Exploiting Approximate Feature Extraction via Genetic Programming for Hardware Acceleration in a Heterogeneous Microprocessor

Hongyang Jia, *Student Member, IEEE*, and Naveen Verma, *Member, IEEE*

***Abstract*— This paper presents a heterogeneous microprocessor for low-energy sensor-inference applications. Hardware acceleration has shown to enable substantial energy-efficiency and throughput gains, but raises significant challenges where programmable computations are required, as in the case of feature extraction. To overcome this, a programmable feature-extraction accelerator (FEA) is presented that exploits genetic programming for automatic program synthesis. This leads to approximate, but highly structured, computations, enabling: 1) a high degree of specialization; 2) systematic mapping of programs to the accelerator; and 3) energy scalability via user-controllable approximation knobs. A microprocessor integrating a CPU with feature-extraction and classification accelerators is prototyped in 130-nm CMOS. Two medical-sensor applications (electroencephalogram-based seizure detection and electrocardiogram-based arrhythmia detection) demonstrate 325× and 156× energy reduction, respectively, for programmable feature extraction implemented on the accelerator versus a CPU-only architecture, and 7.6× and 6.5× energy reduction, respectively, versus a CPU-with-coprocessor architecture. Furthermore, 20× and 9× energy scalability, respectively, is demonstrated via the approximation knobs. The energy-efficiency of the programmable FEA is 220 GOPS/W, near that of fixed-function accelerators in the same technology, exceeding typical programmable accelerators.**

***Index Terms*— Approximate computation, feature extraction, machine learning, programmable accelerator, sensor inference.**

I. Introduction

HARDWARE accelerators have shown to substantially enhance energy efficiency. Going from CPU to GPU to dedicated accelerators, energy-efficiency improvement by up to 1000× is commonly seen [1]. This can be understood by looking at the energy breakdown of a typical CPU instruction, where over 90% of the energy goes toward programmability overheads, such as instruction/operand decoding/fetching. While accelerators largely avoid these, thereby deriving energy-efficiency and performance gains, programmability is also critical. Thus, many architectures have begun to focus on programmable acceleration. However, as seen in Fig. 1, balancing the overheads in state-of-the-art designs still incurs 10–100× degradation in energy efficiency [2]–[5]. Additionally, the approach most commonly used for programmable acceleration has been vector processing and parallelism (as in GPUs), where the aim is to amortize programmability overheads by applying a single instruction to multiple data. Unfortunately, such an approach is only effective if the data types in an application can be represented by vectors and/or parallelization can be made possible in other ways.

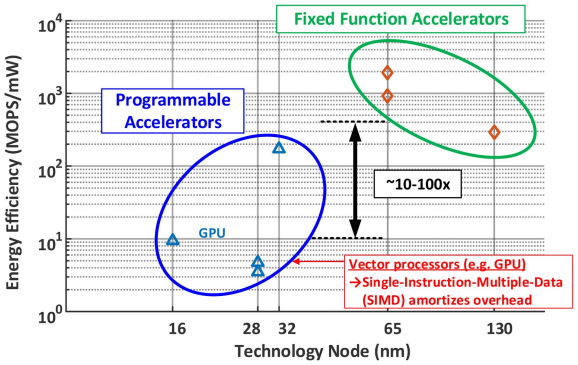


Fig. 1. Energy-efficiency versus programmability tradeoff observed for accelerator designs. Programmable acceleration has mostly focused on vector processing to amortize overheads.

A further problem with programmable acceleration is that accelerators are extremely difficult to program. First, even slight changes in a computation graph can preclude mapping to the accelerator. Second, the mapping process typically requires intimate knowledge of the accelerator's microarchitecture in order to exploit the efficiency gains, making programming difficult even for expert users.

This paper attempts to address both the programmability versus energy-efficiency tradeoff, as well as the challenges of application mapping. This is done by taking the advantage of approximation [6]. Fig. 2 shows the programmability versus energy-efficiency tradeoff introduced previously. The approach taken is to *approximate* the computations using highly structured models, which offer configurability through specific parameters. The structured models enable a high level of accelerator specialization, as well as systematic methods of mapping to the accelerator. As described later, genetic programming (GP) is employed for setting the model parameters.

Manuscript received August 3, 2017; revised October 19, 2017 and November 30, 2017; accepted December 15, 2017. Date of publication January 17, 2018; date of current version March 23, 2018. This paper was approved by Guest Editor Makoto Ikeda. This work was supported in part by AFOSR, in part by NSF under Grant CCF-1253670, and in part by C-FAR/SONIC, two SRC STARnet centers by MARCO and DARPA. *(Corresponding author: Hongyang Jia.)*

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: hjia@princeton.edu; nverma@princeton.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2017.2787762

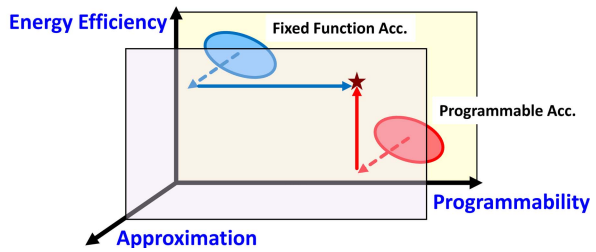


Fig. 2. Approximation of computations via highly structured models is exploited to address energy-efficiency versus programmability tradeoffs.

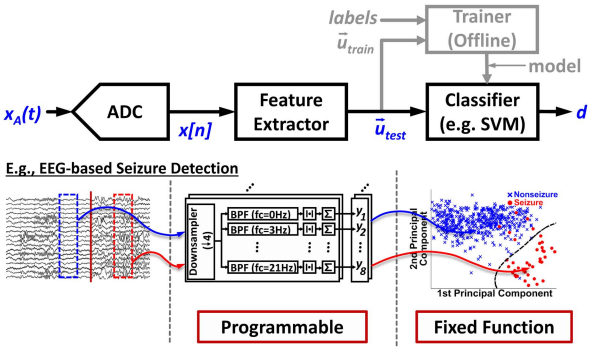


Fig. 3. Block diagram of a typical sensor-inference system.

The remainder of this paper is organized as follows. Section II provides an overview of the rationale behind the microprocessor architecture, including how GP is exploited and how tolerance to approximation errors is enhanced through classifier training, thereby enabling a large range for energy-approximation scalability. Section III describes the architectural and microarchitectural design of the microprocessor. Section IV presents the flow, both for program synthesis (via GP) and program mapping to the accelerator, as well as for classifier training. Section V presents the prototype-measurement and application-demonstration results, and provides the detailed analysis of how the observed energy savings are derived. Finally, Section VI concludes.

II. System Overview and Rationale

Fig. 3 shows a block diagram of a typical sensor-inference system, consisting of feature-extraction and classification stages. The feature extractor takes digitized sensor data $x[n]$, and maps them to feature vectors $\vec{u}_{test}$ that are conducive for pattern recognition. Namely, the feature vectors should exhibit generalization with respect to the inference of interest, so that patterns to be recognized are well expressed, while enabling the definition of metrics for pattern recognition, such as the level of similarity between data instances (often this is done by vector distances). Then, the classifier uses a model, learned through training on feature vectors $\vec{u}_{train}$ from previous data, in order to make decisions d from $\vec{u}_{test}$. For example, Fig. 3 shows the stages in an electroencephalogram (EEG)-based seizure-detection system, where the feature extractor takes an epoch of time-domain EEG data and maps it to a vector space. Here, seizure and non-seizure data are distributed with good separation. Then the classifier, having learned a model (representing a decision boundary between the distributions), declares incoming feature vectors as belonging to seizure or non-seizure classes.

