Intermittent Deep Neural Network Inference

Graham Gobieski
Carnegie Mellon University
gobieski@cmu.edu

Nathan Beckmann
Carnegie Mellon University
beckmann@cs.cmu.edu

Brandon Lucia
Carnegie Mellon University
blucia@cmu.edu

1 INTRODUCTION

The maturation of energy-harvesting technology has enabled new classes of sophisticated, batteryless systems that will drive the next wave of Internet of Things (IoT) applications. These applications require intelligence at the edge and even in the sensor node, e.g., allowing systems to immediately interpret sensed data and make judicious use of scarce bandwidth. However, inference on energy-harvesting devices presents challenges currently unexplored, namely that energy-harvesting devices are severely resource-constrained and operate *intermittently* only when energy is available. Typical systems run at a few MHz, have a few hundred KBs of memory and consume power under 1 mW when active [8, 22]. In comparison, the most energy efficient DNN inference accelerators consume hundreds of mW [5, 6, 9, 10].

An energy harvester's power output is usually insufficient for continuous operation and energy-harvesting systems operate *intermittently*. An intermittently operating device slowly buffers energy in a capacitor and operates in a short burst when the capacitor is sufficiently charged. Software executes in short bursts of active computation (e.g., 100,000 cycles) interrupted by power failures. After power failure, the device recharges (e.g., over one second) and continues executing [2, 3, 7, 8, 12, 17, 19, 23]. Power interruptions require an intermittent device to flush state to non-volatile memory, or lose the state on power failure. Guaranteeing correctness and forward progress in an intermittent execution is a challenge that recent work has begun to address [7, 17, 19].

Current energy-harvesting systems are several orders-of-magnitude too inefficient for practical DNN inference. Continuous sensor classification requires inferences in seconds (or less) to justify local inference vs. cloud offloading. Our initial explorations revealed that a naïve implementation of DNN inference requires more memory than is available on typical energy-harvesting systems [8, 22], even for small networks. Moreover, even if they did fit, a single inference would take more than an hour, precluding execution in (the milliseconds of) an intermittent execution period.

This paper presents *Software-Only Neural Intermittent Computing* (SONIC), a software system that demonstrates the viability of DNN inference on energy-harvesting systems, addressing both efficiency and correctness. SONIC exploits the regular structure of inference to guarantee correct intermittent execution more efficiently than general-purpose intermittent execution frameworks.

We first manipulate recent DNNs to fit into the tiny memory of our energy-harvesting device through aggressive compression [11], focusing on *OK Google*, Google's keyword-spotting network [21], and *LeNet*, a digit recognition network [15]. To this baseline compressed network, SONIC adds DNN-specialized software support for efficient and correct intermittent operation. Unlike existing intermittent execution frameworks, SONIC eliminates wasteful, repeated work and imposes little overhead. SONIC demonstrates the viability of batteryless DNN inference: running on radio-wave-harvesting hardware, SONIC ensures correctness and outperforms a recent general-purpose intermittent framework by 2×.

The remainder of the paper describes SONIC in detail, concluding with a roadmap for future improvements in intermittent DNN inference efficiency and capability.

SysML, February, 2018, Stanford, California, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 SONIC DESIGN AND IMPLEMENTATION

We build SONIC, an intermittent DNN inference framework, and use it to implement the LeNet [15] and OK Google [21] on a Texas Instruments MSP430 microcontroller (MCU). SONIC overcomes two challenges. First, the MCU has little memory and poor performance requiring SONIC to greatly reduce the memory and compute requirements of DNN inference. Second, SONIC must guarantee correct intermittent execution with minimal overhead. Prior work on intermittent execution focused on correctness for general-purpose code, not considering specialized performance optimization. SONIC is the first system to specialize intermittence support to optimize for a particular domain. We employ known techniques [11] to fit the DNN's parameters on the device; this paper focuses on how SONIC efficiently guarantees correct intermittent execution.

Fitting DNN inference on a low-power microcontroller. To fit DNN inference on the MSP430, we leverage Deep Compression [11]. We compress fully connected and convolutional layers, pruning weights below a threshold, and use a sparse matrix representation similar to alternating value-index format. We achieve modest accuracy loss (≈4%) and high compression, reducing the memory footprint by up to 34× and the number of multiply-accumulate operations by up to 7.5×. We use fixed-point math, reducing memory by half and avoiding the MSP430's floating-point emulation. We used these optimizations in all evaluated implementations.

