Evaluating Tradeoffs in Granularity and Overheads in Supporting Nonvolatile Execution Semantics

Kaisheng Ma*, Minli Julie Liao*, Xueqing Li, Zhixuan Huan, Jack Sampson
Computer Science and Engineering Department, Penn State
{kxm505,mjl5868,lixueq,zxh48,sampson}@cse.psu.edu
*authors contributed equally to this work

Abstract— While pausing and resuming execution using nonvolatile storage has long been possible, nonvolatile processing as a fundamental paradigm has only recently been made practical by technology advances allowing on-chip nonvolatile memories. However, even with on-chip nonvolatile storage, the granularity of ensured forward progress that a nonvolatile processor offers can still vary widely from cycle-level guarantees to software-defined checkpoints spanning potentially significant quantities of execution. Choice of supported granularity influences not only the hardware overheads, but also the complexity of avoiding potential inconsistencies between architectural and microarchitectural state in realistic memory systems. In this paper, we examine the overheads, in terms of both complexity and efficiency, for non-volatile processor designs with different granularity of forward progress guarantees.

1. INTRODUCTION

With the development of nonvolatile processors (*NVP*s), energy harvesting is emerging as an increasingly attractive means for powering the internet of things (IoT) [1, 2, 3]. NVPs can endure the power emergencies caused by unstable input power supplies by leveraging nonvolatile memory elements to pause and resume execution without loss of state. The use of distributed nonvolatile flip-flops or integrated memories [4] in NVPs allows this to happen at very short timescales, potentially even a handful of processor cycles, allowing systems using these processors to operate without large energy storage devices.

However, while NVPs can perform useful work in power environments where volatile processors are unlikely to complete their assigned tasks, NVPs are encumbered by overheads, compared to a volatile processor, during periods when power is stable. The degree of overhead that is entailed is closely related to the guarantees of forward progress vs. rollback that a particular NVP offers. A software-defined NVP, for instance, only performing backups to the on-chip non-volatile storage at coarse, user-defined points in a program, will incur limited overheads due to backup costs during uninterrupted execution. However, such a software-directed system can take a long time to perform a backup or restore, and could conceivably lose significant amounts of execution if the checkpoints are far apart and power emergencies are frequent. Moreover, software-directed systems can only reason about user-visible state, may require flush operations to clear microarchitectural structures, and may need substantial energy reserves to perform a backup operation, as the amount of changed data is poorly bounded. On the opposite side of the spectrum, NVPs that handle backup and restore transparently as part of the microarchitecture can operate with minimal reserves, as total state changes between backups can be strongly bounded, but, lacking access to program level semantics, are likely to conservatively persist all executed instructions and memory updates to avoid consistency errors during restore phases.

Deciding on what granularity of ensuring non-volatile progress is appropriate for a given deployment further depends on the expected characteristics of the power input that the system will experience as well as the demands of the IoT tasks that the system performs. In this paper, we examine the affinities between different power environments and different granular approaches for reasoning about durability in an NVP. We then consider the impacts of these affinities on best practices for NVP design, and discuss how to construct NVPs that can gracefully trade among persistence overheads, lost work minimization, and implementation cost considerations in integrating the non-volatile memory elements.

Our work makes the following contributions:

- We analyze piezo, solar and RF WiFi as harvesting sources. Using our lowest-threshold NVP model and simplest backup policy, we show the frequency of interruptions for each power source and describe the expected features in duration and volatility of periods where power supports execution. Specifically, we investigate these power-on periods from the perspective of how much energy is available for backup beyond that needed for basic operation in terms of the number of writes that could be completed to nonvolatile storage.
- We explore functional and basic-block level granularity for persistence in several kernels used in NVPs to contrast with instruction and sub-instruction level hardware backup schemes. We highlight the size diversity of checkpoint costs for software directed schemes. Notably, even the small kernels examined contain regions with checkpoint costs that can vary by an order of magnitude from other regions. However, at the basic block level, the distribution of checkpoint costs is much more modest, and the effective cost lowered even further by a large number of the basic blocks being executed being naturally idempotent.
- Considering the granularity patterns in both codes and power profiles, we provide suggestions for which combinations of NVP design, backup policy, and power source are sensible. We show that the diversity of costs and high variation both within and between benchmarks and power environments motivates dynamic, best effort schemes that utilize hardware capable of executing at the finest granularity of persistence, but that aim to attempt to operate at coarser, software defined granularity whenever possible.

