In-Sensor Classification With Boosted Race Trees

By Georgios Tzimpragos, Advait Madhavan, Dilip Vasudevan, Dmitri Strukov, and Timothy Sherwood

Abstract

When extremely low-energy processing is required, the choice of data representation makes a tremendous difference. Each representation (e.g., frequency domain, residue coded, and log-scale) embodies a different set of trade-offs based on the algebraic operations that are either easy or hard to perform in that domain. We demonstrate the potential of a novel form of encoding, race logic, in which information is represented as the delay in the arrival of a signal. Under this encoding, the ways in which signal delays interact and interfere with one another define the operation of the system. Observations of the relative delays (for example, the outcome of races between signals) define the output of the computation. Interestingly, completely standard hardware logic elements can be repurposed to this end and the resulting embedded systems have the potential to be extremely energy efficient. To realize this potential in a practical design, we demonstrate two different approaches to the creation of programmable tree-based ensemble classifiers in an extended set of race logic primitives; we explore the trade-offs inherent to their operation across sensor, hardware architecture, and algorithm; and we compare the resulting designs against traditional state-of-the-art hardware techniques.

1. INTRODUCTION

In embedded applications, where the computation and sensing are close in both time and space, the exact type of data is something that needs to be carefully considered. Typically, a sensor gathers analog information from the physical world and then converts it into a conventional digital signal. For example, a camera captures incident photons and, through the photoelectric effect, uses their energy to guide the charging of a cell. The voltage on the cell is read out to an analog-to-digital converter (ADC) that converts the measured voltage into a stream of zeros and ones. Although this binary-represented integer is perfectly efficient for storage as bits in a memory and for general-purpose computing operations, it is unclear whether this is the most *energy efficient* solution. We posit that there are other encodings that, although still capturing the relative values of the data to be encoded, are more efficient for in-sensor processing.

One such possible representation is pure analog signaling. There is a long history of machine-learning-like computing with analog devices. Although pure analog design is always an option, it comes with a number of challenges of its own. First, well-understood analog design rules always lag far behind digital rules in available technology

node. High-density, high-performance, and low-energy CMOS analog parts can be hard to achieve because of this gap. Second, although analog design in these aggressive technology nodes is certainly possible, tighter margins for process variations and noise often drive analog designs to use larger gates than their digital counterparts. Ideally, we could keep the good parts of analog behavior, where the computation closely matches the capabilities of the underlying devices, without sacrificing the noise tolerance and layout simplicity of digital designs.

One class of logic that attempts to strike this balance is race logic.¹⁰ The key idea behind race logic is to encode values as a *delay* from some reference. All signals, unlike pure analog approaches, are supposed to be 0 or 1 at all times. However, the *time* at which 0 $\rightarrow$ 1 transition happens encodes the value. Computations are then based on the relative propagation times of signals injected into a configurable circuit. In prior work, it was shown that the basic temporal operators MAX, MIN, and ADD-CONSTANT could efficiently solve a class of dynamic programming algorithms, and both synchronous and asynchronous versions have been evaluated.^{10, 11} The inclusion of INHIBIT²⁰ opens the door to new computations, but the question of the efficiency of this approach to computing on larger and more general problems remains open.

To establish the interesting new capabilities that this more general race logic provides, we propose its application to a sensor-friendly yet machine-learning-ready encoding. For the experimental validation of our hypothesis, we complete an end-to-end evaluation that includes energy, throughput, and area utilization estimates for an ASIC design, a fully functional RTL implementation working in both simulation and on FPGA, SPICE models of the underlying primitives on which our system is built, a fully automated toolchain linking scikit-learn¹⁵ software structures down to device configurations, and accuracy versus energy analysis across a set of decision tree ensembles and design parameters. Even without accounting for the extra energy savings of using an encoding more natural to the sensor, the presented system dramatically reduces the total energy usage required for classification with very low latency.

The original version of this paper is entitled “Boosted Race Trees for Low Energy Classification” and was published in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, New York, NY, USA, 215–228; <https://doi.org/10.1145/3297858.3304036>

2. GENERALIZED RACE LOGIC

Race logic encodes information as timing delays. Computation then may happen through the purposeful manipulation of those delays rather than final logic levels, and the functions forming this logic's foundation are MAX, MIN, ADD-CONSTANT, and INHIBIT instead of AND, OR, and NOT.

