REACT: A Heterogeneous Reconfigurable Neural Network Accelerator with Software-Configurable NoCs for Training and Inference on Wearables

Mohit Upadhyay[1], Rohan Juneja[1], Bo Wang[2], Jun Zhou[3], Weng-Fai Wong[1], and Li-Shiuan Peh[1]
[1]School of Computing, National University of Singapore
[2]Singapore University of Technology and Design, Singapore
[3]Institute of High Performance Computing, A*STAR, Singapore

ABSTRACT

On-chip training improves model accuracy on personalised user data and preserves privacy. This work proposes REACT, an AI accelerator for wearables that has heterogeneous cores supporting both training and inference. REACT's architecture is NoC-centric, with weights, features and gradients distributed across cores, accessed and computed efficiently through software-configurable NoCs. Unlike conventional dynamic NoCs, REACT's NoCs have no buffer queues, flow control or routing, as they are entirely configured by software for each neural network. REACT's online learning realises upto 75% accuracy improvement, and is upto 25× faster and 520× more energy-efficient than state-of-the-art accelerators with similar memory and computation footprint.

***Keywords**: neural networks, accelerators, wearables*

ACM Reference Format:
Mohit Upadhyay[1], Rohan Juneja[1], Bo Wang[2], Jun Zhou[3], Weng-Fai Wong[1], and Li-Shiuan Peh[1], [1]School of Computing, National University of Singapore, [2]Singapore University of Technology and Design, Singapore, [3]Institute of High Performance Computing, A*STAR, Singapore, . 2022. REACT: A Heterogeneous Reconfigurable Neural Network Accelerator with Software-Configurable NoCs for Training and Inference on Wearables. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530406

1 INTRODUCTION

There have been a plethora of specialized hardware accelerators for inference targeted for deployment at the edge[1][5]. However, neural network training is still largely done on the cloud, as it is extremely compute-intensive. Yet, on-chip training support on the edge is becoming increasingly important as the model can be personalized to the user's own data, improving the accuracy while also averting the need to share private user data over the cloud.

Modern *Convolutional Neural Networks* (CNNs) consist of a set of convolutional and pooling layers, used for feature extraction followed by a set of fully connected (FC) layers, used for classification. During transfer learning for edge deployment, feature extracting CNN layers do not require significant changes. Hence, on-chip learning has largely focused on re-training of the FC layers [17] to personalise the model for the user, improving the accuracy.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530406

Backward propagation for training uses inference output to calculate weight and error gradient of previous layers, which help to calculate the updated weights. We leverage this in our design of REACT, an AI accelerator that supports both training and inference within the tight area/power budgets of wearables.

The REACT accelerator comprises a heterogeneous mix of GIGA cores architected to handle training and inference of FC layers, and nano cores that are designed to accelerate inference of convolutional layers. These cores are then interconnected by three specialized NoCs, which efficiently add and propagate weights, output feature maps (during inference) for both nano and GIGA cores and error gradients (during training) for GIGA cores.

Here we summarize the key contributions and results of REACT. **(i)** This work introduces REACT, an accelerator targeting neural network inference and training for wearables. REACT's efficiency is due to a unique fully distributed architecture with heterogeneous cores specialized for different layers, interconnected by three specialized software-configurable NoCs that can efficiently move error gradients, weights, and feature maps on-chip, reducing costly off-chip accesses. Unlike prior works, REACT's NoCs need no hardware support for buffer queues, flow control or routing, relying fully on software synthesis for mapping to diverse neural networks. **(ii)** Our results show the benefits of on-chip training, with pre-trained model accuracy improved by up to 75% across multiple datasets, and speedup of more than 900× over software training on edge CPUs, enabled by the on-chip training acceleration of REACT. **(iii)** Our architectural evaluations also show that besides accelerating training, REACT can perform on-chip inference for the given datasets and benchmarks 2.4× faster at 2× better energy consumption on average, in comparison to state-of-the-art baselines with similar footprint, thus providing a full on-chip solution for accelerating on-device training and learning for wearables.

Section 2 discusses the motivation for on-chip training. Section 3 details the proposed REACT accelerator design. Section 4 describes the software mapping for different neural network types onto REACT. Section 5 evaluates REACT, and compares it with the state-of-the-art. Section 6 briefly surveys the related works while Section 7 concludes the paper.

