Supporting Autonomous Vehicle Applications on the Heterogeneous System Architecture

NANDINBAATAR TSOG, Mälardalen University, Sweden
MARIELLE GALLARDO, Volvo Autonomous Solutions, Sweden
SWETA CHAKRABORTY, Volvo Construction Equipment, Sweden
TORBJÖRN MARTINSON, Volvo Construction Equipment, Sweden
ALEXANDRA HENGL, Mälardalen University, Sweden
MAGNUS MOBERG, Mälardalen University, Sweden
ADEM SEN, Mälardalen University, Sweden
MOBYEN UDDIN AHMED, Mälardalen University, Sweden
SHAHINA BEGUM, Mälardalen University, Sweden
MORIS BEHNAM, Mälardalen University, Sweden
MIKAEL SJÖDIN, Mälardalen University, Sweden
SAAD MUBEEN, Mälardalen University, Sweden

The contemporary processors are unable to meet the increasing data-intensive and computation-demanding requirements in autonomous vehicle software applications. Recently, the new Heterogeneous System Architecture (HSA) has emerged as a promising solution to meet these requirements. The HSA reduces the latency of data exchange between the compute units and cache-coherent shared memory, which is not supported by the non-HSA compliant heterogeneous platforms with acceleration support. The main goal of the paper is to investigate the performance gain by the HSA and conduct a comparative evaluation of the HSA and non-HSA compliant heterogeneous platforms. The paper aims at evaluating these platforms by using two computation-intensive software functions in autonomous vehicles, namely the object detection and vehicle movement. In order to achieve this goal, the CUDA-accelerated source code of the functions is ported from a non-HSA compliant heterogeneous platform to the HSA platform. In this regard, the paper presents the architecture of a proof-of-concept prototype and provides evaluation using the prototype.

CCS Concepts: • **Computer systems organization → Architectures**; **Parallel architectures**.

Additional Key Words and Phrases: Heterogeneous System Architecture, HSA, HIP, OpenCL, parallel computing architectures, convolutional neural network, CNN, MIOpen, code migration, CuDNN.

ACM Reference Format:
Nandinbaatar Tsog, Marielle Gallardo, Sweta Chakraborty, Torbjörn Martinson, Alexandra Hengl, Magnus Moberg, Adem Sen, Mobyen Uddin Ahmed, Shahina Begum, Moris Behnam, Mikael Sjödin, and Saad Mubeen. 2021. Supporting Autonomous Vehicle Applications on the Heterogeneous System Architecture. In *7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26–27, 2021, Novi Sad, Serbia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3459960.3459970

ECBS 2021, May 26–27, 2021, Novi Sad, Serbia

ACM ISBN 978-1-4503-9057-6/21/05…$15.00
https://doi.org/10.1145/3459960.3459970

1 INTRODUCTION

There is a huge interest in developing autonomous vehicles in the construction equipment vehicle industry. Such a vehicle is capable of sensing its environment and navigating as well as performing the construction operations (e.g., excavating, digging, crushing, paving, loading, moving and identifying objects) without human operators. These vehicles are designed to handle unexpected situations and failures without posing any hazards to people and other objects and at the same time meet the demands on productivity. Navigation, object detection and obstacle avoidance are some of the key functions that enable these operations. The algorithms implementing these functions provide a support for localizing and predicting bounding boxes for every perceived or targeted object. These vehicles include many computation-demanding functions that require data-intensive computations [2]. Making full use of sensing data improves the quality of service in these vehicles. Heterogeneous processors, such as the Heterogeneous System Architecture (HSA) [8], provide massive computation power to execute these data-intensive functions. HSA is a new computer architecture standard/specification to handle heterogeneous processors including CPU, multi-cores, Graphical Processing Unit (GPU) and others on the same board with improvements such as cache-coherent shared memory [19].

1.1 Problem Statement and Paper Contribution

The classical computation platforms (single-core or even multi-core processors) are unable to provide the high levels of computational power required by the data- and computation-intensive functions in autonomous construction equipment vehicles. The HSA is an emerging architecture that provides a promising solution to address these requirements.