TABLE I
Energy Breakdown of Sensor-Inference Applications on Microprocessor With Classification Accelerator [7]

	EEG-based Seizure Detection		ECG-based Arrhythmia Detection	
Total cycles (w/o Class. Accel.)	73M		102M	
	Fea. ext.	Class.	Fea. ext.	Class.
Cycle breakdown	1.97M (2.7%)	71.03M (97.3%)	1.497M (1.5%)	100.50M (98.5%)
Total cycles (w/ Class. Accel.)	2M		1.5M	
	Fea. ext.	Class.	Fea. ext.	Class.
Cycle breakdown	1.97M (98.5%)	0.03M (1.5%)	1.497M (99.8%)	0.003M (0.2%)

A key point here is that the feature computations that result in well-separated distributions depend strongly on the application signals and the classifications of interest. Consequently, feature extraction requires a high level of programmability across applications. On the other hand, classification involves specific computational kernels (e.g., support vector machine (SVM), neural network, decision tree), employing the model from training and the input feature vector. Consequently, classification is readily delegated to a fixed-function accelerator. We point out that increasingly diverse machine-learning models are being proposed for addressing different application characteristics. While such diversity again applies primarily to feature extraction (e.g., in deep learning), some level of configurability in classification algorithms and hardware is also showing value [6]–[8].

Indeed, many previous systems have focused on configurable classification accelerators. As an example, Table I shows the energy breakdown in one such system [7]. The top considers the system without using the classification accelerator, while the bottom considers the system using the classification accelerator. Before acceleration, we see that classification dominates the compute cycles in two representative medical-sensor applications. However, after acceleration, the classification cycles are greatly reduced, leaving feature extraction to dominate, consuming 99% of the cycles. Thus, the aim of feature-extraction acceleration is to address this bottleneck but while maintaining a high level of programmability. Sections II-A and II-B overview the approach taken to enable programmable acceleration via energy-scalable approximation.

A. Genetic Programming for Feature Extraction

GP is a program-synthesis framework which takes inputs x_n and outputs y_m, and learns functions, called "GP models," which relate the inputs and outputs [9]. Its proposed application, in this paper, to feature extraction is illustrated in Fig. 4. What is important is that the GP models have very specific structure. Specifically, they are composed of "genes" $G_{m,i}$,

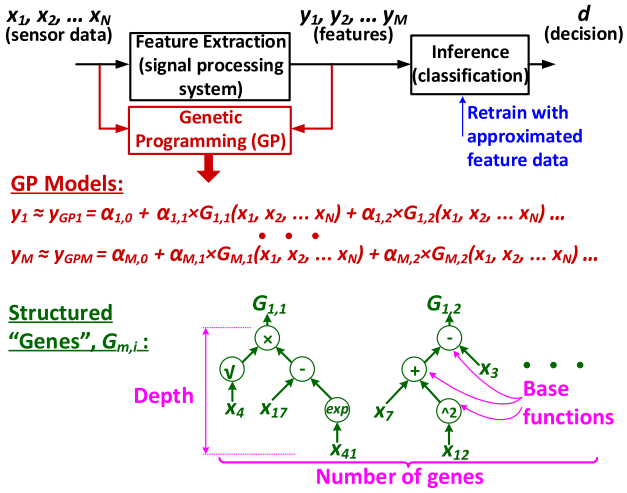


Fig. 4. Illustration of how GP is used in this paper for programmable feature extraction via structured models of computation.

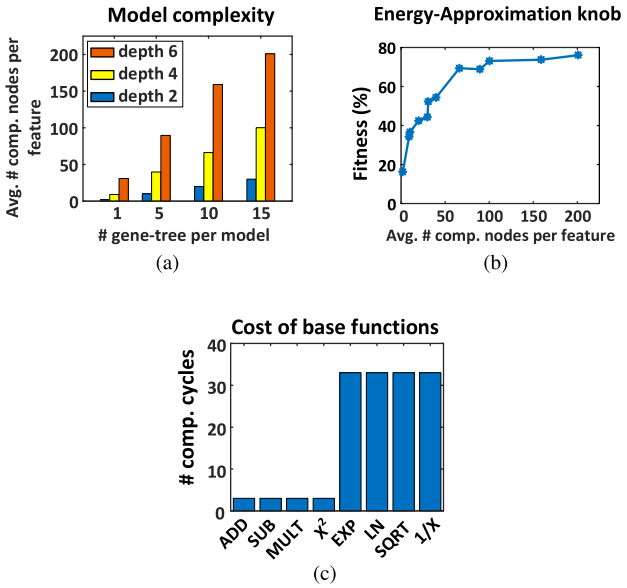


Fig. 5. Demonstration of energy-approximation scalability, seen by (a) how the number of computation nodes varies with constraints on the number and depth of gene trees, (b) how the fitness of GP models varies as a result, and (c) how the number of clock cycles varies for different nodal base functions implemented on the presented accelerator.

which are computation trees in which nodes correspond to "base functions" and edges correspond to intermediate outputs or primary inputs. The final output of a GP model y_{GPm} ($\approx y_m$) is a linear combination over gene-tree outputs. GP employs evolutionary algorithms, involving mutations, perturbations, and crossovers on this structure, in order to iteratively minimize a loss function, which may be set to depend on the error between y_{GPm} and y_m, so as to best map the inputs to outputs. In GP, this gene-tree structure has been important for generalization of the evolutionary search process, enabling fitting to a broad range of computations. In this paper, we exploit the structure toward a high level of accelerator specialization, as well as automatic and systematic mapping of computations to the accelerator.

In addition to specialization, we also exploit GP to enable a knob for energy-approximation scaling. In GP, the number and depth of gene trees as well as the choice of base functions can be set by user-provided constraints, and we see that these determine both the computation complexity and the ability to accurately model functions via GP models. As an example, for a seizure-detection system described later, Fig. 5(a) shows how the number of computation nodes varies, as the depth and the number of gene trees per GP model are constrained. Accordingly, Fig. 5(b) shows the fitness of the approximated function as the number of computation nodes varies. Here, fitness is defined as the average variance explained over all features, we are interested in computing

$$\text{Fitness} = \frac{1}{M}\sum_{j=1}^{M}\left(1 - \frac{\sum_{i=1}^{N}(y_{i,j} - \hat{y}_{i,j})^2}{\sum_{i=1}^{N}(y_{i,j} - \bar{y}_j)^2}\right) \times 100\% \qquad (1)$$

where M is number of different features of interest, N is the number of tested values for each feature over which the approximation is evaluated, $y_{i,j}$ is the true value of the feature for the jth feature of interest and the ith tested value, $\hat{y}_{i,j}$ is the corresponding approximated feature, and $\bar{y}_j$ is the mean of the true value for the jth feature over the tested values. Furthermore, as shown in Fig. 5(c), in the accelerator design described later, ADD/SUB/MULT base functions require ~3 clock cycles, while EXP/LN/SQRT/$1/X$ require ~33 cycles. Thus, we see that GP can enable controllable approximation for energy savings. In Section II-B, we describe a method whereby the energy-approximation tradeoff and scalability range are enhanced through the statistical learning.

B. Classifier Retraining

In the system proposed, feature vectors derived from GP-model approximations are fed to a classification stage (Fig. 4). In order to enhance the energy-approximation knob for feature extraction, retraining of the classification model is performed to the approximated feature-vector data. In [10] an approach called data-driven hardware resilience is described, where such retraining is analyzed in terms of its potential and performance limits in the case of errors due to hardware non-idealities. The resulting model is referred to as an *error-aware model*. Using this approach, which is shown to enable substantial tolerance to errors, this paper employs retraining to construct an error-aware model to address GP-model approximation, particularly as the GP model complexity is scaled as described earlier.

