Nallatech

a molex company

White Paper

FPGA Acceleration of Convolutional Neural Networks

Introduction

Convolutional Neural Networks (CNNs) have been shown to be extremely effective at complex image recognition problems. This white paper discusses how these networks can be accelerated using FPGA accelerator products from Nallatech, programmed using the Altera OpenCL Software Development Kit. This paper then describes how image categorization performance can be significantly improved by reducing computation precision. Each reduction in precision allows the FPGA accelerator to process increasingly more images per second.

Caffe Integration

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center and by community contributors.

The Caffe framework uses an XML interface to describe the different processing layers required for a particular CNN. By implementing different combinations of layers a user is able to quickly create a new network topology for their given requirements.

The most commonly used of these layers are:

- Convolution: The convolution layer convolves the input image with a set of learnable filters, each producing one feature map in the output image.
- Pooling: Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value.
- Rectified-Linear: Given an input value x, The ReLU layer computes the output as x if x > 0 and negative_slope * x if x <= 0.
- InnerProduct/Fully Connected: The image is treated as single vector with each point contributing to each point of the new output vector

By porting these 4 layers to the FPGA, the vast majority of forward processing networks can be implemented on the FPGA using the Caffe framework.

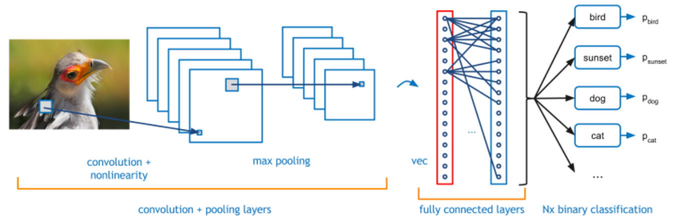

Figure 1 : Example illustration of a typical CNN network

To access the accelerated FPGA version of the code the user need only change the description of the CNN layer in the Caffe XML network description file to target the FPGA equivalent

ImageNet

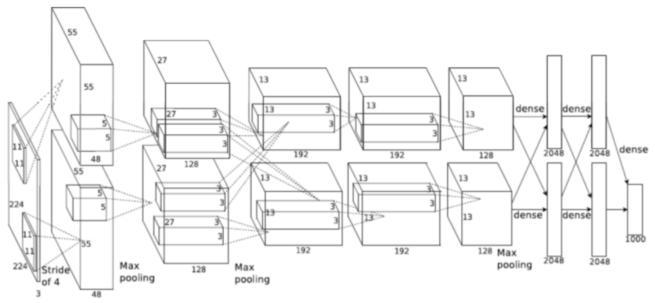
Figure 2 : ImageNet CNN

ImageNet is a well known and well used network, with freely available trained datasets and benchmarks. This paper discusses an FPGA implementation targeted at the ImageNet CNN, however the approach used here would apply equally well to other networks.

Figure 2 illustrates the different network layers required by the ImageNet CNN. There are 5 convolution and 3 fully connected layers. These layers occupy > 99% of the processing time for this network. There are 3 different filter sizes for the different convolution layers, 11x11, 5x5 and 3x3. To create different layers optimized for the different convolution layers would be inefficient. This is because the computational time of each layer differs depending upon the number of filters applied and the size of the input images. Therefore the smallest filter (3x3) is used as the basis for the larger convolutional blocks. The 3x3 convolutional layers are the most demanding due to the number of input and output features processed.

The larger filter sizes can be represented as multiple passes of the smaller 3x3 filters. This adds inefficiency into the processing, but allows for logic reuse between the different layers. The cost of this approach is illustrated in Table 1.

Convolution Size	Efficiency %
11x11	84
5x5	70
3x3	100

Table 1 : Convolution kernel efficiency

用3x3来做不同尺寸的卷积、以及FC

The 3x3 convolution kernel can also be used by the fully connected layers.

