2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors

A Massively Parallel Coprocessor for Convolutional Neural Networks

Murugan Sankaradas, Venkata Jakkula, Srihari Cadambi, Srimat Chakradhar,
Igor Durdanovic, Eric Cosatto, Hans Peter Graf
NEC Laboratories America, Inc. Princeton, NJ.

***Abstract*—We present a massively parallel coprocessor for accelerating Convolutional Neural Networks (CNNs), a class of important machine learning algorithms. The coprocessor functional units, consisting of parallel 2D convolution primitives and programmable units performing sub-sampling and non-linear functions specific to CNNs, implement a "meta-operator" to which a CNN may be compiled to. The coprocessor is serviced by distributed off-chip memory banks with large data bandwidth. As a key feature, we use low precision data and further increase the effective memory bandwidth by packing multiple words in every memory operation, and leverage the algorithm's simple data access patterns to use off-chip memory as a scratchpad for intermediate data, critical for CNNs. A CNN is mapped to the coprocessor hardware primitives with instructions to transfer data between the memory and coprocessor. We have implemented a prototype of the CNN coprocessor on an off-the-shelf PCI FPGA card with a single Xilinx Virtex5 LX330T FPGA and 4 DDR2 memory banks totaling 1GB. The coprocessor prototype can process at the rate of 3.4 billion multiply accumulates per second (GMACs) for CNN forward propagation, a speed that is 31x faster than a software implementation on a 2.2 GHz AMD Opteron processor. For a complete face recognition application with the CNN on the coprocessor and the rest of the image processing tasks on the host, the prototype is 6-10x faster, depending on the host-coprocessor bandwidth.**

I. INTRODUCTION

Machine learning involves embedding intelligence into applications to perform tasks such as recognizing scenes, semantics, and interpreting content from unstructured data. Intelligent classifiers with online learning to perform content-based image retrieval, semantic text parsing and object recognition will be key components of future server and embedded systems.

An important learning algorithm is the Convolution Neural Network (CNN) [1][2]. Originally used to handle recognition in documents and images, CNNs have recently been used in non-vision applications such as semantic analysis [3], thereby increasing their breadth and applicability. Although prior efforts have accelerated another machine learning algorithm, the Support Vector Machine[11][12], similar efforts for CNNs do not exist. CNNs are very compute- and data-intensive, performing hundreds of 1-D or 2-D convolutions and sub-sampling operations with significant intermediate data storage.

In this paper, we architect a massively parallel, programmable coprocessor for CNNs. Our goal is to demonstrate that significant performance improvements can be achieved at low power budgets. Even our high-end FPGA prototype dissipates less than 11W of power. While the target for our solution is an application-specific standard product (ASSP) for embedded domains with real-time constraints, servers where power is critical can also avail of our solution, perhaps even as an FPGA add-on card.

Two key attributes differentiate our work from prior efforts. First, our coprocessor is coupled with off-chip memory with a large bandwidth, implemented using several independently addressable memory banks. Second, we reduce data precision and pack multiple data words into each memory operation. These attributes enable us to implement a "stateful" coprocessor where off-chip memory is used to hold large intermediate state, a critical aspect for CNNs. The high latency of typical off-chip memory access is easily tolerated since the data access patterns are predictable, and all accesses can be pipelined. While several earlier efforts have mapped convolutions [5][6][7] as well as neural networks [8][9][10] to FPGAs, to our knowledge, this is the first effort aimed specifically at accelerating the entire convolutional neural network using a programmable coprocessor.

The large memory bandwidth also enables the use of the coprocessor in applications where multiple inputs are processed by one CNN. For example, a single server would not have the memory or IO bandwidth to examine video feeds from several cameras in real-time. An accelerator with direct connections to multiple camera feeds could potentially provide a more cost-effective solution.

1063-6862/09 $25.00 © 2009 IEEE
DOI 10.1109/ASAP.2009.25

Our coprocessor is based on several parallel convolution primitives and a controller that orchestrates data movement between off-chip memory and the convolution primitives. A CNN is decomposed into convolutions (that match the hardware primitives) and necessary data movement instructions to program the controller. We have a working prototype where the entire coprocessor is mapped to a single Xilinx Virtex-5 FPGA on an off-the-shelf PCI card with 1GB DDR2 memory. The coprocessor operates together with a host which can control the coprocessor using an API. We use the FPGA's hardwired DSP units to implement our basic processing elements, and construct convolution primitives by wiring them together.

The rest of this document is organized as follows. In Section II, we review relevant background and related work. In Section III, we describe the coprocessor architecture. In Section 0, we describe our prototype implementation. We present experimental results in Section V and conclude in Section VI.