2 MOTIVATION

REACT is designed for on-chip learning for wearables, where the user data is similar to the dataset on which network was pre-trained on the cloud (source dataset), just personalized to the user (target dataset). On-chip learning of a pre-trained network is performed on the target dataset, as has been performed in [3][2]. For our evaluations (see experimental details in Section 5), we emulate

transfer learning [17] for the CNN benchmarks and perform on-chip learning for the Multi-layer Perceptron (MLP) benchmarks. Figure 1 shows how on-chip training on target dataset can improve model accuracy by up to 14% and 75% for MLP and CNN respectively. The accuracy prior to on-chip training is shown in Figure 1 (at epoch 0).

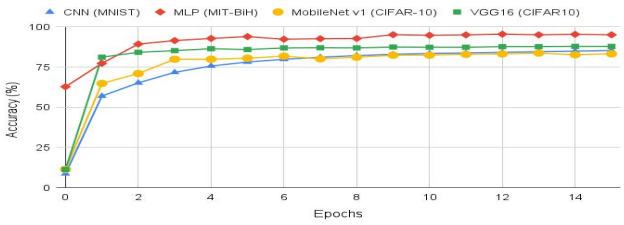

Figure 1: Accuracy improvement using on-chip training.

3 REACT ARCHITECTURE

3.1 Architectural Overview

REACT comprises a mix of GIGA cores, optimized for accelerating training and inference of *fully-connected layers* and nano cores, optimized for inference of the *convolutional layers*. This allows REACT to balance the needs for training and inference within the tight area/power constraints of wearables. REACT performs all computation using bfloat16 arithmetic which has been shown to be more efficient for deep learning applications. Only GIGA cores have additional hardware support for training: A 1D systolic array of multipliers that calculate weights gradients, and the GIGA neuron core concurrently calculates the error gradients. The neuron core updates the weights using the calculated weight gradients. A homogeneous architecture where every core has training support will lead to fewer cores within the same area constraints, and much lower utilization, nullifying the benefits of CNN acceleration since CNN mapping on GIGA core would involve much higher replication and lower utilisation of weights.

REACT cores are interconnected by three types of Networks-on-Chip (NoCs). The NoCs are all fully software configurable, synthesised by the software mapper (see Section 4), like FPGA switch-boxes. They thus save on hardware support for buffering, flow control or routing. The NoCs are specialized for specific functions for efficiency. All cores (GIGA and nano) are interconnected with per-neuron partial sum (PS) NoCs that can perform in-network partial sum computation for inference and training [16]. All cores are also connected by per-neuron weighted-sum (WS) NoCs with a ReLU engine for inference, which routes outputs during inference and on-chip training. The nano cores have a third type of NoC, an adder tree NoC connecting the multipliers and adders within, allowing configuration of the compute units to match the exact CNN kernels. The partial sum and weighted-sum NoCs are meshes as the mesh topology maps readily to the grid floorplanning of REACT cores. The adder tree NoC is a tree topology as that allows prefix addition of variable-sized adds. The PS and WS NoCs are interfaced to small network interface (NIC) buffers to facilitate core reuse and reduce off-chip accesses for input/output feature maps, partial sum values and error gradients. The training buffer is a small memory to support the training process by buffering the fully connected layers' outputs during the forward propagation stage, since the GIGA cores are reused for fully connected layers. REACT has local configuration memories for each core. In Figures 3 and 4, the configuration memory is fed by the control unit which decodes instructions and sends the respective control signals to each core.

Figure 2 shows the REACT architecture with 8 nano cores and 2 GIGA cores, each with 256 neurons, connected to 256 10-node PS mesh NoCs, and 256 10-node WS mesh NoCs. Within each nano core is a six-level adder tree NoC to reduce the 7×7 multiplier outputs. Since number of filters in convolutional layers are typically multiples of 8, such a configuration improves mapping efficiency.

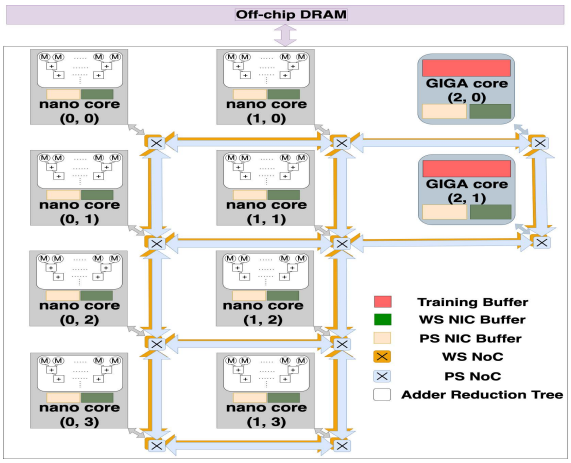

Figure 2: REACT architecture with 2 GIGA cores and 8 nano cores (each with an adder tree NoC), interconnected by 256 per-neuron partial sum NoCs and 256 weighted sum NoCs

