Rethinking Programmable Earable Processors

Nathaniel Bleier
University of Illinois
Urbana, Illinois, USA
nbleier3@illinois.edu

Muhammad Husnain Mubarik
University of Illinois
Urbana, Illinois, USA
mubarik3@illinois.edu

Srijan Chakraborty
University of Illinois
Urbana, Illinois, USA
srijanc2@illinois.edu

Shreyas Kishore
University of Illinois
Urbana, Illinois, USA
kishore4@illinois.edu

Rakesh Kumar
University of Illinois
Urbana, Illinois, USA
rakeshk@illinois.edu

ABSTRACT

Earables such as earphones [15, 16, 73], hearing aids [28], and smart glasses [2, 14] are poised to be a prominent *programmable* computing platform in the future. In this paper, we ask the question: *what kind of programmable hardware would be needed to support earable computing in future?* To understand hardware requirements, we propose *EarBench*, a suite of representative emerging earable applications with diverse sensor-based inputs and computation requirements. Our analysis of EarBench applications shows that, on average, there is a 13.54×-3.97× performance gap between the computational needs of EarBench applications and the performance of the microprocessors that several of today's programmable earable SoCs are based on; more complex microprocessors have unacceptable energy efficiency for Earable applications. Our analysis also shows that EarBench applications are dominated by a small number of digital signal processing (DSP) and machine learning (ML)-based kernels that have significant computational similarity. We propose *SpEaC* — a coarse-grained reconfigurable spatial architecture – as an energy-efficient programmable processor for earable applications. SpEaC targets earable applications efficiently using a) a reconfigurable fixed-point multiply-and-add augmented reduction tree-based substrate with support for vectorized complex operations that is optimized for the earable ML and DSP kernel code and b) a tightly coupled control core for executing other code (including non-matrix computation, or non-multiply or add operations in the earable DSP kernel code). Unlike other CGRAs that typically target general-purpose computations, SpEaC substrate is optimized for energy-efficient execution of the earable kernels at the expense of generality. Across all our kernels, SpEaC outperforms programmable cores modeled after M4, M7, A53, and HiFi4 DSP by 99.3×, 32.5×, 14.8×, and 9.8× respectively. At 63 mW in 28 nm, the energy efficiency benefits are 1.55×, 9.04×, 68.3×, and 32.7× respectively; energy efficiency benefits are 15.7× - 1087× over a low power Mali T628 MP6 GPU.

ISCA '22, June 18–22, 2022, New York, NY, USA

ACM ISBN 978-1-4503-8610-4/22/06…$15.00
https://doi.org/10.1145/3470496.3527396

ACM Reference Format:
Nathaniel Bleier, Muhammad Husnain Mubarik, Srijan Chakraborty, Shreyas Kishore, and Rakesh Kumar. 2022. Rethinking Programmable Earable Processors. In *The 49th Annual International Symposium on Computer Architecture (ISCA '22), June 18–22, 2022, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3470496.3527396

1 INTRODUCTION

Different programmable computing platforms emerge at different times [18] driven by both technology as well as applications. One programmable computing platform that may be on the cusp of explosion is that of "earables" [10] – devices such as earphones [15, 16, 73], hearing aids [28], and smart glasses [2, 14] that are worn in or around the ear and that interact with humans mostly through acoustics. These devices have a large number of sensors [38] which can be leveraged to support a variety of applications (Figure 1).

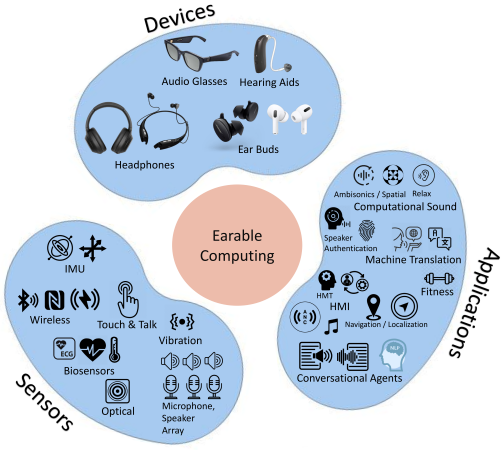


Figure 1: Emerging earable devices will be programmable platforms with a rich set of sensors and will support a variety of applications.

There are several signs that earables may become a prominent programmable computing platform. First, current global market for earables is already over 20 billion dollars [68] and is expected to over 90 billion dollars in next five years [68]. Second, hardware vendors are already building earables with increasingly complex sensors (e.g., proximity sensors, gyroscopes, capacitive sensors, accelerometers, SpO2, EEG, EOG, EMG sensors, microphone and speaker array, etc.) which are being leveraged to support a wide

variety of computing applications [4, 27, 36]. Third, many earable platforms are already programmable and increasingly so. In fact, a large variety of software applications processing different sensor data are already supported in many of today's earables [4, 27, 36]) or are being explored [63, 64, 66, 78].

In this paper, we ask the question: *what kind of programmable hardware would be needed to support earable computing in the future?* Today's programmable hardware for earables [52, 53, 60, 61, 65] need to support only limited computation and sensing and typically consist of either microcontrollers or DSPs or both, running at low to moderate frequencies. The choice of the earable microcontrollers (Cortex M-4 and M-7 cores are popular due to their DSP extensions) and DSPs (Tensilica HiFi DSPs are popular due to their relatively low power), their number, and their operating frequencies are limited by the capacity limitations of today's earable batteries (e.g., AirPods Pro uses a 45.4 mA h, 3.7 V battery [72]). It is unclear if the same processors and programmable hardware will be adequate for future earable applications that are likely to have much higher computational needs and process much more diverse sensor data. Simply choosing more powerful microprocessors and DSPs may not suffice either since the severe energy constraints of the earable batteries still need to be met. On a AirPods Pro 45.4 mA h battery, for example earable hardware should draw only tens of mWs of power for acceptable experience.

To understand the hardware requirements for emerging earable applications, we propose *EarBench*, a suite of representative emerging earable applications (Section 2) with diverse sensor-based inputs and computation requirements. The EarBench applications have been chosen through a comprehensive survey of earable applications research [10], personal correspondence with domain experts [69], and profiling of earable applications to to ascertain their execution requirements.

We analyze (Section 3) the performance of EarBench applications on a spectrum of programmable microprocessors, including those roughly modeled after ARM Cortex-M4 and Cortex-M7 - these cores are a part of several current programmable earable SoCs). We find that there is a 13.54×-3.97× performance gap between the computational needs of EarBench applications and the performance of today's Cortex-M4 and Cortex-M7-based programmable earable hardware. More complex microprocessors (e.g., A53) can meet the performance requirements for several EarBench applications, but their high power requirement (e.g., 276 mW to 339 mW for A53) severely restricts functionality (e.g., number of seconds of audio that can be processed, number of inferences and localizations that can be performed, etc. before battery runs out), rendering them unacceptable.

Our analysis of EarBench applications also shows that such applications are dominated by a small number of diverse digital signal processing (DSP) and machine learning (ML) kernels – GEMM, GEMV, FFT, bilinear filtering, 1-D and 2-D convolution, and LU. Any programmable earable hardware must provide high performance on these kernels.

We propose *SpEaC*[1] — a coarse-grained reconfigurable spatial architecture (Figure 6) to target earable applications efficiently. SpEaC has a fixed-point multiplier and adder tree-based distribute-then-reduce substrate with support for vectorized complex operations that energy-efficiently accelerates the earable ML and DSP kernels and that can be configured for each kernel based on its dataflow. A tightly-coupled scalar, in-order CPU is responsible for configuration and stream-based programming. The CPU also executes code that does not fit well on the reconfigurable substrate, including non-multiply or add operations in the earable DSP kernel code. Unlike recent CGRAs such as SoftBrain [57] whose substrate is designed to support general-purpose computation and lacks direct support for complex operations, SpEaC substrate is optimized for energy-efficient execution of the earable kernels at the expense of generality.

Across our kernels, SpEaC outperforms programmable cores modeled after M4, M7, A53, and HiFi4 DSP [75] 99.3×, 32.5×, 14.8×, and 9.8× respectively.

At 63 mW in 28 nm, the energy efficiency benefits are 1.55×, 9.04×, 68.3×, and 32.7× respectively; energy efficiency benefits are 15.7× - 1087× over a low power Mali T628 MP6 GPU. Kernel-level benefits also translate into application-level benefits. SpEaC meets application-level performance requirements of 9 out of 11 EarBench applications; applications can now run for long periods on an earable class battery.

2 EARBENCH: A SUITE OF EMERGING EARABLE APPLICATIONS

To construct EarBench, a suite of representative emerging earable applications, we conducted a comprehensive survey of earable applications research and practice [12, 44, 59, 62, 70], including over 50 papers on *http://esense.io*, and interviewed three earable computing domain experts [69]. We observed that most research and commercial earable computing applications fall into one of four categories: (1) *Human-Machine Interface* (HMI) applications that allow the earable user to interact with the device (and vice-versa). In the absence of traditional interfaces (e.g., keyboard, mouse, touchscreen), interactions are typically via sound and taps. (2) *Audio* applications that decode, manipulate, and playback audio data streams (e.g., phone calls, podcasts, acoustic augmented reality (AAR)). (3) *Analytics* applications, such as ECG monitors, that are computation-heavy but should ideally execute locally on the earable device. Finally, (4) *Spatial Awareness* applications that localize the user, their movements, head gestures, etc. for applications in AR/VR, gaming, location-specific alerts, etc. These spatial tracking applications are often characterized by sensor fusion engines, and are thus assigned to a separate category. To have a representative, but diverse suite, we then selected from each category 2-4 applications with different computational requirements, as determined by profiling (Table 2), and diverse sensor-based inputs. Diversity was considered also in terms of underlying algorithms (e.g., DSP vs ML, RNN vs CNN vs BERT) and use cases (e.g., authentication vs recognition, classification vs NLP, tracking vs localization). Once the selection was finalized, each application was either custom written or corresponding high quality open source implementation was found. We then packaged the collection of custom written and open source CPU-based software implementations of these applications, including

[1] Spatial Earable Computer

Table 1: EarBench: Inputs, outputs and timing requirements.