II. Background and Related Work

In this section, we review the CNN algorithm and prior efforts related to our work.

A. Background

CNNs are neural networks with 2-dimensional topology. A feed-forward neural network consists of layers of neurons, with each layer feeding only the next layer, and receiving input only from the immediately preceding layer. Each neuron gets its inputs from the outputs of neurons in the previous layer, plus a bias whose value is usually 1. Every neuron applies a weight to each of its inputs, and adds the weighted inputs together. The sum is subjected to a "sigmoid" function whose purpose is to introduce non-linearity and limit the neuron's output to a reasonable range.

A convolutional neural network (CNN) uses small kernels of weights (neurons) that are convolved with a set of inputs. Thus, the neurons are shared across inputs when the kernel windows are moved over the inputs in each layer for the convolution. CNNs also have sub-sampling operations, where some outputs are removed by averaging the values of neighboring outputs. A degenerate case of CNNs is when kernels are of size 1 which results in a normal neural network[1][2].

Evaluating a trained CNN involves performing several convolutions and sub-samplings with considerable data movement. Convolutions constitute the performance bottleneck, but orchestration of data movement is also key for accelerating the computation. The training phase of the CNN, during which the kernel weights are determined, involves back propagation and weight updates. During back propagation, training examples along with expected outputs are fed to the network. The difference between expected and actual outputs is back-propagated through the network, and used to update the weights. One of the core computations during training is the convolution as well. Further details about training are not discussed here as the current work focuses on evaluation of trained CNNs (forward propagation) which is the task most commonly performed by end-users (to classify images, for instance), and often subject to real-time constraints.

B. Related Work

Algorithmic efforts to improve traditional neural network performance have been reported in [3] and FPGA implementations in [8][9][10]. Convolutions and image processing kernels on FPGAs have been extensively studied. For instance, in [5], the authors implement a Spartan-based 2-D convolution core. In [6], the goals are area and on-chip memory reduction for low-cost FPGA implementation. In [7], the authors propose architectural elements to trade-off off-chip memory bandwidth with on-chip memory for 2-D convolutions. [6] bears some similarity to our convolution primitive architecture. However, no prior efforts build a complete CNN virtual coprocessor that includes convolutions, sub-sampling, aggregation and other components. Our fully programmable coprocessor implements all components of any CNN.

Another key difference between our work and prior work is that our coprocessor efficiently uses *off-chip* memory as a scratchpad to manage the large intermediate data between CNN layers. We employ several independently addressable memory banks and lower data precision to achieve a high effective memory bandwidth. Together with predictable data access patterns, the large bandwidth enables us to make off-chip memory an efficient intermediate scratchpad, a feature that is critical for CNNs. This scales the coprocessor performance with data size. By contrast, if CNNs were implemented using a convolution coprocessor with limited memory and intermediate data were managed by a host, significant communication overheads would be incurred.

III. Coprocessor Architecture

CNNs in general comprise convolutional layers and sub-sampling layers. Each layer of a CNN has input planes and output planes, a plane being a collection of neurons. Output planes are produced by convolving or

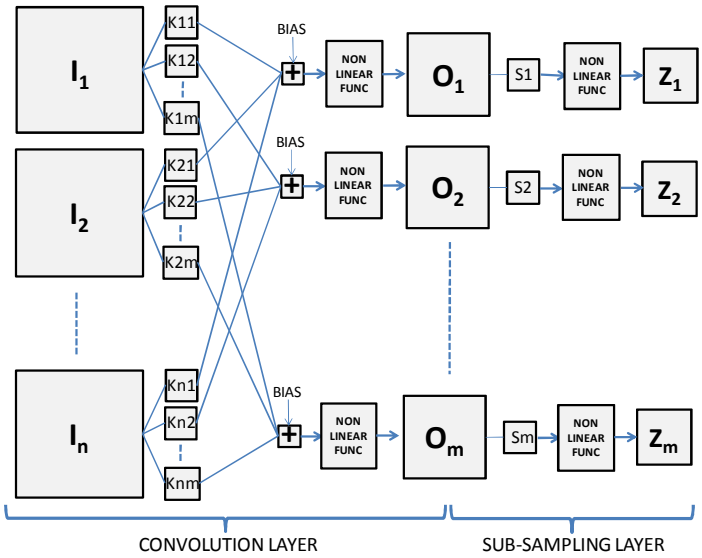

Figure 1: Typical structure of a convolutional neural network.

