The What's Next Computing Architecture

Karthik Ganesan
University of Toronto
karthik.ganesan@mail.utoronto.ca

Joshua San Miguel
University of Wisconsin-Madison
jsanmiguel@wisc.edu

Natalie Enright Jerger
University of Toronto
enright@ece.utoronto.ca

Abstract

Energy-harvesting devices operate under extremely tight energy constraints. Ensuring forward progress under frequent power outages is paramount. Applications running on these devices are typically amenable to approximation. We propose **What's Next**, two anytime approximation techniques for energy harvesting systems: subword pipelining and skim points. By providing an approximate result sooner and moving to the next input upon returning from a power outage, we see up to 1.6× speedup with less than 3% error for an image processing kernel.

1 Introduction

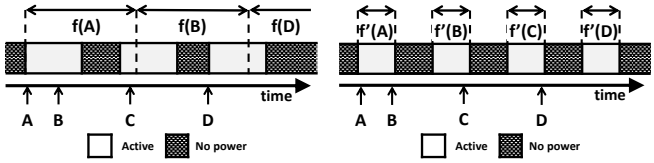
Energy-harvesting (EH) devices are an emerging class of embedded systems that eschew batteries by running directly off of energy gathered from the environment. However, harvested power sources only produce sufficient energy to run these devices for up to a few milliseconds at a time, resulting in frequent power outages [6]. As a result, energy-harvesting systems must often spread processing a single input over multiple power cycles. Thus, when new input data arrives, the system must choose to either continue processing old data or discard it and move on to processing new data.

What's Next (WN) provides a partial answer when energy is scarce but offers the flexibility to refine that answer if more energy is available.[1] To this end, we extend the anytime automaton model [7] which provides early approximate versions of an application's output; as the application continues to run, it refines the output towards the precise result.

Consider the application in Fig. 1. In a conventional EH system (Fig. 1a), the device resumes processing each input after a power outage until the final precise result is achieved. As a result, inputs *C* and *D* arrive while the device is still processing input *B*, forcing the system to drop one of those inputs. Using WN(Fig. 1b), we produce the best possible result for input *A* prior to the first power outage and then begin processing the next available input when power returns. As a result, we start processing input *C* once power returns and hence achieve greater forward progress across all input samples. The output quality is best-effort (i.e., we take the approximate result *as-is* when forced to power down) but forward progress improves substantially. As many energy-harvesting

[1] "I understood the point... when I ask what's next, it means I'm ready to move on to other things, so what's next?" – Jed Bartlet, The West Wing

WAX'18, March 25, 2018, Williamsburg, VA, USA
2018.



(a) conventional **(b)** What's Next

Figure 1. Application running on an EH system.

(a) baseline (100% runtime) **(b)** baseline (50% runtime) **(c)** WN (50% runtime)

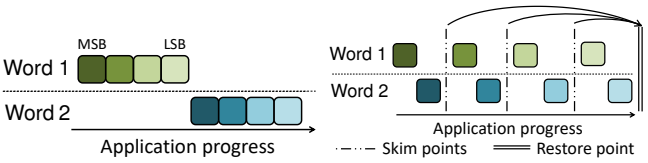
Figure 2. 2D Convolution output: baseline and What's Next applications are naturally amenable to approximation, this trade-off is an effective and appropriate one.

2 Motivation

This section presents an example to demonstrate the drawbacks of current EH systems that What's Next overcomes. EH systems frequently do not have sufficient energy to fully process input data in a single power on cycle. When processing is interrupted, the result produced thus far is likely to be incomplete. In such situations, an approximate result can serve as a facsimile for the complete precise output. To demonstrate this, we compare the output of an image processing kernel from WN vs. a precise implementation (Fig. 2). Fig. 2b shows the output when the device running the precise implementation losses power halfway through processing the image. Clearly the result is incomplete and processing of the image must continue during the next power on cycle to produce an acceptable output. Yet with anytime processing, with the same total power-on time, we can compute a result for the entire image (Fig. 2c). This result is both complete and of acceptable quality. If a higher quality output is needed, the system can run for longer, yielding greater quality over time.

3 What's Next

We propose the What's Next Intermittent Computing Architecture, which dynamically prioritizes the computation of the most significant subwords first. Less significant subwords are



(a) Conventional **(b)** Subword pipelining

Figure 3. Anytime subword pipelining.

