Exploiting Unstructured Sparsity on Next-Generation Datacenter Hardware

Mike Ashby, Christiaan Baaij, Peter Baldwin, Martijn Bastiaan, Oliver Bunting, Aiken Cairncross, Christopher Chalmers, Liz Corrigan, Sam Davis, Nathan van Doorn, Jon Fowler, Graham Hazel, Basile Henry, David Page, Jonny Shipton, Shaun Steenkamp
myrtle.ai

***Abstract*—Recurrent neural networks (RNNs) form a significant proportion of data center deep learning inference (29% [1]). This includes workloads like machine translation, speech synthesis and speech transcription. Such models can be substantially compressed, reducing computational cost. In this paper we show, with explicit reference to a benchmark transcription model, how to train such models to very high levels of sparsity (greater than 95%) with minimal loss of accuracy (less than 0.23%). We then present our scalable tile-based accelerator architecture, which can exploit the unstructured sparsity of such models to accelerate inference. We demonstrate the performance of our MAU accelerator on an Intel® Stratix® 10 FPGA[1,2] data center board highlighting this platform as an ideal target for data center RNN inference tasks. We demonstrate how algorithm-accelerator co-design can achieve extremely high performance on models with high levels of unstructured sparsity.**

I. Introduction

Myrtle's five year focus on Deep Neural Network acceleration (DNN) has enabled the company to build up a thorough knowledge of a diverse range of machine learning workloads. The company has focused its efforts on creating optimized recurrent neural networks. Today, Myrtle is a leading expert in the creation of optimized implementations for cloud-based speech applications, a fact recognized with their codebase and models being provided as the speech inference benchmark for the MLPerf consortium.

Deep learning inference demands are rapidly growing as companies increasingly incorporate these powerful, but computationally challenging, systems into their infrastructure and products [2]–[5]. Facebook notes that a *"significant fraction of the future demand is expected to come from workloads corresponding to DL [deep learning] inference"* and show a nearly $4\times$ rise in server demand for deep learning inference over 7 quarters [4]. Furthermore, Google has previously projected a need to double their total number of data centers in order to meet the computational demands imposed by people using 3 minutes of voice search per day [1].

Recurrent neural networks have achieved state-of-the-art results in language modelling [6]–[10], speech recognition [11]–[13], speech synthesis [14]–[16] and machine translation [17]. Productionized versions of these models typically contain tens to hundreds of millions of parameters but some have been scaled to billions of parameters given enough data [4], [10], [18]. Increasing the size of a model also increases its compute and memory requirements. Reducing the computational cost of these models translates directly to cost and energy savings for service operators.

At the same time, the rate at which general-purpose processors are improving is decreasing. This is due to technological factors such as the end of Dennard scaling and slowdown of Moore's law, architectural factors such as the inability to further exploit instruction level parallelism and parallel processing due to Amdahl's law, and because of a shift in the computing paradigm away from traditional desktop computers to mobile and hyperscale computing [19].

The combination of increasing inference demand, the trend towards larger models and hence their increasing computational and memory requirements, and a decreasing rate of improvement in general-purpose processors means that hardware must be scaled out or sub-optimal models deployed in order to meet peak throughput and minimum latency targets. Scaling out is an expensive proposition, requiring huge capital expense to provide infrastructure in addition to computing hardware. The data center sector is estimated to account for 1.4% of global electricity consumption [20], therefore performance-per-watt improvements in processing have significant impact at scale.

An alternative solution is the use of specialized accelerators designed for deep learning inference. This enables the joint optimization of the model, numerics, and hardware platform resulting in performance gains – latency, throughput, and performance-per-watt – beyond that achievable through optimization of any one of the individual components alone [21]. However, the use of an application-specific integrated circuit (ASIC) may not be desirable. Model architectures, numerics, and other optimization techniques are developing rapidly meaning an ASIC can quickly become obsolete. Furthermore, if the ASIC is unable to be utilized for anything other than the acceleration of today's deep learning models then a significant amount of resources may sit idle the majority of the time due to diurnal and annual load cycles.