sub-sampling input planes. Generally, convolutional layers have *n* input planes $I_1 \ldots I_n$, *m* output planes $O_1 \ldots O_m$ and an *nXm* matrix of kernels *K11...Knm*, as depicted in Figure 1. Each of the *n* input planes is convolved with *n* corresponding kernels to produce *n* convolved planes, which are then summed along with a "bias" and subjected to a non-linear operation to produce one output plane. In the figure, output O_1 is produced by summing the convolutions of $I_1 \ldots I_n$ with *K11...Kn1* with a bias and finally using a non-linear function. The *n* input planes are then convolved with the second set of kernels *K12...Kn2*, and so forth. If the CNN has sub- sampling layers, next set of *n* kernels to create the second output plane, separate sub-sampling kernels may be used. In Figure 1, convolution output O_1 is sub-sampled with sub-sampling kernel *S1* and subjected to a non-linear operation to produce output *Z1*. In the simplest case, sub-sampling averages four neighboring intermediate outputs to produce a single output. Mathematically, the operations required to produce each output in a single layer are given by:

$$O_i = \tanh\left(bias + \sum_{j=1}^{n} I_j \blacksquare K_{ji}\right)$$

$$Z_i = \tanh(O_i . S_i)$$

where $I_j \blacksquare K_{ji}$ represents the convolution operation between I_j and K_{ji}, tanh the non-linear function, and $O_i . S_i$ represents O_i being sub-sampled according to kernel S_i.

A. Coprocessor-based CNN Evaluation

Here we describe the partitioning of the CNN between the host and the coprocessor. CNNs have very different I/O characteristics compared to traditional fully connected networks, which are essentially sequences of matrix-vector multiplications. **Error! Reference source not found.** shows computation and I/O requirements for scanning several convolution kernels over one input plane. A full network requires many such operations for one layer, with nonlinearities between layers. We map all operations onto the coprocessor, since intermediate results are re-used right away. An interesting metric to note is the computation to memory bandwidth ratio, approximately $n*k^2$ for *n kxk* kernels per input which is usually large indicating the compute-intensive nature of the algorithm.

In order to effectively use large off-chip memory as an effective intermediate storage, we reduce data precision. Originally, CNNs are computed with floating point arithmetic which is neither conducive to packing multiple words per memory operation nor is FPGA-friendly. We experimentally determined an acceptable lower fixed-point precision for our CNNs. In particular, we took a CNN for face detection and lowered the precision starting from double precision floating point until the results visually differed. We found that 20-bits fixed point sufficed for kernel weights and 16-bits for all other values in the input and output planes. This provides us significant performance gains while minimizing accuracy loss.

B. Coprocessor Architectural Template

Figure 2 shows a block diagram of our system with the CNN coprocessor implemented as a virtual processor on an FPGA. The coprocessor has access to its own memory resident on its card, as well as the coprocessor's internal on-chip memory. The host hands

off the CNN computation to the coprocessor, and collects the final results. Prior to the hand-off, the host is responsible for composing the input planes (for example, combining image feeds from multiple cameras) and reducing precision.

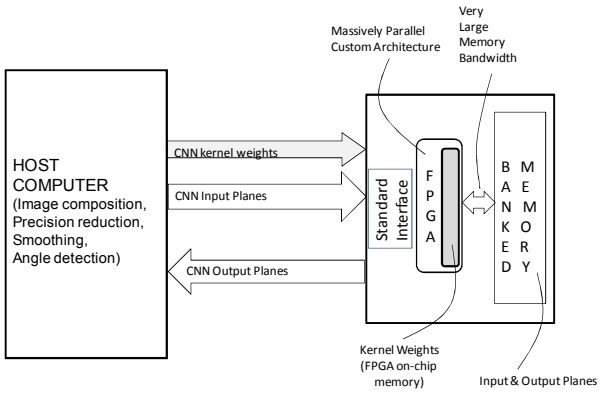

Figure 2: High-level view of our coprocessor system.

Our architecture is characterized by several key attributes. First, we choose a hand-optimized *kxk* 2D convolver as our primitive operator. In the current version, all convolvers are homogeneous, i.e., once determined, *k* is fixed across the coprocessor. Second, we organize the parallel architecture hierarchically as clusters of vector processing elements (VPEs), where a VPE cluster is an array of 2D convolvers, each of which is built using k^2 systolically connected multiply-accumulate (MAC) units. Third, we use banked off-chip data memory and introduce specific instructions to orchestrate data movement between the VPE clusters and the off-chip memory.