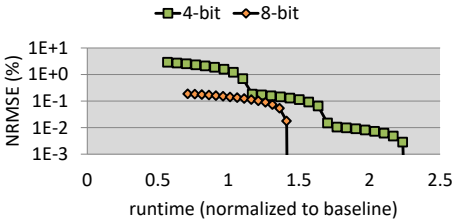


Figure 4. Runtime-quality trade-off

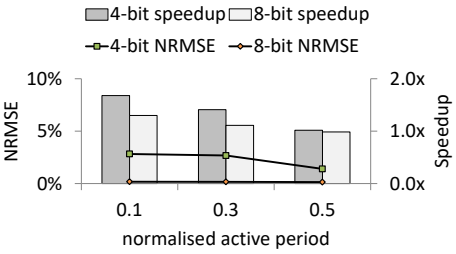


Figure 5. Median speedup and error while varying the active power period.

processed incrementally, improving quality over time. Our first technique, **subword pipelining (SWP)** breaks long-latency operations (e.g., fixed-point multiplication) into subword stages, starting with the most significant subword. The ultra-low power processors used in energy harvesting [3] such as the ARM M0+, do not have a hardware multiplier [1]. A 32×32 multiply is carried out iteratively, taking a total of 32 cycles to complete. Our approach splits each data word into smaller subwords to be processed from most to least significant. Instead of processing each data word to completion as shown in Fig. 3a (data words 1 & 2), the most significant subwords from multiple data elements are processed in a pipelined fashion (Fig. 3b). Upon encountering a power outage, our second technique **skim points**, allows the system to skip processing the rest of the current input data and move onto processing new data. Otherwise, the entire pipeline runs to completion and is guaranteed to produce the precise result.

4 Evaluation

We use a cycle-accurate simulator [2] for the ARM M0+ CPU [1], running at 24 MHz [3]. Our test application is 2D Convolution which applies a 9×9 Gaussian filter on a 128×128 grayscale image. Fig. 4 shows the runtime-quality curve when applying SWP for 4-bit and 8-bit subwords. Runtime (x-axis) is normalized to the conventional precise execution. The y-axis shows the Normalised Root Mean Square Error (NRMSE) in output if the application were to be halted by a power outage at that moment. Quality improves as the application progresses, until the final precise output is reached. An approximate output is available early, allowing the application to be terminated sooner by a power outage. Subword granularity controls the trade-off between how early an approximate output is available and how late the precise output is obtained. With smaller granularities (e.g., 4 bits), it takes longer for the application to run to completion (i.e., produce the precise output). The advantage, however, is that an approximate output is available earlier. WN incurs runtime overhead to reach the precise output, due to the presence of instructions that are not amenable to subword pipelining and must be re-executed due to the iterative nature of WN (e.g., conditional branches, address computation). On devices with frequent power outages, the benefit of producing outputs early and enabling interruptible execution outweighs the cost of longer runtime to the precise result.

Next, we evaluate WN in the presence of frequent power outages on a non-volatile processor based system, commonly used in the energy-harvesting space [4, 5]. Fig. 5 shows the median speedup and error from 13 runs of the application. We vary the *active power period* (i.e., power outage frequency), which is the amount of time that the processor can run for (i.e., processing time between power outages). The active period is normalized to the baseline precise execution time of the application on a single input. For example, an active power period of 0.5 implies that the application is likely to process only half of its current input before losing power, while an active period of 0.1 means the application is interrupted roughly ten times when processing one input. In the presence of frequent power outages (i.e., very low active period), WN achieves a speedup of 1.3× (8-bit) and 1.6× (4-bit) with less than 3% error.

5 Conclusion and Future Work

In this paper, we present What's Next, a novel technique for increasing forward progress on energy-harvesting systems. Specifically, we detail subword pipelining which allows for computations that dynamically prioritize the most significant bits first and skim points, which allows for computation to be skipped when a approximate result is available. We see speedups of upto 1.6× with errors of less than 3%.

As subwords are 4-bit or 8-bit in size, redundant calculations are more likely compared to using the full 32-bit word. Techniques such as memoization can be used to eliminate these and further increase the speedup offered by SWP. WN can also be extended to support low-latency operations by vectorizing them so that subwords from different words can be processed in parallel to yield a speedup.

References

[1] ARM Technologies Ltd. [n. d.]. *ARM Cortex M0+ Technical Reference Manual.* ARM Technologies Ltd. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0484b/CHDCICDF.html.

[2] M. Hicks. 2016. *Thumbulator: Cycle accurate ARMv6-m instruction set simulator.* https://github.com/impedimentToProgress/thumbulator.

[3] Matthew Hicks. 2017. Clank: Architectural Support for Intermittent Computation. In *Proceedings of the International Symposium on Computer Architecture.*

[4] Kaisheng Ma, Xueqing Li, Jinyang Li, Yongpan Liu Liu, Yuan Xie, Jack Sampson, Mahmut Taylan Kandemir, and Vijaykrishnan Narayanan. 2017. Incidental Computing on IoT Nonvolatile Processors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture.*

[5] 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 *Proceedings of the International Symposium on High Performance Computer Architecture.*

[6] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2011. Mementos: system support for long-running computation on RFID-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems.*

[7] Joshua San Miguel and Natalie Enright Jerger. 2016. The Anytime Automaton. In *Proceedings of the International Symposium on Computer Architecture.*