2016 IEEE International Parallel and Distributed Processing Symposium Workshops

clCaffe: OpenCL accelerated Caffe for Convolutional Neural Networks

Jeremy Bottleson*, SungYe Kim*, Jeff Andrews*, Preeti Bindu*, Deepak N. Murthy† and Jingyi Jin*
*Intel Corporation, Folsom, CA 95630, †Northeastern University, MA 02115
*{jeremy.bottleson, sungye.kim, jeff.andrews, preeti.bindu, jingyi.jin}@intel.com, †narasimhamurthy.d@husky.neu.edu

***Abstract*—Recent advances in deep convolutional neural networks enable researchers and developers to apply machine learning to a much broader number of applications. With the proliferation of deep learning applications, widely used deep learning frameworks, such as Caffe, Theano and Torch, have been significantly improved with the support of powerful GPUs and GPU-accelerated libraries. However, lack of frameworks and libraries built on OpenCL could hinder exploration of more diverse compute devices (CPUs, GPUs, DSPs and FPGAs) in future deep learning domains. In this work, we present OpenCL acceleration of a well-known deep learning framework, Caffe, while focusing on the convolution layer which has been optimized with three different approaches; GEMM, spatial domain, and frequency domain. Our work, clCaffe, greatly enhances the ability to leverage deep learning use cases on all types of OpenCL devices, particularly on small form factor devices in which discrete GPUs are rare and integrated GPUs are much more common. Our benchmark shows 2.5x speedup on the Intel integrated-GPU, compared to CPU-only AlexNet on ImageNet dataset. As such, our work provides the deep learning community with the opportunity to embrace a broad range of devices through OpenCL.**

***Keywords*-Deep learning framework; Convolutional Neural Networks; Caffe; OpenCL; Integrated GPU**

I. Introduction

Deep Convolutional Neural Networks (CNN) have recently gained much attention for having performance and applicability that have surpassed traditional approaches for general visual recognition problems. One of the first use cases where CNN performed very well was published by Lecun et al.[1] on gradient-based learning applied to document recognition in 1998. Later, Ciresan et al.[2] came up with a method for quickly training CNN by utilizing GPUs, and led to work by Krizhevsky et al.[3] in classifying the ImageNet dataset using a deep CNN called AlexNet. This was a significant accomplishment as they showed that CNN could produce state of the art results on general image classification. Since first winning the ImageNet Challenge [4] in 2012, CNN approaches have made great breakthroughs in the deep learning and machine learning domains. Subsequent years of the ImageNet competition have included object recognition approaches with increasingly higher accuracy, that are based on CNN. The accuracy of those computerized models have reached the point that they surpass humans for some tasks [5], [6].

In order to deliver fast and accurate prototypes or products based on CNN, having a comprehensive and easy to use framework becomes a crucial component to avoid extensive duplicated development efforts. Most popular frameworks such as Caffe[7], Theano[8] and Torch[9] provide a full set of CNN primitives with state of the art optimization to make the building of complex and efficient network models feasible. Caffe was originally developed by the Berkeley Vision and Learning Center (BVLC) and has been further improved by a large number of deep learning community open source contributors. Theano is a Python library developed at the University of Montreal and allows developers to define, optimize and evaluate mathematical expressions efficiently and intuitively, specifically in scientific investigations. Torch is also an open source machine learning library based upon the Lua scripting language and used by a number of industries, including Facebook, Twitter and Google. All of these frameworks have been significantly improved with the support of GPU-accelerated deep learning libraries. Nevertheless, we identify one major constraint of most deep learning frameworks, which is narrow support of compute devices thus far. For instance, Caffe provides CPU mode and CUDA/cuDNN supported GPU mode only on NVidia GPUs. Similar conditions hold for Theano and Torch as well.

Hence, enabling a CNN framework to run on a broad range of compute devices is one of our major goals for this work. clCaffe is a solution that we introduce, which is an extension of the Caffe framework with OpenCL. The reasons why we chose Caffe as our base framework are that it is one of most popular frameworks in the deep learning community, and has a well-structured code base, which facilitates our OpenCL extension. Inheriting all functionality of Caffe, we design clCaffe to support CNN applications for both training and deployment, while keeping compatibility with the existing CPU and CUDA/cuDNN based GPU modes. Therefore, our extension would broaden the supported devices in Caffe to all types of OpenCL compliant devices including CPUs, GPUs, DSPs and FPGAs for future deep learning applications. Besides the contribution of clCaffe of enabling more devices, we also empower clCaffe by supporting multiple approaches to convolution, such as GEMM (General Matrix Multiplication), spatial domain and frequency domain convolutions, in order to allow developers to switch between different convolution implementations according to their use cases and configurations.