1) Architectural Details

The basic unit in our coprocessor is a hand-crafted primitive 2D convolver, shown in Figure 3. We build the convolver using basic vector or SIMD processing elements (VPEs). In our case, all VPEs in a convolver use a single instruction but operate on multiple data. The figure shows a *kxk* convolution unit with k^2+k VPEs, the final column of VPEs being used to sum partial results. Each VPE consists of a multiply-accumulator (MAC), a programmable register holding a constant weight, and programmable input and output delay units. The left side of the figure shows a series of input FIFOs that may be implemented using two-port memory blocks to feed the first PE in each row. Input image pixels stream into one FIFO, and subsequently into the other FIFOs as shown in the figure. It is easy to verify that this structure performs the 2D convolution operation.

We build the coprocessor using a hierarchical organization of VPEs. As shown in Figure 4, the massively parallel coprocessor has *MN* VPEs organized in *M* clusters, and serviced by *B* off-chip memory banks. Each cluster consists of *N* VPE arrays implementing convolver primitives of size *kxk*. A programmable microcontroller is used to interface VPE clusters to off-chip memory and a host. The microcontroller can be programmed to fetch instructions from certain locations in the off-chip instruction memory, load corresponding data from off-chip data memory, send commands to the different VPE clusters, receive and store results to specific locations in the off-

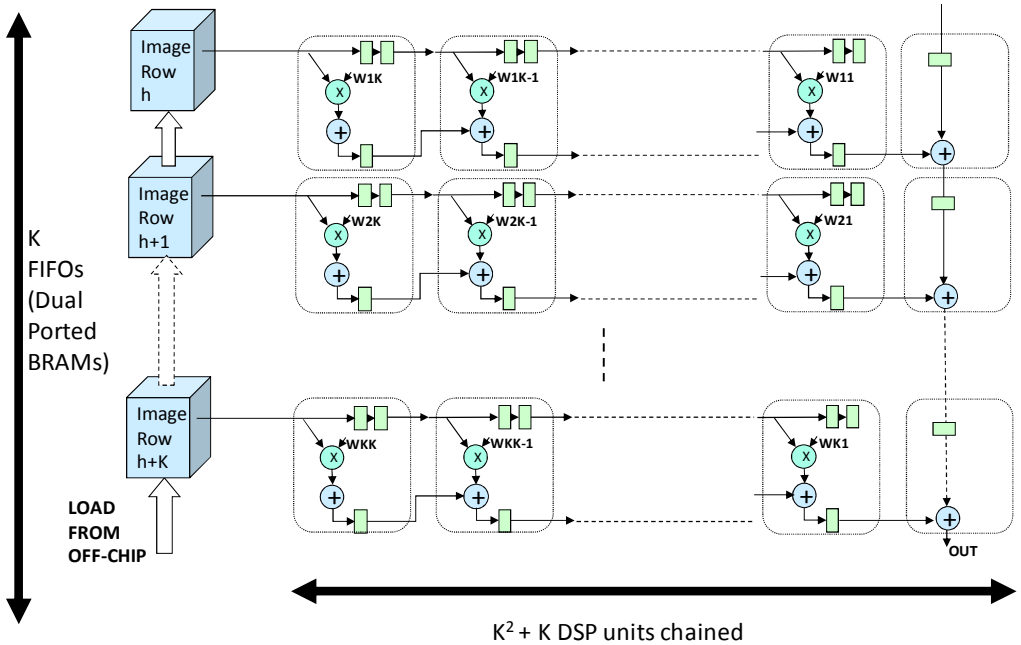

Figure 3: A VPE array implementing the primitive 2D convolver unit.

