Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

FPGA and GPU-based acceleration of ML workloads on Amazon cloud - A case study using gradient boosted decision tree library

Maxim Shepovalov, Venkatesh Akella [*]

Department of Electrical & Computer Engineering University of California, Davis, CA 95616, USA

A R T I C L E I N F O

Index terms:
Cloud computing
FPGA
GPU
Machine learning
Gradient-boosted decision trees
Classification

A B S T R A C T

Cloud vendors such as Amazon (AWS) have started to offer FPGAs in addition to GPUs and CPU in their computing on-demand services. In this work we explore design space trade-offs of implementing a state-of-the-art machine learning library for Gradient-boosted decision trees (GBDT) on Amazon cloud and compare the scalability, performance, cost and accuracy with best known CPU and GPU implementations from literature. Our evaluation indicates that depending on the dataset, an FPGA-based implementation of the bottleneck computation kernels yields a speed-up anywhere from 3X to 10X over a GPU and 5X to 33X over a CPU. We show that smaller bin size results in better performance on a FPGA, but even with a bin size of 16 and a fixed point implementation the degradation in terms of accuracy on a FPGA is relatively small, around 1.3%–3.3% compared to a floating point implementation with 256 bins on a CPU or GPU.

1. Introduction

The increasing computational requirements of next-generation applications coupled with the diminishing improvements in performance through technology scaling has resulted in heterogeneous computing environments, with increasing use of GPUs and FPGAs as accelerators. This trend has percolated even to cloud computing platforms with the leading vendors such as Amazon (AWS) and Microsoft (Azure) adding FPGAs to their offerings over the past year. For example, Xilinx FPGAs (up to eight Virtex UltraScale + VU9P FPGAs with a combined peak compute capability of over 170 TOP/sec) are now available on the Amazon Elastic Compute Cloud (EC2) F1 instances, along with Xilinxs SDAccel Development Environment for cloud acceleration, enabling the user to easily and productively develop accelerated algorithms and then efficiently implement and deploy them onto the heterogeneous CPU-FPGA system. The main advantage of cloud-based acceleration is the ability to scale on demand as the computation requirements increase due to larger datasets or due to latency constraints. Furthermore, there is no upfront cost in terms of tools and hardware. The customer pays only for the amount of resources used and for the duration of use, which makes high performance computing available to a broad range of consumers including small and medium companies that do not have in house networking and hardware expertise.

General-purpose GPUs take advantage of SIMD and SIMT style parallelism and have been in use for over a decade now with a wide body of knowledge on their costs, benefits, and design tradeoffs. They have become the de facto platform of choice especially for implementing machine learning applications. In CPUs/GPUs cache subsystem is fixed (size and connectivity) and managed automatically by the hardware. So, applications that have poor data locality and little data reuse or large working sets exhibit poor performance and poor scalability. FPGAs in contrast have embedded memory blocks called Block RAMs which can be used to create user managed caches tailored to the specific algorithm. Moreover, given that BlockRAMs are in close proximity to the custom compute logic, FPGAs naturally support an in-memory computing paradigm which can be beneficial in many machine learning applications. Recent work [1–4] has shown that FPGAs can be quite *effective* in a variety of high performance computing applications, especially data analytics and networking. But to the best of our knowledge there has been no direct comparisons between highly optimized GPU and FPGA implementations of large scale machine learning applications especially in a cloud computing setting. So, the main **motivation** for this work is to explore the design and cost trade-offs and scalability of FPGA based acceleration of computing in the cloud environment and compare it with optimized CPU and GPU implementations, so that application developers can make more informed decisions about how to

* Corresponding author.
E-mail addresses: mshepovalov@ucdavis.edu (M. Shepovalov), akella@ucdavis.edu (V. Akella).

https://doi.org/10.1016/j.vlsi.2019.09.007
Received 15 January 2019; Received in revised form 22 August 2019; Accepted 13 September 2019
Available online 18 September 2019
0167-9260/Published by Elsevier B.V.

optimize cost/performance in a heterogeneous cloud computing environment.

The intended audience of this work are

- Machine learning users who have large datasets and are currently using GPUs in the cloud for accelerating their applications.
- Cloud computing vendors such as Amazon who are interested in offering new types microservices such as FaaS (Function as a Service) and would like to know the cost benefits tradeoffs of using FPGAs.

Specifically we are interested in answering the following questions in this paper to address the concerns of potential users and vendors of cloud computing services. What are the cost/performance tradeoffs since there is a wide disparity in terms of cost per hour between CPUs, GPU, and FPGAs? How well does an application scale with increasing number of FPGAs? What speedups are necessary for a FPGA based implementation to be competitive compared to a GPU? How does one develop, debug, and deploy FPGA based accelerators in the cloud? Traditional approaches to HW/SW codesign are not directly applicable to cloud-based acceleration, because the FPGA/CPU interface is different. Also, the amount of memory available on the accelerator, and how the memory can be used, are pre-defined by the cloud provider. Also, in large scale machine learning there is often a complex software infrastructure with multiple libraries for data pre/post processing, evaluation, etc. which usually means one has to be more careful in choosing where/how to partition the application. So, kernels have to be selected carefully to not only match the accelerator architecture, but also the memory requirements and the interconnect subsystem. Though there has been prior work in implementing and optimizing machine learning kernels on *stand-alone* FPGAs, the questions mentioned above have not been answered in research literature yet, because FPGAs have only been available in the cloud computing setting in the past year or so.

We present a *case study* using a widely used machine learning library called LightGBM [5]. LightGBM is an efficient implementation of gradient boosted decision trees that achieves extremely high performance in a variety of applications such as multiclass classification, click prediction, regression, and ranking etc. Zhang et al. [6] developed state-of-the-art GPU implementation of LightGBM[1] that serves as a concrete comparison point for our work. We propose to use an FPGA instead of a GPU to accelerate the performance critical kernels from this implementation as described in Section 2. By using the same starting point in terms of code base, we propose to evaluate if, and when FPGA based acceleration is cost effective compared to a GPU based acceleration.