There have been a few other efforts to support OpenCL in deep learning frameworks. For instance, DeepCL[10] is a standalone open source framework built to use OpenCL for accelerating CNN, and clTorch[11] provides developers with an OpenCL backend for Torch. OpenCL-Caffe[12] has been recently open-sourced by AMD on GitHub to provide OpenCL acceleration on AMD GPUs. Particularly, our work is similar to OpenCL-Caffe in that we both use Caffe as a base framework and extend it with an OpenCL implementation. However, we also focus on providing multiple convolution approaches while accelerating Caffe with OpenCL so that developers can properly select the best convolution engine for their CNN models.

Our performance benchmark results on the Intel integrated-GPUs show significant performance improvement, when compared to the CPU-only benchmark results. Because of this we find good use cases on deployment on low-end, power constrained devices where acceleration from giant-sized discrete GPUs might be infeasible. The main contributions of our work are listed as follows:

- We extend a well-known deep learning framework, Caffe, with OpenCL so that developers can create and leverage their CNN applications on multiple devices.
- We provide OpenCL based convolution approaches in GEMM, spatial domain and frequency domain so that developers and researchers can leverage the proper convolution engine for their CNN models.
- We quantitatively analyze training and deployment performance of CNN models on the Intel integrated-GPUs, which is not possible with standard Caffe.
- Finally, clCaffe provides baby steps towards device abstraction in deep learning frameworks, as a reference implementation for OpenCL compliant devices.

In the following sections, we first analyze the compute profile for our initial OpenCL implementation for a few CNN models to identify the compute bottlenecks. Then, the details of our OpenCL accelerated Caffe framework are described. Finally, we show the benchmark results and analysis.

II. CNN Model Compute Profile

As part of our OpenCL accelerated Caffe framework implementation, we profiled and optimized it in order to get a high level of performance. The first step involved profiling some CNN models to determine where most of the computation time was being spent and where optimization efforts should be focused. Among the many different CNN models we found that by targeting AlexNet by Krizhevsky et al.[3], and GoogLeNet(one of the largest CNN models, consisting of 22 layers) by Szegedy et al. [13] we covered most of the common varieties of layer types. Caffe bundles both models with a pre-trained caffemodel so that developers can easily reproduce results. As such, we chose these two networks as our benchmarks.

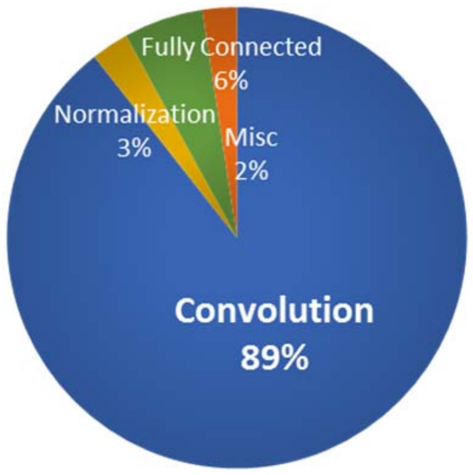


Figure 1. AlexNet GPU performance profiling on initial OpenCL implementation with clBLAS.

AlexNet consists of the following layer types: convolution, rectified linear, normalization, pooling, and fully connected.

- The convolution layer, which is used to convolve an input image with multiple trainable filters and produces feature maps, which are used as an input of the next layer.
- The rectified linear layer is used to add nonlinear properties to a network model. The simplest function to do this is Max(x, 0), and a negative slope parameter is used to specify whether to leak the negative part.
- The normalization layer normalizes the values across all layer channels.
- The pooling layer down samples an image by taking a block of pixels and outputting a single value, either the maximum, average, or stochastic value.
- The fully connected layer, also referred to as the inner product layer, performs vector multiplication.

As shown in Figure 1, the convolution layers in AlexNet are by far the biggest bottleneck representing 89% of the execution time in our initial OpenCL implementation. GoogLeNet also shows similar distribution of execution time. Accordingly, we decided to focus more on optimizing the convolution layer in clCaffe.

