Non-Volatile FPGA-based Intermittent Computing and Its Performance Analysis

Aalaa M.A. Babai[1,a)] Kuan yi Ng[1,b)] Teruo Tanimoto[1,c)]
Satoshi Kawakami[1,d)] Koji Inoue[1,e)]

Abstract: Intermittent computing is an emerging computing paradigm to enable computation in devices powered by energy harvesting. This can enable several battery-less and low maintenance applications. Previous research in intermittent computing has shown that it is possible to support complex computation under sever energy constraints and frequent interrupts with checkpointing. However, the performance is still limited by the system architecture and energy efficiency. In this work we propose a model for a non-volatile FPGA based reconfigurable heterogenous architecture and investigate its potential for future high performance intermittent computing systems.

Keywords: Intermittent Computing, Non-volatile FPGA, Analytical modeling

1. Introduction

Recent years have witnessed a rapid growth in the number of IoT devices and their applications. For many applications, batteries require maintenance and limit the life time of the device. Energy-Harvesting devices can extract energy from the environment utilizing different forms of energy like solar, kinetic and RF. Replacing batteries with EH-devices can enable self-powered devices that require little to no maintenance.

Since energy harvesting devices relies on unpredictable energy sources, they impose multiple challenges. They can only buffer and store a small amount of energy at a time. Additionally, the nature of the power source results in frequent and unpredictable failures. Hence, EH-powered devices must carry computation under strict energy budgets and frequent interrupts, which is known as intermittent computing. Intermittent computing deals with the challenge of guaranteeing correct forward progress of the program execution under very frequent and unpredictable interruptions. The system utilizes the buffered energy to perform computation while checkpointing its progress. When the buffer is empty a failure occurs, and the system must wait for energy to become available again in the next cycle. The interrupts vary with the energy source and can range from every milliseconds to every few minutes and hours [5]. This has motivated extensive research in efficient checkpointing techniques ranging from software to hardware levels.

Although achieving low checkpointing overheads can enable running complex applications in intermittent systems, research have shown that the overall performance is still limited by the underlying architectures capabilities and energy-efficiency [7][9]. Ultra-low-power architectures have been favoured due to their low power operation and high programmability which means they can support various EH sources and multiple applications. However this comes with the cost of performance. Performance in an intermittent systems is limited by the variable power cycles and the charging duration between them. Hence, power utilization, i.e., the ability of the system to translate the available power into performance is a key factor. In an intermittent system this is not a trivial task due to the power unpredictability and the amount of power sacrificed in maintaining forward progress.

FPGAs have demonstrated high performance and energy efficiency in various areas as an adaptive and reconfigurable computing platform. In particular, CPU-FPGA heterogeneous architectures have been investigated as a way to achieve both high performance and energy efficiency in a wide range of applications ranging from data centers to embedded systems [4][1][15]. We notice that although such an architecture can achieve high performance and energy efficiency without sacrificing the desired programmibility, FPGAs have not been considered in accelerating intermittent systems. We observe the potential of a heterogeneous CPU-FPGA architecture in accelerating intermittent computing. In particular, the reconfigurability of FPGAs, which can enable exploiting the area and performance relation under variable power conditions in intermittent execution. We focus on non-volatile FPGAs since their configuration data is not

[1] Kyushu Uniersity, Fukuoka, Fukuoka 819–0395, Japan
[a)] aalaa.babai@cpc.ait.kyushu-u.ac.jp
[b)] kuanyi.ng@cpc.ait.kyushu-u.ac.jp
[c)] tteruo@kyudai.jp
[d)] kawakami@ed.kyushu-u.ac.jp
[e)] inoue@ait.kyushu-u.ac.jp

lost during a power failure.

In this paper, we propose a performance model to understand the potential of an intermittent CPU-FPGA heterogeneous system. We model the acceleration in a single phase of a program running under intermittent conditions. Then we perform a quantitative analysis to compare between two non-volatile and volatile FPGA models. From the analysis, we derive insights for designing a future high performance architecture for intermittent computing based on application specific FPGA acceleration.