The main contributions of this work are as follows:

1. We make a direct comparison of an FPGA implementation with a *highly optimized* GPU implementation using the same code base. We show that on performance critical kernels, FPGA implementation using AWS F1 instances results in a 3X to 10X speed-up compared to a GPU.
2. Our results indicate that look-up-tables (LUTs) (not the Block RAMs) are a limiting to improving the performance on a single FPGA via thread-level parallelism.
3. Exploiting thread-level parallelism on a FPGA is challenging compared to a GPU because there are no atomic operations in a FPGA. We propose an address remapping technique that takes advantage of the large amount of memory available on the F1 instance that avoids memory conflicts.
4. We show that the performance of the FPGA-based implementation does not scale well on AWS at present because there is a single memory that is shared by all the FPGAs. This becomes the bottleneck. As a result, when going from 1 to 8 FPGA, the performance improves only modestly (on average about 2.6X), though the cost per hour increases linearly. So, clearly in future F1 instances a better memory system architecture is required.
5. Though there is a significant improvement in the performance of the compute-intensive kernels compared to a GPU, the overall speed of the *entire* application is far more modest because of Amdahl's law argument. So, the cost of FPGA computing per hour has to come down drastically before FPGAs can become truly competitive with GPUs in the cloud.

[1] https://github.com/huanzhang12/lightgbm-gpu.

The rest of the paper is organized as follows. We start with an overview of decision trees and their computational bottlenecks on a CPU and GPU. Next, we provide the background on AWS platform, and the cost model. Next, we provide details of our implementation followed by detailed results and discussion of the results. We conclude with directions for future work.

2. Background

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal but are also a popular tool in machine learning. Gradient Boosted Decision Tree (GBDT) is an ensemble model of decision trees [7,8], which are trained in sequence. In each iteration, GBDT learns the decision trees by fitting the negative gradients (also known as residual errors). The main cost in GBDT lies in learning the decision trees, and the most time-consuming part in learning a decision tree is to find the best split points. One popular method to improve the training time to use histogram based techniques (for example LightGBM [5]) that buckets continuous feature values into discrete bins and uses these bins to construct feature histograms during training. Histogram is in fact a set of histograms for sample counter, gradient and hessian values. Result of histogram splitting is used to generate new leaf of the tree and prepare new data for generating next layer of the tree.

The computational bottlenecks in the implementation of LightGBM are shown in Table 1. This data was obtained by Ref. [6] through instruction level profiling on a CPU. Function BeforeFindBestSpilt() mainly spends its time on generating three arrays - the indices of the training samples on this leaf and the corresponding hessian and gradient values for them. Its time complexity is O(N), where N is the number of samples at the current leaf. Function ConstructHistogram() goes over one feature at a time to construct a feature histogram and has a complexity of O(N.d) where N is the number of samples and d is the number of features. Function FindBestThreshold() finds the best split point and there are d calls to this function for a total complexity of O(k.d) where k is the number of bins. The data shows that 85% of the computation time is spent in the four functions identified in Table 1, with the ConstructHistogram() function taking up almost 80% of the time. So, we will focus on implementing ConstructHistogram() on the FPGA. In general a sequential implementation of histogram computation is quite trivial. It involves merely reading a value from the memory and incrementing the memory location corresponding to the value. However, implementing the ComputeHistogram() function in LightGBM is challenging for the following reasons. First, we need to compute a histogram for each feature. In general the number of features can be quite large (thousands or tens of thousands) for some datasets. Second, the histogram has to be computed for the training samples for a given leaf node during the

Table 1
Computational Bottleneck in LightGBM GBDT Estimated via Instruction level Profiling in Ref. [6].

Function Name	Higgs	epsilon	Yahoo-LTR
BeforeFindBestSplit()	5.3%	<0.1%	0.3%
ConstructHistogram()	80.8%	87.6%	75.7%
FindBestThreshold()	0.4%	5.1%	6.8%
Split()	6.3%	<0.1%	4.3%

Table 2
Different HW instances on AWS cloud.

	AWS instance Oregon	CPU	cores	Caches				CPU's RAM	accelerator	Accelerator's RAM
				L3	L2	L1i	L1d			
CPU	C4.4xlarge	Intel Xeon E5-2666 v3 2.90 GHz	8	25600k	256k	32k	32k	29 GB	–	–
GPU	P2.xlarge	Intel Xeon E5-2686 v4 2.30 GHz	2	46080k	256k	32k	32k	59 GB	NVIDIA Tesla k80	12 GB
1 FPGA	F1.2xlarge	Intel Xeon E5-2686 v4 2.30 GHz	4	46080k	256k	32k	32k	119 GB	Xilinx Virtex Ultra-Scale + AWS VU9P F1	64 GB
8 FPGAs	F1.16xlarge	Intel Xeon E5-2686 v4 2.30 GHz	16	46080k	256k	32k	32k	960 GB	Xilinx Virtex Ultra-Scale + AWS VU9P F1	64 GB on each FPGA

Table 3
Typical cost associated with different instance types (as of October 2018).

	AWS instance Oregon	EC2 $/hour	EC2 $/month	EBS $/month for 90 GB	upload $0.09/Gb for 2 GB	Total, $/month
CPU	C4.4xlarge	0.796	592.224	9	0.18	601.40
GPU	P2.xlarge	0.9	669.6	9	0.18	678.78
1 FPGA	F1.2xlarge	1.65	1227.6	9	0.18	1236.78
8 FPGAs	F1.16xlarge	13.2	9820.8	9	0.18	9829.98

tree building process. The training samples for a given leaf node are not located contiguously in memory. As a result, the computation involves non-sequential scattering access to a large memory. Third, parallelizing the histogram computation is non trivial. Not only does it require significantly higher memory bandwidth to read multiple items in parallel and write them, but also could result in read/write conflicts, if the same data element is being read each parallel thread. This is not infrequent as many data items are usually quite similar (for example, 0).