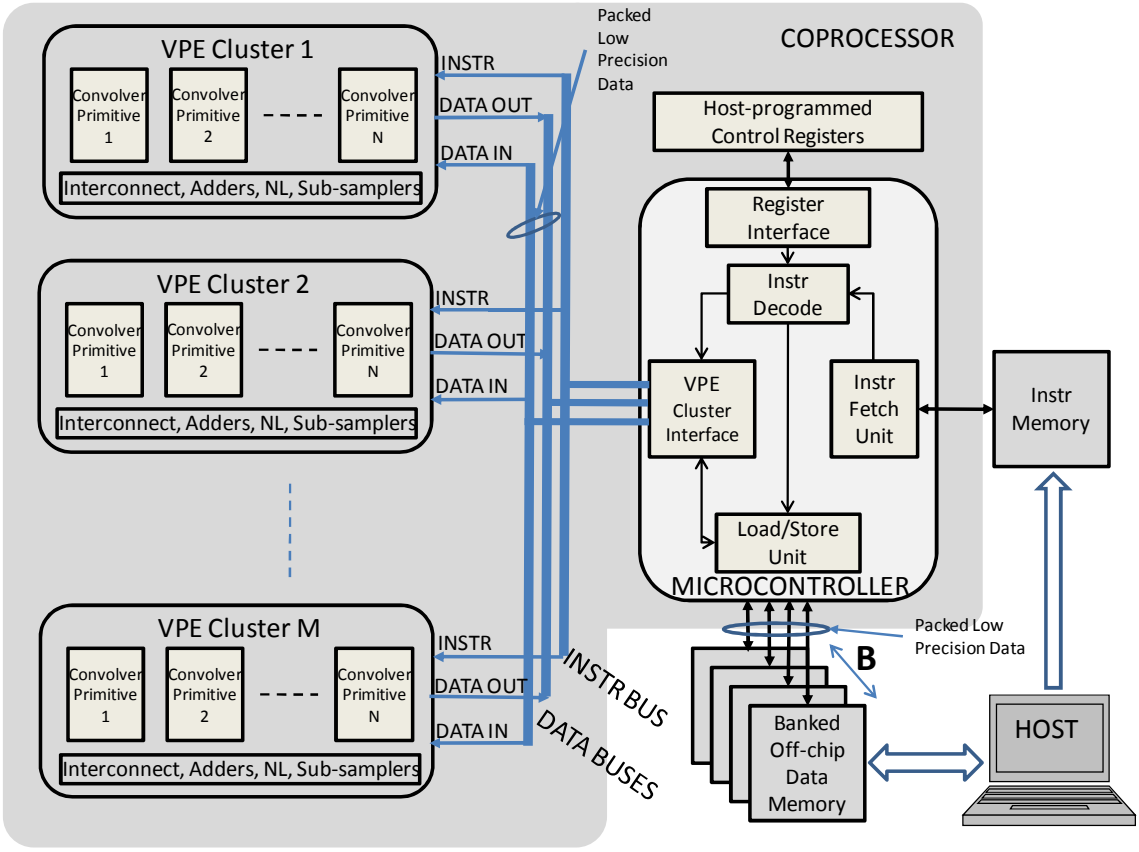

Figure 4: The overall massively parallel architecture with clusters of vector processing elements (VPEs) and microcontroller.

chip data memory. A key point to note is all data buses,oth internal and external to the coprocessor chip, are packed with multiple words of low precision data to increase data bandwidth.

The convolver primitives in a cluster are coupled with adders, sub-samplers and look-up tables. The adders compute partial sums of convolutions while the sub-samplers and look-up tables perform on-the-fly sub-sampling and implement the non-linear functions.

2) Programming

A CNN is processed into instructions for the microcontroller. Each instruction enables a VPE cluster to independently add or skip convolutions, summation, non-linearity and sub-sampling. All CNNs are mapped into the meta-operation represented by a VPE cluster.

Low-level programming is made possible using an API to populate control registers, instruction and data memory, and send results back to a host. The instruction set includes instructions to specify each CNN layer i.e., the number of input planes, number of output planes, their respective sizes as well as the number and sizes of all kernels. Given the sizes, the system loads data into suitable locations in the off-chip memory banks.

We improve performance by vectorizing and streaming data with minimal instructions. For example, each input image that needs to be convolved is only specified by its memory location and size. Using burst-mode access, the images are streamed into the VPEs. Similarly the outputs are streamed and stored with a single store instruction. We thus leverage the predictability of data access patterns in our application to simplify the hardware and improve performance.

3) CNN Pre-processor

Given a CNN, and the target architecture consisting of M VPE clusters, each with N kxk convolver primitives, the pre-processor's task is to decompose all convolutions into kxk convolutions, and organize the computations such that at most M groups of N convolutions are performed and summed together at a time. We do not discuss decomposing larger convolutions into kxk convolutions as that is beyond the scope of this paper. Rather, in each layer of the CNN, the pre-processor organizes the convolutions into M

groups of N, and allocates the input and output plane data to specific, discrete off-chip memory banks.

4) Performance Analysis

We analyze here the coprocessor performance. As per Figure 4, we assume the architecture has M VPE clusters, each with N kxk convolvers and B independently addressable off-chip memory banks each with a memory bus of width w. The CNN core computation shown in Figure 1 consists of n input planes, m output planes and nm kernels, each assumed to be of size no more than kxk. The computation requires a total of nm convolutions, $2m$ non-linear functions and m sub-samplings. On the coprocessor, each VPE cluster can perform and sum N convolutions in parallel, then execute one sub-sampling and associated non-linearity operations. Since the computation is streaming, the number of clock cycles required for this combined operation is $O(I\text{-}size)$ where $I\text{-}size$ is the size of the input plane. Note that we perform "on-the-fly" sub-sampling, i.e., every pair of output values is continually sub-sampled which makes storing entire intermediate images unnecessary.