3.2 GIGA Core

GIGA neuron cores accelerate fully-connected layers. A GIGA core comprises of three logical memories storing upto 256×256 synaptic weights, the transposed weights and the weights gradient values. It has 256 MAC(multiply-accumulate) compute units and associated control logic. The micro-architecture of the GIGA neuron core is shown in Figure 3(a).

The GIGA neuron core was specifically designed to efficiently perform matrix-vector multiplications, as is required during inference and training. During inference, GIGA neuron core multiplies the input feature maps to weights. The partial output feature maps are sent to the 256 per-neuron PS or WS NoCs depending on the mapping. During back propagation phase of training, the input error gradient values, sent from the input port of weighted-sum NoC are multiplied with the transposed weight matrix values to generate the output error gradient feature. The output error gradient may be sent to the weighted-sum NoC directly or to the partial sum NoC based on the mapping. The GIGA neuron core updates weights using the on-chip stored weights, and its gradients after the backpropagation phase of training.

1D systolic array. The GIGA core also houses a 1D systolic array which accelerates training. The systolic array of 256 multipliers is shown in Figure 3(d). It multiplies the error gradient values (row vector) and inference inputs (column vector) streamed in from the

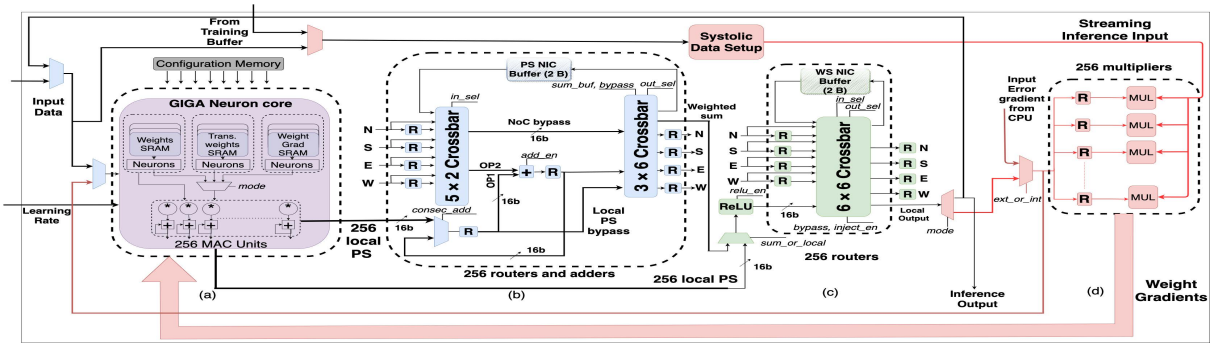

Figure 3: Micro architecture of (a) GIGA neuron core, (b) partial sum router, (c) weighted sum router with the ReLU Unit and (d) 1-D systolic array

local activation input or the training buffer, producing the weight gradient values, which are sent to the GIGA neuron core.

3.3 Nano Core with Adder Tree NoC

The nano neuron core accelerates the convolution neural network layer and consists of two parts: A systolic data setup unit which is responsible for streaming in receptive field(s) for convolution at every clock cycle; and a 7×7 multiplier array feeding an adder tree NoC. The multiplier array outputs are fed into a tree of adders to perform convolution effectively. At each level of the adder tree NoC, a bypass path allows software to map multiple receptive fields for smaller kernels. We discuss the basic mapping of different sized kernels in Section 4. Each core can concurrently hold 64 different kernels with each kernel up to a maximum size of 7×7.

During inference, the inputs to the nano neuron core are the input feature maps and the weights to the SRAM bank from the CPU. The outputs from the core are then sent to each neuron's partial sum NoCs when neuron core outputs are partial sum values or to the weighted sum NoCs if the neuron core outputs are full sum values.