Major contributions of this work are as follows.

- This is the first work to consider FPGA-based acceleration for intermittent computing.
- We perform power and performance modeling for an intermittent system with CPU-FPGA and CPU-NVFPGA.
- We derive insights from the model to aid us in realizing a future CPU-NVFPGA based high performant intermittent architecture.

Our results show that NVFPGA offers an advantage over a volatile FPGA and can provide a significant speedup in accelerating an intermittent system, but this requires careful optimization for the reconfiguration latency and the memory architecture between the CPU and the FPGA.

The rest of this paper is organized as follows, Section 2 introduces intermittent computing and review some of the literature in it. Section 3 explains the FPGA-based intermittent acceleration. Section 4 shows the preliminary analysis with performance modeling. We conclude with our insights and future work in Section 5.

2. Intermittent Computing

In this section we provide a brief introduction of intermittent computing. We focus on discussing the main challenges and performance limitations in intermittent systems to facilitate modeling.

2.1 Checkpointing in Intermittent Computing

Typical intermittent computing systems consist of EH-powered computing device, sensors, and peripherals. The energy harvester charges a small buffer. Harvested energy is used to charge the buffer. When the buffer is full, the device carries on computation as shown in Figure 1. When the buffer is empty a power failure occurs and the device has to wait for energy to become available again in the next energy cycle. Since it is difficult to predict the time at which the energy buffer runs out during execution, and power availability depends on the source, these interrupts are unpredictable. If the program restart its execution with each failure, it is not able to make forward progress [20] and finish its execution. Following the same terminology in [20], the following subsections discusses single-backup and multi-backup systems and their main overheads.

2.1.1 Multi-Backup Systems

Early intermittent systems relied on compilers to make periodic backups during the program execution. The program

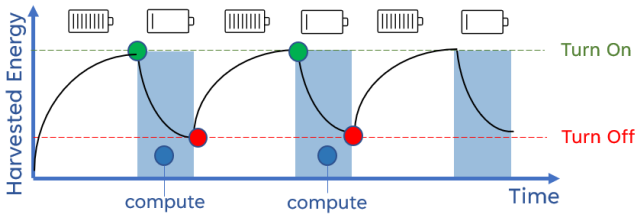

Fig. 1 The power cycles in intermittent execution

saves its state during execution and when a failure occurs, it resumes from the most frequent backup. This backup process is known as checkpointing, and the systems that perform frequent backups are known as multiple-backup systems. The backups in a multiple-backup system can be triggered by different ways. Early works relied on frequent voltage measurements to trigger backups [18]. However, it was shown later on that program re-execution from such backups can result in erroneous states due to inconsistencies caused in the non volatile memory [17]. In other works the backups are triggered by idempotency violations detected by hardware [8] or software [22]. Idempotent sections are code sections that can be re-executed multiple times error-free. Since these sections can re-execute without errors, backups are triggered at their boundaries. However, the most recent work in [3] have shown that program addresses renaming can eliminate backups resulting from idempotency violations and guarantee correct execution. The main challenge in these systems is to reduce the number of backups and re-execution overheads.

2.1.2 Single-Backup Systems

In order to further reduce the checkpointing overheads, some systems suggest to perform a single checkpoint per power cycle. Hibernus [2] relies on voltage measurement devices and voltage thresholds to take a single checkpoint before a power failure. This shifts the checkpointing overhead to the voltage measurement device which can incur up to 40% energy overhead. In order to eliminate checkpointing overheads, Alpaca [14] suggests slicing the program into tasks that can be executed in a single power cycle based on the energy buffer size. The compiler backups the task information only and guarantees the correct re-execution of these tasks in case a power failure occurs during execution. However, they rely on the programmer to divide the program by reasoning about the energy buffer size.