2. BACKGROUND AND RELATED WORK

Energy-Harvesting IoT Systems Energy harvesting IoT systems are typically composed of energy harvester, NVP, sensor, ADC, etc. components[1]. Common sources of harvested energy are solar, piezo, RF and thermal energy [1]. Systems may opt to use energy harvesting in addition to or in lieu of battery power, and for some or all of the component power needs. Motivations for battery-less harvesting-based operation include form factor constraints precluding sufficient batteries for intended lifetimes, biocompatibility and safety concerns, and deployment scenarios where recharging is impractical. Similar factors constrain what little energy storage

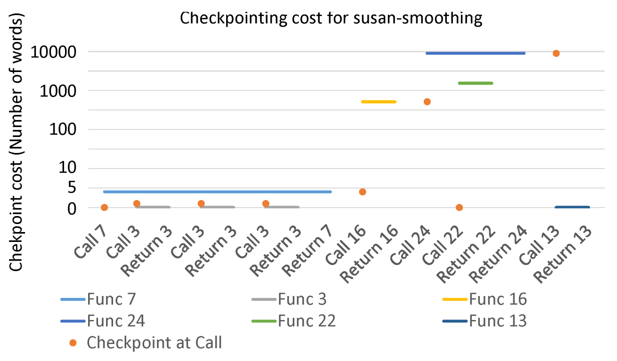

Figure 1: Checkpointing cost of different schemes for susan

capabilities such systems do have, usually either capacitors or supercapacitors. In particular, a key feature of systems relying on harvested energy is that they will experience unscheduled power interruptions. While nonvolatile processing aims to limit the loss of execution due to these power interruptions, NVPs must do so by either turning unscheduled power interruptions into unscheduled backups, which are resource constrained by available energy storage, or by conservatively scheduling backups during periods of more abundant power.

Nonvolatile Processing Architectures Ma et al. explored various architectural design issues for hardware-enforced NVPs in batteryless ambient energy harvesting systems [1, 2] and showed that, While simpler processor architectures have advantages in quick backup and recovery, they can suffer from overall performance loss due to an inability to capitalize on periods of higher than average power. Later studies considered dynamic heterogeneous architectures to reap the benefits of both simple and more complex NVP datapaths [5, 3] utilizing a dynamic prediction unit to match the power history to a pipeline configuration. Other efforts have explored NVP semantics at the language and compiler levels in order to improve commodity processor-plus-NVM platforms [6].

Integrated NVM Checkpointing Nonvolatile memory has long be used for checkpointing. Dong et al. proposed leveraging PCRAM for checkpointing in massively parallel processing systems [7]. The proposed scheme reduces performance overheads to 3% and improves the efficiency of incremental and background checkpointing, focusing on memory state. Checkpoint aware caches have been proposed using combinations of SRAM and STT-RAM [8]. At the intra-pipeline level, to address soft error rates, Swaminathan et al. used STT-RAM to snapshot pipeline structures in the processor [9] and correct detected errors. Specific to checkpointing in nonvolatile processors, it has been shown that inconsistency could arise if checkpointing and resuming is not carefully handled in a software-directed NVP [10]. The proposed checkpointing algorithm is designed to remove consistency-relate errors.

3. WORKLOAD ANALYSIS

In this section, we analyze a typical set of tasks for an NVP from the perspective of understanding the commit requirements for software-directed checkpointing. We investigate software directed commit points at function-oriented and basic-block oriented granularities. For the latter, we focus on identifying idempotent blocks that cannot cause consistency issues and therefore may not need to consume backup resources. For function-based checkpointing, we discuss multiple variants.