Efficient intermittent execution via flexibly sized tasks. SONIC includes new optimizations that improve intermittent execution performance. We implemented a baseline intermittent DNN inference implementation using Alpaca [19], a recent intermittent execution framework. Alpaca programs are split into fixed-size *tasks* that a compiler makes idempotent by buffering values read then written (i.e., WAR values) in non-volatile memory until task completion. After power failure, Alpaca resumes at the start of the failed task. By guaranteeing idempotence, Alpaca ensures correct intermittent execution. However, fixed-size tasks waste work by discarding results from partial task executions. Tasks must be short to complete in a single intermittent charge cycle, but smaller tasks have higher setup/teardown overheads.

SONIC addresses these issues by flexibly sizing tasks to grow and shrink to fit the system's energy budget. SONIC's flexibility relies on the predictable structure of DNN computation, specifically repeated sparse matrix-multiply loops. SONIC persistently

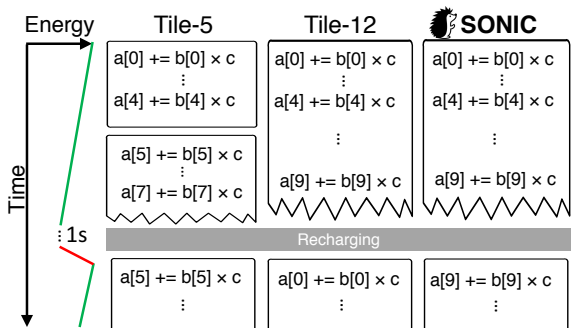
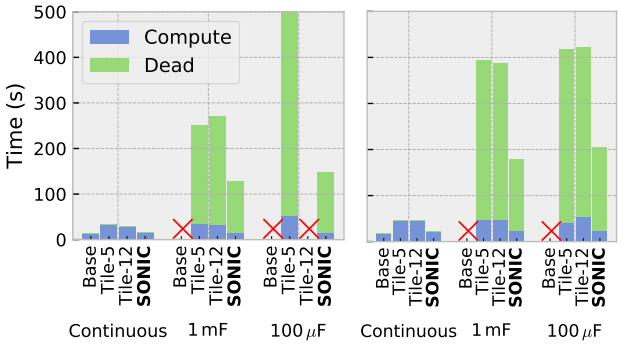


Figure 1: Comparing SONIC to Alpaca's fixed-size tasks [19]. SONIC's flexible tasks guarantee forward progress with varied capacitor sizes with little overhead and wasted work.



(a) LeNet **(b) Google Keyword Spotting**

Figure 2: Execution time for one inference on a TI MSP430. We compare SONIC to a baseline implementation (Base) and two intermittent variants using Alpaca [19] (Tile-5 and Tile-12). We run on continuous power and on harvested energy using two capacitor sizes (1 mF, and 100 µF). The baseline system *does not make progress* on harvested energy. SONIC is comparable to Base on continuous power and more than twice as fast as the next-best intermittent variant.

tracks the completion of each of a task's loop iterations, resuming the last uncompleted iteration after power failure. At most, the computation loses one loop iteration on power failure. SONIC guarantees forward progress when a single DNN inner-loop iteration always completes within a charge cycle. Unlike prior checkpointing systems for intermittent execution [12, 18, 23], SONIC tracks application-level progress and uses Alpaca tasks to avoid checkpointing imposing low overheads.

Figure 1 compares SONIC against the Alpaca implementation using two task sizes, Tile-5 and Tile-12. Time goes top-to-bottom. The capacitor energy is shown on the left; power fails once, requiring a one second recharge. Tile-5's first task completes 5 iterations, and power fails during the second task. Tile-5 resumes in the second task, repeating 3 loop iterations from before the power failure. Tile-12 has larger tasks that better amortize task overheads, and complete more loop iterations than Tile-5 before power fails. However, Tile-12's task does not complete. Tile-12 repeatedly resumes at the start of the first task and *does not make forward progress*. SONIC tracks iteration-by-iteration progress, resuming execution where it left off and repeating at most one loop iteration. SONIC amortizes task overheads like Tile-12, allowing it to complete more work before failing, while guaranteeing progress.

Handling idempotence violations in DNN inference. An intermittent system demands different optimizations than a conventional one. Conventional systems often order (e.g.,) convolution loops to maximize cache locality. Unfortunately, loop ordering complicates intermittent execution by updating matrix elements in place in a single task, which violates idempotence. Re-executing such non-idempotent tasks leads to incorrect results. To ensure idempotence in flexibly sized tasks, SONIC re-orders loops so that each element is updated only once per outer loop (i.e., task). SONIC alternates between two copies of activations being computed, making a single update to one copy before switching to other. SONIC's approach is practical because most energy-harvesting MCUs lack caches [8, 22] with one cycle memory latency, and loop ordering yields no speedup.