Grouping n convolutions into groups of NM, we perform the operation of Figure 1 in $\lceil n/NM \rceil$ "rounds", where a round computes and stores the partial sum of NM convolutions in off-chip memory. If the data precision of each computation is p, each cluster requires a bandwidth of Np bits of input from memory and p bits of output. Since we do not store intermediate data, both inputs and outputs stream simultaneously requiring $\lceil (N+1)p/w \rceil$ ports per cluster With B off-chip memory banks, the number of rounds therefore increases to $\lceil n/NM \rceil * \lceil M(N+1)p/wB \rceil$. We note the data precision linearly affects performance: reducing precision linearly reduces the number of rounds.

For M clusters of size N with kxk convolver primitives, with sufficient memory bandwidth we can perform MNk^2 MACs per coprocessor clock cycle.

IV. Prototype

We designed a prototype of the coprocessor using an off-the-shelf FPGA board from Alpha Data[14]. The board has a Xilinx Virtex-5 LX330T device, 4 banks of DDR2 SDRAM of 256MB each, 2 banks of DDR2 SSRAM of 4MB each and a universal PCI host interface. The card is memory-mapped to 4MB of the host memory space of which 2MB is used to transfer data to and from the card's memory banks (in pages) and the rest mapped to a few FPGA registers for control and programming from the host.

We implemented a CNN coprocessor with a single VPE cluster containing 4 5x5 2D convolver primitive units. We used the FPGA's DSP units to implement the VPEs. Shown in Figure 5, the prototype implements a custom programmable pipeline with adders, sub-sampling and non-linearity units in addition to the convolution engines. With a clock speed of 115MHz and a total of 100 VPEs operating in parallel, the coprocessor is capable of a theoretical maximum of 11.5 GMACs per second.

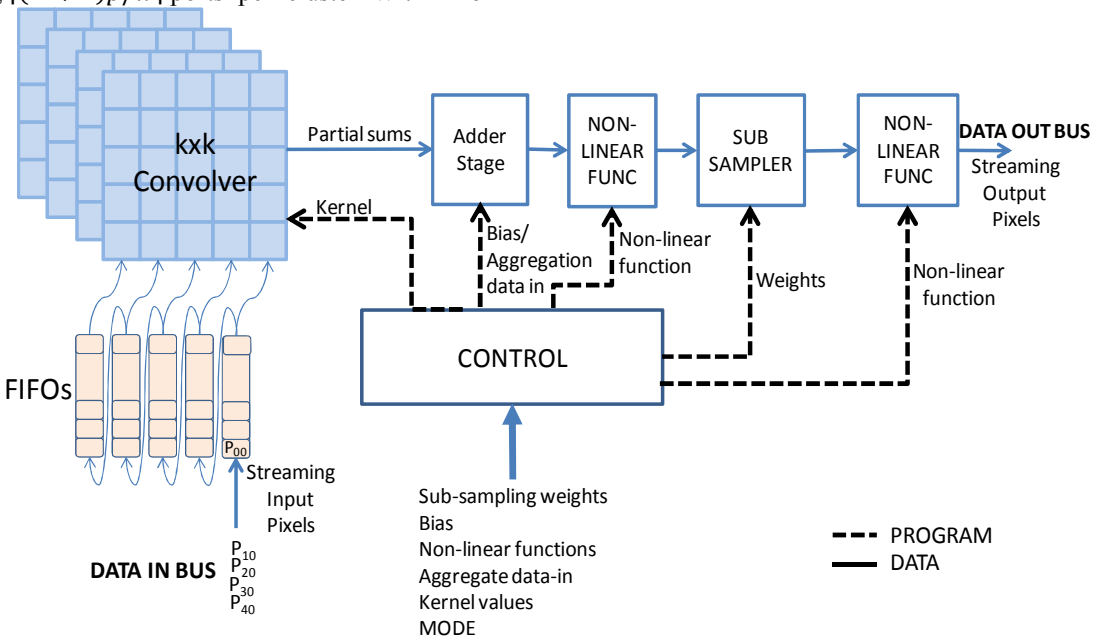

Figure 5: Prototype of FPGA-based CNN processor showing the details of the VPE cluster.

In order to use our prototype, a CNN is decomposed into primitive convolutions and data movement instruction (Section III.B.3)). The resulting convolutions are statically scheduled and mapped to the hardware. The adder stage computes sums of individual convolution results together with a programmable bias. The non-linear function, specified by the host via the API, is then applied to the summed results. A specialized sub-sampler hardware unit is available if required by the specific CNN being processed. Sub-sampling weights are also specified by the host, and programmed into the sub-sampling hardware.