Figure 1 shows the checkpoint cost of different function-level checkpointing schemes for Mibench [11] benchmark *Susan*-(**c**orners, **s**moothing, **e**dges), namely checkpointing at function calls and check-

```
// BlockSize: load first and then store each iteration
for ( BlockSize = 2; BlockSize <= NumSamples; BlockSize <<= 1 ) {          // LoopA
// delta_angle: only used to calculate sm2, sm1, cm2, cm1
  double delta_angle = angle_numerator / (double)BlockSize;
// sm2, sm1, cm2, cm1, w: store first, assigned value each iteration
  double sm2 = sin ( -2 * delta_angle );
  double sm1 = sin ( -delta_angle );
  double cm2 = cos ( -2 * delta_angle );
  double cm1 = cos ( -delta_angle );
  double w = 2 * cm1;
// ar, ai: store first in subloop
  double ar[3], ai[3];
// temp: Unused
  double temp;
  // i: load first and then store each iteration
  for ( i=0; i < NumSamples; i += BlockSize ) {                             // LoopB
    // ar[2], ar[1], ai[2], ai[1]: store first, assign value each iteration
    ar[2] = cm2;
    ar[1] = cm1;
    ai[2] = sm2;
    ai[1] = sm1;
    // j, n: load first and then store each iteration
    for ( j=i, n=0; n < BlockEnd; j++, n++ ) {                              // LoopC
      // ar[0], ai[0]: store first, assign value each iteration
      // ar[1], ar[2], ai[1], ai[2]: load first and then store each iteration
      ar[0] = w*ar[1] - ar[2];
      ar[2] = ar[1];
      ar[1] = ar[0];
      ai[0] = w*ai[1] - ai[2];
      ai[2] = ai[1];
      ai[1] = ai[0];
      // k, tr, ti: store first, assign value each iteration, unused outside the loop
      // RealOut[k], ImagOut[k]: load first and then store each iteration
      k = j + BlockEnd;
      tr = ar[0]*RealOut[k] - ai[0]*ImagOut[k];
      ti = ar[0]*ImagOut[k] + ai[0]*RealOut[k];
      // RealOut[j], ImagOut[j]: load first and then store each iteration
      RealOut[k] = RealOut[j] - tr;
      ImagOut[k] = ImagOut[j] - ti;
      RealOut[j] += tr;
      ImagOut[j] += ti;
    }
  }
  // BlockEnd: load first (in LoopC) and then store each iteration
  BlockEnd = BlockSize;
}
```

Figure 2: Commit sets within versus after loop iterations

pointing for each function. The checkpointing cost for each function is represented by lines, and the checkpointing cost at each call is illustrated with dots. The horizontal axis shows the sequence of event during the execution of the kernel, and the vertical axis shows the number of words that needs to be checkpointed, where the scale is linear below 10 and logarithmic above for clarity. The checkpointing cost for each function is calculated as follows: Supposing the status is already consistent before the call to the function, record the number of words that would need to be checkpointed after the return of the function to get a consistent status again. The checkpointing cost at call is calculated by recording the number of words that would need to be checkpointed between 2 function calls.

Figure 2 shows code from the Mibench [11] *fft* benchmark. This illustrates where checkpointing within an iteration of a loop would be less efficient than checkpointing at the end of an iteration or after the loop. The variables highlighted in blue are those that are read first and then written to within an iteration. Yellow variables are those that are written to first each iteration. Green variables are only used to generate other variables, and are not used in a subloop. Pink indicates an unused variable. If checkpoints are put inside the loop, then the yellow set may need to be checkpointed along with the blue set. However, if checkpointing at the end or in-between iterations, then the yellow set can be discarded due to future writes overwriting the current value. After exiting the loop, no local variables would need to be persisted, greatly reducing the commit cost compared to checkpointing inside the loop.

These trade offs can be seen in Figure 1. For function 7, by us-

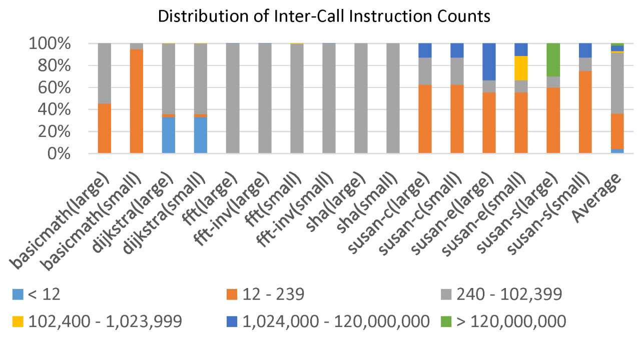

Figure 3: Distribution of inter-call instruction counts

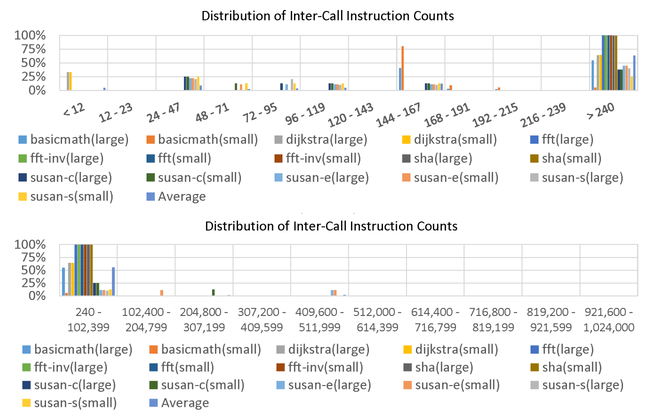

Figure 4: Distribution of inter-call instruction counts (step)