Applications	Inputs / Outputs / Timing Requirements
tinySR	1s audio / Small vocabulary English words / real time ($\leq$ 1 s)
DeepSpeech	5s audio / English text / real time ($\leq$ 1 s)
Ambisonic-2/6/8	10s audio / Binauralized audio / real time ($\leq$ 1 s)
HMT-1	IMU@100 Hz / dead-reckoned location of the user / $\leq$ 1 s
HMT-10	IMU@1 kHz / dead-reckoned location of the user / $\leq$ 1 s
HRTF - 2/6/8	8sec audio / binauralized audio / $\leq$ 1 s
Speaker Auth.	1s audio / Classifies the speaker as one of the 10 authorized speakers, or as unauthorized / real time ($\leq$ 1 s)
ALBERT	question (10 words) / start_logits: the logits of being the start of the answer of each position end_logits: the logits of being the end of the answer of each position. / $\leq$ 3 s
DeepECGCNN	9s ECG signal / Four output classes 1) Normal sinus rhythm, 2) Arrhythmic, 3) Other kind of rhythm, 4) Very noisy, / 9s
Wave2Letter	MFCC with input tensor share (1, 296, 39) / Tensor of time and class probability of shape (1, 1, 148, 29) / real time ($\leq$ 1 s)
TLIO-1DResNet18	IMU@200HZ/ Two 3D vectors, the displacement estimates and their uncertainties / $\leq$ 0.1 s/inference
EKF	IMU data 75k samples / Orientation of IMU / $\leq$ 1 s

build files, input data, and test vectors (several of which are developed in-house) into a suite.

2.1 HMI

TinySR [63] is a real-time speech recognition agent for a small vocabulary (10s to 100s of words), stored as an acoustic model. TinySR can be used by other applications to create speech-based interfaces (e.g., audio playback controlled via commands such as "stop", "next track", etc). These acoustic models use traditional DSP techniques for classification, such as Fast Fourier Transforms (FFT), and Discrete Cosine Transforms (DCT).

DeepSpeech [32] performs large-vocabulary speech recognition, transforming arbitrary speech utterances into text. DeepSpeech uses a recurrent neural network (RNN) to produce a phonetic, textual version of the speech utterance. This is then passed into an N-gram based language model which produces a 'proper' text of the speech utterance. EarBench includes the TensorFlow Lite version of this system, developed by Mozilla.

Wav2letter [22] is an end-to-end speech recognition system built out of a convolutional neural network (CNN). Like DeepSpeech, Wav2letter also performs large vocabulary speech recognition, but it differs in its CNN-based approach instead of an RNN-based one.

Speaker Auth. [37] verifies whether an input speech signal is indeed from the legitimate/registered user. The application uses the Mel-frequency cepstrum coefficients (MFCC) as input to a Gaussian-mixture model classifier.

2.2 Audio

Ambisonic decodes and binauralizes a surround sound audio stream (i.e., creates two versions of the sound as they would arrive at the two ears). This application uses libspatialaudio [76] and is based on a benchmark from ILLIXR XR [33]. Ambisonic in EarBench consists of three constituent implementations, Ambisonic-{2,6,8}, differing in the number of channels in the input encoding.

Head Related Transfer Function is a 3D audio application that aims to synthesize sounds that, to the user, would appear realistic. This requires two steps. First, the gaming environment's echoes need to be embedded into the sound, hence the surrounding's impulse response is convolved with the sound source. Second, the user's head, face, and ears also distort the signal before it enters the ear. Hence, these distortions — together called the HRTF — is also convolved with the output of the first step. The net result is a left/right spatial audio stream that is played back to the user. Of course, the synthesized sound must be updated as the sound source moves in the environment. HRTF implementation in EarBench is taken from the *rg3d* video game engine [54].

2.3 Analytics

DeepECGCNN [43] is a convolutional neural network that classifies ECG data (using 12 hidden layers - 8 convolution for feature extraction and 4 fully connected layers for final classification - and 140M trainable parameters). The network accepts a time series ECG signal of 30 seconds at a sampling interval of 0.003 second, and then classifies it into one of the following 4 classes — normal sinus rhythm, arrhythmic, other rhythm, and very noisy. Since in-ear ECG sensing has been shown to be effective [23], we use it as a representative biosensing-based earable application.

ALBERT [45] is a lite version of BERT [25] used for natural language processing (NLP) tasks — in this case, question answering. EarBench uses a TensorFlow Lite model trained on the Stanford Question Answering Dataset (SQuAD) [67]. Although NLP workloads are typically executed on the cloud, we include ALBERT as a forward-looking application, particularly with the growing concerns around speech privacy, and increasing demand for offline operation (when the earable is not connected to the Internet).

2.4 Spatial Awareness

Head Motion Tracking (HMT) uses an inertial-measurement unit (IMU) consisting of a gyroscope, an accelerometer, and a magnetometer to track the angular position of the user's head [46]. In HMT-1, the IMU provides samples at 100 Hz, and the HMT application updates its estimate of angular position at 10 Hz. In HMT-10, the IMU sampling rate is 1000 Hz, while the estimates are still performed at 10 Hz. HMT allows a user to control the earable (or other devices connected to the earable) via head gestures.

TLIO-ResNet or 1D-ResNet18 is used in Tight Learned Inertial Odometry [49] for localization and navigation with six IMUs sampled at 200 Hz. TLIO-ResNet uses 1D convolution layers. The network outputs estimates of displacement and uncertainty.

Extended Kalman Filtering (EKF) estimates the hidden state of a dynamic system given a discrete time series of noisy observations. The EKF extends the Kalman filter to cases where the state-transition model and observation model are differentiable functions of the hidden state, rather than simply linear. Noise is assumed to be Gaussian while the hidden process upholds the Markov property. The EKF application in EarBench estimates the orientation of a device from the IMU data.

Table 1 shows the inputs, outputs details, and timing requirements of various EarBench applications. For these inputs, Table 3 shows the dynamic instruction count (in millions), resident and working set sizes (in MB), dynamic instruction composition (computation, conditional and unconditional branch, and memory accesses in percentage), and ratio of floating point adds to multiplication instructions[2] for the different EarBench applications (when profiled using armv8-QEMU [9]). We notice that the applications vary widely in complexity: DeepSpeech executes 250× more instructions

[2]Wave2Letter code was integer-based.

Table 2: EarBench: Execution characteristics. 'RSS' and 'WSS' are resident and working set sizes in MB. 'Comp.', 'Cond.', 'Uncond.', and 'Mem.' columns correspond to the percent of dynamic instructions categorized as computation, conditional branching, unconditional branching, and memory accessing, respectively. 'Inst. Count' is the dynamic instruction count (in millions). 'FPA2M' is the ratio of floating point arithmetic to bytes of memory accessed (Wav2letter uses integer arithmetic only).

	RSS	WSS	Inst. Count	Comp.	Cond.	Uncond.	Mem.	FPA2M
TinySR	1.56	0.07	313	36.5	4.4	2.2	56.9	0.518
HMT	1.77	< 0.01	14	49.2	4.4	3.4	42.7	0.878
Ambisonic	6.82	5.42	2,795	51.9	7.2	1.4	39.5	1.27
SpeakerAuth	6.40	3.62	1,520	26.6	15.4	1.5	56.4	1.07
HRTF	8.95	3.97	2,712	59.4	12.6	1.3	26.7	1.49
DeepECGCNN	1084	1071	3,295	72.5	14.4	0.01	13.1	1.00
DeepSpeech	226	195	3,486	30.6	3.3	16.8	49.3	1.07
TLIO	39.8	0.5	87	69.0	17.4	0.37	13.1	1.00
Wav2Letter	29.7	28.1	1,077	79.4	2.01	0.06	18.4	(N/A)
EKF	8.9	0.63	1,351	44.9	14.0	4.76	36.2	0.760
ALBERT	773	767	15,213	73.7	1.77	0.18	24.3	1.00

than HMT, and working set sizes range from under 10 kB to over 1 GB. The applications are characterized by large amounts of computation and data movement and small amounts of control flow (only DeepSpeech has over 20% of its instructions as branches).

The ratio of computation to memory accesses varies. In the convolutional neural networks (TLIO, DeepECGCNN, Wav2Letter), we see high ratios of compute to memory instructions due to the data reuse available in convolution layers. We see considerably more data movement in DeepSpeech, which prominently features dense layers, as data reuse is more difficult to achieve on dense layers than on convolution layers in a register-limited ISA such as ARMv8-A. The computation itself is dominated by adds and multiplies. On neural network benchmarks, we see similar numbers of floating point addition and multiplication instructions (nearly identical on DeepSpeech, DeepECGCNN, and TLIO). Since these workloads feature large amounts of linear algebra (i.e., computing inner products of vectors), it is understandable that the ratio of additions to multiplications is near one.

3 THE EARABLE APPLICATION-HARDWARE PERFORMANCE GAP

Today's earables, including programmable ones, do not run most applications in EarBench (e.g., large-vocabulary speech recognition, natural language processing, tight localization, etc.) and support limited version of other EarBench applications (in form of wake word detection, speaker identification, etc.). To understand how well today's programmable earable processors meet the needs of future earable applications, we simulated the execution of EarBench applications on processors modeled after ARM Cortex-M4@180 MHz and Cortex-M7@480 MHz (methodology details in Section 5). We chose M4 and M7 since they are already found in several of today's programmable earable SoCs (e.g., CYW20721,SAM-S70S, etc.) either as standalone processors or in conjunction with DSPs. We refer to the cores modeled after M4 and M7 as s_180 and ss_480. respectively. We also run the applications on ss_1000, a core modeled after A53@1 GHz, even though it is much more complex than the MCUs on today's earable SoCs and has unacceptably high power consumption (276 mW to 339 mW) for earable batteries (e.g., 45.4 mA h battery for AirPods Pro). We measure the runtime of each application on the three modeled cores, and then compare the runtime with the application's timing requirement (listed in Table 1).