Under the assumption that smaller delays in rise time encode smaller magnitudes and longer delays encode larger magnitudes, a MAX function should output a logical high only when all of its inputs have arrived (e.g., “gone high”). Therefore, only a single AND gate between its input wires is needed for its implementation. Figure 1(a) displays the symmetric nature of this function; the input that arrives first has to wait for the second one to arrive before the output responds. In the case of MIN, the function outputs a logical high when the first input arrives, and thus a single OR gate is all that is needed—Figure 1(b).

Furthermore, as the arrival time of the rising edge is what encodes information, delaying the $0 \rightarrow 1$ transition by a fixed amount of time is equivalent to constant addition (ADD-CONSTANT). Delaying a signal can be performed in multiple ways depending upon the implementation. In conventional synchronous digital logic, a sequence of flip-flops can be used, as shown in Figure 2. Asynchronous delay elements constructed out of current-starved inverters can provide an alternative, more energy-efficient method for performing the desired delay operation.¹¹

Finally, the INHIBIT function, inspired by the behavior of inhibitory postsynaptic potentials in the neurons of the neocortex,²⁰ works as a nonlinear filter that has two inputs: an inhibiting signal and a data signal (that gets inhibited). If the inhibiting signal arrives first, the output is prevented from ever going high (no state transition), which corresponds to ∞ in the race logic world. On the other hand, if the data signal arrives before or at the same time as the inhibiting signal, the former is allowed to pass through unchanged.

Figure 1. Panels (a) and (b) show the implementation of MAX and MIN functions in race logic. Panel (c) represents an example waveform for $x = 2$ and $y = 4$.

Figure 2. In race logic, adding a constant value k to a variable x is equivalent to delaying the rising edge of x by k clock cycles. Panel (a) shows how this delay can be achieved in conventional synchronous digital logic with the use of a shift-register. Panel (b) shows an example waveform for $x = 2$ and $k = 3$.

Figure 3 shows (a) the symbol used for i inhibiting j , (b) the function's state diagram as a Mealy machine, (c) and (d) two possible CMOS implementations, and (e) a waveform depicting its functionality through two examples. An even more efficient implementation, consisting of only a single PMOS pass gate, is also possible with a little customization.

Together this set of four operations allows us to deliberately engineer “race conditions” in a circuit to perform useful computation. The energy efficiency of the scheme comes from the very low number of “total bit flips” required. Compared to traditional approaches, race logic implementations require fewer wires because each of them can hold a multivalued signal (the delay). Moreover, these wires flip from 0 to 1 at most once through a logic evaluation as the signal-front washes across the circuit. Although not all computations are amenable to such an encoding, those that are have the potential to operate with very little energy. An open question answered in this work is if such a logic is applicable to any general learning or classification task.

3. RETHINKING DECISION TREES

Although monolithic neural networks receive the lion's share of attention from the architecture community with respect to machine learning, decision trees have proven to be incredibly useful in many contexts and a promising solution towards explainable, high-performing AI systems. A decision tree, as its name denotes, creates a hierarchy of decisions, which consists of a set of leaves (labels) and a set of decisions to be made (branches) that lead one to those labels. One normally starts at the root and branches down the tree to find the relevant answer. Thus, classification is reduced to a sequence of binary decisions.

3.1. Reverse race trees

Existing race logic implementations, such as the DNA sequence alignment engine,¹⁰ perform computation by observing the

Figure 3. Panel (a) introduces the symbol that from now on we will use to represent the INHIBIT operator. Panel (b) presents the state diagram of the corresponding Mealy machine. Each transition edge is labeled with the value of inputs i and j and the value of the output. The machine starts in state s_0 , which denotes that input j has not been inhibited by i , whereas state s_1 indicates the opposite; j has been inhibited. Panels (c) and (d) show two possible implementations of the operator in a purely digital context. Finally, the waveform in Panel (e) depicts INHIBIT's functionality through two examples: (1) $i = 3$ and $j = 2$, and (2) $i' = 2$ and $j' = 4$. In this and other examples below, we assume that low to high transitions in the input signals are synchronized with the clock.

relative propagation times of signals injected into the circuit. Following this example, one approach to implement decision trees is by virtually turning them upside down; we can think of them as reverse tree networks that route possible packets from the leaves to the root. Initially, a unique delay-encoded label is assigned to each leaf. These labels then race against one another, and where two of them “meet,” only one is allowed to propagate further. In the end, only the label associated with the “correct” leaf survives—the packet at the output of the network is unchanged, whereas all others get discarded along the way.