The aim of the HSA is to ease the development of heterogeneous programming which is complicated in traditional non-HSA compliant heterogeneous platforms. A notable feature of the HSA is the zero memory copy, which allows reduced latency for data exchange between the compute units and cache-coherent shared memory [19]. In comparison to the HSA, the non-HSA compliant heterogeneous platforms lower the latency for the data exchange by using the traditional *pipeline* techniques. However, these techniques incur an additional delay due to copying of data between the compute units and memories, which is avoided by the zero memory copy feature in the HSA. Hence, the HSA is expected to provide better performance in terms of execution times as compared to the non-HSA compliant heterogeneous platforms. However, to the best of our knowledge, there is no existing work that provides a comprehensive comparative evaluation between the HSA and non-HSA compliant heterogeneous platforms that implement various acceleration techniques (e.g., pipelining). Within this context, we take the first step to evaluate these platforms.

The main goal of the work presented in this paper is to investigate the performance gain that can be achieved by running computation-intensive software applications on the HSA platform. For this purpose, we conduct a comprehensive comparative evaluation between the HSA and non-HSA compliant heterogeneous platforms (implementing various acceleration techniques).

A proof of concept is provided by migrating and executing some of the data- and computation-intensive functions of an existing autonomous vehicle application from legacy platforms (both multi-core and non-HSA compliant) to an HSA-compliant platform. In particular, this requires porting a CUDA-based application to the HSA Radeon Open Compute (ROCm) platform. The functions in focus are vehicle movement and object detection. The functions are based on a real-time object detection system named You Only Look Once (YOLO) [15], which implements a convolutional neural network (CNN) that analyses visual imagery in order to detect the objects.

1.2 Paper Outline

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 discusses proposed migration approach for the autonomous vehicle functions to the HSA platform. The implementation of a prototype is presented in Section 4. The evaluation results are presented in Section 5. Finally, conclusions are drawn in Section 6.

2 RELATED WORK

Power et al. [13] use the HSA in a database environment. They advocate that the execution performance can be increased by migrating parallel software to an HSA platform. Mukherjee et al. [9] create both micro benchmarks and test applications to compare the execution performance of OpenCL 1.2, OpenCL 2.0 and HSA 1.0. They show a significant improvement in the performance of the applications by using the HSA platform. Bao et al. [1] implement a framework to accelerate the classification and training processes of an arbitrary CNN. They show that the computation performance can be increased 4-10 folds by using the HSA as compared to using the CPU.

CUDA has several limitations. One major limitation is that it is confined to one specific platform, namely NVIDIA. All CUDA applications that implement Deep Neural Network (DNN) are required to use the proprietary cuDNN library which restricts their portability to other platforms. To overcome such constraints, it is necessary to have a platform-independent open standard programming model which provides more alternatives for hardware and development tools. In this regard, the HIP tool-set[1] is used to port the CUDA-based application to the HSA in the CAFFE framework [16]. The ported code gives better performance on AMD's ROCm platform with the MIOpen[2] library than on the CUDA platform. The performance of different programming frameworks including OpenCL, HC++ and HIP on both an integrated Accelerated Processing Unit (APU) and a discrete GPU is evaluated in [17, 18, 20]. The results demonstrate that with an identical kernel implementation on all frameworks, OpenCL introduced additional runtime overhead on AMD's ROCm platform. On the other hand, HIP shows better performance on both AMD and NVIDIA. In this paper, we employ the HIP tool-set for porting the autonomous vehicle application to the HSA.

3 PROPOSED MIGRATION APPROACH

This section presents the proposed approach to port the autonomous vehicle functions to the HSA platform. Moreover, the section presents a discussion about the development of a CNN on the ROCm Platform.

3.1 Application Porting to the HSA Platform

The system architecture considered in this paper is illustrated in Figure 1. The object detection functionality in the existing vehicle application utilizes the Darknet - a neural network framework [14]. The Darknet utilizes the YOLO neural network for object classification and provides acceleration with CUDA. Since CUDA only complies with Nvidia GPUs (a reference platform), the source code needs to be migrated ensuring compatibility with the target platform - an AMD A10 APU (a target platform). The first step is to set up the system by installing Ubuntu 16.04 and AMD's ROCm[3] 1.7 platform on top of it. The migration of the application from the existing platform is performed using the HIP tool set that comes with the ROCm platform. The HIP tool automatically translates the CUDA-based code (*.cu*-files) to portable C++ code, where all CUDA-function calls are

[1]https://gpuopen.com/tag/hip/
[2]https://gpuopen.com/compute-product/miopen/
[3]https://github.com/RadeonOpenCompute/ROCm

replaced by the HIP-function calls. This method accelerates the porting process since the developer does not need to manually change the CUDA code.