To visualize this, Fig. 6 shows data from an EEG-based seizure-detection system and electrocardiogram (ECG)-based arrhythmia-detection system implemented on the prototype described later. For plotting purposes, we show just the first two principal components of the high dimensional feature-vector data. In the left plots, the class distributions resulting from feature vectors derived from the baseline algorithms are shown. In the next plots, the class distributions resulting from approximated feature vectors derived from GP models constrained to have different complexities are shown. We see

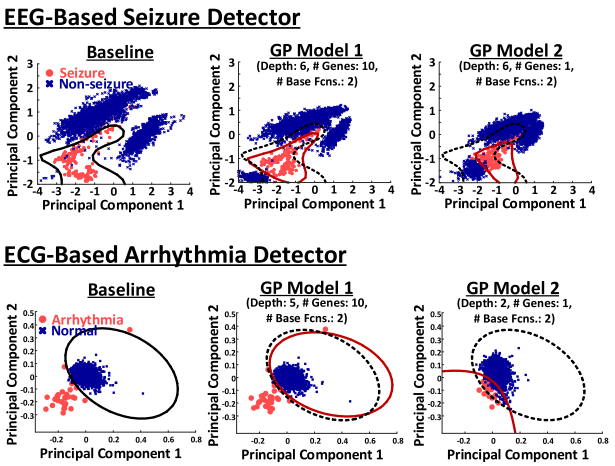


Fig. 6. Illustration of classifier retraining to derive decision boundaries optimized to GP-model approximations of feature-vector data.

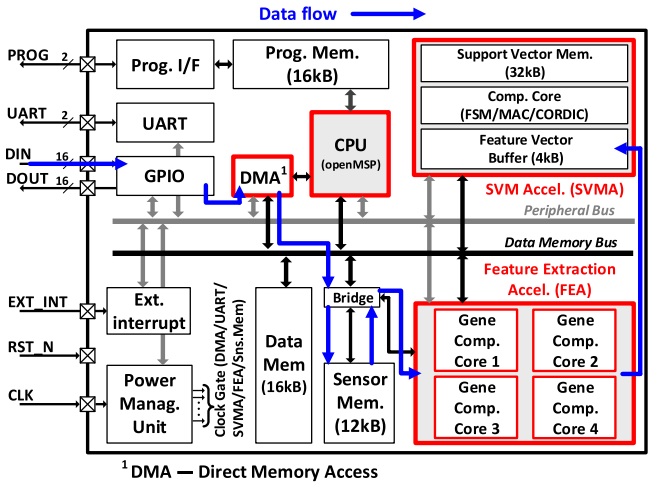


Fig. 7. Architecture of demonstrated heterogeneous microprocessor for sensor inference.

that the decision boundaries associated with feature vectors from the baseline algorithms (shown as dotted lines in next plots) are not optimal in the presence of GP-model approximations. However, classifier retraining, to derive an error-aware model for the approximated feature vectors (represented by decision boundaries shown as red solid-lines), provides much better classwise discrimination, thereby substantially restoring classification performance (as will be shown later). Our recent work looking at the software implementation of GP models provides additional validation of this approach on an algorithmic level [11]; here, we focus on a hardware architecture and application-mapping flow to exploit this.

It is important to note that a new classifier model derived from retraining can have system-level implications. For instance, in the case of an SVM, the new class distributions resulting from approximation can lead to an increased number of support vectors, and thus higher classification complexity. We examine this further in Section V-A.

III. Architectural and Micro-architectural Design

Fig. 7 shows the architecture of the demonstrated heterogeneous microprocessor. The key blocks include the following:

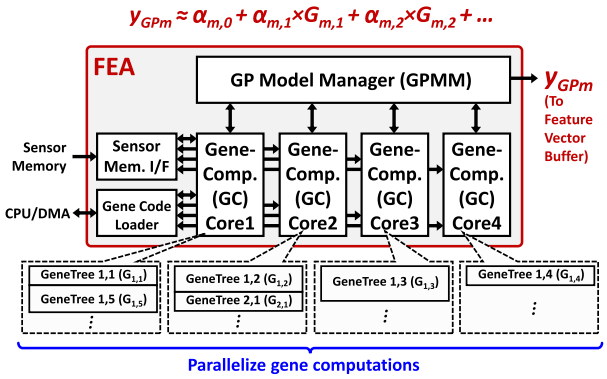


Fig. 8. Details of FEA.

1) CPU, based on an MSP430 instruction set, for top-level software control; 2) direct-memory-access (DMA) module, for automated data movement in applications without incurring CPU overheads; 3) a four-core feature-extraction accelerator (FEA), for programmable acceleration of GP-model computation; and 4) a support-vector-machine accelerator (SVMA) [7], for configurable but comparatively fixed-function acceleration of SVM classification. The accelerators and other modules are memory mapped and interfaced to the CPU via a peripherals bus, supported by the unified memory-address space of an MSP430 architecture.

Fig. 7 (blue arrows) shows the flow of data in an application, achieved automatically under DMA control without intervention of the CPU. The DMA starts by taking digitized (ADC) sensor data from the general-purpose input/output and moving it to the sensor memory. From here, the raw data are pulled by the FEA to compute features, which are then loaded into a feature vector buffer in the SVMA for classification. Finally, the SVMA asserts an interrupt to the CPU after classification is completed. Sections III-A to III-C below describe the block-level details.

A. Feature-Extraction Accelerator

Fig. 8 shows the details of the FEA, consisting of four gene computation (GC) cores, which compute individual gene trees and apply a scaling factor to their outputs. These outputs are provided to a GP model manager (GPMM) to compute the final feature values. The four cores enable parallel computation of the gene trees, which constitute the dominant compute task. Scheduling of the gene-tree computations, which can span different numbers of cycles, is managed by the GPMM. While, requests for fetching input operands made done through the sensor–data–memory interface.

1) Gene Computation Core: Fig. 9 shows the microarchitecture of a GC core. To exploit the tree-structured computation of genes, the design is a stack machine. Here, overheads associated with computational precedence control are avoided, as in reverse-polish notation (RPN) [12], enabling improved energy-efficiency and throughput [13]. In RPN, operands precede the operators applied to them, thereby enforcing a strict computational flow defined by the construction of the expression. A gene tree can be interpreted to an RPN expression,

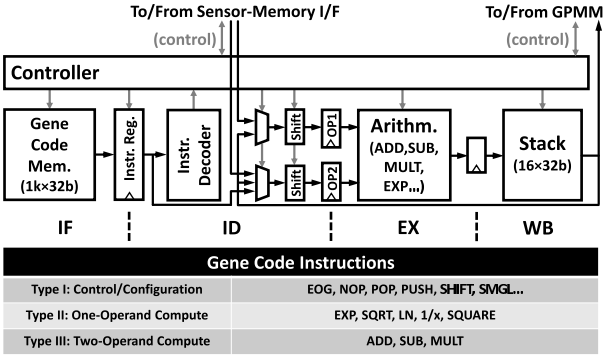


Fig. 9. Details of the GC core and its supported instructions.

by a post-order depth-first search (DFS) algorithm, in which only computation nodes generate instructions and edges do not. Specifically, there is no need for JUMP, BRANCH, etc. instructions, nor instructions for storing intermediate data back to memory, as typically required for general-purpose control and data flow. Simply, the result produced from a nodal base function, corresponding to the output edge of a node (gene trees are anti-arborescence, where all edges point toward the root), is immediately consumed by the next nodal base function or pushed to a shallow stack. As we analyze later in Section V-B, this substantially reduces the data-movement overheads typically incurred in programmable processors, and is critical to the energy savings.