Field-programmable gate arrays (FPGAs) can be used to accelerate deep learning inference whilst avoiding these issues. They are dynamically reconfigurable which enables the rapid iteration and adoption of the latest models, numerics, and other optimizations. This also ensures that the underutilization of resources is avoided as they can be reconfigured to perform other tasks during periods of low load.

In this paper we apply two simple optimization techniques,

that have minimal impact on final test accuracy and are applicable to a wide range of neural networks, to a speech transcription model – DeepSpeech [11] – that is representative of recurrent neural networks deployed in production systems today. We show how these optimizations enable the model to be deployed with Myrtle's MAU accelerator, a high-performance sparse linear algebra accelerator running on an Intel® Stratix® 10 FPGA. The accelerator achieves low-latency processing at batch-size 1.

II. Speech to Text Algorithm

Speech to Text (STT) is the task of transcribing an acoustic speech signal into a sequence of characters or words.

Effective Speech to Text processing has been demonstrated by Baidu's DeepSpeech model. The DeepSpeech model is a neural network architecture for speech recognition [11]. The network contains 5 hidden layers — the first three are fully connected, the fourth is a bi-directional recurrent layer that uses an LSTM cell and the fifth is a fully connected layer. After the fifth hidden layer there is an additional softmax output layer that computes a probability distribution over the desired output alphabet. This architecture is summarized in TABLE I where the *# Parameters* column refers to the specific model we discuss in this report that uses a hidden size of 2048.

A preprocessing step is applied to the input audio stream prior to the network's application. This slides a window of length 25ms over the input audio sequence with a stride of 20ms. 26 Mel-frequency cepstral coefficients (MFCCs) are computed for the audio contained within the window at each position. The i^{th} input to the network is a vector of length 494. This is a concatenation of the 26 MFCCs computed at the i^{th} window position plus the vectors of MFCCs computed over the 9 preceding and 9 succeeding window locations.

The network is applied to this input sequence of vectors to produce an output sequence of the same length. A connectionist temporal classification (CTC) beam search decoder converts from this output sequence of probability distributions over characters in an alphabet to a single sequence of words [22].

TABLE I
DeepSpeech Architecture Summary

Layer	Type	# Parameters (M)
0	Input	-
1	Fully Connected	1.01
2	Fully Connected	4.20
3	Fully Connected	8.39
4	Bidirectional LSTM	100.70
5	Fully Connected	8.39
6	Softmax Output	0.06
	Total	122.75

III. Model Compression

Model compression reduces the size of the model in order to reduce computational cost and memory bandwidth. This reduces the requirements of the processing platform either by reducing execution time for the algorithm or reducing the circuit size required. Both of these reduce power consumption and operating cost.

We utilize two orthogonal forms of model compression: *sparsity* and *quantization*. We utilize these techniques without significant degradation in algorithmic performance.

A. Sparsity

Inducing sparsity within a model decreases the total effective number of parameters by explicitly setting some to zero. With suitable hardware support this significantly improves the effective arithmetic intensity of the system as computations involving parameters with zero values can be implicitly executed without a memory access or multiplication. Furthermore, these parameters do not need to be stored, reducing the overall memory requirements. When targetting an FPGA platform, this enables the storage of weights entirely in on-chip RAM.

Pruning techniques are an effective class of methods that induce sparsity within a model. Let the importance, or *saliency*, of a parameter be the size of the increase in the loss or error function if that parameter were to be set to zero. Parameters with higher saliency will cause the error to increase more when removed. Pruning techniques remove the parameters with lowest saliency. However, finding these parameters is a non-trivial problem [23].

In this paper we discuss and apply *magnitude-based pruning*. This is a specific pruning technique that has been succesfully used to induced high-levels of sparsity in a variety of neural networks [23]–[30]. It makes the non-trivial problem tractable by assuming that a parameter's magnitude closely approximates it's saliency and hence finding the least salient parameters reduces to either a linear scan through, or sort of, the model's parameter vector which are $O(n)$ and $O(n \log n)$ operations respectively.