The decision tree that we use as a running example is presented in Figure 4(a). Figure 4(b) shows the flow of the four temporally encoded labels in the reverse tree for $x = 2$ and $y = 3$, with the corresponding waveform being illustrated in Figure 4(c). Its race logic implementation is depicted in Figure 4(d). The upper two blocks, colored in red and blue, correspond to the tree’s internal nodes ($x \leq 1$ and $y \leq 1$) and are implemented with the use of one INHIBIT and one MIN operators, whereas the bottom one, colored in yellow, is slightly more complicated as the label coming from its *False* path can take more than one values (either label *C* or label *D*).

Note that when reversing a tree, the *if* clauses in its nodes should be revised. For example, for label $D = 3$, $y \leq 1$ in node $n2$ must be rewritten as $y + 1 < 3$. To implement $y + 1 < 3$, feature y must be delayed by one clock cycle. As already discussed, when we rely on off-the-shelf digital circuits, shift-registers must be used to perform constant addition. However, these clocked components are relatively costly, and ideally, their usage should be constrained.

Figure 4. Panel (a) depicts an example decision tree. Panel (b) shows its “reverse” equivalent as well as the flow of four temporally encoded labels for $x = 2$ and $y = 3$. Panel (c) displays the corresponding waveform for the given example. Panel (d) presents its race logic implementation. Note that the leaf label associated with the *False* branch of a node plays the role of j in the INHIBIT operator, and the node’s attribute (x or y in this example) serves as the inhibiting input i . Given that subtraction and variable addition are not supported by race logic, the attribute routed to an INHIBIT’s controlling input must be adjusted accordingly; for example, $y \leq 1$ must be rewritten as $y + 1 < 3$.

3.2. Flat race trees

An alternative way to look at a decision tree is as a set of independent and parallel rather than sequential decision rules that lead to a final prediction when combined accordingly.¹ Each leaf now can be represented as a logical function of the binary decisions encountered at the nodes on its path to the root. In other words, the tree gets “flattened” and each path from the tree root to a leaf corresponds to a unique combination of attribute test outcomes. The big idea behind the parallel execution of all these independent *if* clauses is shown in Figure 5(a). For example, the leftmost leaf is reached only when both $n0$ and $n1$ return *True*, whereas the output of $n2$ is inconsequential. The order that the outcomes of these conditions reveal to appear does not affect the final decision.

Figure 5(b) presents the implementation of such a flat tree in race logic. In contrast to conventional digital logic approaches, where the size and performance of the circuit realizing the desired threshold functions (each node is a binary decision) are directly related to the resolution of the associated attribute and threshold values, this is not the case here. In race logic, a range of magnitudes can be encoded on a single wire with a single edge. Thus, only one INHIBIT gate is required per tree node.

Moreover, because the decisions related to the various tree paths are mutually exclusive and the maximum threshold value is statically known, the transition from the temporal domain to binary happens seamlessly, without the need for any special circuitry. Figure 5(c) presents the truth table describing the functionality of the decoder that associates nodes’ decisions with one of the leaf labels. Figure 5(d) shows the resulting waveform for $x = 2$ and $y = 3$. In the given example, the maximum threshold value is 2; thus, all node decisions can be safely considered final after 2 clock cycles, and the outcome of $n0$, $n1$, and $n2$ conditions can be read at any time after that.

Figure 5. A decision tree can be viewed as a set of independent decision rules that lead to one and only one leaf when combined accordingly. Hence, the threshold functions corresponding to each node (and lead to these decisions) can be executed in parallel, as shown in Panel (a). Panel (b) depicts the race logic implementation of this flattened decision tree with the use of INHIBITS, where thresholds play the role of the gates’ controlling inputs. Panel (c) presents the truth table that defines the decoder’s functionality. Panel (d) displays the resulting waveform for $x = 2$ and $y = 3$.