While the majority of the aforementioned techniques try to limit or avoid architectural modifications, Non-volatile processors [6][23][12][21][11] exploit fast and emerging memory technologies to enable fast backup. These works propose placing non-volatile memories at each level in the memory hierarchy. The processor relies on the built-in voltage detector and triggers a backup before the failure. The backup will copy the volatile registers to the non-volatile flip flops. This reduces the backup operation but consumes high energy during backup operations. The work presented in [13]

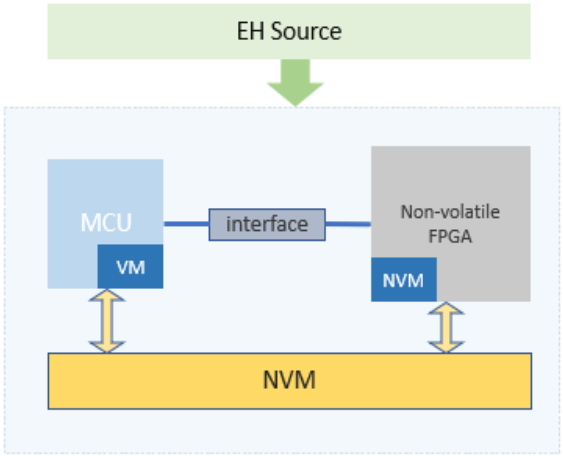

Fig. 2 Architecture of a non-volatile FPGA-based accelerator

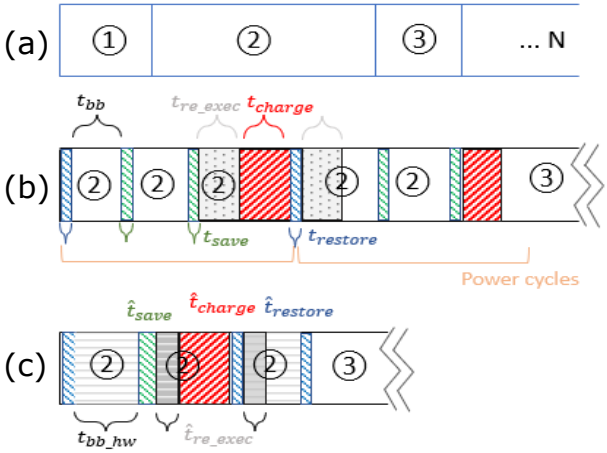

Fig. 3 Intermittent execution of a program. (a) The program with different execution phases. (b) The intermittent execution of phase 2 in the MCU compared to (c) phase 2 accelerated.

investigates various backup and design strategies for such processors.

2.1.3 Performance in Intermittent Computing

The performance in an intermittent systems is decided by the power utilization and charging duration. Although with low checkpointing overheads it is possible to run complex computations, the charging duration and the architecture energy efficiency limit the performance [7]. Ultra-low-power architectures have low power operation thresholds and can operate under a wide range of EH sources with various conditions. However such architectures fail to take advantage of power variability and situations when abundant power is available. One approach to meet performance requirements is to design and optimize the architecture based on the EH source characteristics [13]. However, such approaches relies heavily on these characteristics that can vary with different conditions.

3. NVFPGA-based ImC

Field programmable gate arrays (FPGAs) are hardware that can be tailored to different applications. Typical FPGAs contain a large number of configurable logic blocks (CLBs), comprised of look-up-tables (LUTs) and registers and multiplexers. These logic blocks can be configured to perform different functions. The logic blocks are surrounded by a sea of programmable interconnects. FPGAs are highly configurable and can achieve high energy efficiency and performance compared to CPUs.

With the steady improvements in FPGA design automation tools, FPGAs started to gain attention in a wide variety of areas.

Volatile FPGAs are based on CMOS technology and use SRAM cells for configuration. Since the SRAM is volatile, configuration information and intermediate data will be lost when power is turned off. On the other hand, non-volatile FPGAs use nonvolatile elements in the interconnect, this means their configuration data will not be lost when the power is turned off.