GPU implementation proposed in [6] overcomes these challenges by taking advantage of the thread-level parallelism and GPU atomic operations. Since non-sequential scatter access to feature arrays is expensive, histograms corresponding to multiple features are computed simultaneously. Next, the number of read/write conflicts is reduced by staggering the computation of the gradient and hessian operations of each feature in the bundle of threads. However, it is still possible for the same value to be updated by multiple threads, so atomic operations are necessary. The amount of parallelism on a GPU (number of histograms that can be computed in parallel) is dictated by the size of the local memory. The ConstructHistogram() implementation on a GPU in [6] delivers a speed up between 7 and 8 compared to a CPU based implementation on a 28 core Xeon E5-2683 with 192 GB of memory and a speed-up of 25 over the exact-split finding algorithm of XGBoost [9].

In this work we propose a scalable parallel implementation of the ConstructHistogram() function on a FPGA that is significantly faster than the GPU implementation. We achieve the speed-up over a GPU by eliminating the von Neumann overhead (fetching, decoding, storing back results) by realizing the underlying computation directly in hardware with look-up tables and by scheduling the memory reads and writes to avoid conflicts.

However, before delving into the details of the FPGA implementation we will provide a brief overview of the AWS platform and the cost model.

3. AWS platform

AWS is a service managed by Amazon.com that provides on-demand cloud computing platforms to individuals, companies and governments. This service allows its customers to instantly make a virtual server on Amazons resources instead of manually purchasing hardware, building networking solutions, and then dealing with maintenance and power costs. Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. EC2 instances can be set up according to chosen Amazon Machine Image (AMI). AMI is a template that contains a software configuration (for example, an operating system, an application server, and applications). Hardware selection should reflect chosen AMI. There are several types of instances: T: general computing, R: memory optimized, C: compute optimized, P: GPU accelerated, F: FPGA accelerated. Each AMI can work on more than one hardware configurations. This project is using F1.2xlarge for 1 FPGA solution and F1.16xlarge for 8 FPGA solution. For comparison with CPU only and GPU implementations, data was collected on C4.4xlarge and P2.xlarge. Table 2 shows the available resources of different instances CPU, GPU, and FPGA. Table 3 shows the cost of using the different resources.

The total cost per month is derived assuming 730 h per month, and that the application uses 90 GB of EBS storage and uploads 2 GB data. Note that the cost of using a GPU and CPU are about the same, while an FPGA is almost 2X compared to a CPU/GPU.

4. FPGA implementation of ConstructHistogram() on AWS platform

Our OpenCL kernel for implementing ConstructHistogram() computation is shown in Fig. 1.

The implementation has five steps. The first step is to copy histogram data from global memory to FPGA Block RAM. On the AWS platform the host CPU can only access system level DDR (global memory), so the FPGA has to allocate and manage the Block RAMS (BRAM) that hold the partial histograms. So, the first step in the FPGA implementation is for each thread to fill its part of the BRAM with zeros. This is necessary because the kernel is not reloaded between calls, so the BRAM is not automatically cleared between invocations of the kernel. In addition, a specific thread is used to populate the first k values

```
Input: N - number of data for processing
Input: d, i, g, h: arrays for bins, indices, gradients, and hessians
Input: ofs_inp: offset for input
Input: ofs_out: offset for output
Output: c_out, g_out, h_out: sample counter, gradient, and hessian histograms
All threads on FPGA are running simultaneously:
t_id <- thread ID
local_count <- {}
local_hess <- {}
local_grad <- {}
for j in numBins do
    if (t_id == 0) and (ofs_inp == 0) do
        local_count[t_id*numBins + j] <- c_out[j]
        local_hess[t_id*numBins + j] <- h_out[j]
        local_grad[t_id*numBins + j] <- g_out[j]
    else do
        local_count[t_id*numBins + j] <- 0
        local_hess[t_id*numBins + j] <- 0
        local_grad[t_id*numBins + j] <- 0
synchronization of threads
for j <- t_id to N with step j <- j + numThreads do
    idx <- i[ofs_inp + j]
    bin <- d[idx]
    local_count[t_id*numBins + bin] <- local_count[t_id*numBins + bin] + 1
    local_hess[t_id*numBins + bin] <- local_hess[t_id*numBins + bin] + h[ofs_inp + j]
    local_grad[t_id*numBins + bin] <- local_grad[t_id*numBins + bin] + g[ofs_inp + j]
synchronization of threads
for k <- numThreads/2 to 1 with step k <- k/2 do
    if (t_id < k) and (t_id + k < numThreads) do
        add 3 local arrays from [t_id + k] to [t_id]
    synchronization of threads
if t_id < numBins do
    copy counter, hessian and gradient values for bin t_id from local memory to global
```

Fig. 1. OpenCL kernel for FPGA implementation ConstructHistogram().

of BRAM arrays where k is the number of bins in the histogram. Next, we remap the bin address to avoid data races as described below. The third step involves computing the partial histograms followed by reducing the partial histograms into a single final one. The last step involves copying the results from the Block RAM back to the CPU global memory.

The main idea in the proposed implementation is to use hardware parallelism and fast low-latency BRAM to build one histogram for one feature as fast as possible. The kernel on the FPGA distributes the task of building histogram to multiple threads. Each thread will take only part of input data to build a partial histogram. As all threads operate simultaneously, it is possible to have data races. For example, consider the case when thread #0 and thread #14 are both processing bin 3 in the same 64 bin histogram. They will try to increment the value at index 3 in histograms counter array, as well as increase hessian and gradient values for the same bin. Data races can be avoided by atomic operations on a GPU. Since FPGAs do not support atomics, we propose to address this by forcing each thread to write to non-overlapping parts of the memory. Each thread finds its actual index as follows: *Index_in_BRAM* = *bin_number* + *threadID* ∗ *number_of_bins*

Using the example above, thread #0 will increment bin #3 at index 3 + 0*64 = 3, but thread #14 will increment bin #3 at index 3 + 14*64 = 899. Memory requirements for the histogram computation scale linearly with the number of threads. For each bin we need storage for 3 values with each value being 4 bytes since LightGBM is a 32 bit application. So, a histogram with 64 bins require 768 bytes. Since each thread has its own histogram, with N threads, the total storage will be 768 ∗ N. Virtex UltraScale + VU9P FPGA in the AWS F1 instance has almost 43 MB of embedded memory, so the storage required is unlikely to be a bottleneck.