We modify the model's original training loop to incorporate magnitude-based pruning. The model is first trained as per the original scheme for a number of epochs. The remaining epochs are altered such that an increasing percentage of the least salient parameters are set to zero each time a certain number of steps are performed. This gradually increases the sparsity of the network from 0% up to the target sparsity percentage. This procedure is outlined in Algorithm 1.

Algorithm 1 Magnitude-based Pruning Training Loop

```
# init mask for each parameter vector to prune
for θ_i ← model.to_prune do
    M_i ← J_(θ_i.size)
end for
# train network, gradually increasing sparsity
sparsity ← 0.0
for epoch ← {1, 2, ..., n_epochs} do
    for step, sample ← enumerate(data) do
        # update sparsity, masks and prune parameters
        sparsity ← schema(epoch, step)
        for θ_i ← model.to_prune do
            M_i ← update_mask(θ_i, M_i)
            θ_i ← θ_i ⊙ M_i
        end for
        # exec forward, backward passes and update params
        train_step(model, data)
    end for
end for
```

Algorithm 1 creates a binary mask vector $\mathbf{M}_i$ for each parameter vector $\boldsymbol{\theta}_i$ that is to be pruned. The masks initially contain all ones. The sparsity is gradually increased according to a sparsity `schema` that returns a $\%$ for a given `epoch` and `step` during training.

Fig. 1 summarizes the effect of this process on the distribution of the network's parameters. We modify the training hyperparameters during this process to maximize performance on the validation data. This work is an advancement on [31].

Results are presented in Section V-A.

This method of training sparsity into the model results in unstructured sparsity. We do not require that specific sparsity patterns are created into the model for the accelerator proposed. This allows weights to remain at non-zero, where most significant to algorithm performance.

B. Quantization

Quantizing a model reduces the number of bits used to represent each parameter and/or each activation during inference. In this paper we quantize both the IEEE 754 single-precision floating-point weights and activations of the original model to 8-bit integers. This reduces the size of the model by a factor of 4 resulting in an immediate $4\times$ improvement in arithmetic intensity and a $4\times$ decrease in data bandwidth requirement. In turn this results in a smaller circuit design, and the ability to store weights in on-chip memory, consuming less power than reading from external DRAM memory. [4], [32], [33]

We base our quantization schema on that presented in [34], which is further described in [32], that has been shown to be applicable to a wide range of models including MobileNet v1 and v2, Inception v3, ResNet 50 and 152, and Google Neural Machine Translation (GNMT) [17]. We use a per-layer symmetric linear quantization scheme for the weights where a 32-bit floating-point value x is mapped to an 8-bit integer $x_Q \in [-128, -127, \ldots, 127]$. This schema is defined in (1) where Δ is the *scale* value computed based on the largest absolute value in the weight vector being quantized. We ensure that the pruned weights in the original 32-bit floating-point vector remain set to zero after the quantization process.

$$x_Q = \texttt{clamp}\left(\texttt{round}\left(\frac{x}{\Delta}\right), -128, 127\right) \tag{1}$$

The weights tend to be symmetric around zero hence we apply a symmetric quantization schema. This is not true for activations — layers followed by a ReLU activation function produce vectors that only contain non-negative values. We therefore use an asymmetric linear quantization scheme for the activations where a 32-bit floating-point value x is mapped to an 8-bit integer $x_Q \in [0, 1, \ldots, 255]$. This schema is detailed in (2) where the zero point z and scale Δ are computed based on an exponential moving average of the smallest and largest activations taken over a small sample of the training dataset.

$$x_Q = \texttt{clamp}\left(\texttt{round}\left(\frac{x}{\Delta}\right) + z, 0, 255\right) \tag{2}$$

We apply our quantization scheme after the modified training procedure, as described in Section III-A, has finished.

IV. An Unstructured Sparse Matrix Accelerator

To efficiently compute the compressed model, we implement a dedicated accelerator on FPGA hardware. This enables us to use computational structures that are able to exploit sparsity created in the model, by the training process outlined above.