ing the checkpointing for function scheme, the checkpoint cost is 4. However, due to multiple function calls to function 3 (whose cost when checkpointing for functions is 0), the cost for checkpointing at call would be 1 more than the checkpointing for function scheme for each call to function 3, becoming 7 in total. In this case, it would be more beneficial to use the checkpointing for function scheme instead of checkpointing at each call. However, in the case of function 22 and function 24, checkpointing for function 22 when it returns would require 1529 more committed words than just checkpointing at function calls. Moreover, for functions with large costs (e.g 22 and 24) it is unclear that function-level checkpointing in many NVP environments will ever succeed unless augmented with finer-grained checkpointing mechanisms: Figure 3 and Figure 4 shows that for the check before call scheme, most segments between 2 calls are less than 102.4k instructions long, which may be suitable for less intermittent energy sources, but, for some benchmarks, such as susan-smoothing, there are segments that are longer than 120M instructions, which means that even very steady energy sources like solar energy could face challenges.

The smallest such granularity that makes sense for software directed approaches is the basic block level (*BBL*). Figure 5 shows the checkpointing cost distribution at BBL measured across different benchmarks and input sizes. The vertical axis shows the number of basic blocks in percentage, and the horizontal axis shows the different benchmarks and inputs with the right most bar showing the average. From bottom to top, the stacked bar shows the percentage of basic blocks that have checkpointing cost of 0, 1, 2 and larger than 2 respectively. Different input sizes only slightly affect the percentage distribution, and basic blocks with zero checkpointing cost (idempotent) consistently constitute a large fraction of basic blocks. On average, around 48% of the basic blocks have zero checkpointing cost, and around 33% of the basic blocks have a checkpointing cost of 1. For some benchmarks, such as dijkstra, over 99% basic blocks have checkpointing cost of less or equal to

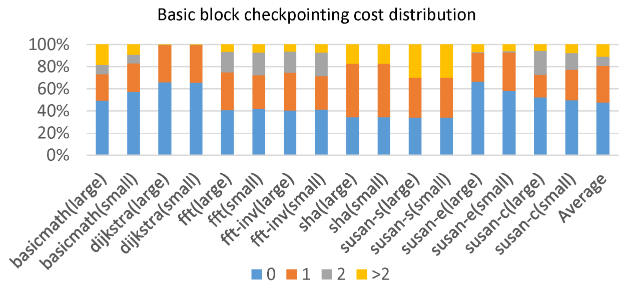

Figure 5: Basic block level checkpointing cost distribution

one. Only 11% of dynamically executed blocks have commit costs larger than 2 per block. This is not particularly surprising, as these codes have basic block sizes that tend to be less than 12 instructions long. This indicates that a BBL approach should be able to achieve progress guarantees similar to an instruction-granularity approach with overheads amortized by both block size and the frequent occurrence of idempotent blocks in these workloads.

4. POWER PROFILE ANALYSIS

In this section, we examine common ambient energy sources including piezo, solar and WiFi in order to observe backup-relevant features of the types of power traces seen from such sources. The aim of this power profile analysis is to build the foundations for mapping between the granularity of the power profiles to the granularity and backup policy for code regions as well as application scenarios. To develop the statistical features of the representative power profiles, we first present a brief case study of real harvested power profiles to provide an intuitive concept about what these power profiles would look like. Then we run a non-pipelined NVP [1] simulator powered by these power profiles for Mibench testbenches with a fixed activation threshold and selective backup policy[1]. This represents a best-case scenario for power-on time, albeit not necessarily a best case scenario for forward progress. We also show the backup number count that NVP performed to show the potential penalty of energy wasted on backup.

4.1 Piezoelectronic Energy

Figure 6 shows three separate profiles captured from daily activities, sampled once every 0.1ms on a wrist-mounted piezoelectric harvester in a watch form factor [12]. The power profile exhibits randomness in both amplitude and granularity, varying from almost 0uW to 2000uW, as well as rapid ramp-ups and drops with little warning. Figure 7 shows the simulation results for power profile 1 in Figure 6, highlighting the system-on duration of periods of sufficient power to run the processor. We assume a processor frequency of 120kHz and active power of 20-40uW. The power-on behavior exhibits randomness, which complicates a direct mapping to code properties, but, as seen in Figure 8, the distribution in power-on durations indicates that most active periods will be between 1600us to 4000us. The profile clearly indicates the need for nonvolatile processing, as a volatile version of our processor powered by these profiles is unlikely to completely execute even a short kernel without a power interruption occurring, given the very low frequencies that the processor must operate at 128kHz, 1600-4000us represents a run of only between 48-120 instructions.