Accordingly, the design implements a single-instruction-pipeline stack machine, where each instruction roughly maps to a gene-tree node. The pipeline consists of four stages: instruction fetch (IF); instruction decode and operand fetch (ID); execution (EX); and write-back (WB). But, we note that IF and WB stages execute simultaneously on consecutive instructions, and the EX stage can be stalled, in the case of delays during operand accessing from the sensor-data memory (operands from the stack and/or constants within the instruction never incur delay). Compared with conventional stack machines, we adopt two major features in our pipeline: 1) inclusion of a specialized arithmetic unit in the EX stage, for accelerating the (linear and non-linear) base functions and 2) extensions to operand fetching and handling within instructions (for instance, fetched operands are bit-shifted to manage computational dynamic range and stored in registers within the pipeline ID stage, thereby avoiding instructions and associated overheads for accessing memory and/or loading constants).

Operands from sensor-data memory and the instruction constants are formatted as 16-bit signed values, to support the precision required in typical sensing applications. Intermediate data are stored as 32-bit signed values, determined based on simulations to preserve computation precision, particularly for the non-linear operations supported. The arithmetic unit consists of the following: a 32-bit fixed-point full adder, to execute addition and subtraction; a 32-bit fixed-point multiplier, to execute multiplication and squaring; and a 32-bit coordinate rotation digital computer (CORDIC), to execute non-linear functions, such as exponential, natural logarithm, square root, and reciprocal. The CORDIC employs pre-scaling and post-scaling as proposed in [14] to effectively extend the dynamic range. Table II shows a comparison between the number of cycles required by the CPU versus a GC core for the various base functions; thus, we see that the component of energy reduction in the FEA due to computation acceleration, particularly for non-linear functions, is expected to be a significant portion of the overall savings (analyzed in Section V-B).

TABLE II
APPROXIMATE CYCLE COMPARISON OF BASE FUNCTION EXECUTION ON FEA VERSUS CPU

	FEA Cycles[1, 2]	CPU Cycles[3]
ADD[4]/SUB[4]	3	150
MULT[4]/SQUARE	3	180
EXP	33	3850
LN	33	2935
SQRT	33	2350
1/X	33	390

[1] IF and WB pipeline stages execute simultaneously.
[2] Assumes contention-free, one-cycle accessing from sensor-data memory (as typically seen in practice, details in Section III-A3).
[3] Based on using C math library with *float* data type, required for non-linear functions and used for ADD/SUB/MULT to avoid overhead (>100 cycles) of data type and decimal-to-integer conversion.
[4] These are two operand functions which might take two memory data or two stack data. In those cases, 1 additional cycle should be added to FEA cycles.

Fig. 9 shows the 32-bit instruction set, containing three types of instructions: 1) Type-I, corresponding to control and configuration; 2) Type-II, corresponding to one-operand computation; and 3) Type-III, corresponding to two-operand computation. The Type-I instructions implement no-operation for workload balancing and scheduling (NOP), end-of-gene and end-of-file for synchronization with the GPMM (EOG/EOF), stack pop and push for facilitating RPN data flow (POP/PUSH), setting bitwise shifting of subsequent operands (SHIFT) for managing computational dynamic range, and writing to dedicated registers in the GPMM (SMGL) to facilitate GPMM scheduling and control. The Type-II instructions implement exponential, natural logarithm, square root, reciprocal, and square operations (EXP/LN/SQRT/1/X/SQUARE). All of these make use of the CORDIC, except for SQUARE, which gets masked as an MULT after the ID stage. When processing Type-II instructions, the GC-core pipeline accesses the operand from sensor-data memory, stack top, or the *constant* field within the instruction. The Type-III instructions implement addition, subtraction, and multiplication (ADD/SUB/MULT). As with Type-II instructions, when processing Type-III instructions, the GC-core pipeline accesses either operand from the sensor-data memory, stack top, or the *constant* field. Operations with two constants are pre-computed at program-synthesis time and are replaced with a PUSH instruction. We note that operations

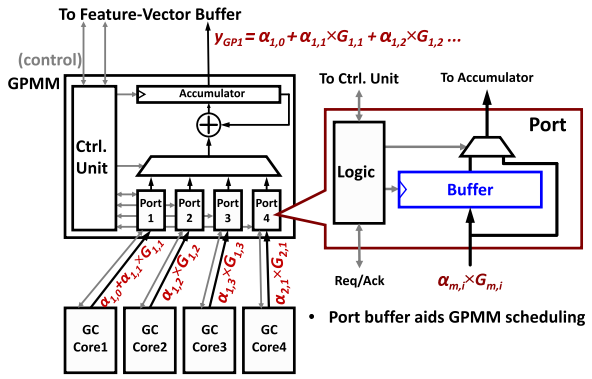


Fig. 10. Details of the GPMM.

with one memory operand and one constant operand are not supported, in order to keep the instruction length small. Instead, such operations are separated into one PUSH (to load constant into stack) and another memory-operand instruction to complete the operation.

The maximum required stack depth to compute a gene tree, $G_{m,i}$, is given as

$$\mathrm{sd}(G_{m,i}) \leq \mathrm{td}(G_{m.i}) + \deg^{-}(G_{m,i}) - 2 \tag{2}$$

where sd is stack depth, td is tree depth, and $\deg^{-}$ is the maximum in-degree of nodes in a gene tree, which is 2 in this design. Here, the tree-depth assumes memory accesses and constants occur at leaf nodes, requiring no stack depth. However, splitting of a memory-constant instruction into two instructions, as mentioned earlier, increases the required depth by one, making the associated constant term in this special case -2, not -3. While Eq. 2 corresponds to the maximum depth required for full trees, the depth taken by smaller trees is correspondingly reduced by swapping left and right subbranches, causing the execution of nodal base functions to be reordered during DFS. Such reordering is supported by the GC core microarchitecture by enabling operands to be selectively swapped before being sent to the arithmetic unit, depending on a control bit within instructions corresponding to non-commutative base functions (such as subtraction). Thus, a stack of depth 16 is adequate for a depth-16 gene tree, which is found by us and others [9], [12] to be a practical limit both easily handled by GP engines and required in profiled applications.

2) Genetic-Programming Model Manager: Fig. 10 shows details of the GPMM block. Primarily, the GPMM pops results from the GC-core stacks, and accumulates them to compute the approximated feature value y_{GPm}. In order to manage multiple GC-core completion requests and mitigate contention among GC cores, the GPMM has a buffer at each GC-core interface port to assist scheduling.

Recall that each GP model will typically contain multiple genes, each requiring a different number of execution cycles.[1] Fine-grained parallelism via the four GC cores is achieved while avoiding computational dependencies, by doing allocation to GC cores at the gene-tree level. Since the scaling factors involved in gene-tree output accumulation are applied within the GC core, minimal synchronization is required in the GPMM (i.e., execution of gene trees can proceed in any order); simply, GP-model gene trees are loaded in the GC cores contiguously (as shown in Fig. 8).

For GP-model computation, each GC core writes the GP-model index of the gene tree into a dedicated register before beginning its execution. The GPMM then reads that GP-model index upon gene-tree completion, and if it corresponds to the GP model currently being computed, and the accumulator is not occupied, the GPMM will read the result from the GC core to perform accumulation and release the GC core for subsequent gene-tree execution. If the completed gene tree does not belong to the current GP model or the accumulator is occupied, the GPMM stores its result and model index into a buffer at the port, and the GC core is again released for subsequent gene-tree execution. Thus, GPMM buffering is only required when the two conditions outlined are not met, which occurs infrequently in practice when gene trees are loaded in the GC cores contiguously. In fact, buffering requirements are greatly eased by workload balancing across the GC cores at program-synthesis time, since the highly structured data flow of gene trees substantially enhances the predictability of execution cycles required.