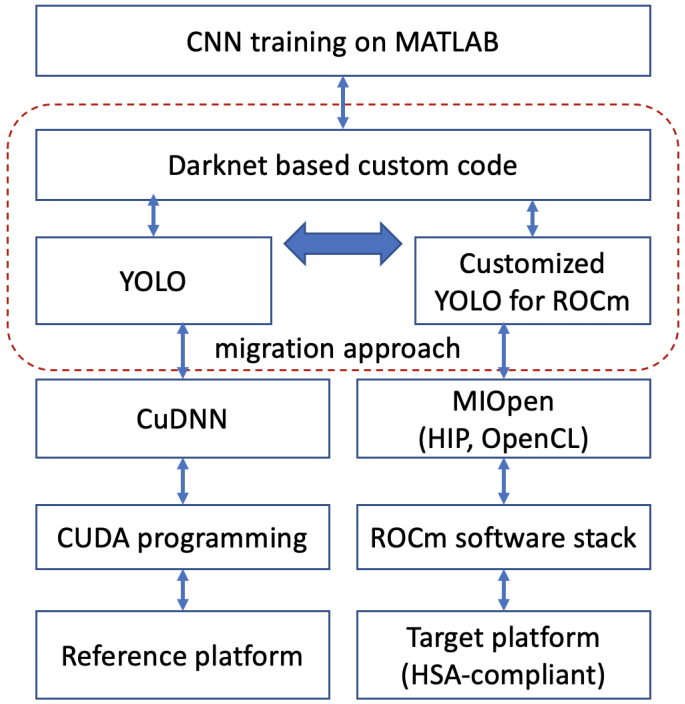


Fig. 1. System architecture.

There are several challenges involved in “hipifying" the CUDA code in the autonomous vehicle applications. For instance, the YOLO source code utilizes CUDA’s own DNN library, cuDNN. Neither cuDNN nor the parts of the YOLO source code that use this library are hipifiable. This holds true despite the fact that there exists an AMD equivalent of cuDNN, called MIOpen. These libraries are similar in a number of ways, e.g., both of them include primitives for deep neural networks and have similar function calls (e.g., *cudnnSetTensor4dDescriptor* and *miopenSet4dTensorDescriptor*).

Nevertheless, the two libraries do not have a one-to-one equivalence and therefore, it is not possible to merely swap the cuDNN and MIOpen function calls, i.e. to hipify them. Without the ability to hipify the aforementioned parts of the YOLO source code, the application needs to be manually ported. This can be achieved by manually replacing the functionality of cuDNN with the functionality of MIOpen (i.e., the HIP implementation of MIOpen).

3.2 Development of a CNN on the ROCm Platform

Exploring the state of the art revealed that there are very few HSA-compliant deep learning frameworks. Some efforts in this context include OpenCL-Caffe [7], clCaffe [3], HIP-Caffe [16] and HC-Caffe [10], HcTorch [11], Cltorch [12] and hipTensorflow [5]. However, none of these frameworks are mature enough to be used in the context of this work. In order to overcome this limitation we create an MIOpen-based implementation of a CNN that could serve as a classifier on the HSA-compliant AMD platforms. This is achieved by leveraging the MIOpen API. A classifier

CNN is set up based on the method in [6] together with cuDNN. This method involves calling MIOpen functions in place of cuDNN functions for neural network primitives, and OpenCL functions in place of the CUDA functions for memory allocation. The execution time of the CUDA sample code is measured on the CUDA platform. The measured execution time is then compared to the execution time of the MIOpen code on the ROCm platform. The DNN toolset in MATLAB is used to design the CNN architecture. The CNN is trained and the weighted values are saved for reuse in the MIOpen implementation discussed in the next section.