Backup count, shown in Figure 9(a), is about 2000 times per 10 seconds, which is extremely high. This indicates that both reducing backups and reducing data per backup will greatly improve NVPs targeting piezo harvesting sources. Given the distribution of run lengths, basic block, rather than function-level software checkpointing techniques, or hybrid sw-hw techniques would be most appropriate for these power profiles.

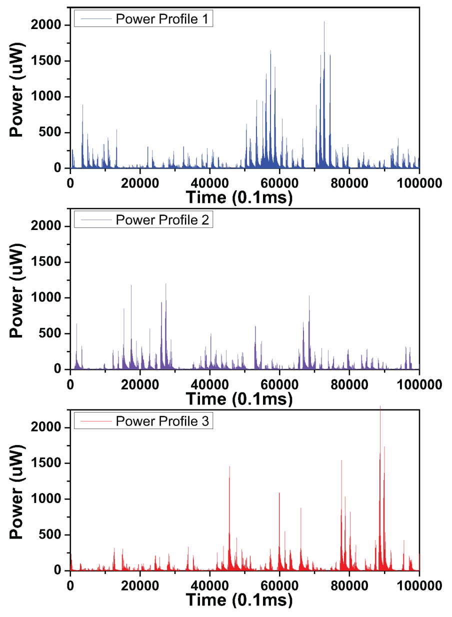

Figure 6: Power profile for wrist-mounted piezoelectronic harvester during daily activities

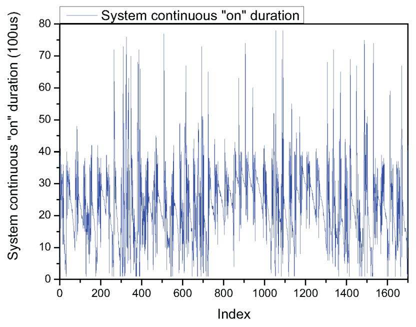

Figure 7: System "on" time duration for piezo Profile 1

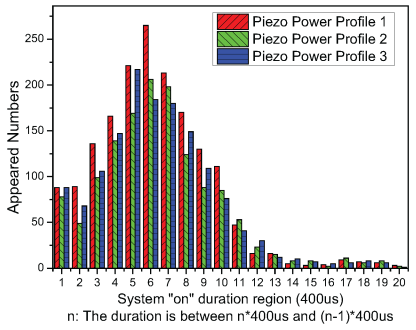

Figure 8: Distribution of powered durations for piezo profiles

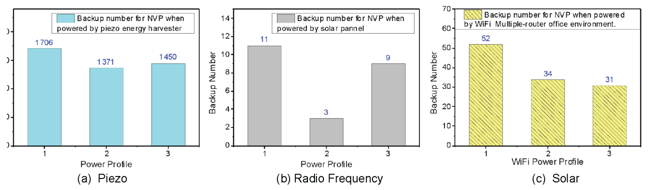

Figure 9: Backup number of a non-pipelined centralized NVM block based NVP powered with (a) Piezoelectronic (b) Solar (c) WiFi energy harvesters

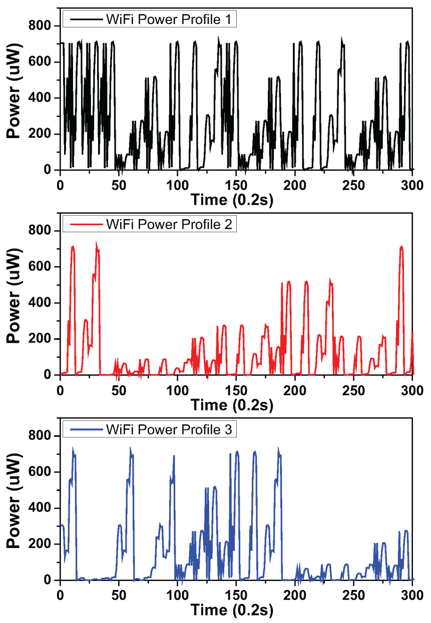

Figure 10: WiFi profiles measured in an office equipped with multiple WiFi routers at sample time 0.2s interval