Figure 2 shows the amount of speed-up required on each platform to meet each application's timing requirement. We notice that s_180 is not able to meet the performance requirements of *any* of our applications. On average, s_180 requires 13.5× speed-up across EarBench applications. ss_480 running at 2.67× higher frequency than s_180 is not able to meet the performance requirements of many EarBench applications. On average, ss_480 requires 4× speed-up across our audio benchmarks. Even ss_1000 is unable to meet the performance requirements of several of our EarBench applications, in spite of being much more complex and power-hungry. On average, ss_1000 requires 1.9× speed-up across EarBench benchmarks.

For the small number of cases where performance requirements are met (mostly by the complex ss_1000), functionality is severely restricted due to energy constraints. Figure 3 shows the number of times an application whose performance requirements were met will run before a 45.4 mA h, 3.7 V battery — similar to one in AirPods Pro earbuds — runs out. Value for a (core, application) pair is empty if application performance requirements were not met by the core for that application. Figure 3 shows that Ambisonic-2 can run 198 times on ss_1000. Considering that our Ambisonic-2 benchmark takes a 10s audio as input (Table 1), this corresponds to only 0.55 hours of audio decoding. Similarly, 201 runs of DeepECGCNN will correspond to ≈ 0.5 hours of ECG data analysis in a continuous usage mode. Therefore, simply using complex microprocessors (e.g., ss_1000 or higher) to support programmable earable computing may be unacceptable.

3.1 Key Computation Kernels

We analyze the EarBench suite to identify the key computation kernels in earable applications. Figure 4 shows the performance breakdown of the EarBench applications across constituent kernels on a single Cortex A72 (profiling was performed on Raspberry Pi 4). Applications were compiled with gcc 9.2 and rustc 1.5 with 'O3' optimizations (which include SIMD auto-vectorization).

Figure 4 shows that EarBench applications are built out of a small number of computationally intensive DSP and ML kernels. At first glance, this commonality of kernels in applications across a wide range of use-cases (e.g., audio playback, text processing, heart-rate monitoring, speech processing, etc.) may seem surprising. However, note that these applications are implemented broadly using techniques from digital signal processing and machine learning; DSP and ML computation are well-known to both be dominated by a small number of computation kernels and share computation kernels between them [56].

3.2 Computational similarity between DSP and ML Kernels

Machine learning and DSP kernels have similar computation. A neural network consists of an alternating sequence of linear transformations and non-linear activation functions. Computing the action of a linear transformation, W, on an input vector, a, is given

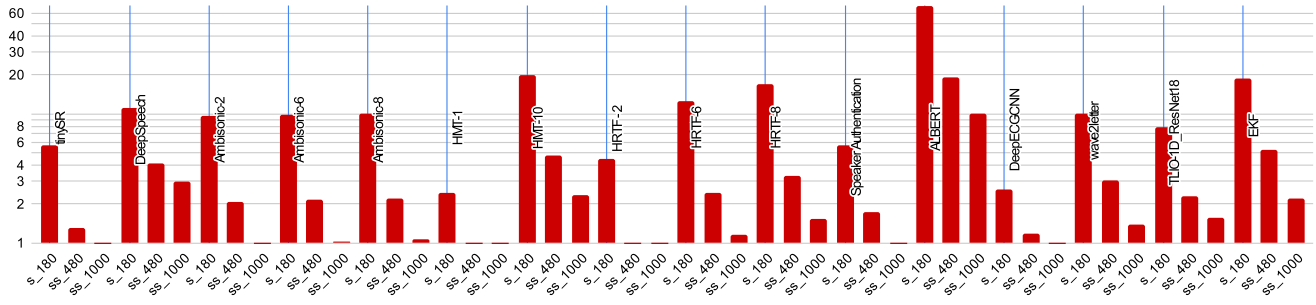


Figure 2: Required Speed-up for different modeled microprocessors across EarBench applications.

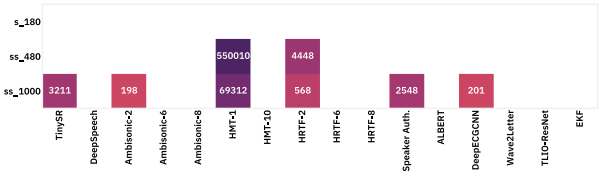


Figure 3: Number of runs possible on a 45.4 mA h, 3.7 V **battery**

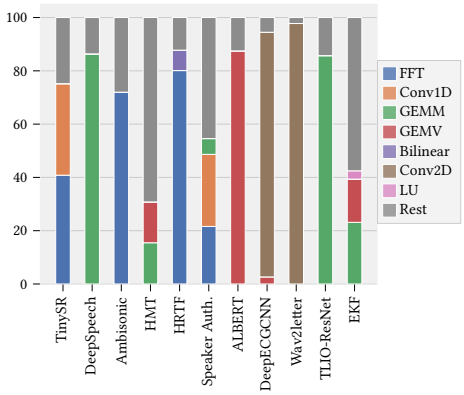


Figure 4: Constituent kernels of EarBench applications.

by $b = Wa$. Given a choice of basis, W and a can be expressed in matrix form, and computing the i'th output of b is

$$b_i = \sum_{n=0}^{N-1} a_j W_{ij}. \tag{1}$$

Similarly, computing the N-point discrete Fourier transform (DFT) of a real or complex vector, x, is given by the linear transformation $X = Wx$, and $W_{nk} = e^{-i2\pi/N}$ is an element of the order N subgroup of the unit circle group. Thus, the 'frequency-domain' output

$$X[k] = \sum_{n=0}^{N-1} x[n] W_{nk} \tag{2}$$

is remarkably similar to the core computation of a neural network. Two factors complicate using a substrate optimized for ML kernels to be used for DSP kernels and vice versa. First, neural networks typically are real-valued, while DFT is complex valued. Second, symmetries in the DFT's transformation enable a 'fast' $O(n \log n)$ Fourier transform by recursively subdividing the N-point DFT into smaller and smaller sub-blocks. Similarly, neural networks often exploit symmetries of their own to greatly reduce required computation and data movement by using convolution layers. Despite these complications, striking similarities remain. Both DFT and neural network computations are made up of real-valued multiplications followed by additions/subtractions, and in both the number of multiplications needed is nearly identical to the number of additions/subtractions needed. This suggests that an efficient processor architecture for earable applications is a neural network accelerator that is augmented with support for complex processing and that can be reconfigured to support different dataflows for the different ML and DSP kernels.

Furthermore, since the dataflows of dot-product and complex multiplication, the key subcomputations of neural networks and FFT, respectively, consist of scattering data streams to multiplication units, followed by reduction (dot-product) and gathering into an output stream (complex multiplication), an architecture which enables scatter-multiply-reduce and scatter-multiply-gather is needed (Figure 5). High bandwidth tree-structures for both the scatter network, and for the reduction/gather network should enable efficient computation of machine learning and DSP workloads.

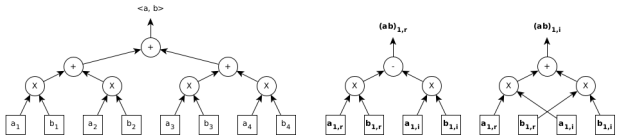


Figure 5: Dataflows for dot-product (left) and complex multiplication (right).

4 SPEAC: A SPATIAL ARCHITECTURE FOR EARABLE COMPUTING

We propose *SpEaC*[3] - a stream-programmed [57] coarse-grained reconfigurable spatial architecture (Figure 6) as an energy-efficient programmable processor for earable applications. At high level,

[3] Spatial Earable Computer

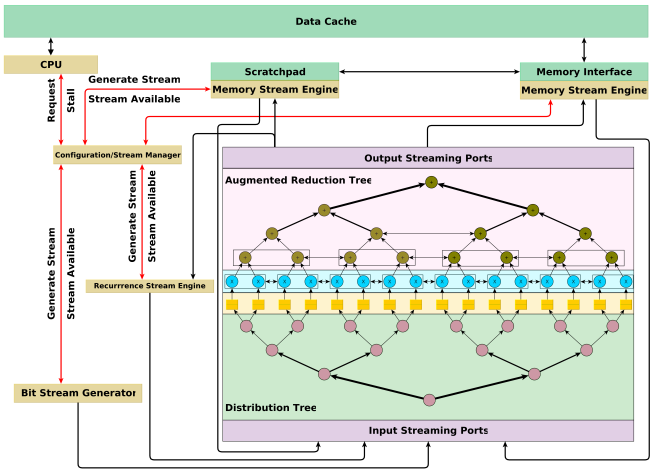
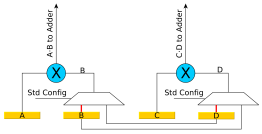
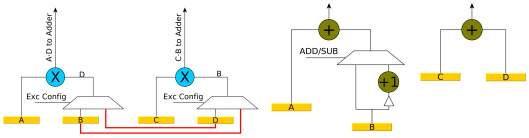


Figure 6: Architecture of SpEaC.



(a) LVM Standard Configuration



(b) LVM Exchanged Configuration

(c) LVA

Figure 7: Logical Vector Multiplier (LVM) and Logical Vector Adder (LVA) support complex arithmetic in SpEaC.