Next let us consider how to parallelize the implementation across multiple FPGAs (as noted before, AWS offers an 8 FPGA configuration). FPGAs on AWS can share their global memory between each other through a dedicated interconnection network without having to use the PCIe network. We take advantage of this dedicated network to avoid recalculation of the same values on different FPGAs, when parallelizing the implementation across multiple FPGAs. The same kernel can be used for all the FPGAs but with different offsets and different number of items for each FPGA. Upon completion, each FPGA writes its histogram to a separate location in the output array. Host needs to make an additional kernel call to reduce these in to the final result.

We assume a 32-bit fixed point implementation with 14 bits for the signed integer part and 18 bits for the fraction. In the FPGA implementation, the first stage of processing involves converting floating point values to this fixed point representation and the last stage of the computation involves converting it back to the fixed point implementation. Note, that in general it may not be necessary to use a 32-bit fixed point representation and one could optimize the hardware by reduc-

Table 4
FPGA utilization.

kernel type	# of threads	BRAM	DSP	FF	LUT
256 bins	94	2790(64.6%)	658(9.6%)	870722(36.8%)	1180452(99.8%)
64 bins	110	700(16.2%)	770(11.3%)	877910(37.1%)	1177110(99.6%)
16 bins	124	178(4.12%)	868(12.7%)	883500(37.4%)	1173288(99.2%)
Available		4320	6840	2364480	1182240

Table 5
Datasets used and their characteristics.

Datasets	Higgs	Microsoft –LTR	Yahoo-LTR	Epsilon	Expo	Bosch
Training Examples	10,000,000	2,270,296	473134	400,000	10,000,000	1,000,000
Features	28	137	700	2000	700	968

ing the number of bits with a concomitant decrease in the accuracy [10,11].

Table 4 shows the utilization of the various FPGA resources as a function of the number of bins. The limiting factor for the number of parallel threads seems to be the LUTs (look-up tables). With 16 bins we can have at most 124 threads and with 64 bins we can have 110 threads.

Next we will describe the results in detail starting with the datasets, accuracy of CPU, GPU, and FPGA implementation due to fixed point implementation and histogram based implementation.

5. Results

5.1. Workloads

We use six datasets shown in Table 5 that represent a wide diversity in terms of the training set size and number of features. Higgs, Epsilon, Expo, and Bosch are large and dense datasets applied for classification tasks. Yahoo-LTR and Microsoft-LTR are for learning to rank tasks.

5.2. Accuracy results for CPU, GPU, and FPGA implementations

As noted above, the histogram based implementation of GBDT inherently tradeoff statistical efficiency for hardware efficiency because it uses a finite number of bins instead of the exact-split method for constructing the decision tree. However, the loss in accuracy is not substantial as we go from 256 to 16 bins. In a FPGA implementation, there is additional loss in accuracy due to fixed point implementation. As the results in Table 6 show compared to a GPU the loss in accuracy in a FPGA is about 1.3%–3.3% as we go from 256 bins to 16 bins for different data sets. The degradation in accuracy is negligible as we go from 1 to 8 FPGAs.

Table 6
Accuracy Tradeoffs for CPU, GPU, FPGA as function of Number of bins. AUC (Area Under Curve) and NDCG (Normalized Discounted Cumulative Gains) are performance metrics to evaluate convergence behavior of machine learning algorithms.).

device	CPU			GPU			FPGA			8 FPGAs		
# of bins	256	64	16	256	64	16	256	64	16	256	64	16
Higgs AUC	0.845432	0.844993	0.840329	0.845591	0.845322	0.839743	0.833548	0.833236	0.828189	0.833365	0.832991	0.828025
Yahoo NDCG1	0.735694	0.732814	0.731817	0.727822	0.729786	0.72847	0.721905	0.722437	0.72008	0.72174	0.722586	0.719938
Yahoo NDCG3	0.740624	0.737505	0.73644	0.736131	0.735911	0.734873	0.72697	0.725119	0.724288	0.726875	0.72501	0.72435
Yahoo NDCG5	0.757726	0.755246	0.75428	0.755084	0.75512	0.753234	0.745804	0.744193	0.743159	0.745671	0.744062	0.743343
Yahoo NDCG10	0.797791	0.796558	0.795503	0.795248	0.796341	0.79409	0.787431	0.787139	0.785684	0.787156	0.787325	0.785452
Microsoft NDCG1	0.525336	0.520073	0.516673	0.524474	0.520645	0.519187	0.512811	0.506948	0.507033	0.512918	0.506804	0.506865
Microsoft NDCG3	0.505291	0.503479	0.502523	0.506736	0.503699	0.503543	0.494055	0.490802	0.490604	0.49417	0.490696	0.49053
Microsoft NDCG5	0.510691	0.510509	0.506849	0.510965	0.510186	0.50872	0.497573	0.496481	0.495618	0.497417	0.496372	0.495439
Microsoft NDCG10	0.527825	0.526975	0.524425	0.528319	0.527829	0.525526	0.515626	0.514292	0.513335	0.515656	0.514428	0.513288
Epsilon AUC	0.950103	0.949912	0.947959	0.950024	0.949912	0.948065	0.928182	0.928094	0.926946	0.928241	0.927977	0.927128
Bosch AUC	0.684312	0.692864	0.687224	0.686276	0.688328	0.688559	0.665279	0.663734	0.663821	0.665322	0.663583	0.664038
Expo AUC	0.774791	0.76851	0.740438	0.773652	0.766492	0.741739	0.753369	0.740674	0.716671	0.753344	0.740636	0.716831

Table 7
Performance of CPU, GPU, and FPGA implementations of the ConstructHistogram() kernel that is a proxy for the training time. The units are in seconds.