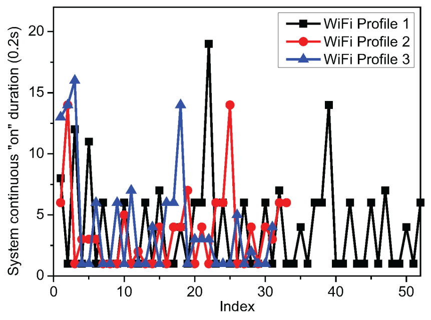

Figure 11: System "on" time duration for WiFi Profiles

4.2 Radio Frequency Energy

Figure 10 shows the power profiles from a radio-frequency harvester collecting energy from an office WiFi environment with multiple routers. The magnitude variation is a notable feature of the RF traces, and can be as large as hundreds of times the average power. We simulate an NVP with a threshold of 80uw and frequency of 512kHz, which has been indicated to be a high-performing configuration in RF environments [5]. The power-on intervals vary from 0.2s to 4s. In terms of instructions, this translates to between roughly 100K to 2M instructions being executed during an activation. In a one minute simulated execution, the NVP backed up between 31 to 52 times as shown in Figure 9(b). Unlike the piezoelectric traces, these run lengths are likely sufficient for courser grained software directed approaches to frequently be successful. However, if larger checkpoint gaps happen to line up with a shorter activation duration, a software-only approach could experience significant lost work if no finer-granularity support is provided, and larger code regions larger than a few million instructions would likewise need to be explicitly decomposed.

4.3 Solar Energy

The profiles in Figure 12 depict solar energy income for a solar panel in a stable location with fixed angle, sampled by the minute. The area is 1 cm^2. 10 percent solar-electronic transformation effi-

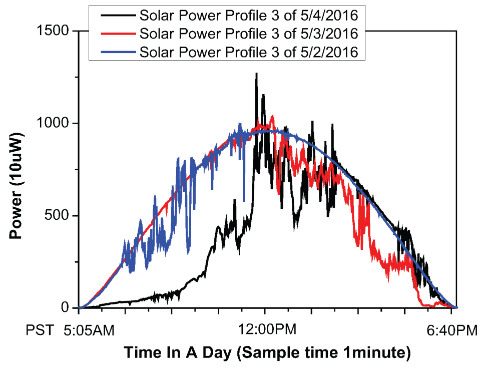

Figure 12: Solar energy harvester power profiles measured by a fixed-location outdoor device

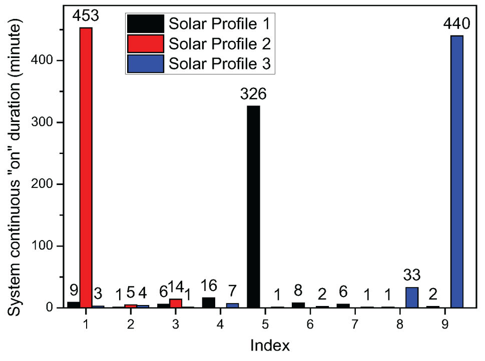

Figure 13: System "on" time duration for solar energy harvester Profiles

ciency is assumed. Solar harvesting can provide higher DC output, and our baseline NVP is assumed to be able to run at 2MHz under solar power, consistent with prior studies [13]. Despite higher power income, Figure 12 still depicts significant variance due to the time of day and weather conditions. However, unlike the previously discussed piezoelectric and radio frequency harvesting, these changes are slower, easier to predict, and only in very few occasions does power income fall to zero.

Statistics on the power-on time are shown in Figure 13. There is one exceptionally long run lasting for several hours, and the rest of the power-on durations are distributed between several minutes to 10s of minutes. Consequently, the backup numbers shown in Figure 9(c) are very low, only 3 to 11 times a day, and the NVP can expect to execute over 100M instructions during a powered-on period. Unsurprisingly, a solar powered NVP would therefore be relatively insensitive to checkpointing granularity and would be more focused on minimizing overheads, although some checkpointing approach is still necessary, as indicated by the regions shown in the workload analysis in Section 3 that require more than 100M instructions to be executed for task completion.

5. DESIGN APPROACHES AND DISCUSSION

In this section, we discuss NVP design approaches considering energy sources, nonvolatility hierarchy, program characteristics, and checkpointing method, and we also introduce potential opportunities to optimize NVPs in these environments.

5.1 Energy Source Features

As analyzed in Section 4.3, even when an NVP has been tuned for a given power source, different energy sources produce starkly varying powered duration distributions and exhibit high variation

Energy Source	Nonvolatility	Checkpointing Method
Piezo	on-chip distributed	Hardware
Solar	on or off chip	Software pre-defined
WiFi	on-chip centralized	Hardware

Table 1: NVP Design approaches under consideration of energy sources, nonvolatility hierarchy, and checkpointing method.

in power income even within an active time period. This variation in input power magnitude provides both challenges for the energy storage system, which must operate at a low threshold, but should capture peak incomes, and opportunities in NVP power management, such as using dynamic frequency scaling and resource allocation to turn surplus energy into additional computation.