The majority of computation to be performed in deep learning networks is matrix-vector multiplication. This is therefore the primary target for efficient computation. We achieve highest acceleration performance by maximising the use of the embedded multiply resource available on the FPGA. Using multipliers efficiently requires that there is sufficient random access memory bandwidth to read activation values from memory to feed all the multipliers on each clock cycle.

A. Tile Based Architecture

Building a large and fast design on the largest FPGAs requires suitable data architectures, in order to be able to move data efficiently around the design, without routing congestion or slowing the achievable clock frequency. We solve this using a scalable tile based architecture. This allows data in the network to flow point to point, therefore providing an extremely large instantaneous bandwidth within the accelerator, and removing bottlenecks for data flow within the accelerator. Using a tile based architecture reduces the routing options between cores, lending itself to layout on the FPGA grid structure, that can be realized using Intel® HyperFlex™ fast interconnect within the FPGA.

Our MAU accelerator is a grid of computation cores. The computation cores are optimized for high throughput and low latency multiply compute on unstructured sparse matrices. The routing infrastructure connects the cores and supports configurable routing, to allow different network topologies to be placed on the accelerator. Each routing element is connected to 5 different endpoints (north, south, east, west and

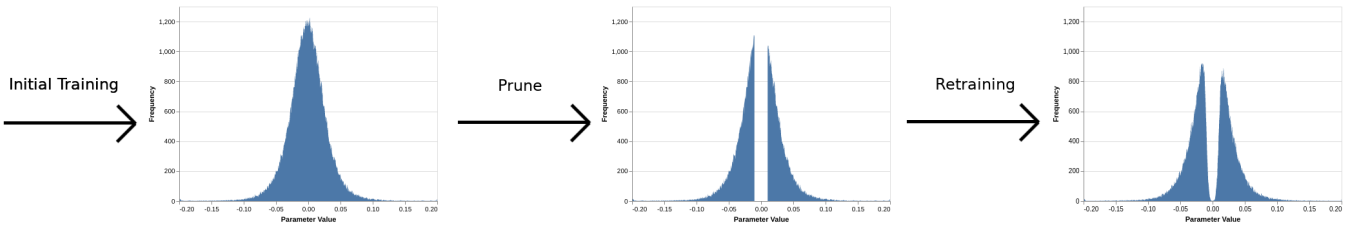

Fig. 1. Prune-retrain process for maintaining model accuracy

core). This limits the size and complexity of routing elements, and limits the amount of routing resource required in any given area of the FPGA.

Fig. 2 shows the interconnection of cores via routing interconnect. Each routing connection supports 3.3GBytes/s full duplex data transfer. This provides sufficient bandwidth to transfer data through the network, given the rate at which data is generated internally. Routing also supports the duplication of data to multiple output ports, to provide the branching needed for a neural network description.

Our accelerator design is targeted to a large Intel® Stratix® 10 FPGA at high resource utilization. To support the toolchain through fitting and routing, we use floorplanning for each tile, in order to achieve the best possible layout for the design.

The accelerator input and output data is mapped to PCIe addressable memory, with a defined entry and exit point into the routing infrastructure. We use a PCIe x16 lane interface to provide sufficient data bandwidth to transfer data to and from the accelerator.

B. Accelerator Design to Exploit Sparsity

Ensuring optimal cycle efficiency of the multiplier resource requires specialized data structures around the multiplier and memory resource. Sufficient high bandwidth memory accesses must be provided so that random access activation lookup can be made every clock cycle so that multipliers are always active.

The accelerator is optimized to work with a model where only 1 in 16 weights per row are non-zero, corresponding to a sparsity level of greater than 93.75%. The accelerator is able to handle unstructured sparsity patterns, where weights can be distributed at random across a matrix row. The accelerator performance is optimal when weights approximate an even distribution across a row, but are not forced into a specific sparsity pattern. Non-optimal weight distribution is handled as additional processing cycles. For our implementation of DeepSpeech, we are able to train the model to provide an unstructured sparsity pattern that provides optimal cycle efficiency without loss in accuracy of the model.