SpEaC consists of two parts - a reconfigurable computation substrate that is responsible for executing kernels and a tightly coupled control core that is responsible for configuring the substrate, generating data streams, and executing kernel operations not directly supported by the substrate. Below we describe the substrate and the control core one by one.

Since earable applications are dominated by a small number of DSP and ML kernels which in turn are dominated by inner-product-based computation, SpEaC computation substrate consists of a set of 15 adders (8 of which are organized in pairs - explained later) and 16 multipliers (organized as 8 pairs - explained later). Table 2 suggests an equal distribution of multipliers and adders; the actual number of units was chosen such that SpEaC power remains around 60 mW. A fat distribution tree [47] feeds SpEaC's 16 multiplier units. The products of the multiplier units are forwarded to 15 adders organized in an 'augmented reduction tree', which allows a single tree to perform multiple inner products concurrently, or to be time-multiplexed to perform a large inner-product across multiple cycles. The final three adder units are equipped with accumulators to enable computation of multiple inner-products in parallel. SpEaC's accumulators may be configured to output either the accumulated value, x, or $\max\{0, x\}$, which means SpEaC supports either linear or reLU activation functions. SpEaC supports max-pooling layers through the reduction tree by allowing the adder units to also perform comparison operations.

SpEaC's functional units support fixed-point arithmetic since fixed-point units are significantly lower power than floating-point units, and since the EarBench DSP and ML kernels are largely resilient to fixed-point and integer quantization in context of audio and machine learning workloads, [11, 19, 30, 31, 48, 80]. SpEaC uses 32-bit fixed-point arithmetic since high quality audio often has a 24-bit depth, while 16-bit fixed-point arithmetic would lower audio quality.

Furthermore, since FFT is a key computational kernel in EarBench, SpEaC functional units support vectorized complex operations. To enable multiplication of complex numbers, two adjacent multipliers form a pair to create a *logical vector multiplier* (LVM). Each LVM can be configured to either perform a 2x32 bit multiplication or a 2x32 bit multiplication in which one of the 64-bit inputs' top and bottom 32-bits are swapped (i.e., exchanged multiplication).

Since the product of two complex numbers may be decomposed into four real-valued partial products of the complex numbers' components, SpEaC uses two LVMs (one in 'Standard' and the other in 'Exchanged' configurations) to generate the partial products. Figure 7 shows the possible configurations for a LVM. In 7a, the LVM is configured to produce two standard or scalar multiplications. In 7b, the LVM is configured to produce products in which the multiplicands have been exchanged.

Also, since $i^2 = -1$, the real component of the complex product is produced by computing the difference of two of the partial products while the imaginary component is produced by computing the sum of the other two partial products. To support this, adjacent adders in level 3 of the tree form *logical vector adders* (LVA). The 'left' adder in a LVA can be configured to perform subtraction as well as addition. Figure 7c shows that a LVA can be configured to either perform parallel 32-bit adds, or a 32-bit subtract and a 32-bit add.

Since complex addition *does not* require multiple or out-of-sequence accesses to a complex number's real and imaginary components, no additional hardware is required to support complex addition (in fact, complex addition may use the same dataflow as real addition).

The control core in SpEaC is a tightly-coupled scalar, in-order CPU responsible for configuration of the SpEaC computation substrate and stream-based programming. It also executes code that does not fit well on the reconfigurable substrate, including non-multiply or add operations in the DSP kernel code (e.g., square-root and divide operations in LU and Cholesky).

The core orders the Configuration/Stream Manager to configure the computational substrate, and to generate data streams. Data streams belong to one of three classes:

- *Streams with 3D affine access patterns from memory*: SpEaC supports 3D affine addressing, meaning that a single stream can represent array access in a depth-3 nested loop.
- *Recurrence streams [57]*, which pipe streaming outputs into inputs;

- *Bitstream*, which is used to provide some amount of dynamic control to the configured substrate - specifically whether the accumulators accumulate or forward to an output stream.

SpEaC supports up to four concurrent data streams and one bitstream.

4.1 SpEaC vs Other CGRAs

SpEaC's architecture most closely resembles the SoftBrain CGRA [57] in its architecture. However, there are three significant differences. First, SoftBrain uses general purpose functional units (since it is aimed at supporting arbitrary workloads). SpEaC replaces SoftBrain's fixed-point general purpose functional units with fixed-point adders and multipliers - common across our DSP and ML-based kernels. As a result, SpEaC's functional units consume 4.42x lower power, on average, in 28 nm technology. Second, SpEaC provides hardware support for complex arithmetic through support for 2-element vectorized operations (2×32 multiplication, 2×32 exchanged multiplication, 2×32 addition, 2×32 sub-add). SoftBrain does not directly support complex arithmetic and, therefore, kernels such as FFT cannot be mapped efficiently. Third, SoftBrain uses a general purpose mesh network to support arbitrary workloads. SpEaC replaces SoftBrain's general-purpose network with a distribute-then-reduce tree due to the specific multiply-add and multiply-accumulate patterns found in earable workloads. This provide a 4× savings in CGRA network power at 28 nm. Overall, SpEaC consumes less than two-thirds the power of SoftBrain at 28 nm.
Other related work includes MAERI [42] and SNAFU [29]. Similar to SpEaC, MAERI substrate is based on multiply-and-add trees. However, its system architecture and programming environment are focused on ML kernels. It is unclear how to map earable DSP kernels such as LU, Bilinear, Cholesky, and FFT directly on MAERI. There is no programming support for non-matrix computation (Bilinear), no hardware support on the substrate for complex arithmetic (FFT), and no control core for running non-multiply and add operations either (Cholesky and LU). SNAFU integrates a control core with a CGRA and should therefore support the earable kernels. However, SNAFU supports a *general purpose* CGRA substrate, not one optimized for linear algebra workloads, found commonly in machine learning and DSP applications. Unlike SpEaC, which targets linear algebraic workloads, SNAFU's general-purpose substrate has a high ratio of multiplier-less ALU FUs to multiplier FUs. This means its performance for linear algebraic workloads is multiplier limited. As a point of comparison, [29] reports SNAFU's performance on dense matrix multiply with 64×64 matrices. SpEaC outperforms SNAFU on this kernel by 72×. While SpEaC's 1 GHz clock rate accounts for 20× speed-up, the additional 3.6× speed-up is a result of the greater parallelism achievable on SpEaC, due to its larger number of multipliers (16 vs 4), and SpEaC's nearly 1-1 ratio of adders to multipliers (while SNAFU has only one multiplier for every three adders). Similarly, while SNAFU's grid-based NoC topology is useful for general purpose programability (since it can support arbitrary dataflows), it consumes more power than a tree-based NoC (for SpEaC, we estimated that the tree-based NoC is ∼ 4× lower power than the grid-based NoC). Previous work [42] has also observed that grid-based NoCs scale poorly in power and area relative to a tree-based NoC for the bandwidth values SpEaC targets. A direct comparison of SNAFU and SpEaC would nevertheless be interesting once SNAFU toolchain is made public.

Table 3: Hardware specifications of modeled processors.

Platform	Hardware Description
s_180	M4 based, 32-bit, 180 MHz, Power min (typical) = 0.96 mW, Power max (typical) = 1.18 mW, Scalar Processor, Single cycle 16/32-bit MAC, Single cycle dual 16-bit MAC, 8/16-bit SIMD arithmetic, 1V, 28 nm, Hardware Divide (2-12 Cycles), gem5 simulator
ss_480	M7 based, 32-bit, 480 MHz, Power min (typical) = 17 mW, Power max (typical) = 20.7 mW, 6-stage superscalar + branch prediction, Single cycle 16/32-bit MAC, Single cycle dual 16-bit MAC, 8/16-bit SIMD, 1V, 28 nm, arithmetic, Hardware Divide, gem5 simulator, 32 kB L1D cache, 32 kB L1I cache
ss_1000	A53 based, 64-bit, 1GHz, Power min (typical) = 276 mW, Power max (typical) = 339 mW, Superscalar Processor, gem5 simulator, L1D cache size = 32 kB, L1I cache size = 32 kB, 28 nm
hifi4_dsp	HiFi4 DSP core, 32-bit, 600 MHz, Power (typical) = 234 mW, 4 VLIW slots 4 32 × 32 fixed point MACs/cycle, MIMXRT685-EVK board, 1.13 V, 28 nm
SpEaC	CGRA based 32-bit architecture, 1 GHz, Power (typical) = 63 mW, Spatial Architecture, gem5 simulator with spatial architecture simulation support, 10.1.0, DSA CGRA generator

4.2 Programming SpEaC

As with other stream-programmed CGRAs [57], SpEaC is programmed by first generating a dataflow graph, scheduling the dataflow graph onto the compute substrate, and then using custom ISA instructions of the control core to generate control and stream instructions for the computational substrate and the stream generators, as well as an instruction to block the control core until all in-flight write streams are complete.

Dot-product computation maps naturally onto the architecture's reduction-tree, with the reduction tree supporting the multiplication and reduction of up to sixteen partial sums in parallel.

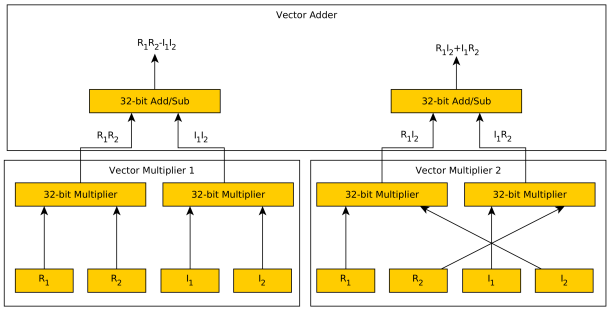


Figure 8: CGRA configured to perform complex multiplication. The distribution network sends two complex numbers, $c_1 = R_1 + iI_1$ and $c_2 = R_2 + iI_2$ from two streams to two LVMs. LVM 1 is configured to produce the two real terms of the product $c_1 \cdot c_2$, while the LVM 2 is configured to produce the two imaginary terms. The real terms are then sent to an adder configured to compute the difference $R_1R_2 - I_1I_2$ (since $i^2 = -1$), while the imaginary terms are sent to an adder configured to compute the sum $R_1I_2 + I_1R_2$.