3) Sensor–Data–Memory Interface: With four GC cores, the FEA can generate eight simultaneous sensor-data memory-access requests (each core can take two operands at a time). To enable a single unified memory space with one port, the sensor–data–memory interface is designed to balance contention-induced delays to facilitate workload balancing among the GC cores at program-synthesis time. Since the sensor–data–memory interface has relatively light load (only gene-tree leaf nodes require sensor-data), a round-robin polling mechanism is employed. In the case of only a single request, access is granted to the corresponding GC core. In the case of multiple requests in the same clock cycle, the sensor–data–memory interface performs scheduling by granting access in round-robin order following from the last GC core granted access during the previous contention event. GC cores wait until access is thus granted.

B. Support-Vector-Machine Accelerator

The SVMA used is similar, in structure, to that presented in [7], essentially corresponding to a dot-product engine, supporting various inputs for SVM computations. Although the SVMA is a fixed-function accelerator, it retains configurability for various SVM kernels (linear, polynomial, and radial basis function), which are seen to strongly impact classification performance and energy depending on the distribution of feature vectors across different applications [7]. The radial-basis-function kernel employs a dedicated CORDIC within the SVMA.

C. Power-Management Unit

The CPU core used supports a software-controlled low-power mode (LPM), providing clock-gating of the CPU and memory bus, leaving only interrupt monitors active.

[1]We assume that all GP models will have the same number of gene trees.

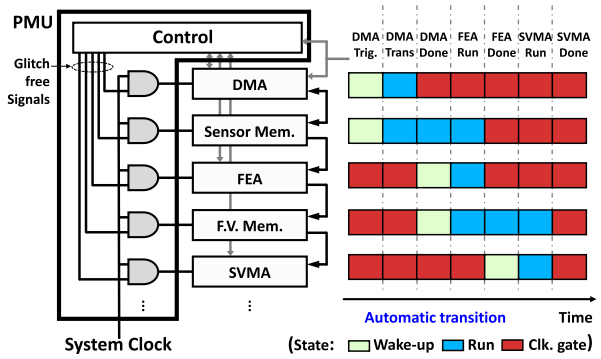


Fig. 11. Details of the PMU and its clock-gating functionality.

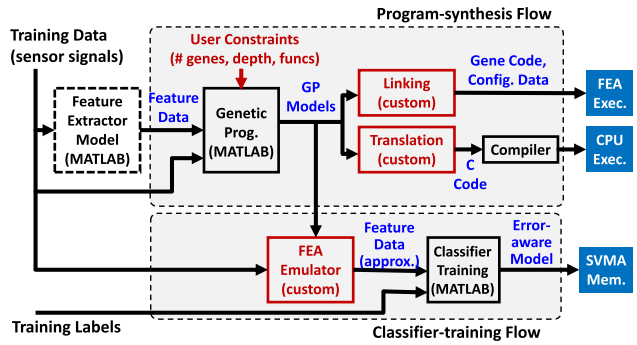


Fig. 12. Overview of program-synthesis and classifier-training flows.

Fine-grained hardware-controlled clock-gating of the other blocks in the architecture is managed by a custom power-management unit (PMU), with details shown in Fig. 11. As mentioned previously, data movement between the FEA and SVMA proceeds under DMA control without CPU intervention. Furthermore, as data progresses, the accelerator stages are clock gated to save active energy. On the right, Fig. 11 shows this clock-gating schedule as data progresses. As seen, thanks to fine-grain controllability of clock gating, blocks remain on for only the minimum required time. If data continually arrives, the stages remain on, supporting accelerator-level pipelined operation; in this case, to support the buffering of real-time sensor data, the DMA can interrupt accesses to the sensor-data memory by the FEA. All other blocks proceed through request/acknowledge signaling.

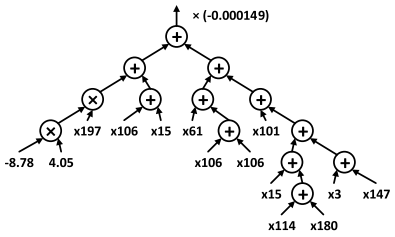
IV. Program-Synthesis and Classifier-Training Flows

Fig. 12 overviews the program-synthesis and classifier-training flows. The program-synthesis flow outputs gene code (and configuration data), corresponding to the instructions required for computing GP models on the FEA; additionally, C code for executing GP models on the CPU is also generated, for comparison and analysis. The classifier-training flow outputs an error-aware classification model for use in the SVMA, corresponding to feature vectors derived from the particular GP models.

A. Program Synthesis Flow

The program-synthesis flow takes as inputs: 1) sensor-signal input samples and 2) a computational model of a baseline feature extractor. For instance, this might be a signal-processing system implemented in MATLAB. Using these two, baseline outputs are generated, which are provided along with the sensor-signal samples as the outputs and inputs that must be related through GP models. Next, a GP toolbox [15] is used to derive the GP models. This is where user constraints on number, depth, and nodal base functions for the genes are provided for energy-approximation scalability. Then, the GP models are fed to a custom linker and translator to generate the gene code and configuration data for FEA execution, as well as C code (which can be compiled) for CPU execution. For illustration, Table III shows how derived GP models are systematically converted into gene code, through symbolic and RPN expressions.

TABLE III
Example for Gene-Code Synthesis

Gene tree			
Symbolic expression of a gene tree	factor = -0.000149 gene = add(add(mult(mult([-8.782928], [4.054360]), x197), add(x106, x15)), add(add(x61, add(x106, x106)), add(x101, add(add(x15, add(x114, x180)), add(x3, x147)))))		
RPN expression of a gene tree	-8.782928 4.054360 × x197 × x106 x15 + + x106 x106 + x61 + x114 x180 + x15 + x3 x147 + + x101 + + + -0.000149287414085 ×		
Gene code	SMGL	0	10
	SHIFT	11	
	PUSH	C_-35.6091519661	
	MULT	S_0	X_197
	ADD	X_106	X_15
	ADD	S_1	S_0
	ADD	X_106	X_106
	ADD	X_61	S_0
	ADD	X_114	X_180
	ADD	X_15	S_0
	ADD	X_3	X_147
	ADD	S_1	S_0
	ADD	X_101	S_0
	ADD	S_1	S_0
	ADD	S_1	S_0
	MULT	C_-0.000149287414085	S_0
	EOG		

B. Classifier-Training Flow

The classifier-training flow takes as inputs: 1) the sensor-signal input samples; 2) the GP models derived in the program-synthesis flow; and 3) classification training labels. Then, by using a custom FEA emulator, bit-true approximated feature-vector data are computed using the sensor-signal

TABLE IV
PROTOTYPE-MEASUREMENT SUMMARY TABLE

Technology	GF 130nm CMOS	
Supply Voltage	0.75 – 1.2 V	
Clock Frequency	5.0 – 29 MHz	
Energy per Clock	0.75 V	1.2 V
(1) CPU	21.8 pJ	57.1 pJ
(2) FEA	86.7 pJ	230.5 pJ
(3) SVMA	68.3 pJ	182.9 pJ
Energy per F.V. (1.2V)[1]:	FEA/SVMA	
(1) Seizure Detect – GP Model 1	3.38 μJ/18.3 μJ	
(2) Seizure Detect – GP Model 2	1.01 μJ/31.6 μJ	
(3) Arrhyth. Detect – GP Model 1	0.35 μJ/0.18 μJ	
(4) Arrhyth. Detect – GP Model 2	0.045 μJ/0.18 μJ	

[1] Energy of all other blocks < 40 nJ/F.V.

samples and GP models. The resulting feature-vector data are then provided along with the training labels to an SVM training toolbox [16], giving the error-aware model, which can be loaded into the SVMA.