The nature of the 3D convolution algorithm is such that it is well suited to being implemented as a data parallel algorithm which is ideal for parallel hardware such as GPUs and multi-core CPUs. Each output pixel can easily be generated in parallel by a separate thread. However, while the basic algorithm is trivial to implement, it has a significant performance issue with regards to memory bandwidth as each individual pixel in the image is used to calculate multiple outputs, causing large amounts of memory to be used that are likely to bottleneck performance below the

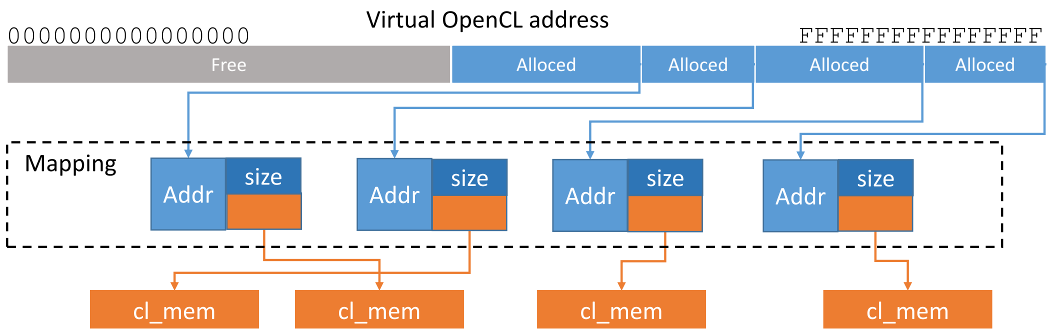


Figure 2. Virtual OpenCL address in clCaffe to handle OpenCL memory with a regular pointer, by mapping OpenCL memory handle (cl_mem) and allocated memory size into a virtual address.

theoretical FLOPS capabilities of devices. As an example, we look at the second convolution layer of AlexNet which consists of 128 5×5 filters with a stride of 1 being applied to the image. This means that for the trivial implementation we access each pixel up to 5×5×128 equal to ∼3200 times per pixel. In our tests, the trivial implementation often achieved far less than 1% of device efficiency. To increase our overall performance and achieve greater utilization of hardware, we also consider several improvements to the basic convolution approach focused on decreasing the number of times a pixel is accessed, thus lowering bandwidth requirements and increasing throughput.

III. OpenCL Accelerated Caffe Framework

Our work, extending the Caffe framework with OpenCL, enables developers and researchers to execute their CNN models on different OpenCL compliant devices (e.g., CPUs, GPUs, DSPs, and FPGAs). While the main focus of our optimization effort was on optimizing the different convolution approaches for integrated GPUs through OpenCL, we also spent time on an appropriate memory management scheme, harnessing shared memory between CPU and integrated GPUs to eliminate explicit memory transfers, referred to as zero-copy.

A. OpenCL Integration to Caffe

Extending the Caffe framework to support OpenCL was divided into multiple efforts on the host-side, the OpenCL kernels on the device-side, and the build configuration. For the host-side, we created several helper classes to manage state, memory, and OpenCL kernel creation and destruction. Each layer implementation on the host does little more than queuing up OpenCL kernels for execution, with the exception of our spatial domain convolution layer. OpenCL kernels on the device-side have been implemented for all layers in the original Caffe, as well as additional math/logic functions, and other miscellaneous modules. Since the majority of execution time for CNN is in the convolution layers, our optimizations focused on the convolution layer, which is why two additional variants of the convolution layer are integrated together with GEMM based convolution: spatial domain and frequency domain. The spatial domain goes through several iterations to tune kernel parameters and generate the most optimal convolution kernel for the given parameters. The build configuration is modified in order to build the host-side code as well as incorporate the OpenCL kernels as linked objects to improve compile time.

Furthermore, there are several OpenCL based libraries that we used which expedited our OpenCL integration by providing primitive math and BLAS (Basic Linear Algebra Subprograms) functions in OpenCL kernels. clBLAS[14], ISAAC[15] and clFFT[16] are utilized for GEMM based and frequency domain convolution. For uniform GPU random number generation, Random123[17] library has been used.