As discussed earlier, SpEaC substrate which uses vectorized operations to support the FFT's complex arithmetic. Figure 8 shows the configuration of LVMs and LVAs necessary to perform complex multiplication.

5 METHODOLOGY

We used Gem5 [13] to evaluate the performance of the EarBench applications (Table 1) and the earable computational kernels. We use a simulator instead of a real development platform since several

of our EarBench applications have much larger memory requirement (Table 2) than what is available on these platforms (and since the platforms do not support the non-RTOS operating system abstractions required by several of our Linux EarBench applications). We compiled the kernels and applications to the ARMv8-A ISA, even though Cortex M4 and M7 feature the ARMv7-M ISA, since Gem5 supports ARMv8-A rather than ARMv7-M.

We compiled EarBench applications on a Raspberry Pi 4 (Cortex-A72) using gcc 9.2.0. We compiled the HRTF benchmark, written in Rust, with rustc 1.50.0. We used 'O3' compiler flag and enabled NEON SIMD auto-vectorization.[4] We used hand-optimized versions of the earable kernels for our evaluations. Kernels were hand-optimized for the MCUs and the parameters of the data structures and operations were defined inside the program itself, allowing the compiler to perform several optimistic optimizations. Our hand-optimized implementations consistently beat the off-the-shelf kernel implementations from the muFFT [8] and LAPACK [3] libraries for our inputs, setting up a conservative comparison of MCUs against SpEaC.

In addition, we generate measured kernel-level results (using CCOUNT register) for a state-of-art Tensilica HiFi4 DSP running at 600MHz on a MIMXRT685-EVK board. We used DSP kernels from the performance-optimized NatureDSP library [34]. Three kernels (LU, bilinear, and Cholesky) that were not present in the NatureDSP library were hand-written and optimized.

For power modeling, we use Gem5's IPC-driven ARM power modeling methodology [13]. Datasheets for the M4 [6], M7 [7], A53 [5], and HiFi4 [17] cores are used for calibration such that the mean and maximum power numbers reported by our models for our applications match the corresponding typical power consumption numbers in the datasheets. Similar methodology has been used in many previous works [40].

We used the DSA-Framework [77] to evaluate SpEaC, including its power and area. We specify the CGRA substrate using a domain-specific language built on top of JSON. DSA-Framework then schedules given DFGs onto the specified CGRA. The framework enables programming of the CGRA via a RISC-V (RV64-MC) control core with a customizable ISA extension for stream-based spatial accelerators. Benchmark kernels are written in C++ with in-line assembly for the extension instructions, and compiled with gcc 10.1.0. We then simulate SpEaC with a cycle-accurate model in Gem5 running at 1 GHz.

6 RESULTS AND ANALYSIS

Figure 9 presents speed-up results for SpEaC against the modeled cores. Across the computation kernels, SpEaC is 5.1 to 195× faster than s_180, 1.2 to 35× faster than ss_1000, and 3.8 to 12.5× faster than the HiFi4 DSP. SpEaC outperforms ss_1000 by an average of 11× and outperforms the HiFi4 DSP by an average of 6.57×.

Table 4 shows the power and area of SpEaC at 28 nm technology. Power and area for the data cache are generated with cacti [55]. The remainder of the numbers are reported by DSAGen Framework and then scaled to 28 nm.

In spite of running at 1 GHz, SpEaC consumes only 64 mW. For comparison, we note that ss_1000 - the only other core running at 1 GHz - and HiFi4 consume 276 mW to 339 mW and 234 mW respectively. So, SpEaC consumes much less power than these cores while providing much higher performance.

[4]Full details of ARMv8-A build will be published with the EarBench suite.

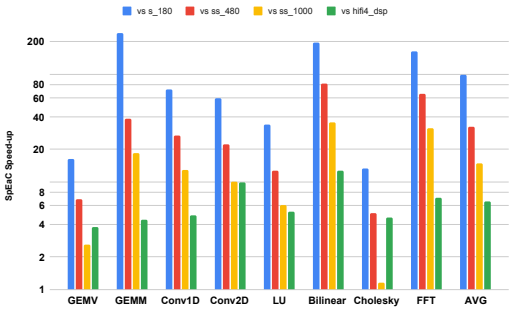


Figure 9: Speed-up of SpEaC over modeled cores.

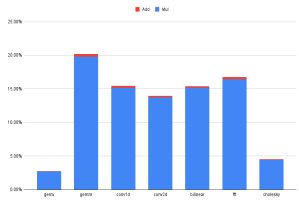


Figure 10: Percentage of total energy consumed by arithmetic functions across kernels.

Table 4: Power and area of SpEaC in 28 nm technology. The CPU core is a scalar, in-order RISC-V core with 32 KiB of cache.

Component	Area (mm^2)	Power (mW)
Substrate	0.04	20.05
Network	0.004	1.2
FU Buffers	0.02	13.6
Stream Engines	0.006	8.38
CPU Core	0.04	7.5
Data Cache	0.06	10.5
Scratchpad	0.039	2.6
Total	0.209	63.8

Figure 11 shows the energy benefit of SpEaC relative to the baseline cores. The benefits are up to 160× compared against the microprocessors, and up to over 42× compared to the DSP.

Figure 10 shows the percentage of total energy consumption consumed by adders and multipliers for the kernels. GEMV, with its low data reuse, has the lowest FU utilization and, correspondingly, has the lowest portion of energy consumed by arithmetic. Similarly Cholesky and LU decomposition, which rely on the scalar control core for portions of their computation, also have low portions of energy consumed by arithmetic. The high overhead of control and data movement reinforce the importance of minimizing the CGRA network costs (using a tree, for example), the use of specialized FUs (using only fixed point adders and multipliers on the substrate, for example), and the associated memory system.

Using kernel speed-up values from simulation, and with the kernel-breakdown of Figure 4, we estimate the application level

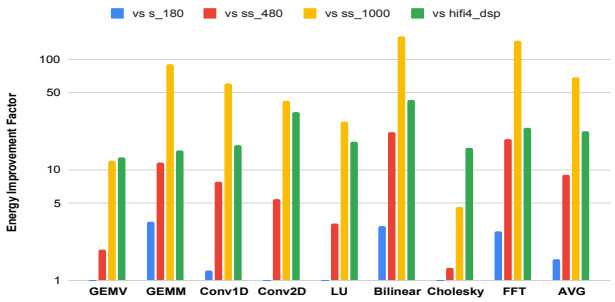


Figure 11: Energy improvement from SpEaC over modeled cores.

Table 5: Latency and power of different non-compute components of an earable system.

	latency	Power
Microphone [35]	~30 µs to 50 µs	~1.17 mW
Speaker [26]	~50 µs	10 mW
IMU [50]	~1 ms to 10 ms	~3.96 mW
BLE TX/RX [71]	8 µs B^{-1}	84 mW / 66 mW

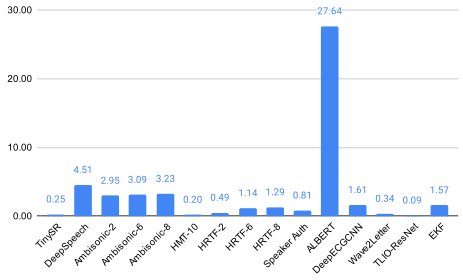


Figure 12: Runtime (in seconds) of different EarBench applications on SpEaC.

speed-up and energy benefits of SpEaC. Compared against ss_1000, significant application level speed-ups are significant: 3.3× for DeepSpeech and Ambisonic, 6.6× to 9.6× for HRTF audio, 3.6× for the ResNet, 2.2× for ALBERT, 6.4× for DeepECGCNN, and 8.3× for WaveToLetter. In fact, these speed-ups are achieved despite ss_1000 consuming 2× to 2.5× more power than SpEaC. These speed-ups are large enough to allow SpEaC to meet the application-level requirements of all applications, except ALBERT and EKF (Figure 2). Recall that s_180, ss_480, and ss_1000 met requirements for 0, 2, and 6 applications respectively (Figure 2).

Figure 12 shows the application level latency (in seconds) of different EarBench applications on SpEaC.

To understand the system-level impact of SpEaC, we also consider non-compute components. Table 5 lists power and latency of several non-compute components. The latency overhead of the non-compute components is much smaller than the estimated latency of Earbench applications on SpEaC (Figure 12). We estimate that, compared to s_180, ss_480 and ss_1000, the speedup of SpEaC

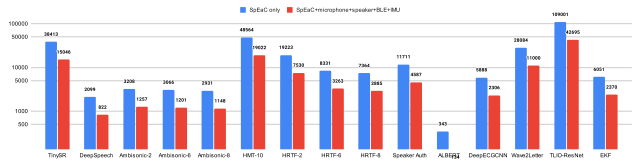


Figure 13: Number of runs on SpEaC on a 45.4 mA h, 3.7 V battery

on-average across all EarBench applications is only reduced by 0.67%, 0.51% and 0.33% respectively, when the latency of *all* the non-compute components in Table 5 is added. We similarly estimated (Figure 13) the number of times different EarBench applications whose performance requirements are met by the SpEaC can be run on the SpEaC given the energy budget of a 45.4 mA h, 3.7 V battery. Our results show that even when microphone, speaker, BLE and IMU sensor are simultaneously consuming battery energy, the EarBench applications can still run hundreds of times. E.g. DeepSpeech on 5s audio can run 849 times before battery needs recharging; this corresponds to large-vocabulary speech recognition for 1.14 hours.