V. PROTOTYPE MEASUREMENTS

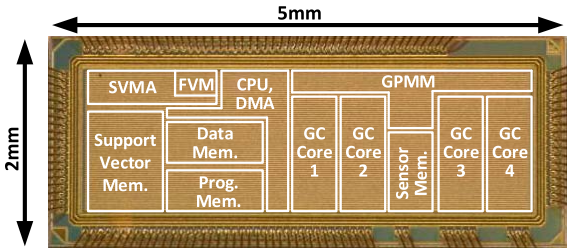
A prototype of the heterogeneous microprocessor is developed in 130-nm CMOS technology, with 96 kB of on-chip SRAM. Measurement summary and the IC die photograph are provided in Table IV and Fig. 13, respectively. The IC operates with supply voltages scaling from 0.75 to 1.2 V, with the maximum clock frequency scaling from 5 to 29 MHz over this range. At the lowest supply voltage (lowest energy point), the CPU consumes 21.8 pJ per clock cycle, the FEA (which has four GC cores) consumes 86.7 pJ per clock cycle, and the SVMA consumes 68.3 pJ per clock cycle.

In addition to voltage scaling for energy efficiency, the prototype supports clock-gated duty cycling via the PMU to maintain the energy efficiency as the required classification rate is reduced. Fig. 14 shows the system power while running an example application end-to-end, at different duty cycles. As seen, the PMU enables large and duty-cycle-proportional power savings, beyond the cases without any clock gating and with CPU-only clock gating (LPM).

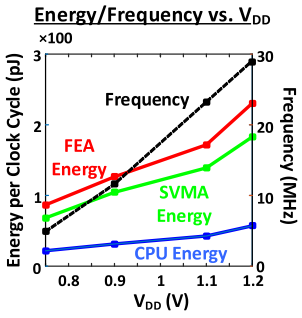
Table V provides a comparison summary of the prototype with previously reported fixed-function and programmable accelerators [3], [5], [17]. The overall energy efficiency of the FEA is estimated to be 221 GOPS/W (assuming no GC-core stalling), which surpasses the programmable accelerators and is near the fixed-function accelerators. Sections V-A and V-B provide further details regarding the application demonstrations and energy analysis.

A. Application Demonstrations

Two medical-sensor applications are demonstrated on the prototype: 1) an EEG-based seizure-detection system, based on the algorithm in [18] and 2) an ECG-based cardiac-arrhythmia detection system, based on the algorithm in [19]. Fig. 15(a) shows the baseline algorithm for the seizure-detection system. The baseline features correspond to the spectral energy distribution of each EEG channel, computed using eight FIR filters and absolute-value accumulators applied to each of six EEG channels, giving a total of 48 features. Fig. 15(b) shows



(a)



(b)

Fig. 13. Prototype IC of heterogeneous microprocessor, showing (a) die photograph and (b) energy breakdown and operating frequency versus supply voltage.

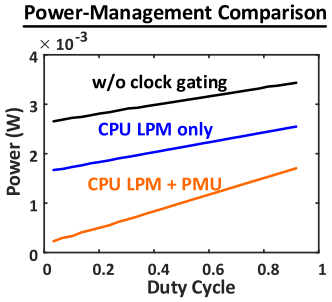


Fig. 14. System power versus application's duty cycle under different power-management configurations with supply voltage at 1.2 V.

TABLE V
COMPARISON TABLE WITH STATE-OF-THE-ART ACCELERATORS

	FIXED FUNCTION		PROGRAMMABLE		
	Moons [17]	Kim [3]	GPGPU Nvidia P100	Bohnenstiehl [5]	This Work
Technology	40 nm	130 nm	16 nm	32 nm	130 nm
Quantization Strategy	1-16 bit fixed	16 bit fixed	Mixed	16 bit fixed	32 bit fixed
Function	CNN	Obj. Recog.	Prog.	Prog.	Prog.
Energy Scale Strategy	Bit precision	--	--	--	GP model approx.
Energy Scale Range	8.7 ×	--	--	--	9 - 20 ×
Energy Efficiency	0.2 - 2.7 TOPS/W	290 GOPS/W	10 GFLOPS/W	172 GOPS/W	221 GOPS/W‡

‡ 1 OP equals 1 16 bit ADD/MULT

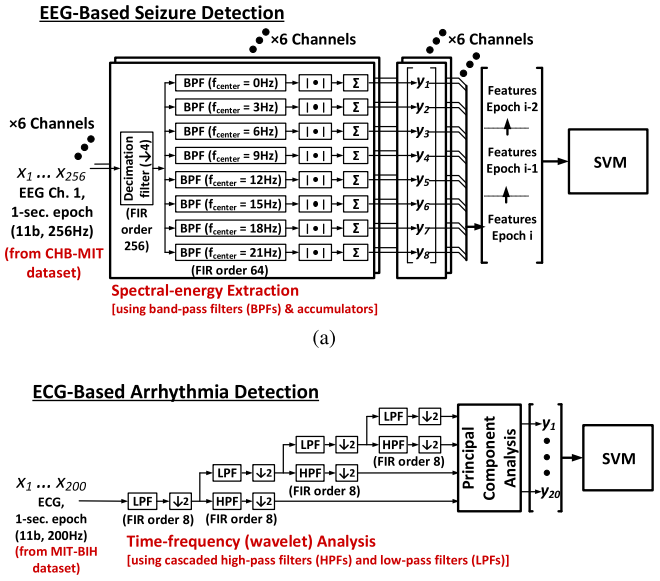


Fig. 15. Baseline algorithms for demonstrated applications. (a) EEG-base seizure detection. (b) ECG-based arrhythmia detection.

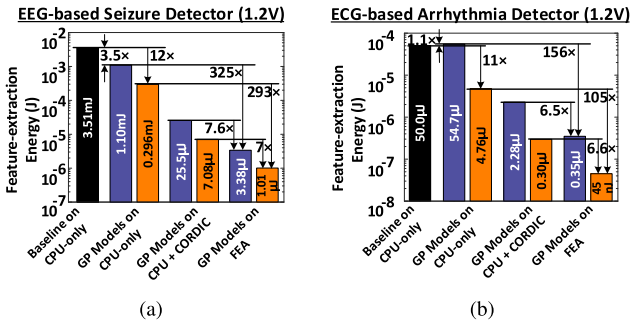


Fig. 16. Energy savings for computing two GP models on FEA versus CPU-only and CPU-with-coprocessor architectures for (a) EEG-base seizure-detection system and (b) ECG-based arrhythmia-detection system.

the baseline algorithm for the arrhythmia-detection system. The baseline features correspond to time-frequency analysis, computed using a four-stage wavelet filter bank followed by principal component analysis, giving a total of 20 features.