B. CPU/GPU Memory Synchronization

Integrating OpenCL into Caffe's synched memory functionality is not straightforward because Caffe performs indexing on the allocated memory as if it were a regular CPU pointer. In the case of OpenCL, which returns a handle to a buffer object, pointer arithmetic is not allowed. Caffe will typically take an offset from the allocated address and pass it as a new pointer address into other functions. For CPU memory pointers this is perfectly valid but for OpenCL this is now an invalid handle value. In OpenCL, offsets can be used with handles but they must be passed to the API. We originally attempted to determine the handle from the modified address by calculating the distance based on the allocated size but it proved problematic because of overlap with the addresses so the wrong buffer object handle would sometimes be selected. We overcame this issue by creating our own virtual addressing with OpenCL memory as shown in Figure 2. After allocating OpenCL memory, we map the handle and the memory size into a virtual address and remove the addressable range from use. This approach now allows for Caffe's pointer offsetting to be used such that the object buffer handle can easily be determined and passed to OpenCL kernels for execution.

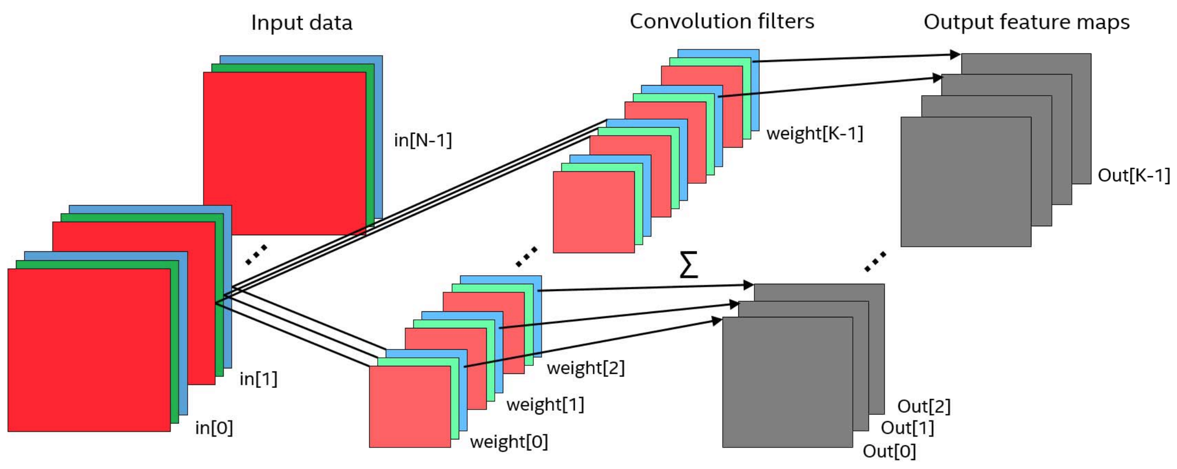


Figure 3. Convolution process generating a feature map per convolution filter from multi-channel images and filters.

C. Convolution Approaches

Figure 3 describes a convolution process for general CNN models. Input data consists of N images with multiple channels. K filters are applied to each input data then an output feature map is generated per convolution filter by summing up the convolution values of all the channels. To implement this process, Caffe uses a GEMM based approach for its convolution layer, utilizing a BLAS library. To provide increased flexibility in our clCaffe, we implement two additional convolution methods: spatial domain and frequency domain. The GEMM based convolution in clCaffe resembles the original implementation in Caffe with the exception that clBLAS or ISAAC (Input Specific Architecture Aware Computations) is used for OpenCL GEMM kernels. The spatial domain convolution is able to be more optimized for Intel integrated GPUs as it does not have any dependencies on other libraries. The frequency domain convolution in clCaffe also provides reasonably good speedup for specific size of convolution with its use of the clFFT library, which isn't necessarily tuned for Intel integrated GPUs. In this section, we present more details for each convolution approach.

1) GEMM based Convolution: This is the default method in Caffe for both forward and backward convolution. In clCaffe, it is implemented through the clBLAS and ISAAC library, and offers advantages in that both forward and backward can easily be implemented using OpenCL GEMM kernel. By mapping to standard BLAS library functions, this method is also able to benefit greatly in instances in which tuned linear algebra libraries exist.

GEMM based convolution has a two step process which introduces some additional memory requirements to store data and convolution filters in column vectors. In the first phase, which Caffe calls the image to column step (im2col), the image data is expanded and rewritten into a larger matrix which will be multiplied with the convolution filter matrix. All of the rows of this new matrix are multiplied by a column of the filter matrix to produce an output image. The convolution filter matrix has a single row for each filter, and produces one output feature map per filter.