3.1 NVFPGA based acceleration

Non-volatile FPGA can offer application-specific acceleration to intermittent computing. A system with an ultra-low-power processor and a reconfigurable non-volatile FPGA is shown is show in Figure 2. The interface is used for reconfiguration and the shared memory is used for checkpointing and shared data. This system design is similar to the previous acceleration systems for intermittent operation proposed in [16][19][10].

A program running intermittently in the micro-controller unit can benefit from the re configurable non-volatile FPGA acceleration based on the available power at a certain duration. A program can be divided into different phases with different features. Based on power availability a phase can be accelerated by running it in the FPGA. However, since power failures can occur during this phase execution, the non-volatile FPGA offer the advantage of keeping the configuration data after the failure. This means the non-volatile FPGA needs to be reconfigured once for each phase of the program even when multiple power cycles occur. On the other hand, a volatile FPGA will need to be reconfigured once for each power cycle during the acceleration of a program phase.

Since this is the first work to consider FPGAs for accelerating intermittent systems, we start with a performance modeling and evaluation. This will be covered in the next section.

4. Preliminary Analysis

This section explains the preliminary analysis performed through performance modeling and model analysis.

4.1 Performance Modeling

We model the speedup in a single phase of the program under intermittent execution compared to a baseline system. The goal of this model is to show the possible speedup in

Table 1 Model Parameters and Their Definitions

	General Parameters
n_{charge}	Average number of power cycles in the baseline during a phase
$\hat{n}_{charge}$	Average number of power cycles in the accelerator during a phase
t_{bb}	Average time between backups in the baseline
t_{bb_hw}	Average time between backups in the accelerator
t_{sw}	Average time spent in computation in a power cycle for the baseline
t_{hw}	Average time spent in computation in a power cycle for the accelerator
λ	Speedup introduced by the accelerator in continuous execution
t_{save}	Average time to backup data after a duration t_{bb} in the baseline
$\hat{t}_{save}$	Average time to backup data after a duration t_{bb_hw} in the accelerator
$t_{restore}$	Average time to restore backup data after a power failure in the baseline
$\hat{t}_{restore}$	Average time to restore backup data after a power failure in the accelerator
$t_{re-exec}$	Average time spent in re-execution after a power failure in the baseline
$\hat{t}_{re-exec}$	Average time spent in re-execution after a power failure in the accelerator
E_{bl}	Average energy to execute a phase from the program in the baseline
ϵ_{bl}	Average energy in a single power cycle in the baseline (less than or equal the capacitor total energy)
E_{acc}	Average energy to execute a phase from the program in the accelerator
ϵ_{acc}	Average energy in a single power cycle in the accelerator (less than or equal the capacitor total energy)
p_{supply}	The supply power during the execution of a single phase of the program
t_{charge}	Average time charge between power cycles in the baseline
$\hat{t}_{charge}$	Average time charge between power cycles in the accelerator
t_{ckpt}	Average checkpointing time in the baseline in a single power cycle
$\hat{t}_{ckpt}$	Average checkpointing time in the accelerator in a single power cycle

intermittent operation when considering FPGA-based acceleration. We assume that a program consists of phases during execution. We measure the speedup for a single phase of the program compared to the baseline since it simplifies reasoning about power availability during execution. The baseline consist of a single EH-powered processor while the FPGA acts as the accelerator. The model parameters are listed in Table 1.

We start with the total execution time of a single phase in the baseline. As shown in Figure 3, the total time $T_{baseline}$ can be characterized by the number of charge cycles multiplied by the average execution time per cycle.

$$T_{baseline} = n_{charge}\Big(t_{sw} + n_{save}t_{save} + t_{restore} + t_{re-exec} + t_{charge}\Big) \quad (1)$$

Where the execution time t_{sw} in a cycle is equal to the time between backups t_{bb} multiplied by the number of backups. The re-execution time depends on when a failure occurs, it can be equal to 0 at minimum or equal to the time between backups t_{bb} at maximum. This is modeled with σ whose value range between 0 and 1.