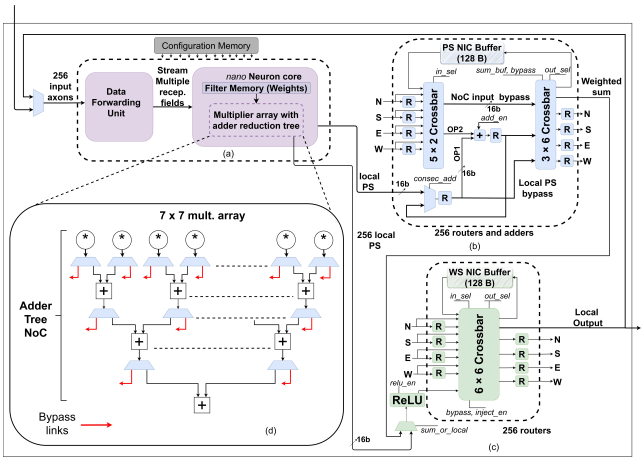

Figure 4: Microarchitecture of (a) nano neuron core with (b) PS NoC routers, (c) WS NoC routers and (d) adder tree NoC

3.4 Partial-Sum NoC

The partial-sum (PS) NoC is leveraged from the Shenjing inference accelerator [16]. Each neuron has its own PS NoC, so each core has 256 PS routers.The PS NoC routers have adders within its datapath to perform in-network summation and aggregation of the feature values (during inference) and error gradient values (during training). It is designed to perform partial sum within a layer mapped to a core. The partial sum routers are configured by the software mapper to add sums coming from any of the nearby cores and any previous result(s) stored in the router.

As shown in Figures 3(b) and 4(b), a 5×2 input crossbar fetches input data from a port (North, South, East, West or PS NIC Buffer) and either registers it for local addition or bypasses the given router to adjacent router using the output links. The 3×6 output crossbar ejects the data to the desired output port or the ReLU unit of the WS NoC.

3.5 Weighted-Sum NoC

The Weighted-Sum(WS) NoC is architected to support both forward propagation of the full-weighted output feature maps (during inference) and error gradients (during back-propagation phase of training), so a single NoC can be software-configured for use along opposite directions during forward and backward passes which occur at different times, essentially time multiplexing the NoC for better utilization and efficiency (see Fig 3(c)). The weighted-sum NoC routers send the error gradient values across different cores. For the last layer, the error gradient is received as an input from off-chip. Otherwise, the 6×6 crossbar routes the error gradient value from its output port and is sent to the systolic array and the GIGA neuron core to calculate the weight gradient and output error gradient values respectively. The output error gradient values are then sent to other core(s) based on the mapping through the input port of the 6×6 crossbar and ejecting it through one of the output ports (North, South, East, West or the Weighted-sum NIC Buffers). The Weighted-sum NIC Buffers are designed to store one set of error gradients feature to reduce the off-chip accesses during training.

4 MAPPING ANN ONTO REACT

REACT's software mapping algorithm enables automated mapping of diverse target neural networks onto REACT, and synthesises

the configuration signals for REACT cores and NoCs. It essentially schedules the cycle-by-cycle operation of the cores and NoCs, ensuring efficient, contention-free use of the resources.

4.1 Mapping for Fully Connected Layers

To map an $m \times n$ FC layer with m inputs and n outputs where both m and n exceed the core size, we need *inv* invocations, where $inv = \lceil \lceil m/v * n/v \rceil / (n_{row} * n_{col}) \rceil$ and v are the number of input and output neurons of one core. We arrange the $n_{\text{row}} \times n_{\text{col}}$ cores in a rectangle. The rows receive the m inputs and columns produce n outputs. Algorithm 1 shows the logical scheduling of partial-sum NoCs to produce the total weighted sum.

Algorithm 1: Mapping of fully-connected layer onto REACT's NoC

```
Input  : n_row × n_col cores in rectangle with local partial-sum
         subroutine PS.Send(i, j, input), and weighted-sum
         WS.Send(i, j, input)
Output : Network trace N
N ← 0, inv ← ⌈⌈m/v * n/v⌉/(n_row * n_col)⌉
// Loading data from WS buffer for inter-layer communication
N.add( WS.Send(WS.Buffer(..) to Inp_data) )
// Partial Sum Computation
for f ← 1 to inv                                  // Partial Sum Computation
do
    L ← 0
    for i ← 1 to n_row and j ← 1 to n_col do
        L.add( PS.Send(i, j, (Output(i, j) to PS.Buffer(i,j)) ) )
    end
    N.add(L)
end
// Send PS output to WS input for all cores
N.add( WS.Send(PS.Buffer(..) to WS.Buffer(..)) )

if train == 1 then
    N.add( WS.Send(WS.Buffer(..) to Inp_data) )
    for f ← 1 to inv do
        L ← 0
        for i ← n_row to 2 and j ← 1 to n_col do
            L.add( PS.Add(i, j) to PS.Add(i − 1, j) )
            L.add( PS.Send(i − 1, j, PS.Buffer(i − 1, j)) )
        end
        N.add(L)
    end
    N.add( WS.Send(PS.Buffer(..) to WS.Buffer(..)) )
end
```