While, at fine temporal granularity, sources (e.g. piezo) may seem to have high energy spikes at random, in the context of the wearer's activity, such spikes may be substantially more predictable. Additional context information can also assist solar (for instance, time of year and weather reports), and mapping information regarding routers and blockages could assist RF prediction. However, given the very limited energy budget of an NVP, such predictors themselves can have large energy costs if run frequently [5].This means that, even if large-scale behaviors can be predicted, which may influence scheduling policies and quality-of-service considerations, prediction alone is unlikely to avoid challenges for power interruptions at the dozens-hundreds of instruction granularities.

Note that the power-on intervals presented depend on the NVP startup threshold, which, while tuned for each power source, was a fixed value. A more dynamic approach could potentially trade among rate of execution, stability, and backup frequency by dynamically adjusting the minimum power-on threshold over a range. A smart NVP system may have options of (1), boosting performance during power interval then sleeping (run-to-halt for NVPs); (2) boosting performance then running very slowly to recharge the capacitor (computational sprinting [14] for NVPs); (3), running slowly enough to make average power meaningful and thereby avoid sleeping. Moreover, an NVP could dynamically trade among any of these three behaviors based upon input power characteristics.

5.2 Program Design

We observe that, for some functions, finishing the function can release numerous intermediate variables, reducing the amount of backup significantly. Pure hardware fine-grained NVPs cannot benefit from this effect and pay very large overheads in persisting these intermediate values. However, knowing that such regions are coming, even in a hardware NVP, does allow for backup reduction opportunities. Namely, even if storage efficiency is compromised by storing such a large amount of energy, if may still be worthwhile to ensure enough power-on time to finish these backup-heavy functions, provided that the inefficient, larger energy store can be engaged selectively. Whether this would be superior to engaging a fine-grained backup approach and persisting the intermediate values would require some prediction about power incomes, but at the large scale of such a region (e.g. millions of instructions or more), such predictions are more likely to be meaningful than for power over the next 10s-100s of instructions. To even better match between programs and profiles, a fat-binary approach could also be employed, where software directed schemes produce multiple possible checkpoints, and which set is enabled varies dynamically based upon power history.

Across the three power sources considered, not all appear to be sensible to rely on a software approach to ensure non-volatility. In particular, while solar power experiences intermittency, the power-on durations are long enough that not only could software direct backups, but any policy dynamism could conceivably execute in the software itself. WiFi power can clearly benefit from software assisted backup, but policy decisions at small scale are likely to remain the role of firmware or hardware. For piezoelectric harvesting, software hints at the basic block level, especially to indicate dead variables or idempotence, can still be useful, but function-level approaches appear impractical. Considering all these program design constraints, we see opportunities for co-design from the programming language level through the compiler, scheduling, firmware, and even operating system level support for NVPs.

5.3 Nonvolatility Hierarchy

The interplay between power and programs affects NVP design at a fundamental level: What nonvolatile elements should be included, and where should they be integrated? There are four broad answers to these questions, each with different pros and cons: (1) off-chip flash, which is easy and cheap to implement using off-the-shelf components guided by software. If power failures are present, but rare, this is the easiest deployment option. (2), an on-chip centralized nonvolatile memory block reduces the cost in both time and energy for backup operations but has limited total space and requires custom chips. (3), distributed nonvolatile flip-flops, and NV-SRAMs. Rather than serial backup, this approach provides potential parallel backups, and shorten the data movement distance, which is more energy efficient. However, it is also the most complex design with the lowest storage density, and large backup currents could cause system instability. (4), Finally, a design could be nonvolatile "all the way down" if new nonvolatile devices and circuits live up to their potential [15, 16, 17, 18, 19, 20]. However, at present, none of these proposed technologies designs are commercially viable, and many have endurance limitations that limit the practicality of processors without volatile state.

5.4 NVP Design Approaches

Taking all factors into consideration, we summarize the NVP design affinity for the considered sources and techniques in Table 1. For piezo systems with very frequent backups, we suggest an on-chip distributed backup topology with a purely hardware controlled backup method. While overheads are high, this is the only sure way to ensure forward progress on a piezo source. The software can provide hints to such a system, but cannot be relied upon for decision making. For solar designs with very few backups a cost-centric approach is recommended; such systems will not be as sensitive to design decisions, making cost and complexity a primary consideration, given that IoT devices are expected to be deployed at scale. For WiFi energy, which is hard to predict, we recommend an on-chip centralized backup solution with hardware controlled backup and restore. While the RF traces provide sufficient flexibility for a centralized storage approach, duration, magnitude, and predictability are still sufficiently poor that hardware guarantees with conservative backups will likely provide superior forward progress compared to software approaches with lower backup costs but more frequent rollbacks.