2) Spatial domain Convolution: What we refer to as spatial domain convolution is a method in which the convolution filter directly operates on an input image to convolve it to produce the output image. Unlike the GEMM based method, it does not use an intermediate image to column step and so avoids some extra memory overhead. In the spatial domain convolution, a filter of values of dimensions W$\times$H with channels C is multiplied with a portion of an image and then summed to produce a single output value. This is repeated for all valid positions of the filter with the image to produce an output image as shown in Figure 3. The resultant output images are stacked into a multi-channel image which forms the input to the next layer.

The primary way we attempt to lower convolution bandwidth requirements is by modifying the OpenCL kernel so that each work item generates multiple output pixels. In addition to generating multiple outputs, we make sure that the kernel is written so that it only ever accesses a portion of the image data only once from global memory. By doing this we significantly lower the bandwidth requirements of the convolution filters. For instance, using the second convolution layer in AlexNet we see that if each work item calculates a 4$\times$3$\times$2 block of output pixels then the number of times we access any values is reduced from $\sim$3200 to $\sim$133.3 times on average. As part of our spatial convolution, we parametrize OpenCL kernels to calculate a block of output pixels with dimensions X$\times$Y$\times$Z. While there exists no hard cap on how many output pixels we can calculate per work item, there does exist a number of factors which limit our ability to do so from a performance standpoint. The key considerations are 1)the overall number of work items which we create, if each work item calculates too

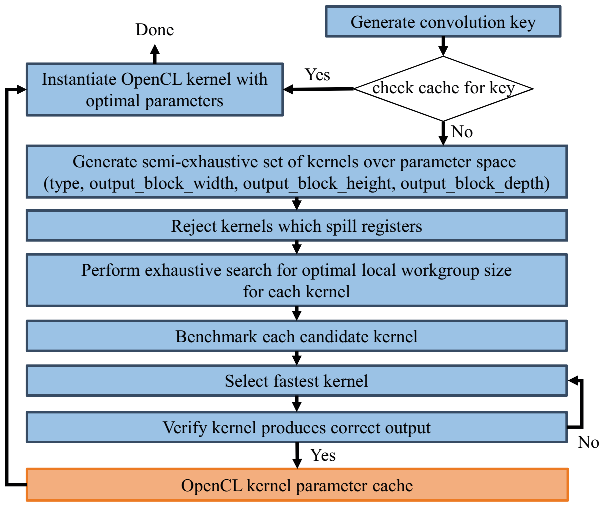

Generate convolution key
Done
Instantiate OpenCL kernel with optimal parameters
Yes
check cache for key
No
Generate semi-exhaustive set of kernels over parameter space (type, output_block_width, output_block_height, output_block_depth)
Reject kernels which spill registers
Perform exhaustive search for optimal local workgroup size for each kernel
Benchmark each candidate kernel
Select fastest kernel
Verify kernel produces correct output
No
Yes
OpenCL kernel parameter cache

Figure 4. Auto tuning process for generating the optimal spatial domain convolution kernel.

Table I
BENCHMARK SYSTEM CONFIGURATION.

	Intel Broadwell	*Intel Cherry Trail*
CPU	Intel® Xeon® processor CPU E3-1200 v4 @ 3.40GHz	Intel® Atom™ x7 processor CPU @1.6GHz
GPU	Intel® Iris™ Pro graphics 6200 48 EUs boost clock 1150 MHz 0.88 TFlops peak perf 128MB eDRAM Bandwidth 25.6 GB/s	Intel® HD Graphics 16 EUs boost clock 600 MHz No eDRAM Bandwidth 25.6 GB/s
OS	CentOS 7.1	Windows 10
OpenCL	1.2	1.2

many output pixels we may not spawn enough work items to saturate our device, and 2)the number of output pixels we can calculate in a work item without spilling registers to global memory, which is primarily a function of how much register memory each work item has available. These factors result in the optimal kernel parameters being highly device specific.

The key in spatial domain convolution is an auto tuning approach to account for the differences in possible devices and ensure good performance across all platforms and devices. Figure 4 shows the auto tuning steps to find the optimal kernel parameters and instantiate the fastest OpenCL kernel. The first time a new convolution is seen, a semi-exhaustive search is performed over the kernel parameter space to find the optimal parameters. Once determined, these optimal parameters are saved in a cache, and future instantiations of the convolution OpenCL kernel will use cached parameters instead of searching for optimal parameters again.