4. END-TO-END ARCHITECTURE

4.1. From sensor to delay coded input

Whenever a different encoding is considered, the cost of translation in and out of that encoding should be taken into account. However, race logic is such a natural direct target for sensors that we can instead consider a case where sensing and processing are tightly integrated. Figure 6 presents an end-to-end architecture for temporal processing.

Because sensory input is analog in nature, most sensors begin with a measured voltage or current, which is then converted to a digital output with the use of ADCs. ADCs traditionally return digital binary values; however, given that information in the time domain is now useful for computation, the design of these components can be significantly simplified. For instance, the costly time-to-digital conversion (TDC) in the ADCs is redundant and can be skipped.⁶ Examples of sensing systems that can provide directly time-encoded outputs, without TDCs, include Dynamic Vision Sensors (DVS),⁹ Asynchronous Time-based Image Sensors (ATIS),¹⁶ Time To First Spike (TTFS)¹⁷ and Time-of-Flight (ToF)¹⁴ cameras, and AER (Address Event Representation) Ear² sound sensors.

Given the properties of race logic and the nature of the above-described sensing systems, raw delay-coded data can be directly provided to temporal accelerators to achieve a more efficient sensor-accelerator integration.

4.2. Programmable race trees architecture

Reverse and flat tree encodings provide two ways of implementing decision trees in race logic. The reverse tree idea is of particular interest as it is unlike any other network. Typically, in a network, the packet contents are inert with respect to routing. For example, in the case of sorting networks, the packet values are used for routing, but they also have numerical content external to the network.¹² In reverse trees, the packets are externally assigned values that are symbolic and contain no useful numerical content—in much the same way that numbers in Sudoku are used

Figure 6. End-to-end temporal architecture for in-sensor processing.

symbolically, but not numerically. However, internal to the network, as part of the routing architecture, packet values do take part in numerical operations.

The idea behind the flat tree approach is much simpler. This simplicity results in a more compact and efficient hardware design—fewer shift-registers and a smaller inter-connection network are needed. Due to these reasons, we consider flat trees as our design of choice for the rest of the paper. Figure 7 presents a programmable architecture for the hardwired design of Figure 5.

To ensure any delay-coded threshold value and any input feature can be routed to the necessary nodes of the tree, we use two configurable crossbars. The decoder, which transforms INHIBITs' outputs into a memory address, is built from AND gates and inverters. Note that although INHIBITs are returning temporal signals, the sampled outputs form a typical binary vector. Prior to the next computation, the circuit must be reset.

Figure 8 presents our system architecture for a tree-based ensemble learner. To keep the overhead of the clocked components low, we organize trees into groups and share the same shift-register and buffer (used for the generation of the delay-coded thresholds and the local buffering of the delay-coded input features). Of course, the cost of the crossbars increases with the size of these groups. This trade-off should be further analyzed for maximum efficiency; crossbars can be replaced by any other more efficient configurable routing network without any effect in the system's functionality. Finally, because the data retrieved from memory are in a regular binary encoding, the implementation of the weighted voting scheme is based on typical binary adders. Once the prediction values of all trees have been summed, a comparison between them takes place to find the class with the highest score and determine the system's final "guess."

5. EVALUATION

5.1. Methodology

To evaluate the proposed designs and identify opportunities for further improvement, we created analytical and empirical

Figure 7. Programmable race logic accelerator for a decision tree of depth 2 (before flattening). The length of the shift-register, used for the temporal encoding of thresholds, is defined by the resolution of input features. The memory block shown on the right side of the decoder is useful in the case of tree ensembles, where a weighted voting scheme follows.

power and area models for the basic components of our architecture. We also build a Python-based development flow, as shown in Figure 9, which attaches to the popular scikit-learn library¹⁵ and leverages the power of hardware templates and PyRTL⁴—a Python embedded hardware design language—to generate synthesizable RTL code. More specifically, once the model is trained, the tool analyzes the importance of input features, explores the learners’ performance against lower resolution data, proceeds with votes (content of tree leaves) quantization, generates either a customized hardware design or a configuration file, and performs cross-checking verification. To obtain the desired implementation results, we use open-source tools^{22,24} and a publicly available 14 nm standard

Figure 8. Lower-level diagram of the architecture presented in Figure 6. The shown circuit implements a configurable race logic accelerator for tree-based ensemble learners.