$$t_{sw} = n_{save} \times t_{bb} \quad (2)$$

$$t_{re\_exec} = \sigma \times t_{bb} \quad (3)$$

Charging time during a power failure can be modeled with the average energy per cycle ϵ_{bl} and the power p_{supply} during the phase execution.

$$t_{charge} = \frac{\epsilon_{bl}}{p_{supply}} \quad (4)$$

Moreover, the total number of charges needed is equal to the total energy E_{bl} required to run a phase of the program divided by the energy per cycle ϵ_{bl} .

$$n_{charge} = \frac{E_{bl}}{\epsilon_{bl}} \quad (5)$$

For simplicity, we represent the total checkpointing overhead as

$$t_{ckpt} = n_{save}t_{save} + t_{restore} + \sigma t_{bb} \quad (6)$$

The final equation is then given by

$$T_{baseline} = n_{charge}t_{sw}\Big(1 + \frac{t_{ckpt}}{t_{sw}} + \frac{t_{charge}}{t_{sw}}\Big) \quad (7)$$

Next, we model the FPGA-based accelerator execution time, following the same approach in the baseline we represent the total execution time T_{acc} as follows

$$T_{acc} = \hat{n}_{charge}t_{hw} + \hat{n}_{charge}\hat{t}_{ckpt} + \hat{n}_{charge}\hat{t}_{charge} + n_{rec}t_{rec} \quad (8)$$

Where $\hat{t}_{ckpt}$ and $\hat{t}_{charge}$ are the accelerator checkpointing and charging costs respectively. Following the same approach as the baseline, the total number of charges is given by

$$\hat{n}_{charge} = \frac{E_{acc}}{\epsilon_{acc}} \quad (9)$$

Additionally, t_{rec} is the time required to configure the FPGA and can be given as

$$T_{rec} = Area \times Latency \quad (10)$$

Where the *Area* represents the resources provided per a

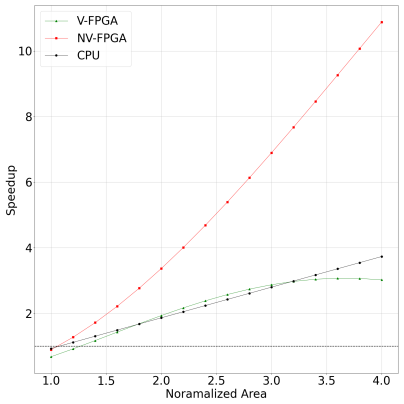

Fig. 4 Speedup and area for FPGA, non-volatile FPGA and CPU

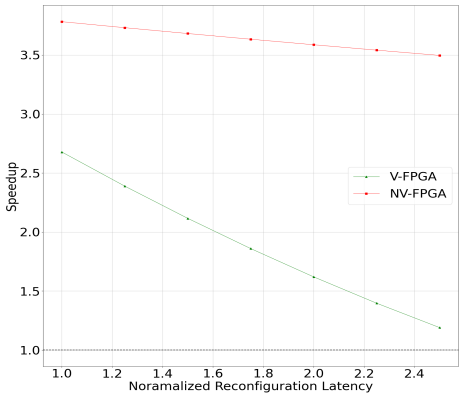

Fig. 5 Reconfiguration latency and the effects on speedup for volatile and non-volatile FPGA