Processing takes place as follows: 1. Read a matrix row from memory, this is stored in compressed format, so that all weights for a single row can be read in one cycle (64 weights, corresponding to a 1024 wide matrix when uncompressed). 2. Read the activations corresponding to each compressed weight. Our memory architecture allows sufficient random access memory bandwidth to read activations for the majority of cases, where the unstructured sparsity pattern for the row is optimal for the accelerator. Where there is insufficient bandwidth, the processor will stall for a cycle giving a reduction in overall performance. 3. Multiply each weight-activation pair and then accumulate across the row to compute the dot-product for the row. 4. Where a row has been split across multiple cores (two cores are required for a row length of 2048), the row results are combined to give a true accumulation for the row, before passing to the next level in the network.

C. Computation Cores

In addition to matrix-vector multiplication, the accelerator must support LSTM pointwise operations, the addition of biases and ReLU non-linearities. These are applied after the multiplication to support LSTM processing fully on-chip, without adding significant complexity to the design. To implement the DeepSpeech LSTM network, the accelerator is configured to define the weights, routing and use of non-linearities and biases. We use our compiler to map the network description onto the accelerator.

We divide the DeepSpeech model computation across multiple computation cores. Each core is sized to calculate a single matrix for 4 rows in parallel, for up to 1024 columns per row, where each row is compressed to 1/16th of its original size. Where the row size for a network operation is longer than 1024 columns, the row is calculated across multiple cores.

Fig. 3 shows the architecture of the core. Each core can compute 256 multiply operations and 256 additions per clock cycles, giving an equivalent performance figure of 8.192 GOPS/MHz for an uncompressed row. At a target design of 34 cores and 250MHz, this yields a headline processing capacity of 69.6 TOPS. When operational factors including the loading of the activations into memory are taken into account, the true processing capacity of the design has a headline figure of 67.5 TOPS.

D. Mapping DeepSpeech Network onto the Architecture

The MAU accelerator is agnostic of the network architecture placed upon it, the network is configured at run time rather

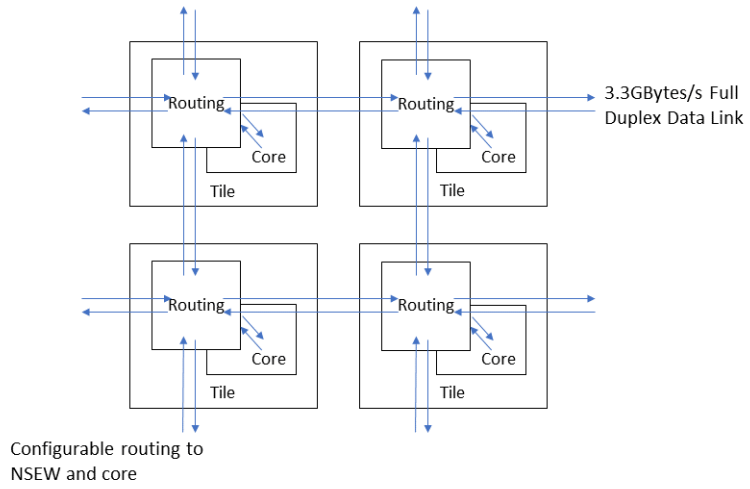

Fig. 2. Routing Infrastructure

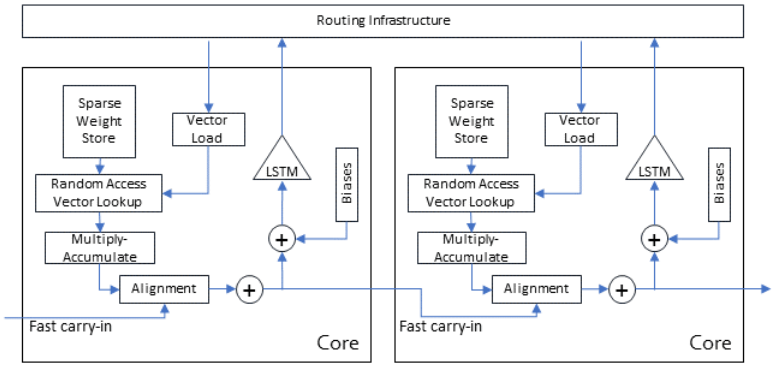

Fig. 3. Computation Core Architecture

than compile time in order to describe the network processing that must be placed upon it. This allows the FPGA image to be used for multiple network descriptions without the need to recompile the design.