Fig. 16(a) and (b) shows the energy to compute each feature vector in the seizure-detection and the arrhythmia-detection systems, respectively. For each application, four cases are shown: 1) the baseline algorithm running on a CPU-only architecture; 2) two different GP models, representing different approximation points, running on the CPU-only architecture; 3) the same two GP models running on an architecture with a CPU and a coprocessor for accelerating GP-model base functions (i.e., similar to [20], [21]); and 4) the same two GP models running on the FEA. To enable the consistency of energy measurements, the CPU corresponds to the simple openMSP430 core employed in the prototype (having limited computational extensions due to the accelerator-centric architecture, e.g., having 16-bit multiplier but no floating-point unit), and the coprocessor corresponds to the arithmetic

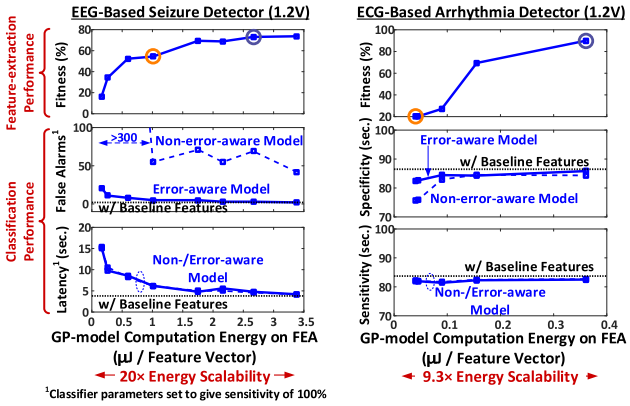


Fig. 17. Fitness and classification performance versus energy of feature extraction from variously constrained GP models (blue lines) compared with baseline computation (black lines) for (a) EEG-base seizure-detection system and (b) ECG-based arrhythmia-detection system.

unit (with CORDIC) employed in the FEA (modeled as being accessible directly from the peripherals bus); the coprocessor eliminates all floating-point and floating-point/integer conversion operations in the CPU.

As seen, for the seizure-detection system, the FEA results in 325× and 293× energy reduction for the two GP models considered compared with the CPU-only architecture, and 7.6× and 7.0× energy reduction compared with the CPU-with-coprocessor architecture. For the arrhythmia-detection application, the FEA results in 156× and 105× energy reduction for the two GP models considered compared with the CPU-only architecture, and 6.5× and 6.6× energy reduction compared with the CPU-with-coprocessor architecture. Detailed breakdown, analyzing the source of the energy savings, is presented in Section V-B.

In addition to enabling a high level of specialization for significant energy savings via the FEA, the different GP models, derived by setting constraints on the gene depth, number, and nodal base functions, enable substantial energy scalability. Fig. 17(a) considers the seizure-detection system, and Fig. 17(b) considers the arrhythmia-detection system, showing the feature-extraction and classification performance versus the energy per feature-vector for different GP models, with the feature-extraction performance of the two models from above circled (black dotted lines correspond to classification performance with baseline features, while blue solid/dashed lines correspond to classification performance with GP-model features using error-aware/non-error-aware models). As seen, the error-aware model significantly enhances classification performance, enabling a wide range for energy-approximation scalability (spanning an energy range of 20× for the seizure-detection system and 9.3× for the arrhythmia-detection system), thus enabling optimization across applications.

In addition to the FEA energy, it is interesting to note that the impact GP model scaling has on the classifier. When an error-aware model is used, the approximated features can potentially require more complex classification models, for instance in the form of an increased number of support vectors.

TABLE VI
APPROXIMATE COMPUTE CYCLE BREAKDOWN OF AN EXAMPLE GP MODEL

		CPU-only		FEA	
		number of operations	cycles per op.	number of operations	cycles per op.
Base function computation	ADD	105	150	89	3
	MULT	33	180	33	3
	total	21700		366	
Overhead	type	data type conversion		GC config. & push	
	cycles	10500		73	
Intermediate data movement		1800		–	
Total cycles		34000		439	

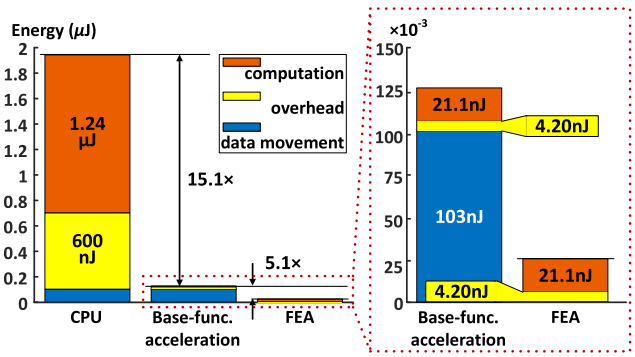


Fig. 18. Computation energy breakdown of an example GP model.

As shown at the bottom of Table IV, this is indeed seen in the case of the seizure-detection system, elevating the SVMA energy and opposing the FEA energy savings. On the other hand, in the arrhythmia-detection system, an SVM kernel formulation is employed wherein energy does not scale with the number of support vectors [7]. This suggests directions for further exploration of classifier architectures and algorithms suited to adaptive systems.

B. Energy Savings Analysis

First considering the FEA compared with the CPU-only architecture, detailed analysis shows that the energy savings observed earlier primarily come from three sources: 1) enhanced efficiency of computing nodal base functions, via the arithmetic unit; 2) reduction of data-movement and control-flow operations, thanks to the structure of gene trees; and 3) reduced energy for accessing instructions and data from memory, thanks to fewer instructions and optimized data placement enabled by computational specialization in GC cores. We illustrate these by breaking down the energy consumption for an example GP model, in terms of CPU/FEA core energies and memory energies. The GP model used corresponds to the first feature in the ECG-based arrhythmia-detection system [Fig. 15(b)], constrained to 20 genes with maximum depth of 7 and using only addition and multiplication base functions, chosen because these prove to yield high detection performance (note, that these lead to conservative energy-saving estimates compared with other non-linear functions, as seen in Table II).

Table VI provides a breakdown of the number of cycles consumed for GP-model execution on the CPU and FEA cores. We categorize according to base-function computations, overheads for data preparation before base-function computations (e.g., *float-int* type conversions in CPU, configuration and bit-shifting in FEA), and intermediate data movement to/from memory. For comparison, we assume the FEA employs only one GC core. For base-function computation, we observe ∼21700 versus ∼366 cycles on the CPU and FEA, respectively. While this accounts for how frequently ADD and MULT operations occur, in addition to the cycles per instruction shown in Table II, it also accounts for the slight savings in ADD operations due to specialized gene-tree output accumulation in the GPMM. For data-preparation overheads, we observe ∼10500 versus 73 cycles on the CPU and FEA, respectively. For data movement, we observe ∼1800 versus no cycles on the CPU and FEA, since in the FEA all intermediate results are retained within the GC-core hardware (and the energy for this is included in FEA core measurement).

Taking these cycle breakdowns, we now analyze the energy consumed, using the energy per cycle measurement for the CPU (57.1 pJ) and the energy per cycle measurement, normalized for one GC core, for the FEA (57.6 pJ), both taken at 1.2 V. Fig. 18 shows the corresponding energy breakdown for the CPU and FEA cores. As seen, 15.1× energy reduction is achieved by reducing base-function computation cycles through acceleration (including data-preparation overheads). Another 5.1× energy reduction is achieved by eliminating cycles for data movement. This yields a total of 77× energy reduction for the simple GP-model considered, with no non-linear base functions.

In addition to energy reduction within the FEA versus CPU core, FEA execution also benefits from reduced memory operations. The openMSP430 CPU, which has a three-stage pipeline, is measured to access data and program memory (D/P-MEM) 1.5× per clock cycle, during execution of the GP model, and each memory (both are 16 kB with identical structure) consumes 42 pJ per cycle. On the other hand, instruction fetching from gene-code memory (GC-MEM) in the FEA consumes 35 pJ (although FEA employs 32-bit instructions, while CPU employs 16-bit instructions, the gene-code memories are much smaller, only 4 kB each). Thanks to specialization, GP models are mapped to much fewer FEA instructions than CPU instructions, requiring 5.8× smaller memory footprint for the GP model considered, and no intermediate data are moved to memories external to the core, only reads from the sensor-data memory (SD-MEM) are required (42 pJ). Accordingly, Table VII shows the memory-access breakdown for the CPU and FEA, and Fig. 19 shows the memory-energy breakdown, amounting to 191× lower energy in the FEA. Taking the total energy numbers for the CPU and FEA cores as well as the memories, we observe 112× energy reduction for the simple GP-model example considered.