Finally, to validate our estimates, we ported TinySR to SpEaC, implementing the FFT component on the CGRA. Based on TinySR's estimates, we had expected to see a 2.7× speed-up on SpEaC. Simulation of the SpEaC port of TinySR showed a 2.5× speed-up, a −7% error versus our analytic estimate. A 2.5× performance improvement on TinySR also corresponds to a 5.8× improvement in energy efficiency, and a 14.4× improvement in EDP.

6.1 SpEaC vs Low-power GPUs and ML Accelerators

We evaluated a subset of earable kernels on four shader core group of a Mali T628 MP6 GPU (Table 6) on a 28 nm Exynos 5422 SoC programmed in OpenCL using the T628-tuned CLBlast [58] library for BLAS functions, and vexcl [24] for FFT implementation. [5] Kernel level results for T628 versus SpEaC are shown in Figure 14 for inputs scaled by different factors up to 32 or until GPU memory runs out (T628's high kernel launch cost is amortized for larger inputs).

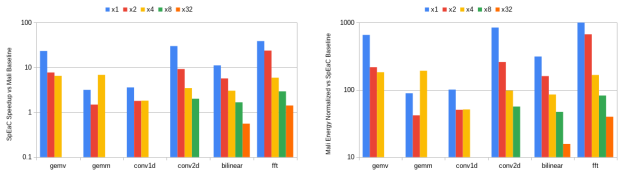
We find that SpEaC and T628 performance values are, in general, within an order of magnitude of each other for all but the smallest inputs. We attribute SpEaC's strong performance relative to the T628 to 1) SpEaC's tight integration with the control core (mobile GPU kernel launch overhead has been shown to contribute significantly to machine learning inference latencies [39], and indeed, T628 performance relative to SpEaC improves with kernel size), 2) its higher clock frequency (1 GHz vs 600 MHz), and 3) its decoupled access/execute architecture which uses streaming engines to overlap address calculations with useful computation on the SpEaC substrate (while T628 uses the same ALUs to perform address calculations and useful computation). At the same time, T628 power is more than an order of magnitude higher (30× for an estimated $P_{\text{typical}} = 1.8$ W) due to use of floating point arithmetic (vs fixed

[5] We do not include Cholesky or LU since they require a significant number of memory barriers and operate on small dimensional matrices in our applications making them un-competitive on T628.

Table 6: Hardware configuration of the modeled accelerators.

Processor	Hardware configuration
Mali T628	4 Shader cores @ 600 MHz with 2 ALUs. SIMD f32×4 Add, SIMD f32×4 Mul, $\mathbb{R}^4$ inner product, scalar f32 Add, scalar f32 Mul per ALU. Rasterizer, triangle set-up unit, Z-buffer per core.
Eyeriss	16 PEs @ 1 GHz, one MAC per PE, 16-bit MACs, 0.5KB SRAM per PE, 108KB global buffer
MAERI	16 Multiplier Switches (PEs) @1 GHz, Distribution Bandwidth = 4 elements, Reduction Bandwidth = 1 elements

point adders and multipliers in SpEaC), general-purpose architecture (SpEaC uses a tree of adders and multipliers customized to our kernels), and presence of dedicated graphics pipelines (that add static power). This leads to more than an order-of-magnitude better energy-efficiency for SpEaC over T628 for the evaluated kernels for all kernel/size pairs.



(a) Performance **(b) Energy**

Figure 14: Performance and energy of a 4 shader-core T628 GPU for different input sizes vs SpEaC.

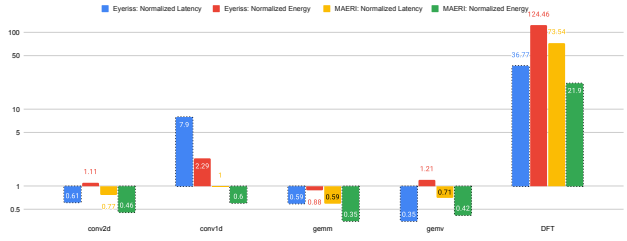


Figure 15: ML accelerators vs SpEaC.

ML accelerators are a poor standalone fit for earable computing since they are fundamentally limited in their generality; it is unclear how to map earable DSP kernels such as LU, Bilinear, Cholesky, and FFT directly on ML accelerators such as Eyeriss [21] or MAERI [42] due to their lack of programming or hardware support for non-matrix computation, complex arithmetic, or non-multiply or add operations. For other kernels, we nevertheless compared SpEaC against scaled down versions of Eyeriss and MAERI using NN_dataflow and MAERI-mapper [41] respectively. For energy evaluations of scaled down MAERI, we used MAERI RTL generator [41] to generate synthesizable RTL which we then synthesized using Synopsys design-compiler. The hardware parameters of the modeled architectures are listed in Table 6.

Figure 15 shows the latency and energy of the ML kernels on scaled-down Eyeriss and MAERI, normalized with respect to SpEaC. Our results show that Eyeriss and MAERI outperform SpEaC on most ML kernels. This is because SpEaC has lower amount of on-chip memory (e.g., 40 KB between data cache and scratchpad in SpEaC vs 80 KB in MAERI) and incurs the overheads of the control core (MAERI and Eyeriss results do not include control core overheads). We also show results for discrete Fourier transform (DFT). Since FFT cannot be easily mapped to Eyeriss or MAERI, we investigated if the choice of an ML accelerator-friendly algorithm - we perform DFT via matrix operations - can give competitive performance. However, SpEaC significantly outperforms the ML accelerators on DFT. Our results show that, compared to DFT on Eyeriss, the equivalent FFT on SpEaC is 36.77× faster and 124× more energy efficient. Compared to DFT on MAERI, the FFT on SpEaC is 73.54× faster and 21.9× more energy efficient.

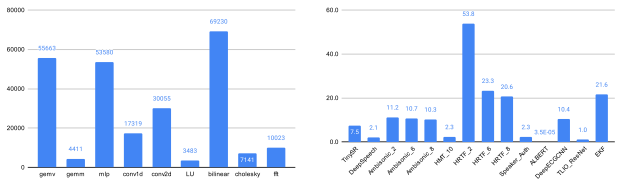
Overall, the results show that GPUs are far too energy-inefficient for earable kernels (while also being unable to directly support a subset of them). ML accelerators, on the other hand, cannot support several earable kernels. SpEaC provides energy-efficient execution for all earable kernels.

6.2 SpEaC vs a collection of accelerators

One possibility is to support multiple fixed function accelerators on the earable SoC each targeting a different set of kernels. For example, an SoC with a CPU, an ML accelerator, and an FFT accelerator looks attractive for our applications. However, there are several disadvantages of such an approach. One, the field of earable computing is relatively new and algorithms are still evolving. Any system with fixed functions ASICs may become obsolete quickly. SpEaC, on the other hand, can be configured easily to any new algorithm whose dataflow graph is dominated by adds and multiplies. Second, a collection of special purpose accelerators will fail to provide the requisite performance for kernels that do not map well to an existing accelerator. For example, LU, Bilinear, and Cholesky see significant speedups on SpEaC. However, they cannot be mapped directly on an ML accelerator or an FFT accelerator and will need to be run on the CPU at high performance cost (Figure 9). Similarly, a given kernel may not fit its fixed function accelerator well depending on the input. For example, FFT size in tinySR, depends on input audio length and, for our microphone input, varies from 32-point to 1024-point, HRTF calls a 4096-point FFT, Ambisonic uses a 512 or 1024-point FFT, etc. Many FFT accelerators are optimized for a specific size [79] and will not run other FFTs of other sizes efficiently [1]. SpEaC, on the other hand, can efficiently support arbitrarily-sized FFTs. Third, SpEaC may even have efficiency advantages for our specific kernels over using a separate FFT accelerator. SpEaC can already be thought of as an $ML+FFT$ accelerator. Its design is heavily based on MAERI, an ML accelerator, and its use of 3D-affine streams and logical vector units enables efficient computation of FFT on the same substrate. As a point of comparison, the area and power overheads of the LVM versus a pair of multipliers are 8.8% and 11.6% respectively, which corresponds to a total overhead of 1.7% and 3.6%, respectively. Adding a dedicated FFT accelerator of equal or greater performance would require an additional 16 multipliers, an area overhead of 19.3% without accounting for additional FFT accelerator control, compute, and routing, and power gating logic. Finally, a single design (SpEaC)

that can target many applications may be more cost-effective than a design with many unique ASICs.

6.3 SpEaC vs Offloading



(a) Kernels (b) Applications

Figure 16: Communication energy of BLE transceiver when inputs and outputs are transmitted over BLE for remote compute offloading, normalized with respect to the energy consumed by SpEaC when kernels and applications are computed locally.

A natural question comes up of weather to offload compute to mobile device (using BLE, for example) - this will invovle sending inputs and outputs over the communication protocol - or compute it locally on SpEaC? Figure 16-a shows the communication energy of BLE transceiver normalized with respect to the energy consumed by SpEaC when we run the kernels locally on SpEaC. Our results show that computing kernels locally on SpEaC is on average $\approx 4OOM$ more energy efficient compared to communicating inputs/outputs over BLE. This is not surprising since BLE consumes significant amount of energy ($\approx 0.7\,\mu J\,B^{-1}$ [71]) when sending large matrices, which are common inputs to our kernels, due to its long communication latency. Similarly, figure 16-b shows the communication energy of BLE transceiver normalized with respect to the energy consumed by SpEaC when we run the entire application locally on SpEaC. Our results show that for compute intensive applications, e.g. Albert, it is more energy efficient to offload compute to remote device, whereas for all other applications computing applications locally on SpEaC is more energy efficient than offloading the compute to remote device over BLE. On average computing locally on SpEaC is $\approx 12.7\times$ more energy efficient across all our applications compared to communicating inputs/outputs over BLE. Other than energy benefits, local computation may often be preferred also for improved latency, privacy, and usability (since a remote device may not always be available or accessible).

