myriad 2 VPU for low-power robotics applications.

In the context of HPC, the use of co-processors is also frequent, specially with the emergence of the "CPU + Accelerator" model in this field [24]. Byun et al. [43] study the Intel Xeon Phi architecture [44] as co-processor for machine learning applications. Tan et al. [45], on the other hand, propose the use of FPGAs as co-processor to accelerate Next-Generation Sequencing (NGS) applications. Notwithstanding, we observe that the integration of low-power co-processors for computation offloading is, in most cases, not considered.

Lastly, we note that the work by Ionica et al. [26] shares some similarities. Here, the authors provide a comprehensive overview of the Myriad 1 VPU chip for scientific computing. In this regard, an implementation of a custom DGEMM operation that uses CMX tiling is provided, and performance results in terms of Gflops and Gflops·W^{-1} (estimated through the TDP) are illustrated as well. While their work focuses on the opportunities that the Myriad 1 VPU chip brings for general-purpose computing, we present the technical aspects of the Myriad 2 VPU and illustrate performance results during inference on convolutional networks using multiple chips. Thus, we consider both works complementary.

VII. Conclusion

The emergence of machine learning and data-centric applications on HPC poses several constraints on general-purpose processors [19], [20]. As such, power consumption and thermal dissipation become major concerns. In this work, we have provided an overview of the Vision Processing Unit (VPU) as co-processor for inference on HPC. In particular, we have explored the most relevant technical details of the Myriad 2 VPU [13], [14], in the context of the Intel Neural Compute Stick (NCS) platform [16]. To support our experiments, we have also presented a small inference framework, named NCSw. This framework contains a parallel, multi-VPU implementation that efficiently coordinates the execution on more than one NCS device.

Using a pre-trained network model based on the GoogLeNet work by Szegedy et al. [3], we have observed that the performance during inference on a single VPU chip is only 4× slower in comparison with reference CPU and GPU implementations. By employing a multi-VPU configuration, however, we have demonstrated equivalent performance results. Yet, the expected thermal-design power (TDP) can still be reduced by a factor of 8×. Moreover, we have confirmed negligible confidence differences by estimating the top-1 error rate, despite requiring FP16 arithmetic precision on the VPU for performance reasons.

As future work, we expect to conduct a thorough study of the possibilities of the Myriad 2 VPU as co-processor for task offloading on HPC. This would imply extending our work and integrating the VPU chip as a conventional vector processor for general-purpose computing. In addition, we expect to compare the VPU with highly-specialized accelerator chips, such as the NVIDIA Volta V100 architecture [40]. This would give us a better understanding of the performance and power consumption benefits in contrast with recent, novel architectures designed for machine learning.

Acknowledgment

The experimental results were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at PDC Centre for High-Performance Computing (PDC-HPC). The work was funded by the European Commission through the SAGE project (Grant agreement no. 671500 / http://www.sagestorage.eu).

REFERENCES

[1] H. Brink, J. W. Richards, D. Poznanski, J. S. Bloom, J. Rice, S. Negahban, and M. Wainwright, "Using machine learning for discovery in synoptic survey imaging data," *Monthly Notices of the Royal Astronomical Society*, vol. 435, no. 2, pp. 1047–1060, 2013.

[2] S. E. Thompson, F. Mullally, J. Coughlin, J. L. Christiansen, C. E. Henze, M. R. Haas, and C. J. Burke, "A machine learning technique to identify transit shaped signals," *The Astrophysical Journal*, vol. 812, no. 1, p. 46, 2015.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 26, no. 1, pp. 97–107, 2014.

[6] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, 2015, pp. 448–456.

[8] SAGE Consortium, "The SAGE project: Data storage for extreme scale," http://bit.ly/2eSYPRH, 2016, [On-Line].

[9] M. Feldman, "Oak Ridge readies Summit supercomputer for 2018 debut," in: Top500.org, http://bit.ly/2ERRFr9, 2017, [On-Line].

[10] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[11] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, pp. 1740–1750.

[12] D. Schneider, "US supercomputing strikes back," *IEEE Spectrum*, vol. 55, no. 1, pp. 52–53, 2018.

[13] D. Moloney, B. Barry, R. Richmond, F. Connor, C. Brick, and D. Donohoe, "Myriad 2: Eye of the computational vision storm," in *2014 IEEE Hot Chips 26 Symposium (HCS)*. IEEE, 2014, pp. 1–18.

