Compas'2023 : Parallélisme/ Architecture/ Système
LISTIC/USMB - Annecy France, 4-7 juillet 2023

RESURRECT : A New Runtime for Intermittent Computing

Antoine Bernabeu, Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, Olivier H. Roux

Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France
antoine.bernabeu@ls2n.fr

Résumé
While intermittent systems powered by harvested energy without batteries allow to deploy sensor node in previously inaccessible area due to batteries constraints like size and maintenance cost, there is a trend to leverage the complexity of applications running on these systems. This will bring us one step closer to autonomous edge-computing nodes. To this end, we introduce RESURRECT, a runtime support for intermittent computing nodes embedding an energy consumption aware model to maximize the forward progress of the application. Built on top of an RTOS to allow more sophisticated and intelligent application, RESURRECT provide up to 77% of execution speed-up compared to state-of-the-art intermittent runtime.

Mots-clés : Intermittent System, Batteryless, Energy constrained System

---

1. Introduction

The joint democratization of high-performance non-volatile memory technologies and ultra-low power electronics has enabled the appearance of battery-less systems. Battery-less systems are powered by energy harvested from their environment backed by a capacitor to smooth out the unpredictable and intermittent supplying of power to the computing unit and the peripherals. These systems are called *intermittent* as they switch on and off according to the available energy. Without batteries, the lifespan of such systems is increased, the maintenance cost is greatly reduced and, their size can be reduced allowing new applications in various fields. Moreover, batteries contain a large number of toxic metals (lead, nickel, cadmium) and corrosive fluids that require special treatments for recycling [5] and may lead to environmental contamination.
Two main approaches have been studied to enable intermittent execution on low-power systems. The first is the insertion of checkpoints into the application code to save into the non-volatile memory necessary volatile data before a power-failure to preserve forward progress. Checkpoints can either be inserted manually by the developper [7, 10], triggered by hardware [2] or automatically by the compiler [9]. The second is an approach based on a transactional computing model where the application is decomposed into restartable and never inconsistent tasks. Forward progress is guaranteed at task-level, thus inducing a trade-off between kernel overhead and re-execution of code for sizing tasks (small tasks will to more runtime overhead while big task will lead to more re-execution in case of power failure). This *task-based* approach was proven to be more efficient than *checkpoint-based* approaches [6, 8].
Previous work on intermittent kernel focus on guaranteeing memory consistency and forward progress in a best-effort manner [11, 12, 8, 1]. These kernels are not aware of the energy

consumption of the system they are running on, leading to overhead due to re-execution of code un-commited before a power-failure.
In this paper, we introduce *RESURRECT*, an energy consumption aware intermittent kernel.
Contributions : We make the following contributions in this paper :

- **Intermittent Computing Execution Model :** Using a high-level abstraction of an application running under the intermittent computing paradigm, we designed an energy consumption aware computing model allowing to leverage the complexity of applications running on batteryless device with low overhead.
- **A Runtime Execution Support called *RESURRECT* :** *RESURRECT* implements the designed energy aware intermittent computing execution model and for this purpose, an RTOS has been modified to support this model.

2. *RESURRECT* : model definition

We consider a concurrent, task-based, intermittent model of computation. In this model, a workload is composed of activities. Before introducing activities, with give a definition of modes that are part of activities.
Mode : A mode is associated with a subset of the devices of the platform. When a mode is active, then all the devices associated with this mode are active. From an energy consumption point of view, a mode is characterized by a constant voltage slope, which is the sum of the different slopes associated with the operation of these circuits.
The finite set of mode of the system is $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. In our model, each mode m_i is coupled with a linear function f_{m_i}, where $\frac{df_{m_i}}{dt}$ describes the evolution of the voltage at the terminals of the energy buffer per unit of time when the mode is the only active mode in the system and no energy is harvested.
A measurement campaign was carried out to verify the compliance of the model and especially the linear evolution of the voltage across the super-capacitor for each mode.
Activity : An activity is a part of the application implemented in software (for example the execution of a computation intensive program) or in hardware (for example a copy of data by the DMA) whose execution activates one (and only one) mode of the platform. An activity a_i is characterized by :