4.2 Mapping Convolution Layer

To map a convolution layer with input size $h \times w \times c_{\text{in}}$ and kernel size $k \times k \times c_{\text{in}} \times c_{\text{out}}$ onto $n_{\text{row}} \times n_{\text{col}}$ cores, *inv* invocations are performed in a time-multiplexed fashion. Each neuron core can perform convolution on $I \times I$ sized partition of the image, thus the number of invocations required to cover all input and output channels are $inv = \lceil (h \times w)/(I \times I \times n_{row} \times n_{col}) \rceil$.

Each nano neuron core can fit upto 64 output channel filters. The weights are mapped such that each neuron completes the convolution in one input channel before moving to the next.

To deal with the underutilization of the multiplier array, we designed an adder tree NoC that has software-configurable bypass links at each level of the adder tree as discussed in section 3.3. These links help to spatially schedule multiple computations corresponding to a kernel and bypass the rest of the adder tree. The software mapper can thus configure and schedule multiple smaller kernels spatially, or one kernel with size equal to or larger than 5×5.

Algorithm 2: Mapping of convolution layers onto REACT's NoC

```
Input  : n_row × n_col cores, with F × F multiplier array, with
         PS.Send(i, j, input), and WS.Send(i, j, input).
         And the convolution layer of input size h × w × c_in,
         and kernel size k × k × c_cin × c_out.
Output : Network trace N
N ← 0
// Adder tree NoC bypass logic
intra_fold ← ⌊(F * F)/2^⌈log2(k*k)⌉⌋
sch ← ⌈log2(k * k)⌉                    // Map weights spatially for utilization
sch_arr ← [i*2^⌈log2(k*k)⌉ for i in range(sch)]
inv ← ⌈(h*w)/(I*I) * 1/(n_row*n_col)⌉                  // Number of invocations
N.add( WS.Send(WS.buffer(..) to Inp_data) )
for f ← 1 ; f < inv ; f = f + 1                  // Mapping onto Nano cores
do
    for in_ch ← 1 to c_in do
        for out_ch ← 1 to ⌈c_out/64⌉ do
            for intra_out_ch ← 1 to
              min(64, c_out − (64 * out_ch)) do
                L ← 0
                for x, y ← 1 to n_row, n_col     // Nano core operation
                do
                    L.add( PS.Send(x, y, (inputs[in_ch][..],
                      kernels[in_ch][out_ch], bypass,
                      sch_arr)) )
                end
            end
        end
        N.add(L)
    end
end
N.add( WS.Send(PS.buffer(..) to WS.Buffer(..)) )
```

As a convolution layer scans through the input image, there will be overlap at the boundary of each core. To produce the convolution sum, along each channel, the neighbouring cores first exchange their partial-sums to produce the convolution of the boundary pixels. Then, among the channels, the partial sums are accumulated to complete the convolution.

The mapping between nano and GIGA cores is similar to the way FC layers are mapped. The nano core outputs are directly mapped to the same number of GIGA cores receiving these outputs as their input axons.

5 EVALUATION

5.1 Target Neural Networks

In our experiments, we use 7 different benchmarks to evaluate REACT architecture, representative of real world workloads. They are 2-layered MLP and 4-layered CNN neural network architectures for the MNIST[11] dataset, a 5-layered MLP neural network for the MIT-BiH ECG dataset and 9-layered CNN for CIFAR-10 [8] dataset. REACT is able to fit neural network layers of varying sizes without accuracy loss due to the partial sum NoCs. The MIT-BiH [9] MLP network was chosen as the MIT-BiH dataset [13] is used to detect cardiac arrythmia, a key application on wearables. We also use another CNN network with 6 layers targeting the EMNIST [4] dataset and use the VGG-16 and MobileNet v1 benchmarks with the CIFAR-10 dataset. We also evaluate MobileNet v1 on ImageNet to show the scalability of REACT. MobileNet v1 illustrates that REACT can support deeper networks as well on wearables.