cell library.³ The operational voltage and frequency are 0.55 V and 1,000 MHz, respectively. In our energy and throughput calculations, we assume a 500 MHz clock and, to compensate for the lack of a wire load model, we do not scale our power numbers; under this assumption, each operation consumes twice its nominal energy.¹⁹

5.2. Implementation results

In recent years, an explosion of hardware accelerated machine learning activity has resulted in a wide variety of ASIC architectures, which we can use for comparison. While MNIST is very simple, it is complex enough to demonstrate the principles involved and, because it is the most commonly used dataset in the context of extremely low-power classifiers, facilitates a comparison with state-of-the-art.

An accuracy versus energy comparison between the proposed race trees – represented by green dots – and state-of-the-art low-power classifiers is shown in Figure 10. Moreover, Figure 11 illustrates an accuracy versus energy-delay product comparison, which better elucidates the efficiency gap between race trees and its counterparts. The technique used for training the race trees is gradient boosting. We note that we do not perform any parameter fine-tuning to improve learners’ performance and that race logic does not introduce any sources of inaccuracy.

In more detail, a classifier consisting of 1,000 race trees of depth 6 (before flattening) gets 97.45% accuracy and dissipates 31.35 nJ of energy per prediction. A more efficient solution, consisting of 200 trees of depth 6, achieves a performance of 95.7% with energy numbers as low as 7.8 nJ per prediction. By increasing the trees’ depth to 8, the accuracy increments by 0.5%. This improvement comes at the expense of 16.1 nJ of additional energy per prediction. More results can be found in Table 1.

6. CONCLUSION

If machine learning is the engine, then raw data is the fuel, and most approaches consume a great deal of it. As machine learning techniques continue to find new and compelling applications across a wide range of computing

Figure 9. Overview of the developed toolchain. For the training part, the open-source scikit-learn framework¹⁵ is used. The tool (a) provides the user with the options to quantize input features and/or trees’ votes, (b) supports the automatic generation of race trees circuitry or the configuration of a programmable architecture directly from scikit-learn structures, (c) assists trade-off analysis with the use of analytical architectural models, and (d) facilitates hardware design evaluation through cross-checking with software models.

tasks, the desire to bring that computational power into even our lowest power devices will only continue to grow. Applying these complex algorithms without resorting to the use of significant amounts of energy remains an important challenge and the choice of data representation is an important cross-layer factor that impacts everything from the sensor to the end product of learning. In this paper, we show that the natural relationship between modern decision tree algorithms, new advances in race logic, and the underlying sensors themselves provide new opportunities

Figure 10. Accuracy vs. energy scatter plot for state-of-the-art machine learning accelerators: a,²³ b,⁷ c,¹⁸ d,⁸ e,⁵ f.⁵ For easier comparison, all results have been scaled to 28 nm. Green dots represent race trees.

Figure 11. Accuracy vs. energy-delay product scatter plot for state-of-the-art machine learning accelerators: b,⁷ c,¹⁸ e,⁵ f.⁵ Green dots represent race trees.

for extremely efficient classification. Although it is rare to come across a change that appears simultaneously beneficial across all three of the sensor, learning algorithm, and architecture layers, a delay code seems to be one such rarity. Others have already shown that it is advantageous from an analog perspective to leave signals as they are, and trivially convert them to a race encoding, than to convert them to a pure digital representation. At the algorithm level, little is needed in the way of changes—one must just be mindful of the depth and configuration of the existing decision tree algorithms. At the architecture level, the improvements for these considerations are dramatic both in hardwired and programmable configurations. The resulting system has a shallow critical path and induces exceedingly few bit transitions as the computation propagates through race trees. Example designs that demonstrate this behavior are available at our GitHub repository.^a

Looking forward, the end of traditional CMOS scaling has reenergized the search for novel models of computation and a questioning of digital/analog boundaries. As our ability to deliver useful computation becomes fully bounded by energy consumption, the objective shifts from performing operations at the lowest possible latency to the highest possible efficiency. We believe that this work is an important step in that direction and invites a serious reconsideration of the interfaces between the analog and digital worlds. When the goal is to process sensor data locally, the choice of data representation does not only affect the way computing happens but also dictates the structure and efficiency of the required converters, which often impose a nonnegligible overhead to the system's performance. Although we have not explicitly considered the gains in efficiency possible at the circuit level from avoiding the full transition to binary, there is reason to believe it could be significant.¹³ In fact, there may be many other advantages to temporal models of computation as a more general proposition, perhaps even as a way of encoding more general learning systems inspired by neural-computation²⁰ or enabling emerging circuit technologies.²¹

Acknowledgments

This material is based upon the work supported by the National Science Foundation under Grants No. 1763699, 1740352, 1730309, 1717779, 156393.