6.4 Configurable precision SpEaC (cpSpEaC)

The primary reason to use a 32 bit datapath is to support audio applications (since high-quality audio has 24b depth). However, several Earbench applications can be implemented at lower precision without a significant drop in accuracy. [81] shows that an 8-bit implementation of BERT maintain 99% of the FP32 accuracy. ARM ML Zoo [82] provides an 8 bit implementation of wav2letter that is able to achieve 87.7% accuracy on Librispeech dataset. TLIO uses ResNet18 as a backend, which can be at 8-bit precision without significant decrease in accuracy drop [74] [51]. [20] presents an 8 bit implementation of EKF that was able to achieve the required

Table 7: Area and power overhead of supporting multiple datatypes with the configurable-precision (cpSpEaC) variants of SpEaC. The 'tree' variants replicate the functional units, one for each datatype supported and are denoted denoted by $t(x)$ subscript, where x is a list of operand bit-widths supported. As in SpEaC, each tree contains 16 multipliers. The 'multi' variants, denoted by subscript $m(x)$, contain vector functional units capable of supporting $1\times$ 32-bit operation, $2\times$ 16-bit operations, and $4\times$ 8-bit

	Area (mm^2)	Power (mW)
Separate NoC/Buffers		
$cpSpEaC_{t(32,\ 16)}$	0.221	(40.4, 65.3)
$cpSpEaC_{t(32,\ 16,\ 18)}$	0.227	(33.9, 65.4)
Shared NoC/Buffers		
SpEaC (32-bit)	0.209	63.8
$cpSpEaC_{t(32,\ 16)}$	0.220	67.8
$cpSpEaC_{t(32,\ 16,\ 18)}$	0.223	69.0
$cpSpEaC_{m(32,\ 16)}$	0.212	63.8
$cpSpEaC_{m(32,\ 16,\ 8)}$	0.216	63.8

prediction accuracy for the battery management system (BMS) in mobile robots. SpEaC can be modified to support multiple precision datatypes at small overhead. Table 7 shows the area and power overheads of several approaches to multiple precision support. In the 'tree' approach, the substrate's functional units are duplicated for each datawidth. If each tree is supplied its own NoC and buffers, coarse-grained power gating can provided significant power savings when using 16-bit or 8-bit datatypes, at the cost of additional area for the duplicated structures. When the trees share the same NoC and buffers, coarse-grained power gating may be difficult to use, however, the power overhead of the duplicated functional units is less than 10% of SpEaC's overall power. In the 'multi' approach, each functional unit is replaced with an equivalent vector-unit, which can perform 1×32 bit operation, 2×16 bit operations, or 4×8 bit operations. SpEaC's 32 bit adders are replaced with four 8 bit adders, and the carry-outs of the lower-significance adders are masked by control signals, and then fed as the carry-ins to the higher-significance adders. This slow, ripple-carry adder still meets timing at 28 nm. SpEaC's 32 bit multipliers are replaced by sixteen 8 bit multipliers, arranged in a 4×4 grid. The multiplier at position i, j computes the product of the i'th byte of the first operand with the j'th byte of the second operand. These partial products are sufficient to compute the 32 bit, 16 bit, and 8 bit products. SP&R reports negligible difference in power and an area increase of no more than 11.2% between the standard multiplier and the multi-precision vector multiplier.

7 SUMMARY AND CONCLUSION

In this paper, we addressed the question: what kind of programmable hardware would be needed to support earable computing in the future? We proposed EarBench, a suite of representative emerging earable applications. Our analysis of EarBench applications showed that, on average, there is a 13.54× and 3.97× performance gap between the computational needs of EarBench applications

and the performance of the microprocessors that several of today's programmable earable SoCs are based on; more complex microprocessors were shown to have unacceptable energy efficiency for Earable applications. We profiled EarBench applications to identify the commonly occurring computation kernels in earable applications. Based on the characteristics of these kernels, we propose *SpEaC* — a stream-programmed, control core-integrated CGRA optimized for earable computation kernels. SpEaC, on average, outperforms cores modeled after M4, M7, A53, and HiFi4 DSP by 99.3×, 32.5×, 14.8×, and 9.8×, respectively. The energy efficiency benefits are 1.55×, 9.04×, 68.3×, and 32.7×, on average; energy efficiency benefits are 15.7 × −1087× over a low power Mali T628 MP6 GPU.

8 ACKNOWLEDGEMENTS

We thank Prof Romit Roy Choudhury for his compelling vision of earable computing and he and his students for multiple discussions about earable applications. We thank the anonymous reviewers as well as the members of the Passat group for their feedback and the NSF for partial support.

REFERENCES

[1] Berkin Akın, Franz Franchetti, and James C. Hoe. 2014. Understanding the design space of DRAM-optimized hardware FFT accelerators. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, Manhattan, NY, 248–255. https://doi.org/10.1109/ASAP.2014.6868669

[2] Amazon. 2021. *Echo Frames*. Amazon. https://www.amazon.com

[3] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. 1999. *LAPACK Users' guide*. SIAM, Philidelphia, PA. https://www.netlib.org/lapack/lug/

[4] Apple. 2021. *Airpods Max*. Apple. https://www.apple.com/airpods-max/

[5] ARM. 2021. *Cortex-A53*. Arm Ltd. https://developer.arm.com/documentation/ddi0500/j/Cortex-A53

[6] ARM. 2021. *Cortex-M4*. Arm Ltd. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4

[7] ARM. 2021. *Cortex-M7*. Arm Ltd. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7

[8] Hans-Kristian Arntzen. 2021. muFFT. https://github.com/Themaister/muFFT

[9] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX annual technical conference, FREENIX Track*, Vol. 41. Califor-nia, USA, USENIX Association, Bostn, MA, 46. https://www.usenix.org/legacy/event/usenix05/tech/freenix/full_papers/bellard/bellard.pdf

[10] K Swetha Bharati and Ashok Jhunjhunwala. 2015. Implementation of machine learning applications on a fixed-point DSP. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, Manhattan, NY, 1458–1463. https://doi.org/10.1109/CCECE.2015.7129495

[11] K Swetha Bharati and Ashok Jhunjhunwala. 2015. Implementation of machine learning applications on a fixed-point DSP. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, Manhattan, NY, 1458–1463. https://doi.org/10.1109/CCECE.2015.7129495

[12] Thomas Bible. 2016. Binaural Audio for Narrative VR. https://www.oculus.com/story-studio/blog/binaural-audio-for-narrative-vr/

[13] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7. https://dl.acm.org/doi/abs/10.1145/2024716.2024718

[14] Bose. 2021. *Bose Frames Tenor*. Bose. https://www.bose.com/en_us/products/frames/bose-frames-tenor.html

[15] Bose. 2021. *Bose QuietComfort Earbuds*. Bose. https://www.bose.com/en_us/products/headphones/earbuds/quietcomfort-earbuds.html

[16] Bose. 2021. *Bose Sport Earbuds*. Bose. https://www.bose.com/en_us/products/headphones/earbuds/bose-sport-earbuds.html

[17] Cadence Design Systems, Inc 2020. *Tensilica HiFi DSP Family*. Cadence Design Systems, Inc.

[18] Martin Campbell-Kelly, William Aspray, Daniel P Snowman, Susan R McKay, and Wolfgang Christian. 1997. Computer A history of the information machine. *Computers in Physics* 11, 3 (1997), 256–257. https://aip.scitation.org/doi/pdf/10.1063/1.4822551

[19] Wei-Hsin Chang and Truong Q Nguyen. 2008. On the fixed-point accuracy analysis of FFT algorithms. *IEEE Transactions on Signal Processing* 56, 10 (2008), 4673–4682. https://ieeexplore.ieee.org/abstract/document/4626107/

[20] Arezki Abderrahim Chellal, José Lima, José Gonçalves, and Hicham Megnafi. 2021. Battery Management System For Mobile Robots based on an Extended Kalman Filter Approch. In *2021 29th Mediterranean Conference on Control and Automation (MED)*. IEEE, IEEE, Manhattan, NY, 1131–1136. https://ieeexplore.ieee.org/abstract/document/9480196/

[21] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2017), 127–138. https://ieeexplore.ieee.org/abstract/document/7738524

[22] Ronan Collobert, Christian Puhrsch, and Gabriel Synnaeve. 2016. Wav2Letter: an End-to-End ConvNet-based Speech Recognition System. *CoRR* abs/1609.03193 (2016). arXiv:1609.03193 http://arxiv.org/abs/1609.03193

[23] R De Lucia, G Zucchelli, V Barletta, A Di Cori, M Giannotti Santoro, M Parollo, L Segreti, S Viani, V Della Tommasina, L Paperini, et al. 2020. The in-ear region as a novel anatomical site for ECG signal detection: validation study on healthy

volunteers. *Journal of Interventional Cardiac Electrophysiology* 60 (2020), 1–8. https://idp.springer.com/authorize/casa?redirect_uri=https://link.springer.com/article/10.1007/s10840-020-00709-x

[24] Denis Demidov. 2012. VexCL: Vector expression template library for OpenCL. https://2013.nscf.ru/TesisAll/Section%203/09_1430_DemidovDE_S3.pdf

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[26] Digi-Key. 2005. *Speaker CMS-151125-076SP-67.* https://www.puiaudio.com/media/SpecSheet/AR01032MR-2-R.pdf

[27] EarableAI. 2021. *Smart Brain Care Wearable.* EarableAI. https://earable.ai/preorder-en/

[28] Eargo. 2021. *EARGO NEO.* Eargo. https://shop.eargo.com/

[29] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. 2021. Snafu: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, Manhattan, NY, 1027–1040. https://doi.org/10.1109/ISCA52012.2021.00084

[30] Yifan Gong and Yu-Hung Kao. 2000. Implementing a high accuracy speaker-independent continuous speech recognizer on a fixed-point DSP. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, Vol. 6. IEEE, IEEE, Manhattan, NY, 3686–3689.