3) Frequency domain Convolution: The Fast Fourier Transform (FFT) based convolution reduces the convolution complexity in spatial domain when filter sizes are large. The key idea is that convolutions in the spatial domain are equivalent to element-wise multiplications in frequency domain. Although it is known that FFT based convolution does not have significant advantages for CNN models with small convolution filter sizes (i.e., 1×1, 3×3, 5×5), many researchers and industries are taking an interest in investigating the potentials of FFT based convolutions. Mathieu et al.[18] presented a CUDA based Cooley-Tukey FFT implementation to accelerate CNN training in order to better perform many FFTs over small inputs. Vasilache et al. [19] conducted extensive performance analysis with FFT based convolution, called fbfft, for multiple filter sizes and problem sizes. Their evaluation shows cuFFT achieves $1.4 - 14.5\times$ speedup over cuDNN, and their fbfft itself is at least $1.4\times$ faster than cuFFT.

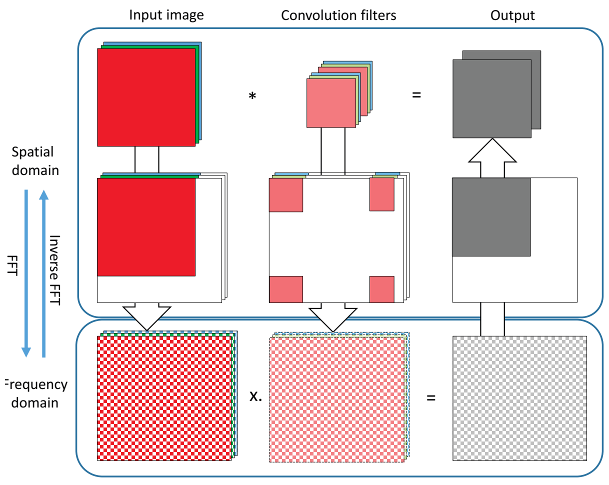
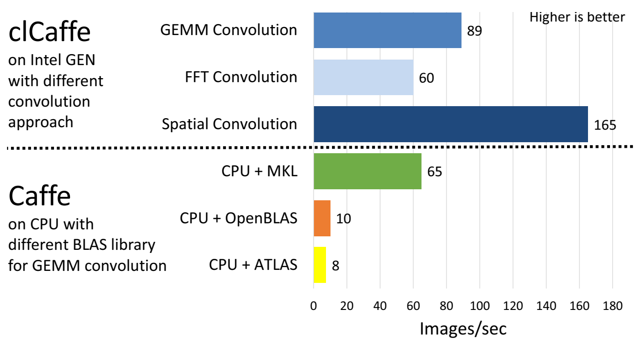
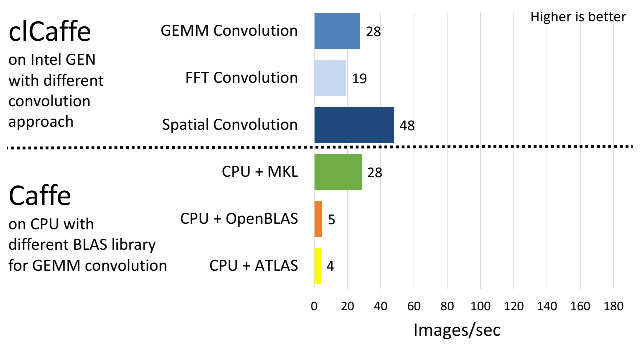


Figure 5. Frequency domain convolution process. An input image and filters are padded with zeros and saved in additional memory. Convolution in frequency domain is done by element-wise multiplication of complex numbers. Finally, we perform inverse FFT to get an output feature map.

Since we work on OpenCL devices, we decide to use clFFT (v2.4.0) for early investigation into FFT based convolutions on OpenCL devices within clCaffe. For FFT based convolutions, we first pad data and filters with zeros before performing FFT. Thus, additional memory is inevitable to store the intermediate zero-padded data and filters in spatial domain as well as transformed data and filters in frequency domain. Figure 5 shows the process in our frequency domain convolution. By using batched 2D FFT and in-place transform of convolution filters, we reduce memory use. In our experiments, we apply FFT to convolution layers with 5×5 filter size and stride of 1 in AlexNet and GoogLeNet. When stride is greater than 1, FFT based convolution does not show speedup over spatial domain convolution because the large stride reduces the complexity of spatial domain convolution. Empirically, we find that FFT based convolution is beneficial

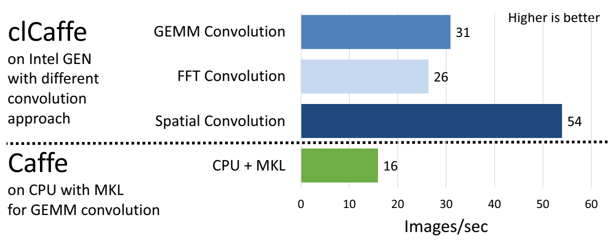


(a) Forward pass only.