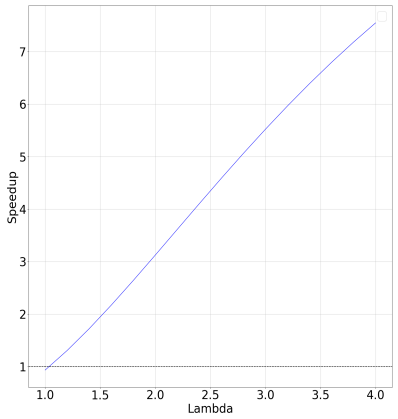

Fig. 6 Speedup with variable values of λ

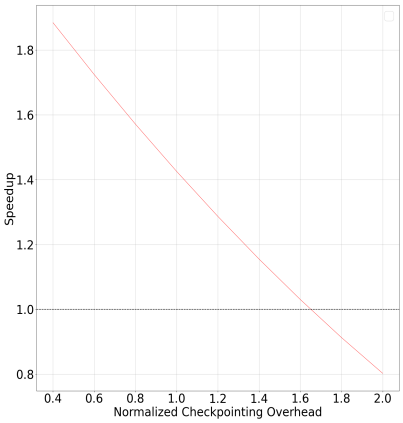

Fig. 7 Effect of the accelerator checkpointing overhead on the speedup

unit area of the FPGA and the *Latency* is the the time required by the interface to configure a single unit area of the FPGA. n_{rec} is the reconfiguration time during the execution of a single phase of the program, for a non-volatile FPGA, this value is always equal to one.

$$T_{acc} = \hat{n}_{charge}\Big(t_{hw} + \hat{t}_{ckpt} + \hat{t}_{charge} + \frac{n_{rect}}{\hat{n}_{charge}} t_{rec}\Big) \tag{11}$$

$$T_{acc} = \hat{n}_{charge} t_{hw}\Big(1 + \frac{\hat{t}_{ckpt}}{t_{hw}} + \frac{\frac{n_{rec}}{\hat{n}_{charge}} t_{rec}}{t_{hw}} + \frac{\hat{t}_{charge}}{t_{hw}}\Big) \tag{12}$$

Finally, we assume that t_{hw}, the pure execution time of the accelerator is

$$t_{hw} = \frac{t_{sw}}{\lambda} \tag{13}$$

That is λ is the acceleration provided by the accelerator. For FPGA we assume that this value is linearly proportional to the FPGA area. Then, the final equation is given as

$$speedup = \lambda \frac{n_{charge}}{\hat{n}_{charge}} \frac{\Big(1 + \frac{t_{ckpt}}{t_{sw}} + \frac{t_{charge}}{t_{sw}}\Big)}{\Big(1 + \frac{\hat{t}_{ckpt}}{t_{hw}} + \frac{\frac{n_{rec}}{\hat{n}_{charge}} t_{rec}}{t_{hw}} + \frac{t_{charge}}{t_{hw}}\Big)} \tag{14}$$

4.2 Evaluation

In this section we analyse the previous model to understand how different parameters affect the speedup.

4.2.1 Speedup with Lambda for non-volatile FPGA

First we study the effects of the speedup and λ in the non-volatile FPGA. We investigate a situation when high supply power is available. We assume a single-backup approach similar to [2] and refer to it as Just-in-time checkpointing. We assume checkpointing overhead of 25% of the the execution time ($t_{cktp} = t_{sw} \times 0.25$) for the baseline and that for the accelerator ($\hat{t}_{cktp} = t_{hw} \times 0.25$). Similarly, we assume a maximum latency overhead of 50% of the execution time ($\hat{t}_{rec} = t_{hw} \times 0.50$) at $\lambda = 4$. Additionally, we assume a constant high supply power value during the execution of the phase of the program which is reasonable assumption for EH sources with small rapid changes. The number

Table 2 Parameters for CPU, VFPGA, NVFPGA Comparison

	CPU	VFPGA	NVFPGA
t_{rec}	0	$area \times latency$	$area \times latency$
λ	$\sqrt{area}$	$area$	$area$
n_{rec}	0	n_{charge}	1

of charges are calculated from the total E_{bl} and E_{acc} for the baseline and the accelerator respectively. Their values are calculated from their respective execution times. We compare with the total program execution time in the continuous case to obtain these values. We also assume the same power for the different values of λ. We report the overall speedup in a single phase of the program execution when increasing the accelerator performance λ. We model λ to be linearly proportional to the resources provided per a unit area, hence increase in lambda will increase the reconfiguration time. Figure 6 shows the speedup under these conditions. The break even point occurs at a λ of 1.2. This is due to reduction in the number of power cycles since any reduction in the number of power cycles result in a reduction in overheads. Since we are assuming the same checkpointing overhead, and since reconfiguration is a one time cost, i.e., in a one charging cycle only, the reduced charging cycles due to the accelerator result in speedup. However, since we are assuming the same power for different values of λ for the porpose of investigation, these values serve as an upper bound for the speedup.