6. CONCLUSION

We have shown the significant diversity of power durations both within and across power sources and within programs for NVPs as well. For several potential combinations, these granularities can be mismatched for pure software and pure hardware approaches, which are either too optimistic or too conservative, respectively. While our work showcases the need for hardware support for relatively fine-grained (dozens of instructions) backup granularity, it also strongly motivates utilizing best-effort software approaches on top of such fine-grained hardware in order to benefit from longer power-on durations, when they are present.

ACKNOWLEDGMENTS

This work was supported in part by NSF awards 1160483 (ASSIST), and NSF CCF-1409798 subaward KK1522.

7. REFERENCES

[1] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 526–537, IEEE, 2015.

[2] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, no. 5, pp. 32–40, 2015.

[3] K. Ma, X. Li, K. Swaminathan, Y. Zheng, S. Li, Y. Liu, Y. Xie, J. J. Sampson, and V. Narayanan, "Nonvolatile processor architectures: Efficient, reliable progress with unstable power," *IEEE Micro*, vol. 36, no. 3, pp. 72–83, 2016.

[4] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, *et al.*, "Ambient energy harvesting nonvolatile processors: from circuit to system," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 150, ACM, 2015.

[5] K. Ma, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Dynamic machine learning based matching of nonvolatile processor microarchitecture to harvested energy profile," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 670–675, IEEE Press, 2015.

[6] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," in *ACM SIGPLAN Notices*, vol. 50, pp. 575–585, ACM, 2015.

[7] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Hybrid checkpointing using emerging nonvolatile memories for future exascale systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 2, p. 6, 2011.

[8] M. Xie, M. Zhao, C. Pan, H. Li, Y. Liu, Y. Zhang, C. J. Xue, and J. Hu, "Checkpoint aware hybrid cache architecture for nv processor in energy harvesting powered systems," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, p. 22, ACM, 2016.

[9] K. Swaminathan, R. Mukundrajan, N. Soundararajan, and V. Narayanan, "Towards resilient micro-architectures: Datapath reliability enhancement using stt-mram," in *2011 IEEE Computer Society Annual Symposium on VLSI*, pp. 236–241, IEEE, 2011.

[10] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue, "Fixing the broken time machine: consistency-aware checkpointing for energy harvesting powered non-volatile processor," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 184, ACM, 2015.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, IEEE, 2001.

[12] T. Xue and S. Roundy, "Analysis of Magnetic Plucking Configurations for Frequency Up-Converting Harvesters," *Journal of Physics Conference Series*, vol. 660, p. 012098, Dec. 2015.

[13] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *ESSCIRC (ESSCIRC), 2012 Proceedings of the*, pp. 149–152, IEEE, 2012.

[14] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, "Computational sprinting," in *IEEE international symposium on high-performance comp architecture*, pp. 1–12, IEEE, 2012.

[15] S. George, K. Ma, A. Aziz, X. Li, A. Khan, S. Salahuddin, M.-F. Chang, S. Datta, J. Sampson, S. Gupta, *et al.*, "Nonvolatile memory design based on ferroelectric fets," in *Proceedings of the 53rd Annual Design Automation Conference*, p. 118, ACM, 2016.

[16] X. Li, K. Ma, S. George, J. Sampson, and V. Narayanan, "Enabling internet-of-things: Opportunities brought by emerging devices, circuits, and architectures," 2016.

[17] K. Ma, X. Li, J. Sampson, Y. Liu, Y. Xie, and V. Narayanan, "Nonvolatile processor optimization for ambient energy harvesting scenarios," 2015.

[18] X. Li, H. Liu, U. D. Heo, K. Ma, S. Datta, and V. Narayanan, "Rf-powered systems using steep-slope devices," in *New Circuits and Systems Conference (NEWCAS)*, pp. 73–76, 2014.

[19] H. Liu, X. Li, R. Vaddi, K. Ma, S. Datta, and V. Narayanan, "Tunnel fet rf rectifier design for energy harvesting applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 400–411, 2014.

[20] K. Ma, H. Liu, Y. Xiao, Y. Zheng, X. Li, S. K. Gupta, Y. Xie, and V. Narayanan, "Independently-controlled-gate finfet 6t sram cell design for leakage current reduction and enhanced read access speed," in *2014 IEEE Computer Society Annual Symposium on VLSI*, pp. 296–301, IEEE, 2014.