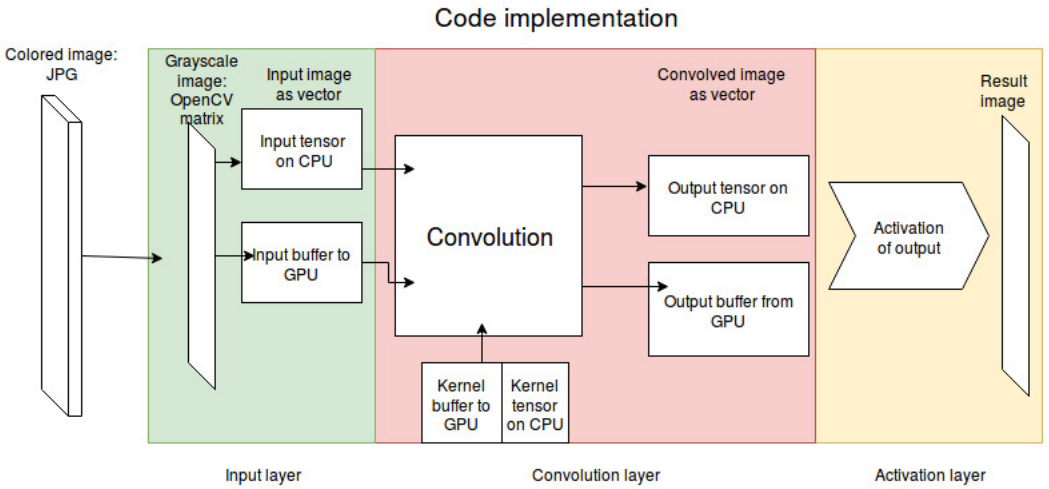


Fig. 2. Prototype architecture: the green, red and yellow fields show the input, convolution and activation layers respectively.

4 IMPLEMENTATION

This section presents the implementation of a prototype to demonstrate the CNN. The high-level functions in MATLAB call the low-level functions from other frameworks, such as MIOpen and CuDNN, for memory management and creation of various types of CNN layers.

4.1 The MIOpen Implementation

The prototype architecture is a three-layer architecture as shown in Figure 2. The layers include: (i) *Input* layer, (ii) *Convolution* layer, and (iii) *Activation* layer.

4.1.1 Input Layer. This layer includes an input image and an input tensor. The input image is stored as a gray-scale OpenCV-matrix. The input tensor is defined to have two dimensions which are appropriate for gray-scale images like the ones used in the work presented in this paper.

4.1.2 Convolution Layer. This layer applies a filter - called kernel - that convolves around the input image and outputs a *feature map*. This layer is set up by defining a *kernel tensor*, *kernel weights*, the *method* of convolution and an *output tensor*. The kernel tensor that convolves around the input image is defined to be a two-dimensional filter since the dimensions of the kernel have to match with the dimensions of the input image. The values for the kernel weights describe a three-by-three edge detector filter as shown below.

$$kernel\_weights = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The stride and padding of this filter are both defined as 1, for simplicity. The cross-correlation convolution method is applied. The output of this layer, i.e., the feature map is stored in the *output tensor*.

4.1.3 Activation Layer. The activation layer operates on the output provided by the convolution layer and uses the softRELU function $log(1 + e^x)$ to calculate the result.

4.2 MIOpen with OpenCL support

MIOpen supports two programming models, HIP and OpenCL. When using MIOpen with OpenCL, memory allocation has to be done with the help of the OpenCL APIs. Allocating memory for the entire size of the input image, the kernel, the output image, as well as the convolution workspace, is necessary. This is done by creating and initializing the necessary command queue and OpenCL context. After that, the OpenCL memory buffers are created on the device. This operation requires correct size of the buffers in bytes. Therefore, these values are calculated in prior. The content of each object is copied from the host memory to their respective device buffers.

4.3 Training of the CNN Architecture

MATLAB has built in neural network toolbox that provides algorithms for training and visualizing deep neural networks. The contribution of this tool in this work is two fold. First, it enables the rapid design process and quick accuracy-evaluation of various CNN architectures. Second, it facilitates the training of the network, through which the appropriate weight values can be obtained. The training dataset and the final architecture with the best performance is discussed below.

4.3.1 Training Dataset. There are different ways of training a CNN for image analysis. First, training the model from scratch, i.e. by back propagation. Second, by applying *transfer learning* that is based on the knowledge of one type of problem which can solve a similar problem. Third, by extracting features (weights) from a pre-trained CNN for reuse in the model. We use the first approach in the work presented in this paper and accordingly train the model from the scratch. In this method, images from the MNIST[4] handwritten digit dataset are used for both the training and evaluation. We hand pick 600 images randomly. The images are sorted by digits 0-9 into different folders, each folder containing 60 images. These image folders have been separated by the *splitEachLabel* MATLAB function into *training* and *test* categories, containing 480 and 120 images, respectively. Some of the digits in this dataset exhibit slight differences, which can be hard for humans to tell apart (e.g., a 0 looks like a 6). The network is trained using the *training* category. The plot in Figure 3 depicts the final validation of the accuracy and the loss value. As the loss value approaches zero, the training process is completed.