Our compiler takes a description of the model and weights and generates the configuration for the accelerator.

To implement DeepSpeech on the FPGA, we use 34 computation cores in the accelerator. We map the DeepSpeech to the cores as shown in TABLE II. We map all but layers 5 and 6 to the FPGA. These layers are processed on the CPU co-processor. This yields a mapping that is 80.1% efficient in terms of utilising the effective TOPs of the accelerator for DeepSpeech.

TABLE II
MAPPING DEEPSPEECH TO COMPUTATION CORES

Layer	Type	Number Cores	Core Efficiency (%)
1	Fully Connected	2	12.5
2	Fully Connected	4	25
3	Fully Connected	4	50
4	Bidirectional LSTM	24	100
Total		34	80.1

V. PERFORMANCE

A. Model Compression

We train the DeepSpeech model described in Section II on the training subsets of the permissively licenced and commonly used LibriSpeech corpus [35]. The LibriSpeech training data contains approximately 1000 hours of audio samples, each of which is approximately 1 to 30 seconds in length, based on volunteer readings of public domain texts.

Performance is evaluated by comparing the mean word error rate (WER) over LibriSpeech's clean validation dataset during development and LibriSpeech's clean test set for the final model. The WER of a single transcription is defined as the sum of the number of word insertions, deletions or substitutions required to convert from the model output to

the target transcription divided by the length of the target transcription.

We compare a dense single-precision floating-point baseline model to a dense 8-bit integer model, a sparse single-precision floating-point model and a sparse 8-bit integer model. The results are summarized in TABLE III where each statistic is the median of 5 runs.

The test set has only been evaluated once, just prior to publication.

TABLE III
DEEPSPEECH SPARSITY AND QUANTIZATION RESULTS

Sparsity (%)	Precision	Validation WER	Test WER
0.0	32-bit float	6.33	6.06
0.0	8-bit integer	6.38	6.17
96.0	32-bit float	6.32	6.25
96.0	8-bit integer	6.35	6.29

The model compression techniques in combination show a negligible degradation in word error rate for the validation set. When applying the trained results to the test set, the compressed model shows an absolute degradation of 0.23% in word error rate compared to the dense floating-point implementation.

B. Accelerator Performance

Mapping the compressed model to Myrtle's MAU accelerator allows the performance of our approach to be compared to standard CPU and GPU processing models. We present key parameters and processing performance in TABLE IV, for our accelerator in comparison to a NVIDIA V100 GPU and an Intel® Xeon®. These are top of the range platforms within their respective classes. The GPU and CPU models are not sparsified, giving reference implementations for effective throughput. For the CPU and GPU profiling we use larger batch sizes, where the batch size is chosen to maximize performance on the given platform.

We measure power of the FPGA during inference using an Intel® provided board test system for the Stratix® 10 development platform. We sum the internal power rail measurements for the platform and then assume a power supply efficiency of 85% to calculate the total input power of the platform. Power measurements on the CPU are taken during inference using the turbostat power monitoring software. Power measurements for the GPU are taken using NVIDIA System Management Interface program.

In addition to raw performance, we also compare latency for the different processing models. These results are shown in TABLE V. We use a batch size of 1 for FPGA, CPU and GPU to allow lowest latency to be compared. We measure latency as the time taken to compute an audio input of length 1 second.

The accelerator demonstrates a $165\times$ improvement in effective throughput when compared to the Xeon and an $638\times$ improvement in performance per Watt. Compared to the GPU, throughput of the FPGA accelerator comparable, but the performance per Watt of the FPGA is $3.6\times$ greater than the GPU. This provides a significant improvement in the power consumption required for speech inference and other RNN workloads.