Advait Madhavan acknowledges support under the Cooperative Research Agreement between the University of Maryland and the National Institute of Standards and

^a <https://github.com/UCSBarchlab/RaceLogic>

Table 1. Synthesis results for hardwired race trees produced by Yosys²⁴ using a publicly available 14nm standard cell library³.

# Trees	Depth	Inp. res. (bits)	Mem. bits (per vote)	Accuracy	Latency (CCs)	Power (mW)	Area (mm ²)	Freq. (MHz)
1,000	6	8	8	97.48%	273	521	0.46	1,000
1,000	6	4	4	97.45%	33	475	0.45	1,000
200	8	4	4	96.18%	31	384	0.33	1,000
200	6	4	4	95.72%	31	125	0.13	1,000

Technology, Physical Measurement Laboratory. Award 70NANB 14H209 through the University of Maryland.

Dilip Vasudevan was supported by the Advanced Scientific Computing Research (ASCR) program and funded by the U.S. Department of Energy, Office of Science. Lawrence Berkeley National Laboratory operates under Contract No. DE-AC02-05CH11231.

Last but not least, the authors would like to thank James E. Smith, Jennifer Volk, Georgios Michelogiannakis, David Donofrio, John Shalf, and the anonymous reviewers for their helpful comments.

References

1. Bermak, A., Martinez, D. A compact 3d vlsi classifier using bagging threshold network ensembles. *IEEE Trans. Neural Netw. Learn. Syst.* 14, 5 (2003), 1097–1109.
2. Chan, V., Liu, S.C., van Schaik, A. A ear: a matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuits Syst. I Regul. Pap.* 54, 1(2007), 48–59.
3. Chen, S., Wang, Y., Lin, X., Xie, Q., Pedram, M. Performance prediction for multiple-threshold 7nm-FinFET-based circuits operating in multiple voltage regimes using a cross-layer simulation framework. In *2014 SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)* (Millbrae, CA, 2014), 1–2. doi: 10.1109/S3S.2014.7028218. <https://ieeexplore.ieee.org/document/7028218>.
4. Clow, J., Tzimpragos, G., Dangwal, D., Guo, S., McMahan, J., Sherwood, T. A pythonic approach for rapid hardware prototyping and instrumentation. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* (Ghent, 2017), 1–7, doi: 10.23919/FPL.2017.8056860. <https://ieeexplore.ieee.org/document/8056860>.
5. Esser, S.K., Appuswamy, R., Merolla, P., Arthur, J.V., Modha, D.S. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, 2015, 1117–1125. <https://papers.nips.cc/paper/2015/hash/10a5ab2db377feedfdcaab192ead4ac0e-Abstract.html>.
6. Guo, X., Qi, X., Harris, J.G. A time-to-first-spike cmos image sensor. *IEEE Sens. J.* 7, 8(2007), 1165–1175.
7. Kim, J.K., Knag, P., Chen, T., Zhang, Z. A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning. In *2015 Symposium on VLSI Circuits (VLSI Circuits)* (Kyoto, 2015), C50–C51, doi: 10.1109/VLSI.2015.7231323. <https://ieeexplore.ieee.org/document/7231323>.
8. Kung, J., Kim, D., Mukhopadhyay, S. A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015, 85–90.
9. Lichtsteiner, P., Posch, C., Delbruck, T. A 128×128 120db 15μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-St. Circ.* 43, 2(2008), 566–576.
10. Madhavan, A., Sherwood, T., Strukov, D. Race logic: a hardware acceleration for dynamic programming algorithms. *Comput. Architect. News* 42, 3(2014), 517–528.
11. Madhavan, A., Sherwood, T., Strukov, D. Energy efficient computation with asynchronous races. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)* (Austin, TX, 2016), 1–6. doi: 10.1145/2897937.2898019. <https://ieeexplore.ieee.org/document/7544351>.
12. Najafi, M.H., Lilja, D.J., Riedel, M., Bazargan, K. Power and area efficient sorting networks using unary processing. In *2017 IEEE International Conference on Computer Design (ICCD)* (Boston, MA, 2017), 125–128. doi: 10.1109/ICCD.2017.27. <https://ieeexplore.ieee.org/document/8119200>.