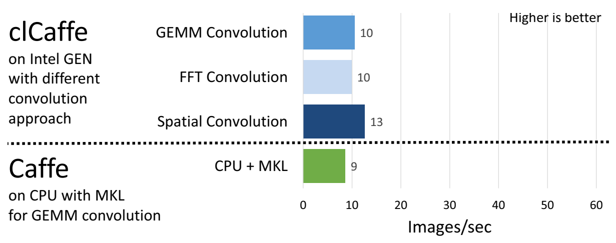


(b) Forward and backward passes.

Figure 6. AlexNet benchmark results using clCaffe, compared with the Caffe CPU-only mode using different BLAS libraries.



(a) Forward pass only.



(b) Forward and backward passes.

Figure 7. GoogLeNet benchmark results using clCaffe, compared with the Caffe CPU-only mode using MKL.

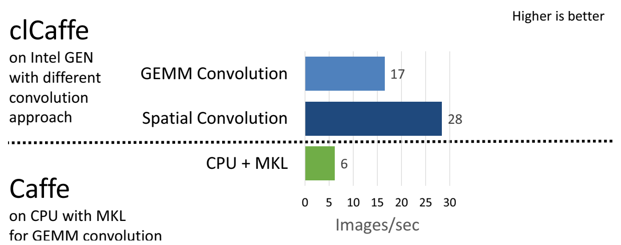


Figure 8. AlexNet benchmark results using clCaffe, compared with the Caffe CPU-only using MKL on Intel Cherry Trail (Forward pass only).

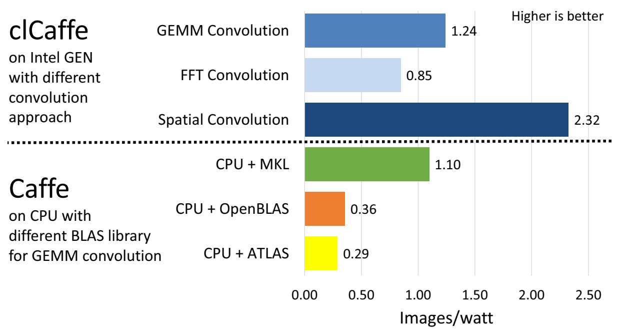


Figure 9. Power efficiency measurement for AlexNet forward pass only, compared with the Caffe CPU-only mode using different BLAS libraries.

when (filter size / stride) is greater than 3.

IV. Benchmark Results and Discussion

We evaluate the performance of clCaffe with AlexNet and GoogLeNet. While AlexNet has become almost as an industrial standard for speed evaluation, GoogleNet is also surging fast for its high accuracy, and smaller data communication model. We perform benchmarks on an Intel Broadwell system, equipped with an Intel® Xeon® processor CPU E3-1200 v4 @ 3.40GHz and Intel® Iris™ Pro graphics 6200, as well as Microsoft Surface™ 3 equipped with an Intel® HD Graphics (Cherry Trail) to demonstrate the value of clCaffe for small factor devices. The configuration details of our benchmark systems are listed in Table I. Time measurement was collected for both models with batch size 256 for AlexNet and 32 for GoogLeNet on Intel Broadwell, and batch size of 64 on Intel Cherry Trail. For power measurement, we used WattsUp power meters[20] to collect the power increase of the overall system during the benchmark by subtracting the base power consumption in the idle time from the total power. In this section, unless stated otherwise, all results were collected on the Intel Broadwell system.