device	CPU			GPU			FPGA			8 FPGAs		
# of bins	256	64	16	256	64	16	256	64	16	256	64	16
Higgs	1073	1058	909	371	335	329	130	104	95	45	41	36
Yahoo	821	710	571	324	204	131	113	67	45	37	25	19
Microsoft	978	966	922	577	491	476	212	166	151	65	54	50
Epsilon	4589	3938	3609	1183	463	270	345	150	110	137	101	53
Bosch	1901	1677	1514	257	184	137	65	58	53	26	25	20
Expo	706	682	629	338	330	363	87	51	37	24	20	19

5.3. CPU vs GPU vs FPGA performance

First, we validated our implementation by benchmarking the results of our GPU implementation with the implementation in Ref. [6]. The results agree though we used the Tesla K80 available on AWS while the reference implementation [6] used a GTX 1080. As noted in Section 2, ComputeHistogram() is responsible for about 80% of the training time in the LightGBM GBDT library. So, we will compare the performance of the ComputeHistogram() kernel on CPU, GPU, and FPGA instances available on AWS. The results are shown in Table 7. With more bins there is more parallelism to exploit, so training performance improves on all the platforms. To better understand the impact of bin sizes and datasets on performance, we present a graphic visualization of the data in Fig. 2. On a complex data set such as epsilon (where complexity is defined by the number of features), the improvement in performance with bin size 256 is 3.42X and 2.45X with a bin size of 16. On the Expo dataset, the improvement is more substantial, almost 10X with a bin size of 16 over a GPU.

In an accelerator-based implementation it is always useful to know the amount of time wasted due to transferring the data from the CPU to the accelerator and the results back to the CPU. Table 8 shows that FPGA implementations suffer from a larger overhead (the time when a FPGA is stalled) expressed as the percentage of the overall execution time. We speculate that this is because we are still in the early days of FPGAs being available as an accelerator in cloud computing setting. General purpose GPU computing has matured over the years, with better integration with a CPU. We believe the FPGA/CPU communication will improve over the years, and this overhead will decrease.

5.4. Scalability and cost/performance benefits

The biggest benefit of cloud computing is the ability to scale the computation and pay for it on demand as opposed to provisioning the computational infrastructure for the peak demand. So, it is important to know how well FPGA based acceleration scales for a complex application such as LightGBM library. Scalability results are shown in Table 9.

AWS provides an instance with 1 FPGA and an instance with 8 FPGAs, so we present the results for these two cases. Though the ComputeHistogram() function is parallelizable across multiple FPGAs, the shared memory that the FPGAs have to use to communicate with the CPU is the bottleneck. As a result, the performance does not scale well with the number of FPGA. For example as we go from 1 to 8 FPGAs the improvement is 3.63X at most and in the case of more complex data sets such as epsilon it is around 1.49X.

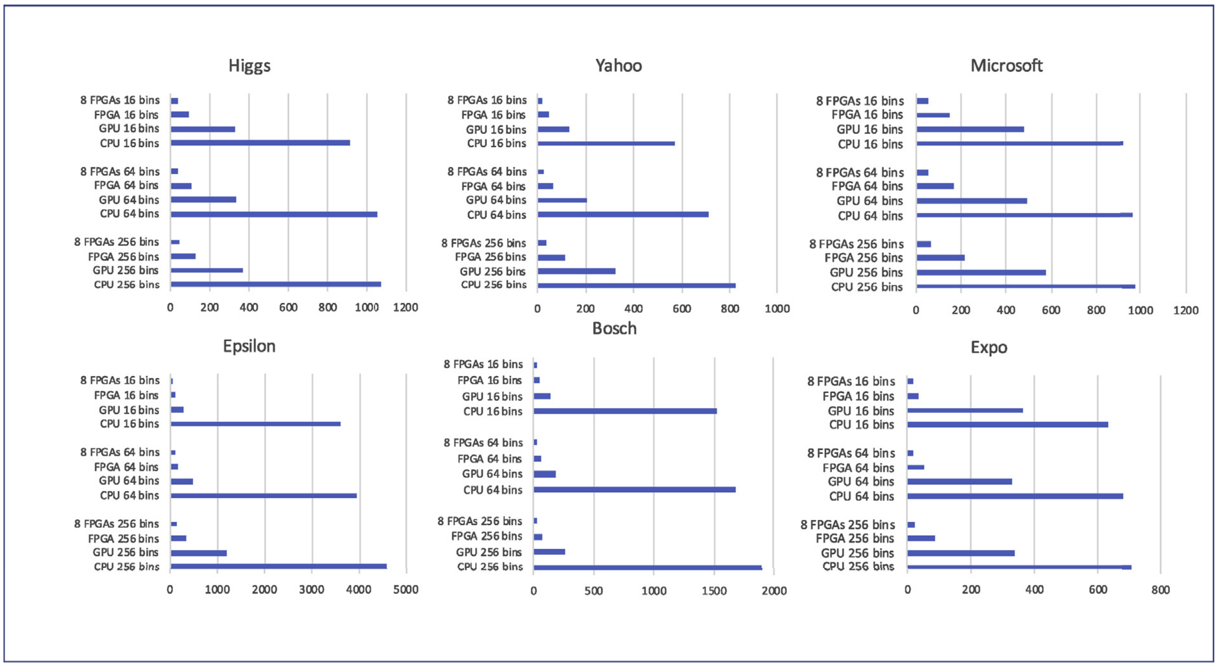

Fig. 2. Graphical visualization of performance in seconds for different data sets and different implementations.

Table 8
Overhead of GPU vs FPGA as Percentage of Overall Execution Time.

device	GPU			FPGA			8 FPGA		
# of bins	256	64	16	256	64	16	256	64	16
Higgs	0.4866	0.5322	0.7939	1.7879	2.31	3.6665	5.2791	6.2831	10.4786
Yahoo	0.4133	0.6848	1.1881	1.5126	2.6326	4.4369	5.1458	7.8595	11.1142
Microsoft	0.2766	0.2926	0.399	0.9639	1.1287	1.5941	3.2954	3.5637	5.1724
Epsilon	0.3263	0.8896	1.5574	1.4819	3.4973	5.0631	3.9846	5.4251	11.3574
Bosch	1.1507	1.3619	1.8419	5.8792	5.618	6.0072	15.0686	13.6083	17.2501
Expo	0.429	0.4719	0.6327	2.2166	3.8979	7.8476	8.6043	10.4785	15.7977

Table 9
Scalability of FPGA Performance. With 8 FPGA instances the average speed up is only about 2.62, though the cost increases by a factor 8. CPU memory is shared by all the 8 FPGA instances which is the bottleneck.