To verify the performance of the CNN architecture, the *test* image category is used. Four of each digits 0-9 in this category are distorted using Adobe Photoshop. Distortion of the test images refers to changing the position or scaling of the image. As a result of the distortion, some digits appear at different positions in the image and with different digit-size-to-image-size ratio.

[4]https://www.kaggle.com/scolianni/mnistasjpg

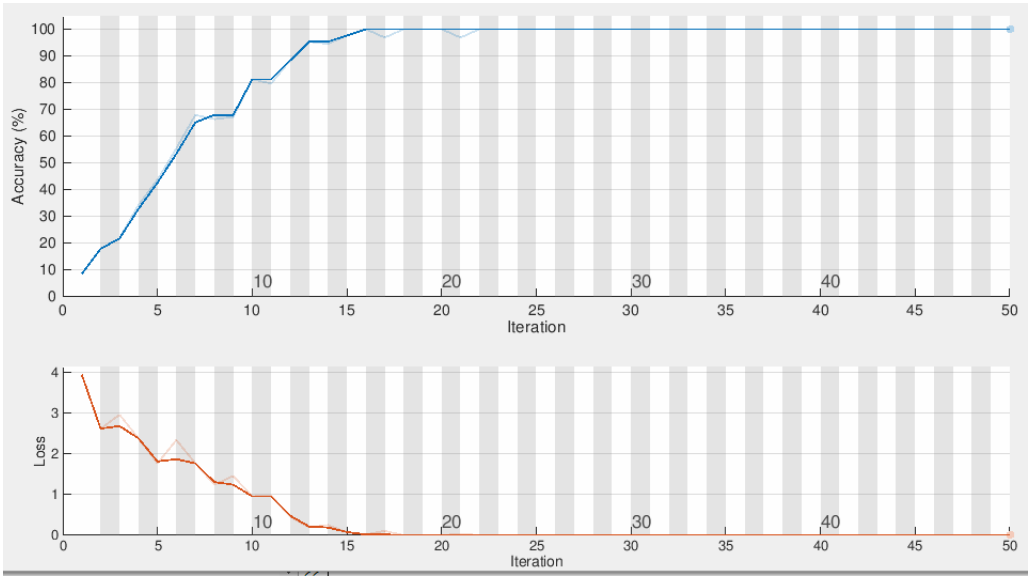


Fig. 3. Training Progress.

4.3.2 The Network Layers. The architecture of the CNN is illustrated in Figure 4. The architecture consists of an input layer, some hidden layers and an output layer. The hidden layers consist of a convolution layer with six 5x5 filters, one activation layer using the ReLU method, and one max-pooling layer with a pooling size of 2x2. The output layer consists of a fully connected layer with output size 10, for the digits 0-9, a softmax layer and finally, a classification layer.

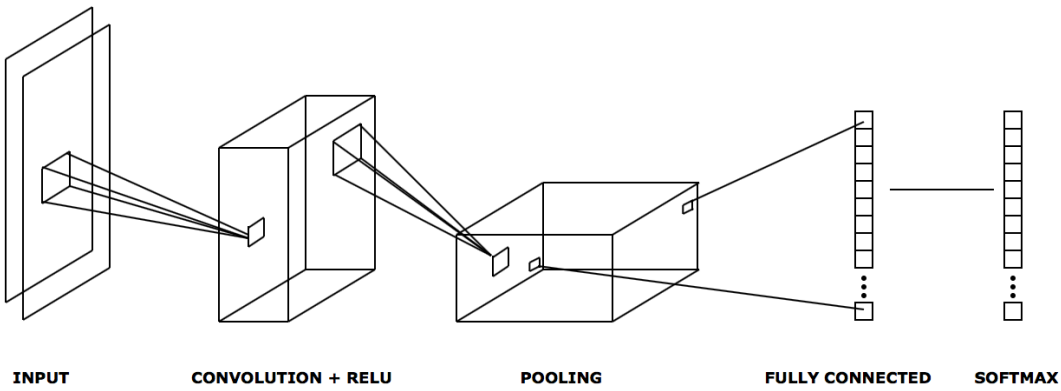


Fig. 4. Illustration of the CNN Architecture.