5.2 Experimental methodology

To emulate transfer learning, CNN benchmarks targeting CIFAR-10 and MNIST (target datasets) are pre-trained on ImageNet and EMNIST (source datasets) respectively. The on-chip training and inference evaluations are performed on the target datasets. For all other benchmarks, we emulate on-chip learning of a pre-trained model by training the FC layers on a subset of their respective source datasets.

We modeled REACT in RTL, then faithfully replicated the operations performed by the RTL during forward and backward passes of the training process in a high-level simulator using python libraries to ensure tractable simulation times. REACT was synthesized at 120 MHz for real-time inference for the given benchmarks. The accuracy results (Figure 1) were obtained using these simulations where pre-trained models were obtained from pytorch.

The power and area results of REACT's cores and NoCs are obtained through RTL modelling and synthesis. The design was implemented and verified on SystemVerilog HDL and its functionality was verified using the Synopsys VCS tool. The synthesis was run using the Synopsys Design Compiler to synthesize the RTL design to gate-level netlists on a commercial 22nm CMOS process and SRAMs used were foundry's memory-compiled SRAM cells.

5.3 Architecture Synthesis Results

5.3.1 Area. Each GIGA core (GIGA Neuron Core, NoC routers and the 1D systolic array) in the REACT architecture is synthesized at 0.8V, obtaining a core size of 1.993 mm^2. Each nano core (nano Neuron Core and NoC routers) in the REACT architecture is also synthesized at 0.8V and takes 1.148 mm^2. The total area of the REACT configuration in Figure 2 is 13.17 mm^2. Table 1 shows the area distribution of the GIGA and nano core.

5.3.2 Power. The power consumption is obtained from synthesis as mentioned above. The total power consumed by a GIGA core is 114.7mW when synthesized at 120MHz. The power distribution of the GIGA neuron core is shown in Table 1. The GIGA neuron core takes a significant percentage of the power as it has have the majority of the computational elements in the architecture. The 1D systolic array and the WS NoC are gated during the inference phase and are only operational during on-chip training.

The total power consumed by a nano core is 91.78 mW when synthesized at 120MHz. The power distribution of the nano neuron core has been shown in the Table 1. As expected, power consumption of a nano core is lower than GIGA core due to its lower complexity. Since the nano cores can process upto 64 different output channels, the PS and WS NoCs in the nano cores have 64× larger NIC buffers than the respective buffers in the GIGA cores. These NIC buffers hold neural network data on-chip while cores are reused.

5.4 Evaluation Results

Table 2 shows the latency for on-chip training on REACT, in comparison to ARM CPU, Nvidia GPU on the Jetson nano[6] board (power output of 5W) and Edge-TPU board[5] (power output of 2W) timing measurements. The Edge-TPU can only train the last classifier layer, unlike others which can train all FC layers. REACT is faster than the CPU and GPU baselines by 426× and 183× respectively on average for all given benchmarks. Edge-TPU board is 15% faster than REACT, with 4.6× higher average power consumption but Edge-TPU achieves only a 5% accuracy improvement using its on-chip training.

Core	Sub-module	Active Power (mW)	Active Power (% per core)	Area (mm^2)	Area (% per core)
GIGA core	GIGA neuron core	76.031	66.29	1.552	77.87
	256 per-neuron PS NoC routers	14.133	12.32	0.188	9.43
	256 per-neuron PS NIC buffers	1.439	1.25	0.0098	0.49
	256 per-neuron WS NoC routers	12.448	10.85	0.126	6.32
	256 per-neuron WS NIC buffers	1.439	1.25	0.0098	0.49
	1D systolic array	2.013	1.75	0.058	2.91
	Training Buffer	5.759	5.02	0.04	1.98
	Config. Memory	1.440	1.26	0.010	0.50
nano core	nano neuron core	8.28	9.02	0.265	23.09
	256 per-neuron PS NoC routers	26.59	28.97	0.188	16.38
	256 per-neuron PS NIC buffers	15.28	16.65	0.371	32.32
	256 per-neuron WS NoC routers	24.91	27.14	0.126	10.98
	256 per-neuron WS NIC buffers	15.28	16.65	0.188	16.38
	Config. Memory	1.440	1.57	0.010	0.86