Dataset	1 FPGA	8 FPGA	Speedup
Higgs 256	130	45	2.89
Higgs 64	104	41	2.54
Higgs 16	195	36	2.64
Yahoo 256	113	37	3.05
Yahoo 64	67	25	2.68
Yahoo 16	45	19	2.37
Microsoft 256	212	65	3.26
Microsoft 64	166	54	3.07
Microsoft 16	151	50	3.02
Epsilon 256	345	137	2.52
Epsilon 64	150	101	1.49
Epsilon 16	110	53	2.08
Bosch 256	65	26	2.50
Bosch 64	58	25	2.32
Bosch 16	53	20	2.65
Expo 256	87	24	3.63
Expo 64	51	20	2.55
Expo 16	37	19	1.95

Next, we will evaluate the cost/performance for different implementations using the cost numbers from Table 3. The results are summarized in Table 10 and graphically visualized in Fig. 3. The results show that on the ComputeHistogram() kernel which is the computational bottleneck in the LightGBM application, FPGAs seem to be cost-effective. But, as discussed in the next section, this may not translate to a significant speed up on the *entire* application, as the rest of the application has to run on a CPU and there is additional CPU/FPGA communication overhead as shown in Table 11. Also, scaling to multiple FPGAs at least on this application does not appear to be cost-effective because of the poor scalability due to the shared memory bottleneck on the AWS platform today.

6. Discussion

6.1. FPGA implementation issues

Bin size is a critical parameter for implementing GBDTs and it was somewhat surprising to note that the performance degradation (in terms of statistical efficiency) with 16 bins and a fixed point implementation on a FPGA was not too bad (around 1.3%–3.3%). The number of LUTs (not the BRAM) seems to be the bottleneck in terms of increasing the number of threads (hence the parallelism) in the FPGA implementation. With 16 bins the number of threads is 123 and with 64 bins the number of thread is 110, with the LUT utilization in the 99% range.

The CPU on the C4.4xlarge instance being used on AWS is an Intel Xeon E5-2666 v3 2.9 GHz with 8 cores and 25 MB L3 cache and the GPU on the p2.xlarge instance is a half[2] of Tesla k80 with 24 GB DDR5 memory, 4992 Nvidia CUDA cores, 240 GB/s bandwidth and the FPGA is Virtex UltraScale + VU9P. Our implementation on the FPGA runs at about 276 MHz. The speed up over a CPU is understandable given the limited amount of parallelism available on a CPU with 8 cores. On a FPGA we have anywhere from 94 to 124 threads as shown in Table 4. The improvement over a GPU is more interesting because the k80 GPU has 2496 cores, so there is definitely more resources available to exploit thread-level parallelism. However, as noted in Section 2, the ComputeHistogram() function for each leaf node requires non-sequential memory reads, since the data samples for that particular leaf are not necessarily contiguous in the memory. Moreover, the amount exploitable parallelism is limited by the amount of shared memory per thread block which is around 48 KB, and, as described in Ref. [6], parallel histogram computation invariably has data races when the same bin is being updated by concurrent threads, which have to be avoided by using atomics. Furthermore, the ComputeHistogram() kernel has very low computational intensity (arithmetic operations per byte of data read), since the underlying computation is very simple, just accumulation. It appears that on a GPU, the overhead of fetching and decoding instructions and moving the data through the memory hierarchy (so called von Neumann computing overhead) is not effectively amortized in this application. On the other hand, on a FPGA, each thread is compiled into dedicated hardware that is directly interfaced to the Block RAMs, so the von Neumann overhead is largely absent. This results in the significant speed up compared to a GPU implementation. Lastly, the high LUT utilization in our FPGA implementation is because of the way the openCL compiler mapped the fixed point arithmetic blocks. We think that a manual implementation of the processing elements using the DSP blocks to realize the arithmetic function could reduce the amount of LUTs being used.

6.2. Main takeaway

The main results of the study in terms of cost performance analysis are summarized in Table 11. Though there is a significant improvement in the performance of the compute-intensive kernels compared to a GPU (anywhere from 3X to 10X), the overall speed of the entire application is far more modest because the bottleneck kernel only consumes 70%–85% of the total execution depending on the dataset. So, the cost of FPGA computing per hour has to come down drastically before FPGAs can become truly competitive with GPUs in the cloud. On the other hand, given FPGAs excel at certain computations (such as computing histograms, data compression, video encoding etc.), cloud vendors could use FPGAs to enable HW microservices [12]. For that the communication overhead between the FPGA and CPU has to be improved since the current overhead of 1%–5% for a single FPGA and 3%–17% for the 8 FPGA case seems to be quite high.

6.3. Power estimation

Using the Xilinx power estimation tools available on AWS, we estimate that the average power for the implementation operating at 276 MHz is between 10 W and 15 W depending on the dataset. Power efficiency is an important concern especially in a cloud computing setting and as expected our FPGA based acceleration of the LightGBM library appears to be significantly more power efficient than the corresponding GPU and CPU implementations. This agrees with the observations of other researchers such [1,2,13] who have also reported the energy efficiency of FPGAs when compared to a GPU. The Tesla K80 used in the p2.xlarge instance has a TDP of 300 W, while the Xeon E5-2666 has a TDP of 135 Watts.

7. Related work

A majority of related work on GPU and FPGA comparison is in the area of deep neural networks implementations not in the area of decision trees. For example, in Ref. [3], Intel researchers show that Stratix 10 FPGA based implementation of deep neural network are slightly better than Titan X Pascal GPU on dense floating point matrix multiplication (GEMM), but with reduced bit width they can be significantly better. The same research group also compared CPU, GPU, ASIC, and FPGA on implementing inference in recurrent neural networks in Ref. [14]. Researchers in Ref. [15] present a scalable deep learning accelerator on a FPGA that is 36x better than CPU. In Ref. [13] researchers compare the CPU, GPU, and FPGA implementations

[2] https://aws.amazon.com/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/.