V. Experimental Results

A. FPGA Resource Usage

Table 1: FPGA resource usage for prototype CNN processor.

	Maximum Clock (MHz)		DSPs	LUTs	Regs	Block RAM
	Proc Array	DDR2 Ctlrs				
CNN Processor: N=4, M=1, k=5,B=4	115	230	107 (55%)	35263 (17%)	39501 (19%)	3 (1%)

Table 1 shows the FPGA resource usage for our CNN processor prototype. The FPGA is a Xilinx Virtex-5 LX330T device with speed grade -2. The design ran at 115 MHz, while the DDR2 controllers, provided by Alpha Data along with the off-the-shelf card [14], were clocked at 230 MHz. Dual clock domains were necessary to match the widths of the data buses from the DDR2 controllers with the rest of the FPGA design. Overall, the resource usage for our prototype processor consisting of 1 VPE cluster with 4 5x5 convolvers and 4 off-chip memory banks was modest in terms of logic and registers. The DSP usage was, as expected, larger, since we used them to implement the VPEs.

B. Speedup, Power

Here we report the raw speedup achieved by the FPGA-based CNN processor. We use a CNN for a face recognition application. The application examines every 64x64 pixel "window" in each frame of VGA resolution moving images, and detects up to 1 full, un-obscured face per window. 9 bits are used to encode the presence of a face and its pose (angle). The inputs are VGA images (640x480 pixels) at a 16 bit per pixel resolution.

The CNN (Table 2) has four layers. The first two layers use both convolutions and sub-sampling. The third only has convolutions, while the fourth has neither (i.e., it is a regular neural network layer). A non-linearity operation follows every convolutional and sub-sampling layer. We employ fast look-up table approximations of the hyperbolic tangent (tanh) as the non-linear function. All sub-sampling operations perform simple averaging of neighboring pixels in a 2x2 square.

The first layer uses 8 kernels to produce 8 outputs. From these outputs, the second layer uses 160 kernels producing 20 outputs (i.e., from Figure 1, n=8, m=20). The third layer uses 400 kernels to produce 20 outputs, which are then processed in the final layer to produce 9 final outputs. These outputs identify the presence of faces and their angular pose in 113x153 different 64x64 viewing windows of the input VGA image.

Table 3 summarizes our results. We compare the FPGA speed a non-optimized software implementation of the CNN, and with an SSE2 optimized case where special SIMD instructions are used to perform many of the computations in the CNN. For the software, we used a 2.2 GHz AMD Opteron processor (single core), and for the hardware, we used our prototype Xilinx Virtex5 LX330T FPGA. Note these are "raw" speedup numbers and do not consider the one-time image transfer time to the hardware, and also other image processing tasks that may be required for the entire face recognition application (we present them in the next section). We observe a 31x speedup for the FPGA over the base software implementation, and nearly a 4x speedup over SSE2 optimized software. The total number of MACs required for evaluating one input image through this CNN is 0.54 billion, and the FPGA achieves a rate of 3.37 GMACs per second. Although our prototype is capable of achieving a theoretical peak of 11.5 GMACs per second, data movement between the VPE clusters and off-chip memory reduces the actual performance.

According to our power analysis tools (Xpower), the power estimation for the FPGA itself is about 7W, and for the on-board DDR2 SDRAM and SSRAM with 100% and 50% utilization respectively is 2.1W resulting in a total device and memory power of under 11W.

Table 2: Characteristics of the face recognition CNN.

LAYER	CONVOLUTION			SUB-SAMPLING	
	INPUT PLANES	KERNELS	INTERMEDIATE PLANES	KERNELS	OUTPUT PLANES
1	1 480x640	8 5x5	8 476x636	8..2x2	8 238x318
2	8 238x318	160 5x5	20 234x314	20..2x2	20 117x157
3	20 117x157	400 5x5	20 113x153	--	20 113x153
4	20..113x153	180 1x1	9 113x153	--	9 113x153

Table 3: Speedup of prototype for the CNN of Table 2

Base CNN Software		SSE2-Optimized CNN Software		Prototype CNN processor		Prototype Speedup	
Time (sec)	GMACs/sec	Time (sec)	GMACs/sec	Time (sec)	GMACs/sec	Over base-case	Over SSE2 optimized case
4.97	0.11	0.6305	0.86	0.16	3.37	31x	4x

C. End to End Application: Face Recognition

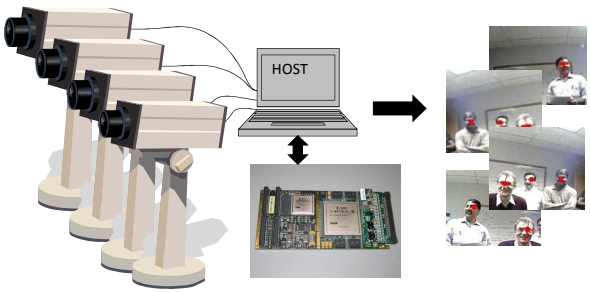

Figure 6: Setup for multiple camera face recognition.