ImageNet Layer	Multiply Adds (M)
Convolution (11x11)	130
Convolution (5x5)	322
Convolution (3x3) 1	149
Convolution (3x3) 2	112
Convolution (3x3) 3	75
Inner Product 0	37
Inner Product 1	17
Inner Product 2	4

Table 2 : ImageNet layer computation requirements when using 3x3 filters

FPGA logic areas

FPGA devices have two processing regions, DSP and ALU logic. The DSP logic is dedicated logic for multiply or multiply add operators. This is because using ALU logic for floating point large (18x18 bits) multiplications is costly. Given the commonality of multiplications in DSP operations FPGA vendors provided dedicated logic for this purpose. Altera have gone a step further and allow the DSP logic to be reconfigured to perform floating point operations. To increase the performance for CNN processing it is necessary to increase the number of multiplications that be implemented in the FPGA. One approach is to decrease the bit accuracy.

Bit Accuracy

Most CNN implementations use floating point precision for the different layer calculations. For a CPU or GPGPU implementation this is not an issue as the floating point IP is a fixed part of the chip architecture. For FPGAs the logic elements are not fixed. The Arria 10 devices from Altera have embedded floating DSP blocks that can also be used as fixed point multiplications. Each DSP component can in fact be used as two separated 18x19 bit multiplications. By performing convolution using 18 bit fixed logic the number of available operators doubles compared to single precision floating point.

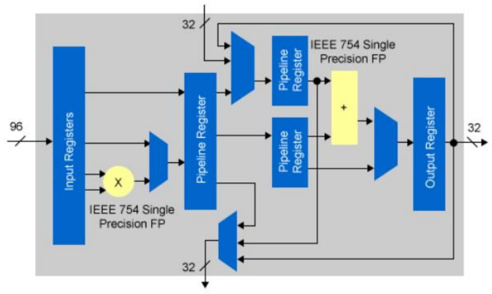

Figure 3 : Arria 10 floating point DSP configuration

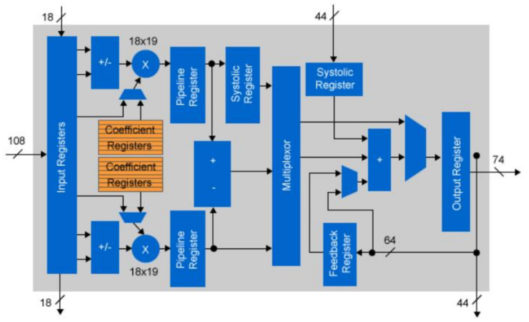

Figure 4 : Arria 10 fixed point DSP configuration

Depending upon the CNNs application tolerance, the bit precision can be reduced further still. If the bit width of the multiplications can be reduced to 10 bits or less, (20 bit output) the multiplication can then be performed efficiently using just the FPGA ALU. This doubles the number of multiplications possible compared to just using the FPGA DSP logic.

OpenCL library functions

Altera has provided the ability to include IP components into their compiler tool flow. This allows optimized functions to be created and included using standard library notation. The library components allow an experienced HDL programmer to create highly efficient implementations in the same way an assembly language programmer would for x86.

For the CNN layers used by ImageNet it was ascertained that 10 bit coefficient data was the minimum reduction that could be obtained for a simple fixed point implementation, whilst maintain less than 1% error versus a single precision floating point operation. Therefore an optimized library for a 10 bit 3x3 convolution was created. This library was then implemented as many times as possible, limited by FPGA resource available.

Resource	GX 1150
Logic Elements (K)	1,150
ALM	427,200
Register	1,708,800
Variable Precision DSP Block	1518
18x19 Multiplier	3036

Figure 5 : Arria 10 GX1150 resources

The Arria10's largest available device is the GX 1150. This device has resource for ~512 convolution blocks, plus the application control logic.

Implementation

Increasing the number of parallel convolution kernels increases the input bandwidth requirements. To avoid the global memory becoming a bottleneck multiple images are calculated at once allowing the convolution filter weights to be reused for each different image. This is particularly important for the fully connected layers where a new set of filter weights is required for each point to point connection, with the speed at which weights are retrieved from global memory to the bottleneck. Fortunately the convolution layers reuse the weight data for each point in a feature image. The smallest convolution feature image is 13x13 pixels, therefore the convolution weights need only be updated every 169 in the worse case.