[14] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on Vision Processing Unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, 2015.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[16] Intel Corporation, "Intel democratizes deep learning application development with the launch of the Movidius Neural Compute Stick," in: intel.com, http://intel.ly/2tH2Eh7, 2017, [On-Line].

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[18] Movidius / Intel Corporation, "Movidius Neural Compute SDK for the Neural Compute Stick platform," in: GitHub.com, http://bit.ly/2G1i9rm, 2017, [On-Line].

[19] B. Jang, D. Schaa, P. Mistry, and D. Kaeli, "Exploiting memory access patterns to improve memory performance in data-parallel architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 105–118, 2011.

[20] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect Memory Prefetcher," in *Proceedings of the 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[21] M. Rhu, M. Sullivan, J. Leng, and M. Erez, "A locality-aware memory hierarchy for energy-efficient GPU architectures," in *Proceedings of the 2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2013, pp. 86–98.

[22] S. Lloyd and M. Gokhale, "In-memory data rearrangement for irregular, data-intensive computing," *Computer*, vol. 48, no. 8, pp. 18–25, 2015.

[23] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 25–32.

[24] J. Nickolls and W. J. Dally, "The GPU computing era," *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, vol. 30, no. 2, 2010.

[25] C. Gregg and K. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 134–144.

[26] M. H. Ionica and D. Gregg, "The Movidius Myriad architecture's potential for Scientific Computing," *IEEE Micro*, vol. 35, no. 1, pp. 6–14, 2015.

[27] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. v. d. Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[28] D. Moloney, "1TOPS/W software programmable media processor," in *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, 2011, pp. 1–24.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[30] W. Gropp, T. Hoefler, R. Thakur, and E. Lusk, *Using advanced MPI: Modern features of the message-passing interface*. MIT Press, 2014.

[31] Intel Corporation, "Myriad Development Kit (MDK): Development suite for the MA2x5x family of VPUs," in: Movidius.com, http://bit.ly/2DF6bpY, 2016, [On-Line].

[32] I. B. Peng, S. Markidis, E. Laure, D. Holmes, and M. Bull, "A data streaming model in MPI," in *Proceedings of the 3rd Workshop on Exascale MPI (ExaMPI)*. ACM, 2015, p. 2.

[33] F. Kainz, R. Bogart, and P. Stanczyk, "Technical introduction to OpenEXR," *Industrial Light and Magic*, 2009.

[34] G. A. Miller, "WordNet: A lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[35] D. Gershgorn, "The data that transformed AI research, and possibly the world," in: Quartz, http://bit.ly/2uwyb8R, 2017, [On-Line].

[36] N. Oh, "Intel launches Movidius Neural Compute Stick," in: AnandTech, http://bit.ly/2DY5e8X, 2017, [On-Line].

[37] D. Pena, A. Forembski, X. Xu, and D. Moloney, "Benchmarking of CNNs for low-cost, low-power robotics applications," *Proceedings of the Workshop on New Frontiers for Deep Learning in Robotics (RSS 2017)*, 2017.

[38] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML09)*. ACM, 2009, pp. 873–880.

[39] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics Conference (AISTATS 2011)*, 2011, pp. 215–223.

[40] NVIDIA Corporation, "NVIDIA Tesla V100 GPU Architecture," http://bit.ly/2nzeLM7, 2017, [On-Line].

[41] Intel Corporation, "Intel Nervana Neural Network Processor: Architecture update," in: intel.com, http://intel.ly/2qXgMXb, 2017, [On-Line].

[42] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "DSP.Ear: Leveraging co-processor support for continuous audio sensing on smartphones," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys 2014)*. ACM, 2014, pp. 295–309.

[43] C. Byun, J. Kepner, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, P. Michaleas, L. Milechin, J. Mullen, A. Prout, A. Rosa, S. Samsi, C. Yee, and A. Reuther, "Benchmarking data analysis and machine learning applications on the Intel KNL many-core processor," in *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*. IEEE, 2017, pp. 1–6.

[44] G. Chrysos, "Intel Xeon Phi coprocessor - The architecture," *Intel White Paper*, 2014.

[45] G. Tan, C. Zhang, W. Tang, P. Zhang, and N. Sun, "Accelerating irregular computation in massive short reads mapping on FPGA co-processor," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1253–1264, 2016.