The convolution layer is trained with iterations of 50 epochs, by the *sgdm* (stochastic gradient descent with momentum) method. The CNN receives a 28x28 gray-scale input image. Figure 5 shows 30 of the 600 test images.

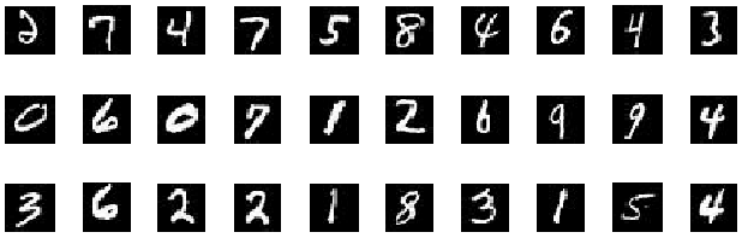


Fig. 5. Example: 30 out of the 600 test images.

5 EVALUATION

This section explains the results of the implementation by both MATLAB and MIOpen.

5.1 Experiment Setting

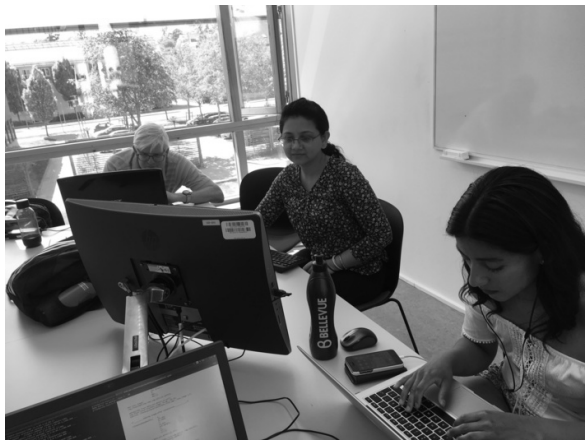
5.1.1 Hardware configuration. As introduced in Section 3 and presented in Figure 1, we consider a MATLAB model for preparing the weights of the CNN architecture used at the hardware level of abstraction. At this level, two platforms are considered: one as a target and the other as a reference. The target platform is an Acer Aspire E15 laptop with an AMD A10-8700P APU, which employs A10 1.8GHz CPU and Radeon R6 GPU 800MHz, with 8GB RAM. The platform is able to install ROCm software stack and supports HSA fully. The platform is able to use a machine learning library, MIOpen, and programming models, HIP and OpenCL. On the other hand, as the reference laptop, we consider a Lenovo P50s laptop with Nvidia's Quadro M500M 1029MHz GPU and Intel i7-6500U 2.5GHz CPU, with 8GB RAM for the CUDA programming. The release date of these platforms is around 2015. Both AMD A10-8700P APU and Nvidia Quadro M500M's are fabricated on a 28nm process. Besides, Intel's CPU is manufactured in 14nm. Furthermore, AMD A10-8700P is the first HSA-compliant and ROCm supported platform. Therefore, we consider that these platforms are an appropriate representative for evaluation.

5.1.2 Experiment configuration. Two challenges are considered in this paper. First, we evaluate how the proposed migration approach completes successful. In order to handle this evaluation, we consider the comparison study of execution time for processing through the prepared CNN architecture between the target and reference platforms. The second challenge is how efficiently the prepared CNN architecture handles. We conduct experiments with non-distorted and distorted input images.

5.2 Evaluation of the Proposed Migration Approach using MIOpen

Figure 6a illustrates the grayscale input image that is sent to the convolution layer. The result of the input image by the filter application is depicted in Figure 6b. This output feature map is a result of an edge detector filter as we can only see the outlines of the image. Figure 6c describes the results after the activation layer, where the activation mode is the softRELU ($log(1 + e^x)$) method.

We consider and measure the execution time of the prepared CNN architecture on the target and reference platforms using two different images. The first image (Figure 7) is 10x10 and the second (Figure 8) is 706x529 pixels. The measurement shows that the first (small) image takes an average 1.20 seconds and the second (large) image takes an average 1.60 seconds after running 20 times each. These numbers seem high for three reasons. For the first, the measurements include the execution time of the whole application, not only the execution time of the convolution layer. Secondly, there is a high overhead per image for running the application. In a real use case, the images are



(a) An input image to convolution layer.