4.2.2 Speedup with Checkpointing Overhead

We also investigate the effect of the accelerator checkpointing overhead on the overall speedup. We follow the same conditions in Section 4.2.1 with varying checkpointing overhead.

Although Just-in-time approaches are simpler to implement, they are application-agnostic which means their overheads can still be reduced when considering application specific behaviour. Figure 7 shows the potential of checkpointing overhead reduction in the accelerator at a λ of 1.2, i.e., the break-even point for the conditions in Section 4.2.1.

4.2.3 Quantitative Comparison of Volatile FPGA, Non Volatile FPGA and CPU

In this section we compare between three platforms, namely a non volatile FPGA, volatile FPGA and CPU for acceleration. To perform this comparison we set the parameters shown in Table 2. Since our focus is on these platforms specific parameters we assume an ideal checkpointing overheads of 8%. Figure 4 shows the comparison results. The Volatile FPGA has to be reconfigured with each power cycle, so with a small reconfiguration overhead of 10% per area, it shows a comparable performance to the CPU. However, as the area increases the overhead dominates the execution time per a charging cycle and it limits the program forward progress. The non-volatile FPGA offers larger speed up than the CPU due to the linear relation between λ and area. This shows FPGA benefits gain over a bigger core. Additionally, the reconfiguration overhead is a one time cost that appears in the first charging cycle only.

We further investigate this gap between the volatile and non-volatile FPGA in Figure 5. We start with a low letency overhead of 9% and examine the effects on a λ of 2. Even at a small reconfiguration latency the gap between the volatile and non-volatile FPGA is noticeable. For the same area, the volatile FPGA reconfiguration latency occurs in every power cycle since it has to be reconfigured with every power loss. The gap expands further with little increase in latency.

We note that for the case of the volatile FPGA, it is possible to maintain the reconfiguration at the cost of saving some energy budget for maintaining the volatile FPGA state during charging time. This requires introducing new parameters to our model and we leave this for future work.

5. Conclusions

In this work, we investigated the potential of FPGA-based acceleration in an intermittent system. We propose a model for the acceleration in a single phase of an intermittent program based on available power at a time. We also performed quantitative analysis to compare between two non-volatile FPGA and volatile FPGA models. Our analysis showed that it is possible to achieve significant speedups with non-volatile FPGAs. Achieving up to 10× times speedups with a non-volatile FPGA-based accelerator requires optimizing the reconfiguration and checkpointing overheads. To achieve this, future work should prioritize investigating the interface and memory architecture of the non volatile FPGA-based intermittent acceleration system in addition to devising a power-aware reconfiguration mechanism.

Acknowledgments This work was partially supported by MIC under a grant entitled R&D of ICT Priority Technology (JPMI00316).

References

[1] Abdelouahab, K., Pelcat, M., Serot, J. and Berry, F.: Accelerating CNN inference on FPGAs: A survey, *arXiv preprint arXiv:1806.01683* (2018).
[2] Balsamo, D., Weddell, A. S., Merrett, G. V., Al-Hashimi, M., Brunelli, D. and Benini, L.: Hibernus: Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems (2014).
[3] Bhattacharyya, A., Somashekhar, A. and Miguel, J. S.: NvMR: non-volatile memory renaming for intermittent computing, *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pp. 1–13 (2022).
[4] Chiou, D.: The microsoft catapult project, *2017 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE Computer Society, pp. 124–124 (2017).
[5] de Winkel, J., Donne, C. D., Yildirim, K. S., Pawelczak, P. and Hester, J.: Reliable timekeeping for intermittent computing, *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pp. 53–67 (online), DOI: 10.1145/3373376.3378464 (2020).
[6] Ganesan, K., San Miguel, J. and Enright Jerger, N.: The What's Next Intermittent Computing Architecture, *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 211–223 (online), DOI: 10.1109/HPCA.2019.00039 (2019).
[7] Gobieski, G., Lucia, B. and Beckmann, N.: Intelligence beyond the Edge: Inference on Intermittent Embedded Systems, *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pp. 199–213 (online), DOI: 10.1145/3297858.3304011 (2019).
[8] Hicks, M.: Clank: Architectural support for intermitent computation, Vol. Part F128643, Institute of Electrical and Electronics Engineers Inc., pp. 228–240 (online), DOI:

10.1145/3079856.3080238 (2017).

[9] Islam, B. and Nirjon, S.: Zygarde: Time-Sensitive On-Device Deep Inference and Adaptation on Intermittently-Powered Systems, *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, Vol. 4, No. 3 (online), DOI: 10.1145/3411808 (2020).

[10] Lee, S. and Nirjon, S.: Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems, *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pp. 138–152 (2019).

[11] Liu, Y., Li, Z., Li, H., Wang, Y., Li, X., Ma, K., Li, S., Chang, M.-F., John, S., Xie, Y. et al.: Ambient energy harvesting nonvolatile processors: From circuit to system, *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6 (2015).

[12] Ma, K., Li, X., Li, J., Liu, Y., Xie, Y., Sampson, J., Kandemir, M. T. and Narayanan, V.: Incidental computing on IoT nonvolatile processors, *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Vol. Part F1312, pp. 204–218 (online), DOI: 10.1145/3123939.3124533 (2017).

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

[14] Maeng, K., Colin, A. and Lucia, B.: Alpaca: intermittent execution without checkpoints, *Proceedings of the ACM on Programming Languages*, Vol. 1, pp. 1–30 (online), DOI: 10.1145/3133920 (2017).

[15] Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S. et al.: Can FPGAs beat GPUs in accelerating next-generation deep neural networks?, *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 5–14 (2017).

[16] Qiu, K., Jao, N., Zhao, M., Mishra, C. S., Gudukbay, G., Jose, S., Sampson, J., Kandemir, M. T. and Narayanan, V.: ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors, *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, pp. 315–327 (2020).

[17] Ransford, B. and Lucia, B.: Nonvolatile memory is a broken time machine, *Proceedings of the workshop on Memory Systems Performance and Correctness*, pp. 1–3 (2014).

[18] Ransford, B., Sorber, J. and Fu, K.: Mementos: System support for long-running computation on RFID-scale devices, *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pp. 159–170 (2011).

[19] Resch, S., Khatamifard, S. K., Chowdhury, Z. I., Zabihi, M., Zhao, Z., Cilasun, H., Wang, J.-P., Sapatnekar, S. S. and Karpuzcu, U. R.: MOUSE: Inference in non-volatile memory for energy harvesting applications, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 400–414 (2020).

[20] San Miguel, J., Ganesan, K., Badr, M., Xia, C., Li, R., Hsiao, H. and Jerger, N. E.: The eh model: Early design space exploration of intermittent processor architectures, *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, pp. 600–612 (2018).

[21] Swaminathan, K., Pisolkar, R., Xu, C. and Vijaykrishnan, N.: When to forget: A system-level perspective on STT-RAMs, *17th Asia and South Pacific Design Automation Conference*, pp. 311–316 (2012).

[22] Woude, J. V. D. and Hicks, M.: Intermittent computation without hardware support or programmer intervention, *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pp. 17–32 (2016).

[23] Xie, M., Zhao, M., Pan, C., Li, H., Liu, Y., Zhang, Y., Xue, C. J. and Hu, J.: Checkpoint aware hybrid cache architecture for NV processor in energy harvesting powered systems, Institute of Electrical and Electronics Engineers Inc., (online), DOI: 10.1145/2968456.2968477 (2016).