We used our CNN processor to accelerate the full face recognition application with the CNN described in the previous section. The setup is shown in Figure 6. We used 4 cameras with different views, each producing images at QVGA (320x240) resolution. The host processor was responsible for composing the 4 images together into a single 640x480 image, and reducing the pixel precision from 24 to 16 bits. 3 layers of the CNN were run on the FPGA, while the last layer, a very small part of the CNN, was run on the host. The host was also responsible for peak detection as well as head pose angle decoding from the CNN result maps produced by the FPGA processor. We observed an overall speed of 6 frames per second using the coprocessor (compared to 1 frame per second on the 2.9 GHz Core 2 Duo host PC). The FPGA speed was limited by the PCI bandwidth and motherboard-related issues, which if resolved (for instance with PCI express), would increase the accelerator speed to 10 frames per second.

VI. Conclusions

We presented a massively parallel coprocessor for Convolutional Neural Networks (CNNs), a class of important machine learning algorithms. The coprocessor is characterized by parallel clusters of vector processing elements, each cluster built using hand-optimized 2D convolver units, along with other special-purpose hardware tailored to CNNs. A key attribute of our coprocessor is the use of off-chip memory located on the coprocessor card as a scratchpad for intermediate CNN data. This is made possible by architecting a high memory bandwidth and also lowering the data precision to pack multiple words per memory operation. We achieve speeds of 3.4 GMACs for CNN evaluation, 31x faster than a base software implementation. For a complete face recognition application, our prototype achieves a speedup of 6x over software.

References

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

[2] LeCun, Y.; Bottou, L.; Orr, G.; Muller, K., "Efficient BackProp", in Orr, G. and Muller K. (Eds), *Neural Networks: Tricks of the Trade,* Springer, **1998**.

[3] Simard, P.; Graf, H. P.,"Backpropagation without Multiplication", *NIPS 1993*: pp 232-239.

[4] Collobert, R.; Weston, J., "A unified architecture for natural language processing: deep neural networks with multitask learning," *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, vol. 307, pp.160-167, Jul 2008.

[5] Benkrid, K.; Belkacemi, S., "Design and implementation of a 2D convolution core for video applications on FPGAs," *Digital and Computational Video, 2002. DCV 2002. Proceedings. Third International Workshop on*, pp. 85-92, 14-15 Nov. 2002.

[6] Cardells-Tormo, F.; Molinet, P.-L., "Area-efficient 2-D shift-variant convolvers for FPGA-based digital image processing," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol.53, no.2, pp. 105-109, Feb. 2006.

[7] Hui Zhang; Mingxin Xia; Guangshu Hu, "A Multiwindow Partial Buffering Scheme for FPGA-Based 2-D Convolvers," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol.54, no.2, pp.200-204, Feb. 2007.

[8] Coric, S.; Latinovic, I.; Pavasovic, A., "A neural network FPGA implementation," *Neural Network Applications in Electrical Engineering, 2000. NEUREL 2000. Proceedings of the 5th Seminar on*, pp.117-120, 2000.

[9] Savich, A.W.; Moussa, M.; Areibi, S., "The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study," *Neural Networks, IEEE Transactions on*, vol.18, no.1, pp.240-252, Jan. 2007.

[10] Gironés, R. G.; Palero, R. C.; Boluda, J. C.; Cortés, A. S., "FPGA Implementation of a Pipelined On-Line Backpropagation," *J. VLSI Signal Process. Syst.,* vol. 40, no. 2, pp.189-213., Jun 2005.

[11] Catanzaro, B.; Sundaram, N.; Keutzer, K., "Fast Support Vector Training and Classification on Graphics Processors," *Machine Learning, 25th International Conference on, (ICML 2008)*, Jul. 2008.

[12] Durdanovic, I., Cosatto, E.; Graf, H., "Large Scale Parallel SVM Implementation", in L. Bottou, O. Chapelle, D. DeCoste, J. Weston (eds.), *Large Scale Kernel Machines*, pp. 105-138, MIT Press.

[13] Zhuo, L.; Prasanna, V. K., "High Performance Linear Algebra Operations on Reconfigurable Systems", in *ACM/IEEE Conference on Supercomputing*, *Proceedings of the 2005,* November 2005.

[14] www.alpha-data.com/adm-xrc-5t2.html.