The MAU accelerator achieves a factor of $1000\times$ improvement in latency compared to the Xeon® and $29\times$ improvement in latency compared to the GPU. For batch-1 low latency processing, the effective throughput of the GPU falls dramatically, while the MAU accelerator provides performance and low latency simultaneously.

In addition to accelerating DeepSpeech, we generate a DeepBench accelerator using a DeepBench specific accelerator architecture, utilising the same fast random-access memory architecture as the DeepSpeech accelerator. The DeepBench benchmark is a standard benchmark that measures the performance of basic operations involved in training and inference of deep neural networks. We configure the accelerator to measure performance for Sparse GEMM result on a 2560 x 7680 matrix at batch size 1 as outlined in http://github.com/baidu-research/DeepBench. Our results against this measurement are shown in TABLE VI. We generate a random matrix with a 95% sparsity characteristic and measure peak throughput, under a continuous stream of activation inputs. The performance measurements for the DeepBench accelerator do not include loading weights or activations onto the FPGA. We compare this to results for NVIDIA GPU Titan XP that currently holds the benchmark.

The FPGA accelerator demonstrates better TOPS performance than the GPU for this benchmark. Significantly, the FPGA accelerator achieves this performance with batch 1 processing, enabling high performance to be achieved in conjunction with low latency processing.

VI. CONCLUSION

This paper shows how optimization of a DeepSpeech implementation in conjunction with dedicated accelerator design can achieve significant acceleration without degradation in algorithm performance. We show how to achieve FPGA based acceleration on an unstructured sparse matrix with our MAU accelerator.

Evaluation of our approach to algorithm-accelerator co-design shows that we can achieve a $3.6\times$ improvement in TOPS/W over a high performance GPU. This supports our assertion that use of FPGA processing in the data center can have a significant impact on power consumption for speech inference at scale. This is of primary importance for service operators to achieve lower operating costs, and reduce their global energy footprint.

Moreover, accelerating a sparse, quantized model on FPGA allows us to perform low-latency speech transcription at batch size 1 with throughput equivalent to dense matrix-multiply accelerators that require high-batch sizes and have a higher latency. Our MAU accelerator gives comparable throughput to a NVIDIA V100 GPU while simultaneously achieving a $29\times$ improvement in latency. Significantly improved batch 1 latency of the MAU accelerator better supports applications for

TABLE IV
ACCELERATOR PERFORMANCE FOR DEEPSPEECH VS CPU AND GPU[3]

	Myrtle MAU Accelerator	**CPU**	**GPU**
Platform	Intel® Stratix® 10 FPGA	2S Xeon Gold 6140M	NVIDIA V100
Frequency (MHz)	250	2300	1530
Sparsity (%)	96	0	0
Quantization	8-bit integer	32-bit fp	16-bit fp
Batch Size	1	16	256
Effective Throughput (TOPS)	54.0	0.327	53.37
Power (W)	60.5	220	216
Performance per Watt (Effective GOPS/W)	893	1.4	247

TABLE V
ACCELERATOR LATENCY FOR DEEPSPEECH VS CPU AND GPU[3]

	Myrtle MAU Accelerator	**CPU**	**GPU**
Platform	Intel® Stratix® 10 FPGA	2S Xeon Gold 6140M	NVIDIA V100
Frequency (MHz)	250	2300	1530
Sparsity (%)	96	0	0
Quantization	8-bit integer	32-bit fp	32-bit fp
Batch Size	1	1	1
Effective Throughput (TOPS)	54.0	0.032	1.12
Latency (ms)	0.343	349	10.1

TABLE VI
DEEPBENCH ACCELERATOR VS GPU[3]

	Myrtle DeepBench Accelerator	**GPU Sparse GEMM**
Platform	Intel® Stratix® 10 FPGA	NVIDIA Titan XP
Sparsity (%)	95	95
Quantization	8-bit integer	32-bit fp
Matrix Size	M=7680,K=2560	M=7680,K=2560
Batch Size	1	1500
Throughput (TOPS)	2.35	1.88
Effective Throughput (TOPS)	47	37.6

human-machine voice interaction, without incurring additional cost of computation.