Table 10
Cost for ComputeHistogram() kernel on CPU, FPGA, and GPU.

device	CPU			GPU			FPGA			8 FPGAs		
# of bins	256	64	16	256	64	16	256	64	16	256	64	16
Higgs	0.237252	0.233936	0.20099	0.09275	0.08375	0.08225	0.059583	0.047667	0.043542	0.165	0.150333	0.132
Yahoo	0.181532	0.156989	0.126254	0.081	0.051	0.03275	0.051792	0.030708	0.020625	0.135667	0.091667	0.069667
Microsoft	0.216247	0.213593	0.203864	0.14425	0.12275	0.119	0.097167	0.076083	0.069208	0.238333	0.198	0.183333
Epsilon	1.014679	0.870736	0.79799	0.29575	0.11575	0.0675	0.158125	0.06875	0.050417	0.502333	0.370333	0.194333
Bosch	0.420332	0.370803	0.334762	0.06425	0.046	0.03425	0.029792	0.026583	0.024292	0.095333	0.091667	0.073333
Expo	0.156104	0.150798	0.139079	0.0845	0.0825	0.09075	0.039875	0.023375	0.016958	0.088	0.073333	0.069667

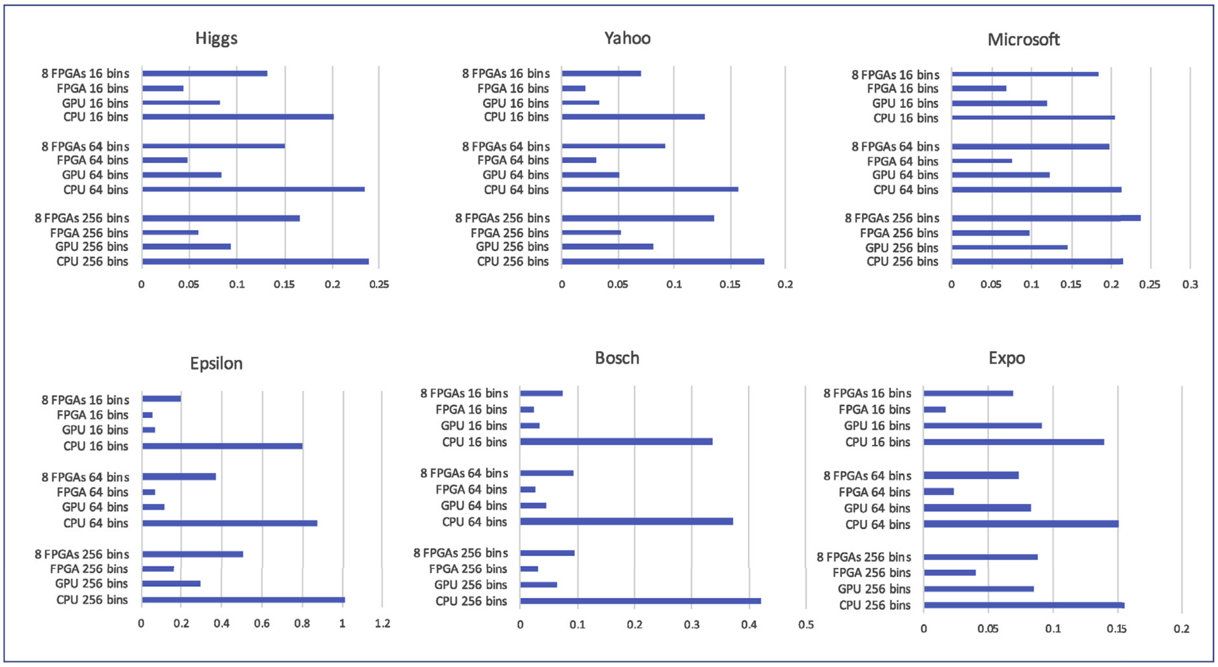

Fig. 3. Graphical Visualization of Cost of ComputeHistogram() kernels on Different Datasets and Different Bin Sizes.

using a subset of the Rodinia benchmarks. Their goal was to demonstrate that OpenCL based implementations can be implemented efficiently on a FPGA. They also show that Altera Stratix V FPGA-based implementations exhibit a 3.4X better power efficiency compared to NVIDIA K20c GPU. In Ref. [16], researchers compare the effectiveness of FPGAs, GPUs, and multi-core CPUs for accelerating classification using models generated by compact random forest machine learning classifiers. In Ref. [17] three different architectures are proposed for a random forest classifier on a ZYNQ evaluation board, and in Ref. [18] decision tree ensemble classifier was implemented on the Intel HARP (CPU + FPGA) platform. However, neither do these implementations consider training, nor do they deal with gradient boosted decision trees.

In Refs. [1,2] Microsoft researchers make a case for of FPGAs as accelerators in a cloud computing framework which had a profound impact in the landscape of heterogeneous computing platform and the resurgence of FPGAs as a viable alternative to GPUs. LightGBM [5] introduces many new algorithmic ideas in improving the performance of GBDT training on traditional multicore CPUs. Zhang et al. in Ref. [6] developed the highly optimized GPU implementation which has been merged with the main LightGBM codebase. Our starting point is Zhangs implementation. To the best of our knowledge our work is the

Table 11
FPGA Implementation of ComputeHistogram() kernel from LightGBM library - Summary of Performance Improvement over CPU and GPU.