TABLE VII
MEMORY-ACCESS BREAKDOWN OF AN EXAMPLE GP MODEL

	CPU Memory Access	FEA Memory Access
D/P – MEM	51000	–
GC – MEM	–	195
SD – MEM	–	105

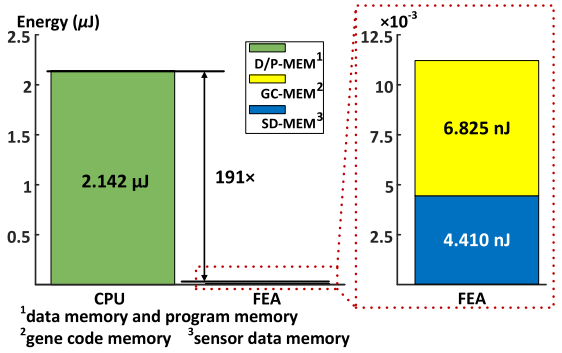


Fig. 19. Memory access energy breakdown of an example GP model.

In addition to large FEA energy savings compared with the CPU-only architecture, we also see significant FEA energy savings compared with the CPU-with-coprocessor architecture. In this case, both the FEA and CPU-with-coprocessor architecture benefit from computation acceleration, and so the FEA energy savings primarily come from the specialization of data flow. Specifically, the FEA, optimized for tree-structured GP-models via a stack architecture, first removes intermediate data movement, corresponding to the approximately 5.1× energy reduction, and second provides tighter coupling to the arithmetic unit, bringing the total energy savings to the 6–7× seen in Fig. 16(a) and (b).

VI. CONCLUSION

While hardware acceleration has been a highly promising approach to achieving energy savings in embedded sensing systems, it has mostly been applied to fixed computations. Examples of sensor-inference systems show that this leaves computations that require a high level of programmability, namely feature extraction, as dominating energy. To address this, a heterogeneous microprocessor based on a programmable FEA was presented. The approach to programmable acceleration exploits approximation, by employing GP to yield structured models of computation for approximating functions. The structure enables a high level of accelerator specialization while retaining programmability. Two medical-sensor applications implemented on a prototype IC in 130-nm CMOS demonstrate 325× and 156× energy reduction for feature extraction compared to a CPU-only architecture, and 7.6× and 6.5× energy reduction for feature extraction compared to a CPU-with-coprocessor architecture. Furthermore, controllable energy scalability of 20× and 9.3×, with respect to the level of feature approximation, is demonstrated for the two applications.

REFERENCES

[1] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[2] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2014, pp. 269–284.

[3] J. Y. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H. J. Yoo, "A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, Jan. 2010.

[4] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 80–81.

[5] B. Bohnenstiehl *et al.*, "A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32 nm 1000-processor array," in *IEEE Symp. VLSI Circuits (VLSI-Circuits) Dig. Tech. Papers*, Jun. 2016, pp. 1–2.

[6] H. Jia, J. Lu, N. K. Jha, and N. Yerma, "A heterogeneous microprocessor for energy-scalable sensor inference using genetic programming," in *IEEE Symp. VLSI Circuits (VLSI-Circuits) Dig. Tech. Papers*, Jun. 2017, pp. C28–C29.

[7] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *IEEE J. Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, Jul. 2013.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[9] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.

[10] Z. Wang, K. H. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 8, pp. 1459–1470, Aug. 2015.

[11] J. Lu, H. Jia, N. Verma, and N. K. Jha, "Genetic programming for energy-efficient and energy-scalable approximate feature computation in embedded inference systems," *IEEE Trans. Comput.*, to be published, doi: 10.1109/TC.2017.2738642.

[12] W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *Proc. 11th Eur. Conf. Genetic Program.*, Mar. 2008, pp. 73–85.

[13] P. J. Koopman, Jr., *Stack Computers: The New Wave*. New York, NY, USA: Halsted Press, 1989.

[14] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. AFIP Spring Joint Comput. Conf.*, 1971, pp. 379–385.

[15] D. P. Searson, D. E. Leahy, and M. J. Willis, "GPTIPS 2: An open-source software platform for symbolic data mining," in *Proc. Int. MultiConf. Eng. Comput. Sci. (IMECS)*, Mar. 2010, pp. 551–573.

[16] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, 2011. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

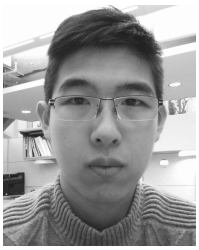
[17] B. Moons and M. Verhelst, "An energy-efficient precision-scalable ConvNet processor in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, Apr. 2017.

[18] A. H. Shoeb and J. V. Guttag, "Application of machine learning to epileptic seizure detection," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2013, pp. 975–982.

[19] E. D. Übeyli, "ECG beats classification using multiclass support vector machines with error correcting output codes," *Digit. Signal Process.*, vol. 17, no. 3, pp. 675–684, May 2007.

[20] J. Kwong, Y. K. Ramadass, N. Verma, and A. P. Chandrakasan, "A 65 nm sub-V_t microcontroller with integrated SRAM and switched capacitor DC-DC converter," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 115–126, Jan. 2009.

[21] S. R. Sridhara *et al.*, "Microwatt embedded processor platform for medical system-on-chip applications," in *IEEE Symp. VLSI Circuits (VLSI-Circuits) Dig. Tech. Papers*, Jun. 2010, pp. 15–16.



Hongyang Jia (S'13) received the B.Eng. degree in microelectronics from Tsinghua University, Beijing, China, in 2014, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree.

His research focuses on ultra-low-energy system design for inference applications. His current research interests include CMOS IC design, which leverages approximate computing technique for model complexity reduction, automatic program synthesis and mapping for feature extraction on various sensing platforms.

Mr. Jia received Analog Devices Outstanding Student Designer Award in 2017.

Naveen Verma (S'03–M'09) received the B.A.Sc. degree in electrical and computer engineering from The University of British Columbia, Vancouver, BC, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2005 and 2009, respectively.

Since 2009, he has been with the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, where he is currently an Associate Professor. His research focuses on advanced sensing systems, including low-voltage digital logic and SRAMs, low-noise analog instrumentation and data-conversion, large-area sensing systems based on flexible electronics, and low-energy algorithms for embedded inference, especially for medical applications.

Dr. Verma is a Distinguished Lecturer of the IEEE Solid-State Circuits Society, and serves on the technical program committees for the International Solid-State Circuits Conference (ISSCC), the VLSI Symposium, DATE, and the IEEE Signal-Processing Society (DISPS). He was a recipient or co-recipient of the 2006 DAC/ISSCC Student Design Contest Award, the 2008 ISSCC Jack Kilby Paper Award, the 2012 Alfred Rheinstein Junior Faculty Award, the 2013 NSF CAREER Award, the 2013 Intel Early Career Award, the 2013 Walter C. Johnson Prize for Teaching Excellence, the 2013 VLSI Symposium Best Student Paper Award, the 2014 AFOSR Young Investigator Award, the 2015 Princeton Engineering Council Excellence in Teaching Award, and the 2015 IEEE TRANSACTIONS ON COMPONENTS, PACKAGING AND MANUFACTURING TECHNOLOGY Best Paper Award.