Table 1: Synthesis results for GIGA and nano cores

Benchmarks	REACT @120MHz	ARM CPU @921.6MHz	Jetson Nano GPU @153MHz	Edge TPU @500MHz
2-layer MLP (MNIST)	0.0382	12.935	4.833	2.03
4-layer CNN (MNIST)	0.0176	7.444	6.683	1.19
6-layer CNN (EMNIST)	0.1243	152.245	15.788	4.26
5-layer MLP (MIT-BiH) [9]	0.0680	50.592	11.158	-
9-layer CNN (CIFAR-10)	0.1429	169.557	18.551	4.52
MobileNet v1 (CIFAR-10)	6.9109	5135.592	2817.617	6.22
MobileNet v1 (ImageNet)	31.1612	10854.013	4161.098	10.74
VGG-16 (CIFAR-10)	53.51	1934.52	295.79	23.56

Table 2: REACT Training Latency (in ms)

For performance comparison against architectural baselines, we used the SCALE-SIM [14] and the MAESTRO [10] architectural simulators to get the latency and utilization numbers for the other baselines. The four baselines are Eyeriss (modelled in MAESTRO) and systolic arrays having similar computational elements and memory as REACT but with weight stationary (WS), output stationary (OS) (similar to ShiDianNao[7]) and input stationary (IS) dataflows respectively, since these are the key dataflows used in neural network accelerators. For the sake of brevity, we shall refer to the configurations as Eyeriss-like, TPU-like, ShiDianNao-like and Sys. Array-IS respectively. For the REACT performance numbers, we use the high-level Python functional simulator described earlier, utilising the mapping algorithm.

Figure 5 shows the inference latency of REACT and the 4 baselines described above. REACT is faster by 3.5×, 2.25×, 1.5× and 2.2× on average over all given benchmarks in comparison to Eyeriss, TPU, ShiDianNao-like and Sys. Array-IS respectively. REACT has a much higher mapping efficiency for FC layers because of its GIGA cores in comparison to the baselines. Owing to its NoCs, REACT has higher utilization than the TPU configuration because of its better utilisation of its on-chip memory. REACT has an average of 56.9% compute utilization in comparison to the 10.1% utilization for the ShiDianNao-like, which has the highest utilization among all the baselines. The Systolic Array configurations also suffer from lower utilization in comparison to REACT, since systolic array architectures are not as effective for matrix-vector computations. REACT's NoCs provide 8× higher on-chip bandwidth in comparison to Systolic Array configurations with similar compute resources.

For energy comparisons, we obtained the RTL model of a TPU-like architecture from [15] to model power and area. The TPU-like architecture is also synthesized at the same clock frequency as

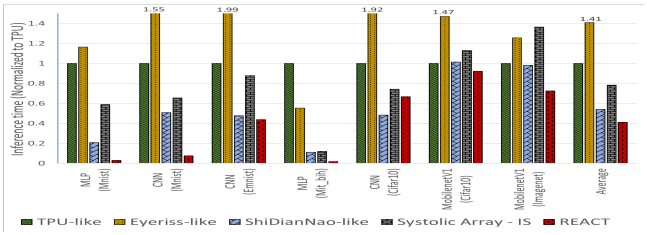

Figure 5: Inference Latency Comparison of REACT with baselines

REACT (120 MHz). In Figure 6, We can see the inference energy distributions for REACT and TPU, normalized to the TPU inference energy. REACT has a significantly lower energy consumption for shallower networks, which are typically ones targeted for wearables, it has lower energy by 2.1× on an average for all given benchmarks in comparison to the baselines respectively. This shows that REACT's specialized cores and software-configurable NoCs enable highly energy-efficient distributed computation on chip.

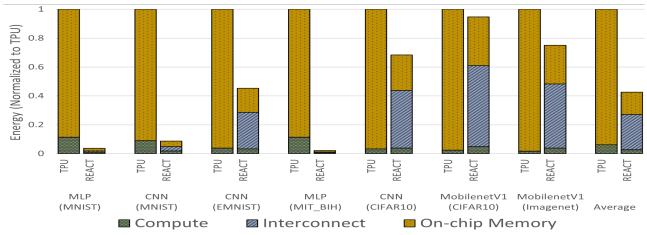

Figure 6: Energy Consumption during inference for given benchmarks

6 RELATED WORKS

There have been many custom hardware accelerator chips developed for ANN acceleration. We focus on the few that support on-chip training. As an apples-to-apples comparison of REACT with other training accelerators is difficult given the different technology node, datasets, neural networks, we briefly mention the differences between some of these prior accelerators and REACT.