(b) A result image by the filter-application.

(c) The activation layer result.

Fig. 6. Results.

processed in batches, rather than one-by-one. Such processing reduces the overhead. The last, the original CUDA code on the reference platform takes approximately 0.60 seconds (20 times) on the small image, which is twice as fast as on the target platform. Further, the reference platform has about 1.4 times more computing capability and the version of MIOpen is still development stage. It is important to note that this study is based on the comparison study between the original CUDA (optimized) code on the reference platform and the non-optimized migrated code on the target platform. In addition, the difference in CPU power on the target and reference platforms affects the experimental result. This is reason why the effectiveness of the HSA contradicts the expected outcomes. Moreover, we can state that we have successfully migrated the CUDA code to the target platform, which is HSA-compliant. We confirm that the HIP version of MIOpen can accelerate the execution time of the proposed migration approach as the latest version of HIP programming model is fully interpretable with CUDA programming.

Fig. 7. Test image (small - 10x10 pixels).

Fig. 8. Test image (large - 706x529 pixels).

5.3 Evaluation of the Prepared CNN Architecture using MATLAB

Figure 9 describes the weight values that are obtained by training the CNN architecture using MATLAB. The CNN architecture on MATLAB correctly categorized the test images 91% of the time when tested on non-distorted images, which is an acceptable result. However, when tested on the distorted category, only 50% of the images were classified correctly. Figure 10 illustrates a sample of a non-centered, i.e. distorted digit. The distorted category result might be due to some of the images being too small and low resolution. Another reason could be the relatively low number of training images.

```
5x5x1x6 single array

ans(:,:,1,1) =

   -0.0086   -0.0008    0.0043    0.0087   -0.0081
    0.0036    0.0084    0.0068   -0.0016    0.0132
    0.0099    0.0009   -0.0056   -0.0018    0.0188
    0.0123    0.0136    0.0071    0.0042    0.0220
    0.0092    0.0065    0.0123    0.0050    0.0081
```

Fig. 9. Matlab weights.

Fig. 10. Distorted image.

6 CONCLUSION AND FUTURE WORK

The MIOpen library for deep learning is a relatively new framework, which has a limited footprint in the research community. After exploring the state of the art, we found that there are no implementations except for the source code of the *MIOpen Driver* application that is implemented by AMD. The software application considered in this paper implements a combination of the input, convolution and activation layers. Currently, this implementation supports visualization of the resulting feature maps produced by the convolution. For instance, the convolution of an edge-detector filter on the input image produces the expected visual effect. The MATLAB framework is utilized to visualize and demonstrate the proposed idea for the CNN architecture. To create a simple but complete classifier CNN, only a fully connected layer - and optionally a pooling layer - are missing, which constitutes the ongoing work. When these layers are implemented, it would be possible to do the classification, provided that a sufficient amount of filters are supplied to the system. Such filters could be obtained by training an identical CNN via MATLAB and reusing the trained filter weights. Another way to obtain the necessary weights would be to implement back-propagation by the MIOpen API. This would allow the training of the CNN.

Another ongoing work is to implement a higher-level framework - such as TensorFlow, Caffe and Torch - by integrating the MIOpen library [4]. Such a framework would enable faster creation and simpler customization of different deep learning architectures, as well as it would hide the low-level GPU allocations from the user. MIOpen supports two different frameworks for GPU acceleration: HIP and OpenCL. OpenCL has been used in this work because it is an older, thus more mature, technology. However, the HIP framework seems quite promising, and in hindsight, it might have been the easier way to port the CUDA code. The reason for this is that the HIP and the CUDA APIs are very similar, unlike the OpenCL and CUDA. The reference platform achieves twice faster performance by using 1.4 times more computing capability because the experiment on the target platform has been handled under the limited conditions of development version of MIOpen. Furthermore, the comparison study is performed between the original code on the reference platform and the corresponding non-optimized code on the target platform. Therefore, it would be an interesting future work to compare the execution time of the original CUDA code with

that of both the OpenCL and the HIP implementations. Using the implementation of the ongoing works, we will conduct a comparative evaluation of the original CUDA code and both the OpenCL and HIP implementations to understand and demonstrate the advantages and disadvantages of HSA vs. non-HSA compliant heterogeneous platforms.