13. Naraghi, S. Time-based analog to digital converters, 2009. <https://oatd.org/oatd/record?record=handle%5C%3A2027.42%5C%2F64787>.
14. Niclass, C., Soga, M., Matsubara, H., Kato, S., Kagami, M. A 100-m range 10-frame/s 340×96-pixel time-of-flight depth sensor in 0.18-μm cmos. *IEEE J. Solid-St. Circ.* 48, 2(2013), 559–572.
15. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.
16. Posch, C., Matolin, D., Wohlgenannt, R., Hofstätter, M., Schöen, P., Litzberger, M., Bauer, D., Garn, H. Biomimetic frame-free hdr camera with event-driven pwm image/video sensor and full-custom address-event processor. In *2010 IEEE on Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2010, 254–257.
17. Qi, X., Guo, X., Harris, J.G. A time-to-first spike CMOS imager. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)* (Vancouver, BC, 2004). IV–824. doi: 10.1109/ISCAS.2004.1329131. <https://ieeexplore.ieee.org/document/1329131>.
18. Reagen, B., Whatmough, P., Adolf, R., Rama, S., Lee, H., Lee, S.K., Hernández-Lobato, J.M., Wei, G.Y., Brooks, D. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)* (2016). IEEE Press, 267–278. doi: 10.1109/ISCA.2016.32. <https://dl.acm.org/doi/10.1145/3007787.3001165>.
19. Shalf, J., Dosanjh, S., Morrison, J. Exascale computing technology challenges. In *High Performance Computing for Computational Science – VECPAR 2010*. J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, 1–25.
20. Smith, J. Space-time algebra: A model for neocortical computation. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (Los Angeles, CA, 2018), 289–300. doi: 10.1109/ISCA.2018.00033. <https://ieeexplore.ieee.org/document/8416835>.
21. Tzimpragos, G., Vasudevan, D., Tsiskaridze, N., Michelogiannakis, G., Madhavan, A., Volk, J., Shalf, J., Sherwood, T. A computational temporal logic for superconducting accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20*. Association for Computing Machinery, New York, NY, USA, 2020, 435–448.
22. Vasudevan, D., Butko, A., Michelogiannakis, G., Donofrio, D., Shalf, J. Towards an integrated strategy to preserve digital computing performance scaling using emerging technologies. In *High Performance Computing*. J. M. Kunkel, R. Yokota, M. Tauber, and J. Shalf, eds. Springer International Publishing, Cham, 2017, 115–123.
23. Whatmough, P.N., Lee, S.K., Lee, H., Rama, S., Brooks, D., Wei, G. 14.3 A 28nm SoC with a 1.2 GHz 568nJ/prediction sparse deep-neural-network engine with > 0.1 timing error rate tolerance for IoT applications. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. (San Francisco, CA, 2017), 242–243. doi: 10.1109/ISSCC.2017.7870351. <https://ieeexplore.ieee.org/document/7870351>.
24. Wolf, C., Glaser, J. Yosys—A free verilog synthesis suite. In *Proceedings of Austrochip* (2013). <https://www.semanticscholar.org/paper/Yosys-A-Free-Verilog-Synthesis-Suite-Wolf-Glaser/65b4a1136599d74ada27ce5226f02dda06d2ccda>.

Georgios Tzimpragos (gtzimpragos@cs.ucsb.edu), Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA, USA.

Advait Madhavan (advait.madhavan@nist.gov), Physical Measurements Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, USA.

Dilip Vasudevan (dilipv@lbl.gov), Lawrence Berkeley National Laboratory, Berkeley, CA, USA.

Dmitri Strukov (strukov@ece.ucsb.edu), Electrical and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA, USA.

Timothy Sherwood (sherwood@cs.ucsb.edu), Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA, USA.

This work is licensed under a Creative Commons Attribution International 4.0 License.