Dataset	Speed up over P2.xlarge (1 GPU)			Speedup over C4.4xlarge (1 CPU)		
	256 bins	64 bins	16 bins	256 bins	64 bins	16 bins
Higgs	2.85	3.22	3.46	8.25	10.17	9.57
Yahoo	2.87	3.04	2.91	7.27	10.60	12.69
Microsoft	2.72	2.96	3.15	4.61	5.82	6.11
Epsilon	3.43	3.09	2.45	13.30	26.25	32.81
Bosch	3.95	3.17	2.58	29.25	28.91	28.57
Expo	3.89	6.47	9.81	8.11	13.37	17.00

first FPGA based acceleration of LightGBM - whether standalone or in a cloud computing setting. Recently, there has been work in the area of using FPGAs in the cloud [4,19–21] where researchers provide tools and design flow to *share* FPGA resources in a datacenter to accelerate large scale applications. Applications such as molecular dynamics, 3D FFT, and deep neural networks were used as case studies. The use of limited precision arithmetic that can be tailored to given workloads and datasets is a key advantage of FPGAs [10,11]. Though we used 32 bit precision to stay compatible with the GPU code from Ref. [6], the resource utilization of our implementation could be improved with reduced precision. In the area of FPGA based histogram implementation [22], show how dual-ported RAMs can be used to avoid read/write conflicts but the amount of parallelism is restricted to only two threads, and the work reported in Ref. [23] focuses on reducing the energy consumption by serializing the computation.

8. Conclusions and future work

The goal of this work is to help a developer, or an organization understand the trade-offs of implementing high performance computing applications using heterogeneous computing platforms offered by cloud computing vendors. It is important to understand the cost/performance and scalability of an application on CPUs, FPGA, and GPUs so that one could choose the appropriate deployment strategy to minimize the cost. We show that FPGAs can provide significant speed up on performance critical kernels compared to a GPU. However, the communication overhead of moving data to and from the FPGA is still quite high. This gets worse with multiple FPGAs. So, clearly there is a need to rearchitect the CPU/FPGA platform to minimize this overhead. Our results indicate that with further optimization, using handcrafted hardware implementations instead of using OpenCL FPGAs can be more cost effective. One key impediment to FPGA based implementations was the tools and the need to program in a low level hardware oriented language such as Verilog. However, it looks like SDAccel and the associated simulation and debug environments have made progress in reducing programming barrier. However, in the long run we need an alternative to OpenCL (perhaps Chisel [24]?) so that it is easier to create customized hardware and memory systems efficiently.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.vlsi.2019.09.007.

References

[1] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, et al., A cloud-scale acceleration architecture, in: The 49th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Press, 2016, p. 7.

[2] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, et al., A reconfigurable fabric for accelerating large-scale datacenter services, ACM SIGARCH Comput. Architect. News 42 (3) (2014) 13–24.

[3] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y.T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, et al., Can fpgas beat gpus in accelerating next-generation deep neural networks? in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ACM, 2017, pp. 5–14.

[4] J. Sheng, C. Yang, A. Sanaullah, M. Papamichael, A. Caulfield, M.C. Herbordt, Hpc on fpga clouds: 3d ffts and implications for molecular dynamics, in: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2017, pp. 1–4.

[5] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: a highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.

[6] H. Zhang, S. Si, C.-J. Hsieh, Gpu-acceleration for Large-Scale Tree Boosting, 2017, arXiv:1706.08359.

[7] J.H. Friedman, Stochastic gradient boosting, Comput. Stat. Data Anal. 38 (4) (2002) 367–378.

[8] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, third ed., Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[9] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794.

[10] P. Colangelo, N. Nasiri, A. Mishra, E. Nurvitadhi, M. Margala, K. Nealis, Exploration of Low Numeric Precision Deep Learning Inference Using Intel Fpgas, 2018, arXiv:1806.11547.

[11] A. Nannarelli, Tunable floating-point for energy efficient accelerators, in: 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), IEEE, 2018, pp. 29–36.

[12] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, C. Delimitrou, An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '19, ACM, New York, NY, USA, 2019, pp. 3–18 [Online]. Available:, https://doi.org/10.1145/3297858.3304013.

[13] H.R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, S. Matsuoka, Evaluating and optimizing opencl kernels for high performance computing with fpgas, in: High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for, IEEE, 2016, pp. 409–420.

[14] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, D. Marr, Accelerating recurrent neural networks in analytics servers: comparison of fpga, cpu, gpu, and asic, in: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2016, pp. 1–4.

[15] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, X. Zhou, Dlau: a scalable deep learning accelerator unit on fpga, IEEE Trans. Comput. Aided Des. Integr Circuits Syst. 36 (3) (2017) 513–517.

[16] B. Van Essen, C. Macaraeg, M. Gokhale, R. Prenger, Accelerating a random forest classifier: multi-core, gp-gpu, or fpga? in: Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on, IEEE, 2012, pp. 232–239.

[17] X. Lin, R. Blanton, D.E. Thomas, Random forest architectures on fpga for multiple applications, in: Proceedings of the on Great Lakes Symposium on VLSI 2017, ACM, 2017, pp. 415–418.

[18] M. Owaida, H. Zhang, C. Zhang, G. Alonso, Scalable inference of decision tree ensembles: flexible design for cpu-fpga platforms, in: 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE, 2017, pp. 1–8.

[19] N. Tarafdar, N. Eskandari, T. Lin, P. Chow, Designing for fpgas in the cloud, IEEE Design Test 35 (1) (2018) 23–29.

[20] A.M. Caulfield, E.S. Chung, A. Putnam, H. Angepat, D. Firestone, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, et al., Configurable clouds, IEEE Micro 37 (3) (2017) 52–61.

[21] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, et al., A configurable cloud-scale dnn processor for real-time ai, in: Proceedings of the 45th Annual International Symposium on Computer Architecture, IEEE Press, 2018, pp. 1–14.

[22] A. Shahbahrami, J.Y. Hur, B. Juurlink, S. Wong, Fpga implementation of parallel histogram computation, in: 2nd HiPEAC Workshop on Reconfigurable Computing, 2008, pp. 63–72. Published.

[23] A. Sanny, Y.-H.E. Yang, V.K. Prasanna, Energy-efficient histogram on fpga, in: 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), IEEE, 2014, pp. 1–6.

[24] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Aviienis, J. Wawrzynek, K. Asanovi, Chisel: constructing hardware in a scala embedded language, in: Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE. IEEE, 2012, pp. 1212–1221.