ACKNOWLEDGMENTS

The work in this paper is supported by the Swedish Knowledge Foundation (KKS) via the DPAC project and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE and PROVIDENT projects. We thank our industrial partners, especially Volvo Construction Equipment for their valuable input to this work.

REFERENCES

[1] Zhenshan Bao, Qi Luo, and Wenbo Zhang. 2017. An Implementation and Improvement of Convolutional Neural Networks on HSA Platform. In *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 594–604.

[2] Lucia Lo Bello, Riccardo Mariani, Saad Mubeen, and Sergio Saponara. 2018. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics* 15, 2 (2018), 1038–1051.

[3] Jeremy Bottleson, SungYe Kim, Jeff Andrews, Preeti Bindu, Deepak N Murthy, and Jingyi Jin. 2016. clcaffe: Opencl accelerated caffe for convolutional neural networks. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 50–57.

[4] Fredrik C Bruhn, Nandinbaatar Tsog, Fabian Kunkel, Oskar Flordal, and Ian Troxel. 2020. Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space. *CEAS Space Journal* 12, 4 (2020), 551–564.

[5] codeaudit. September, 2017. hipTensorflow: the HIP Port of Tensorflow. https://github.com/codeaudit/hiptensorflow/blob/hip/README.ROCm.md. Online; accessed 4 February 2021.

[6] Peter Goldsborough. October, 2017. Convolutions with cuDNN. http://www.goldsborough.me/cuda/ml/cudnn/c++/2017/10/01/14-37-23-convolutions_with_cudnn/. Online; accessed 4 February 2021.

[7] Junli Gu, Yibing Liu, Yuan Gao, and Maohua Zhu. 2016. Opencl caffe: Accelerating and enabling a cross platform machine learning framework. In *Proceedings of the 4th International Workshop on OpenCL*. ACM, 8.

[8] HSA Foundation. May, 2018. HSA Platform System Architecture Specification, Revision: Version 1.2. http://www.hsafoundation.com. Online; accessed 4 February 2021.

[9] Saoni Mukherjee, Yifan Sun, Paul Blinzer, Amir Kavyan Ziabari, and David Kaeli. 2016. A comprehensive performance analysis of HSA and OpenCL 2.0. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 183–193.

[10] Multicoreware. August, 2016. HCC backend Implementation for Caffe. https://bitbucket.org/multicoreware/hccaffe. Online; accessed 4 February 2021.

[11] Multicoreware. December, 2016. HCC backend Implementation for Torch7. https://bitbucket.org/multicoreware/hctorch. Online; accessed 4 February 2021.

[12] Hugh Perkins. 2016. Cltorch: a hardware-agnostic backend for the torch deep neural network library, based on opencl. *arXiv preprint arXiv:1606.04884* (2016).

[13] Jason Power, Yinan Li, Mark D Hill, Jignesh M Patel, and David A Wood. 2015. Toward GPUs being mainstream in analytic processing: An initial argument using simple scan-aggregate queries. In *Proceedings of the 11th International Workshop on Data Management on New Hardware*. ACM, 11.

[14] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[16] Ben Sander. July, 2017. We ported CAFFE to HIP – and here's what happened ... https://gpuopen.com/ported-caffe-hip-heres-happened/. Online; accessed 2 August 2019.

[17] Yifan Sun, Saoni Mukherjee, Trinayan Baruah, Shi Dong, Julian Gutierrez, Prannoy Mohan, and David Kaeli. 2018. Evaluating Performance Tradeoffs on the Radeon Open Compute Platform. (2018).

[18] Nandinbaatar Tsog, Matthias Becker, Marcus Larsson, Fredrik Bruhn, Moris Behnam, and Mikael Sjodin. 2016. Real-Time Capabilities of HSA Compliant COTS Platforms. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 364–364.

[19] Nandinbaatar Tsog, Moris Behnam, Mikael Sjödin, and Fredrik Bruhn. 2018. Intelligent Data Processing using In-Orbit Advanced Algorithms on Heterogeneous System Architecture. In *IEEE Aerospace Conference, 2018*.

[20] Nandinbaatar Tsog and Marcus Larsson. 2016. Time Predictability of GPU Kernel on an HSA Compliant Platform.