Figure 6 shows AlexNet benchmark with our convolution approaches. In Figure 6(a), AlexNet forward-only benchmark results are compared with the performance of the Caffe CPU-only mode. Among CPU-only results, MKL (Intel® Math Kernel Library) provides the fastest performance among three BLAS libraries (MKL, OpenBLAS, ATLAS) that we evaluate, by classifying 65 images/sec. Three convolution approaches in clCaffe provide varying time performance. GEMM based convolution and spatial domain convolution show better forward pass speed than CPU-only using MKL. The fastest performance, 165 images/sec, from spatial domain convolution outperforms the CPU-only mode using MKL by a factor of 2.5×. For training including forward and backward passes, spatial domain convolution in clCaffe also presents better performance than Caffe CPU-only mode with MKL as shown in Figure 6(b), being capable of training 48 images/sec, thus exceeding best CPU-only speed by 1.7×. Moving to small mobile form factor devices, Figure 8 presents results of clCaffe for spatial and GEMM convolutions running on a Microsoft Surface^{TM} 3 (Intel Cherry Trail). When compared with the CPU-only result using MKL, which shows 6 images/sec, our spatial convolution is capable of classifying 28 images/sec. This 4.6× performance benefit enables CNN-based real-time applications on mobile devices with small form factors. When considering we depend on existing OpenCL based libraries for GEMM based convolution and frequency domain convolution, the better performance BLAS and FFT libraries provide, the higher speedup each convolution approach shows. For GoogLeNet benchmark as shown in Figure 7, all our convolution implementations outperform the Caffe CPU-only mode up to 3.4× in classification, and up to 1.4× in the training flow.

Results of power consumption comparison, for AlexNet forward pass only, is also presented in Figure 9. GEMM based convolution and spatial domain convolution classify 1.24 images/watt and 2.3 images/watt, respectively being 1.13× and 2.1× more power efficient than Caffe CPU-only mode using MKL. Note that we utilized a ISAAC library for GEMM based convolution in clCaffe from Figure 6 to Figure 9.

Based on our benchmark results, we clearly see the potential of OpenCL accelerated deep learning framework in deployment using a pre-trained model, specifically image classification in our experiments. Since giant-sized powerful GPUs are not always available on small form factor devices, this might be useful to speedup CNN use cases on such devices.

V. Conclusions and Future Work

In conclusion, by extending a popular and well-known CNN framework with OpenCL support, as well as offering multiple convolution engines, clCaffe enables a much more diverse ecosystem of devices and network models which can be utilized for CNN processing. Looking at our performance results, we found that depending on the device and CNN network model our optimization efforts were able to increase our performance on our test SoC's by 2.5$times$-4.6× over the CPU when using the integrated GPU. The framework is adaptable to support high-end SoC devices with integrated GPUs, as well as a number of smaller power envelope devices (i.e., ultra low power GPUs), many core CPUs, etc. clCaffe is available to download from https://github.com/01org/caffe/tree/clcaffe. Furthermore, our current framework provides a solid foundation on which to build further support for other OpenCL devices such as FPGAs and discrete accelerators.

While clCaffe provides a good initial implementation of OpenCL accelerated CNN, there still exist many avenues for future improvement. Work remains to be done to test and optimize for small form factor devices, including extending the auto tuning to allow much greater performance portability between architectures. Investigating efficient utilization of multiple heterogeneous OpenCL devices simultaneously would also be a good possible future work, which has the potential to offer great benefits in both the client and server spaces. Finally, we hope that clCaffe encourages more investment and development towards creating highly efficient libraries for OpenCL as we feel the lack of such libraries is one of the main obstacles preventing OpenCL from being more widely utilized.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[2] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, “Flexible, high performance convolutional neural networks for image classification,” in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1237.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] 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)*, pp. 1–42, April 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015.

[6] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the ACM International Conference on Multimedia*, ser. MM ’14. New York, NY, USA: ACM, 2014, pp. 675–678.

[8] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[9] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[10] H. Perkins. Deepcl: Opencl library to train deep convolutional neural networks. Accessed: Oct. 14, 2015.

[11] H. Perkins. cltorch: An opencl backend for torch. Accessed: Oct. 14, 2015.

[12] AMD research lab. Opencl-caffe: Opencl version of caffe developed by amd research lab. Accessed: Oct. 14, 2015.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[14] clblas: a software library containing blas functions written in opencl. Accessed: Oct. 14, 2015.

[15] P. Tillet. Isaac: Input specific architecture aware computations. Accessed: March. 7, 2016.

[16] clfft: a software library containing fft functions written in opencl. Accessed: Oct. 14, 2015.

[17] J. Salmon, M. Moraes, R. Dror, and D. Shaw, "Parallel random numbers: As easy as 1, 2, 3," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–12.

[18] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *CoRR*, vol. abs/1312.5851, 2013.

[19] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A GPU performance evaluation," *CoRR*, vol. abs/1412.7580, 2014.

[20] Wattsup power meter. https://www.wattsupmeters.com/. Accessed: Oct. 14, 2015.