[31] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*, Francis Bach and David Blei (Eds.). PMLR, Lille, France, 1737–1746. https://proceedings.mlr.press/v37/gupta15.html

[32] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep Speech: Scaling up end-to-end speech recognition. *CoRR* abs/1412.5567 (2014). arXiv:1412.5567 http://arxiv.org/abs/1412.5567

[33] Muhammad Huzaifa, Rishi Desai, Xutao Jiang, Joseph Ravichandran, Finn Sinclair, and Sarita V. Adve. 2020. Exploring Extended Reality with ILLIXR: A New Playground for Architecture Research. *CoRR* abs/2004.04643 (2020). arXiv:2004.04643 https://arxiv.org/abs/2004.04643

[34] IntegrIT, Limited 2020. *NatureDSP Signal for HiFi4.* IntegrIT, Limited.

[35] TDK InvenSense. 2020. *Bottom Port PDM Low-Power Multi-Mode Microphone With High AOP Mode.* TDK InvenSense. https://www.cdiweb.com/datasheets/invensense/ds-000357-t3902-v1.0.pdf

[36] Jabra. 2021. *Jabra Sport Pulse.* Jabra. https://www.jabra.com/sports-headphones/jabra-sport-pulse-wireless#100-96100010-02

[37] François Jarrier-Gellez. 2022. FragJage/SpeakerVoiceIdentifier. https://github.com/FragJage/SpeakerVoiceIdentifier

[38] Fahim Kawsar, Chulhong Min, Akhil Mathur, and Allesandro Montanari. 2018. Earables for personal-scale behavior analytics. *IEEE Pervasive Computing* 17, 3 (2018), 83–89.

[39] Sumin Kim, Seunghwan Oh, and Youngmin Yi. 2021. Minimizing GPU Kernel Launch Overhead in Deep Learning Inference on Mobile GPUs. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (Virtual, United Kingdom) *(HotMobile '21)*. Association for Computing Machinery, New York, NY, USA, 57–63. https://doi.org/10.1145/3446382.3448606

[40] Rakesh Kumar, Keith I Farkas, Norman P Jouppi, Parthasarathy Ranganathan, and Dean M Tullsen. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, IEEE, Manhattan, NY, 81–92.

[41] Hyoukjun Kwon. 2021. *MAERI GITHUB.* maeri-project. https://github.com/maeri-project

[42] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices* 53, 2 (2018), 461–475.

[43] Ruggero Donida Labati, Enrique Muñoz, Vincenzo Piuri, Roberto Sassi, and Fabio Scotti. 2019. Deep-ECG: Convolutional neural networks for ECG biometric recognition. *Pattern Recognition Letters* 126 (2019), 78–85.

[44] Waverly Labs. 2022. https://www.waverlylabs.com/

[45] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *CoRR* abs/1909.11942 (2019). arXiv:1909.11942 http://arxiv.org/abs/1909.11942

[46] Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. 2014. Head tracking for the Oculus Rift. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, IEEE, Manhattan, NY, 187–194.

[47] Charles E Leiserson. 1985. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers* 100, 10 (1985), 892–901.

[48] Bede Liu et al. 1976. Fixed-point fast Fourier transform error analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24, 6 (1976), 563–573.

[49] Wenxin Liu, David Caruso, Eddy Ilg, Jing Dong, Anastasios I Mourikis, Kostas Daniilidis, Vijay Kumar, and Jakob Engel. 2020. TLIO: Tight Learned Inertial Odometry. *IEEE Robotics and Automation Letters* 5, 4 (2020), 5653–5660.

[50] MaximIntegrated. 2014. *Low-Power, Ultra-Accurate 6 DoF IMU.* MaximIntegrated. https://datasheets.maximintegrated.com/en/ds/MAX21105.pdf

[51] Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, Deepika Bablani, John V. Arthur, Izzet B. Yildiz, and Dharmendra S. Modha. 2018. Discovering Low-Precision Networks Close to Full-Precision Networks for Efficient Embedded Inference. *CoRR* abs/1809.04191 (2018). arXiv:1809.04191 http://arxiv.org/abs/1809.04191

[52] Mediatek. 2021. *Mediatek MT2533.* Mediatek. https://www.mediatek.com/products/wearables/mt2533

[53] Microchip. 2021. *IS2062/64.* Microchip. http://ww1.microchip.com/downloads/en/DeviceDoc/60001409D.pdf

[54] mrDIMAS. 2022. mrDIMAS/rg3d. https://github.com/mrDIMAS/rg3d

[55] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.

[56] Subhashini Narayan and E Sathiyamoorthy. 2019. A novel recommender system based on FFT with machine learning for predicting and identifying heart diseases. *Neural Computing and Applications* 31, 1 (2019), 93–102.

[57] Tony Nowatzki, Vinay Gangadhar, Newsha Ardalani, and Karthikeyan Sankaralingam. 2017. Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, IEEE, Manhattan, NY, 416–429.

[58] Cedric Nugteren. 2018. CLBlast: A Tuned OpenCL BLAS Library. In *Proceedings of the International Workshop on OpenCL* (Oxford, United Kingdom) *(IWOCL '18)*. Association for Computing Machinery, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/3204919.3204924

[59] Nuraphone. 2021. How It Works: Music in full colour: Personalized sound. https://www.nuraphone.com/pages/how-it-works

[60] Nuvoton. 2021. *Nuvoton audio DSP.* Nuvoton. https://www.nuvoton.com/export/resource-files/TRM_ISD94100_Series_EN_Rev1.09.pdf

[61] NXP. 2021. *NXP LPC54114.* NXP. https://www.nxp.com/docs/en/application-note/AN12593.pdf

[62] James Peckham and Sharmishta Sarkar. 2019. Bose Frames review. https://www.techradar.com/uk/reviews/bose-frames-review

[63] Petersn. 2020. petersn/tinysr. https://github.com/petersn/tinysr

[64] Qualcomm. 2021. *Qualcomm AptxHd.* Qualcomm. https://www.aptx.com/aptx-hd

[65] QuickLogic. 2021. *QuickLogic.* QuickLogic. https://www.marketwatch.com/press-release/quicklogics-amazon-qualified-reference-design-brings-alexa-to-hearables-2021-02-18?siteid=bigcharts&dist=bigcharts&tesla=y

[66] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. *CoRR* abs/1606.05250 (2016). arXiv:1606.05250 http://arxiv.org/abs/1606.05250

[67] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. *CoRR* abs/1606.05250 (2016). arXiv:1606.05250 http://arxiv.org/abs/1606.05250

[68] Research and Markets. 2020. Hearables Market by Products, Type, Connectivity Technology, and End User: Global Opportunity Analysis and Industry Forecast, 2019-2026. https://www.researchandmarkets.com/reports/5021786/hearables-market-by-products-type-connectivity

[69] Romit Roy Choudhurry, Yu-Lin Wei, and Zhijian Yang. 2022. private communication.

[70] Scribd. 2019. Bragi pivot press release. https://www.scribd.com/document/404134059/Bragi-pivot-press-release

[71] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. 2012. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *2012 IEEE wireless communications and networking conference workshops (WCNCW)*. IEEE, IEEE, Manhattan, NY, 232–237.

[72] Ruby Singh. 2021. *Apple AirPods Pro: A Complete Review.* WirelessEarBuds.best. https://www.wirelessearbuds.best/product/apple-airpods-pro-a-complete-review/

[73] Sony. 2021. *Sony WF-1000XM3 Wireless Noise-Canceling Headphones.* Sony. https://www.sony.com/electronics/truly-wireless/wf-1000xm3

[74] Sudarshan Srinivasan, Pradeep Janedula, Saurabh Dhoble, Sasikanth Avancha, Dipankar Das, Naveen Mellempudi, Bharat Daga, Martin Langhammer, Gregg Baeckler, and Bharat Kaul. 2019. High Performance Scalable FPGA Accelerator for Deep Neural Networks. *CoRR* abs/1908.11809 (2019). arXiv:1908.11809 http://arxiv.org/abs/1908.11809

[75] Tensilica, Inc 2010. *Xtensa Instruction Set Architecture.* Tensilica, Inc.

[76] Videolabs. 2020. videolabs/libspatialaudio. https://github.com/videolabs/libspatialaudio

[77] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. 2020. Dsagen: Synthesizing programmable spatial accelerators. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, IEEE, Manhattan, NY, 268–281.

[78] Zhijian Yang, Yu-Lin Wei, Sheng Shen, and Romit Roy Choudhury. 2020. *Ear-AR: Indoor Acoustic Augmented Reality on Earphones.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3372224.3419213

[79] Hasan Erdem Yantir, Wenzhe Guo, Ahmed M Eltawil, Fadi J Kurdahi, and Khaled Nabil Salama. 2019. An ultra-area-efficient 1024-point in-memory fft processor. *Micromachines* 10, 8 (2019), 509.

[80] SD You and Yo-Cheng Hou. 2004. Implementation of IMDCT for MPEG2/4 AAC on 16-bit fixed-point digital signal processors. In *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings.*, Vol. 2. IEEE, IEEE, Manhattan, NY, 813–816.

[81] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, IEEE, Manhattan, NY, 36–39.

[82] ARM ML Zoo. 2021. *ML-zoo - speech recognition - wav2letter.* Arm Ltd. https://github.com/ARM-software/ML-zoo/tree/master/models/speech_recognition/wav2letter/tflite_int8