Figure 6 : Nallatech 510T Accelerator

The target hardware for this implementation was the Nallatech 510T – a GPU-sized FPGA accelerator card compatible with most server platforms designed to support Intel Xeon Phi or GPGPU accelerators. The Nallatech 510T features two Altera Arria 10 GX 1150 FPGAs with ~60 GBytes/sec external memory bandwidth for loading weights, input and output data. Typical power consumption of the 510T is only 150W – less than half the power consumption of a high-end GPGPU. An added bonus of using 10 bit for the FPGA implementation is the tippling in the amount of weight data can be read from global memory versus floating point data.

Using the Nallatech 510T accelerator, 16 parallel images can be processed with each image having 64 kernels processed in parallel. This was achieved by generating 8 output features and 8 pixels per feature in parallel. This gives a total of 1024 parallel 3x3 kernels.
This was implemented by creating OpenCL kernel system for 1 image and replicating this as many times as possible dependent upon resource constraints. The convolution weights can be reused for each image so there is minimal increase to global memory requirements when scaling to multiple parallel images.

Results

By applying the above FPGA system, each image takes 9 millisecs to be categorized by the FPGA . With 12 parallel images handled by 510T this gives an average time of 748 usecs per image.

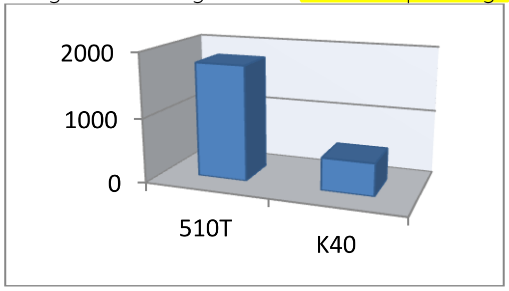

Figure 7 : Images categorized per second.
Nalllatech 510T versus Nvidia K40[1]

Power

The Nvidia K40 GPGPU has nominal power consumption of 235 Watts compared to the 150W of the Nallatech 510T. This gives the FPGA implementation a significant performance/power advantages versus the GPGPU.

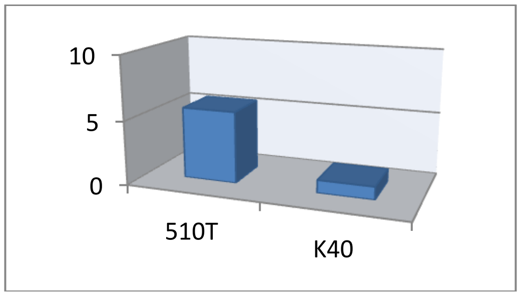

Figure 8 : Relative Image/Power.
Nallatech 510T versus Nvidia K40[1]

Conclusion

The unique flexibility of the FPGA fabric allows the logic precision to be adjusted to the minimum that a particular network design requires. By limiting the bit precision of the CNN calculation the number of images that can be processed per second can be significantly increased, improving performance and reducing power.

The non-batching approach of FPGA implementation allows single frame latency for object recognition, ideal for situations where low latency is crucial. E.g. object avoidance. Each image can be processed in 9 milliseconds. This permits images to be categorized at a frame rate greater than 100 Hz.

The scalability of the FPGA implementation allows complex CNNs to be implemented on increasing smaller and lower powered FPGA's at the expense of some performance. This allows lower performing applications to implemented on extremely low powered FPGA devices, particularly useful for embedded solutions, E.g. Near sensor computing.

Figure 9 : Miniaturized packaging module for near sensor processing

By packaging FPGAs with sensor hardware it is possible to use the power of CNN image recognition near the sensor ensuring low latency is maintained.

[1] Caffe implementation of ImageNet. http://caffe.berkeleyvision.org