Evaluation. We implemented SONIC on a radio-wave-harvesting system with a Texas Instruments MSP430FR5994 powered by a PowerCast P2110 1m away from its receiver and 1 mF and 100 µF capacitors. We evaluated SONIC with LeNet [15] and OK Google [21], comparing to Alpaca implementations with two task sizes, and a conventionally optimized baseline.

Figure 2 shows that SONIC has similar performance to the baseline optimized system and outperforms fixed-size tasks by 2×. More significantly, the baseline does not make forward progress on harvested energy, and neither does the Tile-12 fixed task implementation on LeNet with a 100 µF capacitor. SONIC's flexible tasks complete as much work as possible within each charge cycle and guarantee forward progress for both capacitors.

3 CONCLUSION AND FUTURE DIRECTIONS

SONIC demonstrates the feasibility of DNN inference on energy-harvesting systems, opening the door to new classes of IoT applications. We have shown that there is ample room for optimization of existing intermittent systems for DNN inference, and that the optimizations needed for efficient intermittent DNN inference differ significantly from conventional systems. However, there remains headroom for performance improvement to make SONIC appealing for many IoT applications. Our ongoing work is to develop software and hardware techniques to improve intermittent DNN inference efficiency by orders of magnitude. We will extend our evalution of SONIC with end-to-end prototypes [16] and other DNN models [14]. We will investigate other software techniques to reduce memory and compute requirements [1, 4, 20], and approximation techniques, e.g., *skipping* partially executed task work.

One of the biggest opportunities is to improve the architecture. Current ultra-low-power microcontrollers poorly suit DNN inference, spending little of their energy on computation (i.e., MACs) and most on control and register accesses [13]. We envision a specialized, ultra-low-power, data-parallel architecture for DNN inference with native intermittence support, incorporating SONIC's techniques into the architecture. Unlike recent DNN accelerators, energy-harvesting systems need only optimize for energy efficiency, not performance, requiring research in new directions.

REFERENCES

[1] Jose M. Alvarez and Mathieu Salzmann. 2017. Compression-aware Training of Deep Networks. In *Neural information processing systems*.

[2] D. Balsamo, A. Weddell, A. Das, A. Arreola, D. Brunelli, B. Al-Hashimi, G. Merrett, and L. Benini. 2016. Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP, 99 (2016), 1–1. https://doi.org/10.1109/TCAD.2016.2547919

[3] D. Balsamo, A.S. Weddell, G.V. Merrett, B.M. Al-Hashimi, D. Brunelli, and L. Benini. 2014. Hibernus: Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems. *Embedded Systems Letters, IEEE* PP, 99 (2014), 1–1. https://doi.org/10.1109/LES.2014.2371494

[4] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 176–189.

[5] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proc. of the 19th intl. conf. on Architectural Support for Programming Languages and Operating Systems*.

[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proc. of the 43rd annual Intl. Symp. on Computer Architecture (Proc. ISCA-43)*.

[7] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*.

[8] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *ASPLOS*.

[9] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proc. of the 42nd annual Intl. Symp. on Computer Architecture (Proc. ISCA-42)*.

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pdream, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proc. of the 43rd annual Intl. Symp. on Computer Architecture (Proc. ISCA-43)*.

[11] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding. In *Proc. of the 5th Intl. Conf. on Learning Representationas (Proc. ICLR'16)*.

[12] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 228–240. https://doi.org/10.1145/3079856.3080238

[13] Mark Horowitz. 2014. Computing's energy problem (and what we can do about it). In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 10–14.

[14] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[15] Yann Le Cun, LD Jackel, B Boser, JS Denker, HP Graf, I Guyon, D Henderson, RE Howard, and W Hubbard. 1989. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine* 27, 11 (1989), 41–46.

[16] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho. 2017. Human activity recognition from accelerometer data using Convolutional Neural Network. In *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*.

[17] Brandon Lucia and Benjamin Ransford. 2015. A Simpler, Safer Programming and Execution Model for Intermittent Systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2015)*. ACM, New York, NY, USA, 575–585. https://doi.org/10.1145/2737924.2737978

[18] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture exploration for ambient energy harvesting nonvolatile processors. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 526–537.

[19] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent Execution without Checkpoints. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. ACM, Vancouver, BC, Canada.

[20] Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. 2015. Compressing deep neural networks using a rank-constrained topology.. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[21] Tara N Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[22] Alanson P. Sample, Daniel J. Yeager, Pauline S. Powledge, Alexander V. Mamishev, and Joshua R. Smith. 2008. Design of an RFID-Based Battery-Free Programmable Sensing Platform. *IEEE Transactions on Instrumentation and Measurement* 57, 11 (Nov. 2008), 2608–2615.

[23] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In *Proceedings of OSDI'16: 12th USENIX Symposium on Operating Systems Design and Implementation*. 17.