TrainWare [2] is an on-device training accelerator for CNNs which focuses specifically on the weight update operation. The two key features are a specialized local buffer to reduce off-chip memory access and unique dataflow with a PE array. TrainWare only focuses on accelerating the weight updates stage unlike REACT, which accelerates all the stages of training while also having better area efficiency. Choi et. al.[3] propose an accelerator that supports inference and training for CNNs. The architecture is heterogeneous, with different types of cores for inference and training computation for CNNs and MLPs. This work has lower accuracy in comparison to REACT since it uses 8b fixed point arithmetic, which has much lower precision. Lu et. al.[12] also propose an accelerator that supports inference and training for CNNs by having multiple PS clusters and activation computation units. REACT's NoC-centric design leads to better area efficiency and higher utilization. Sticker[18] has an architecture that performs inference for sparse neural network models while also supporting tuning for FC layers. It has three major modules, a sparsity-aware controller, a multi-sparsity compatible convolution PE array and an efficient online tuning PE for sparse FC layers. This work has lower precision data in comparison to REACT which would lead to lower accuracy for inference and on-chip training.

A primary difference between the REACT accelerator's architecture and these prior accelerators is that the REACT architecture distributes the weights, output feature maps and error gradients across the cores and interconnects them with specialized NoCs rather than primarily relying on a single unified buffer. In short, REACT has a NoC-centric design which allows for higher utilization of memory and compute elements at finer granularity, key constraints on edge devices which improve battery life.

7 CONCLUSION

This work introduces REACT, a neural network accelerator for wearables that can support on-chip training and inference. REACT specialises its cores to specific neural network layers, and interconnects them with specialised software-configurable NoCs with in-network compute, thus enabling very lean, efficient hardware that is scheduled cycle-by-cycle by REACT's software mapper for different neural network applications. We see REACT as a full on-chip solution for personalized learning and inference for wearables.

8 ACKNOWLEDGEMENTS

This research is supported by the National Research Foundation, Singapore, under NRF-RSSS2016-05, and in part by Programmatic grant no. A1687b0033 from the Singapore government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain). We are also grateful to Prof. Tushar Krishna and Sheng-Chun (Felix) Kao from Georgia Institute of Technology for their help using the SCALE-Sim and MAESTRO tools.

REFERENCES

[1] Y. Chen et al. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep CNN. *IEEE JSSC* (2017).
[2] Seungkyu Choi et al. 2018. TrainWare: A Memory Optimized Weight Update Architecture for On-Device CNN Training. In *ISLPED '18*.
[3] S. Choi et al. 2019. A 47.4μJ/epoch Trainable Deep CNN Accelerator for In-Situ Personalization on Smart Devices. In *ASSCC 2019*. 57–60.
[4] Gregory Cohen et al. 2017. EMNIST: Extending MNIST to handwritten letters. In *2017 IJCNN*. 2921–2926.
[5] coral.ai. 2020. *Edge TPU Documentation.* https://coral.ai/docs/
[6] developer.nvidia.com/embedded/jetson modules. 2020. *Jetson Modules.*
[7] Z. Du et al. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ISCA 2015*. 92–104.
[8] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images.* Technical Report.
[9] Gaurav Kumar et al. [n. d.]. Arrhythmia Detection in ECG Signals Using a MLP Network. ([n. d.]).
[10] Hyoukjun Kwon et al. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *MICRO '52*. 754–768.
[11] Y. Lecun et al. 1998. Gradient-based learning applied to document recognition. *IEEE Proceedings* 86, 11 (1998), 2278–2324.
[12] C. Lu et al. 2019. A 2.25 TOPS/W Fully-Integrated Deep CNN Learning Processor with On-Chip Training. In *ASSCC 2019*.
[13] G. B. Moody et al. 2001. The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine* 20, 3 (2001), 45–50.
[14] Ananda Samajdar et al. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *ISPASS 2020*. 58–68.
[15] Ananda Samajdar et al. 2021. TPU: Tensor-Processing-Unit. https://github.com/scalesim-project/scale-sim-v2/tree/main/code-examples/systolic-array-rtl.
[16] Bo Wang et al. 2020. Shenjing: A low power reconfigurable neuromorphic accelerator with partial-sum and spike networks-on-chip. In *IEEE DATE*.
[17] Jason Yosinski et al. 2014. How transferable are features in deep neural networks? *CoRR* abs/1411.1792 (2014). arXiv:1411.1792 http://arxiv.org/abs/1411.1792
[18] Z. Yuan et al. 2018. Sticker: A 0.41-62.1 TOPS/W 8Bit Neural Network Processor with Multi-Sparsity Compatible Convolution Arrays and Online Tuning Acceleration for Fully Connected Layers. In *VLSI 2018*.