Specialized accelerators are able to achieve high levels of performance, but must keep pace as model architectures rapidly change. We use an FPGA solution to enable performance gains through the use of dedicated computing architectures, whilst having the flexibility to update the design to exploit new techniques. Using FPGA based acceleration provides a robust and future proof method for exploiting the latest model optimizations for inference.

Myrtle's MAU accelerator, described in this paper, is available now for deployment on Intel Programmable Acceleration Cards. Contact us today on stratix_eval@myrtle.ai to evaluate the solution, verify this paper's results and find out how you can benefit from next generation hardware.

REFERENCES

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[2] D. Amodei and D. Hernandez, "Ai and compute," *Heruntergeladen von https://blog. openai. com/aiand-compute*, 2018.

[3] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro *et al.*, "Applied machine learning at facebook: a datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 620–629.

[4] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur *et al.*, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[5] T. Wood and T. Merritt, "Varying speaking styles with neural text-to-speech," Nov 2018. [Online]. Available: https://developer.amazon.com/de/blogs/alexa/post/7ab9665a-0536-4be2-aaad-18281ec59af8/varying-speaking-styles-with-neural-text-to-speech

[6] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[7] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[8] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding with unsupervised learning," Technical report, OpenAI, Tech. Rep., 2018.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[11] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[12] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep

speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[13] E. Battenberg, J. Chen, R. Child, A. Coates, Y. G. Y. Li, H. Liu, S. Satheesh, A. Sriram, and Z. Zhu, "Exploring neural transducers for end-to-end speech recognition," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 206–213.

[14] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[15] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio *et al.*, "Tacotron: Towards end-to-end speech synthesis," *arXiv preprint arXiv:1703.10135*, 2017.

[16] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan *et al.*, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[17] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[18] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," in *International conference on machine learning*, 2013, pp. 1337–1345.

[19] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kaufmann Publishers, 2017.

[20] P. B. Maria Avgerinou and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," 2017. [Online]. Available: https://developer.amazon.com/de/blogs/alexa/post/7ab9665a-0536-4be2-aaad-18281ec59af8/varying-speaking-styles-with-neural-text-to-speech

[21] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. S. Khudia, J. Law, P. Malani, A. Malevich, N. Satish, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. M. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *CoRR*, vol. abs/1811.09886, 2018. [Online]. Available: http://arxiv.org/abs/1811.09886

[22] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International Conference on Machine Learning*, 2014, pp. 1764–1772.

[23] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[24] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Twelfth annual conference of the international speech communication association*, 2011.

[25] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *2012 IEEE International conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 4409–4412.

[26] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[27] A. See, M.-T. Luong, and C. D. Manning, "Compression of neural machine translation models via pruning," *arXiv preprint arXiv:1606.09274*, 2016.

[28] S. Han and B. Dally, "Efficient methods and hardware for deep learning," *University Lecture*, 2017.

[29] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[30] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[31] S. Cao, C. Zhang, Z. Yao, W. Xiao, L. Nie, D. Zhan, Y. Liu, M. Wu, and L. Zhang, "Efficient and effective sparse lstm on fpga with bank-balanced sparsity," 02 2019, pp. 63–72.

[32] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[33] W. Dally, "High-performance hardware for machine learning," *NIPS Tutorial*, 2015.

[34] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[35] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

NOTES

[1]© Intel Corporation. Intel and Stratix are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

[2]© Myrtle Software Limited. MAU Core and MAU Accelerator are trademarks of Myrtle Software Limited in the U.K. and/or other countries.

[3]Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of May 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product or component can be absolutely secure.

Intel does not control or audit third-party data. You should review this content, consult other sources, and confirm whether referenced data are accurate.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Myrtle does not control or audit third-party data. You should review this content, consult other sources, and confirm whether referenced data are accurate.

Myrtle disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.