- a mode $mode(a_i) \in \mathcal{M}$,
- a worst case execution time $wcet(a_i) \in \mathbb{N}$,
- a worst case energy consumption $wcec(a_i) = wcet(a_i) \times \frac{df_{mode(a_i)}}{dt}$,
- an absolute deadline $deadline(a_i)$.
- a reward $\mathcal{R}_i$ that denote the progress of the application when the activity is done.

The reward linked to an activity is fixed by the developper of the application. The greater the reward, the greater the progress of the application when the activity is done. This value allows to make a choice in the scheduling of activities when the energy supply is low. The execution of an activity should not be split over two execution cycles. In other words, the runtime only start the execution of an activity if it has enough energy to complete it. This implies that the worst case energy consumption of an activity is not greater than the size of the energy buffer of the platform : $\forall a \in A$, $wcec(a) \leq E_{max}$ where A is the set of activities.

3. *RESURRECT* : design and implementation

3.1. Design

RESURRECT has been designed to enable more complex application running under the intermittent paradigm and to optimize the energy usage on intermittent systems. Based on a high-level model, we generate steps that optimize the forward progress of the application according to the energy buffered and the current state of the system.

A step is a set of activities that maximize the reward reachable within a predefined energy budget. We used a model based on Cost Time Petri nets (c-*TPN*) to model an application. This abstraction use the model defined in Section 2. Optimal steps are computed by solving an optimization problem under constraints on the Cost Time Petri net model of the systems Since this algorithm relies on an exhaustive exploration of the path of the state space of the model, it scales poorly. Thus, we have designed heuristics to guide the exploration towards good yet suboptimal solutions. We compute a step iteratively for each couple of a state of the system and a starting stock of energy. A step depends only on this information. Thus, reaction to external events are scheduled in the steps that come after the step where the event occurs.

Since the computation of steps is based on worst-case scenario and does not take into account energy harvesting, it is possible that, at the end of a step, the stock of energy is sufficient to make another step before to put the system in sleep mode. Thus, for each state of the system, we compute a step for several energy levels (*i.e.*, for several starting energy budget). The result can be expressed as a graph where the vertices are the different state of the system and the edges are steps. Edges are guarded by an energy value : the minimal energy to ensure the completion of the step. Figure 3 illustrates the high-level model of a program (figure on the left) and the graph with the different steps generated for this program (figure on the right). A detailed explanation of step computation can be found in [4].

3.2. Implementation

RESURRECT design rely on steps, computed offline and target-dependent, that are embedded in the intermittent system. When starting the system, a step will be elected. This choice is based on the current energy level (charge of the super-capacitor), the detection of external events, and on the current state of the system. The elected step is the one that starts from the current state of the system, that handle events if detected, and with the highest energy requirement among those that have an energy requirement not greater than the available energy. When the step is elected, the activity sequence is executed.

At the end of a step, when all activities of the step execution are finished, *RESURRECT* checks the current energy level. The state of the system is updated and if possible, a new step is elected. If no step are eligible due to a low level in the energy buffer, a checkpoint is performed, the system enter in a low-power mode and wake-up periodically to try to elect a step. When the harvesting is no longer sufficient to keep the system in low-power mode, the system shutdown. Upon reboot, *RESURRECT* will restore data from the checkpoint and will elect the next step. Figure 1 illustrates the management of the system with *RESURRECT*.

We implemented *RESURRECT* on a RTOS called *Trampoline* [3]. *Trampoline* is an RTOS released under a free license. It is compliant with OSEK/VDX and AUTOSAR standards. Trampoline supports, among others, the MSP430 platform in 16-bit and 24-bit memory models. On MSP430, Trampoline takes less than 10kb of memory, which is an advantage for these small platforms. OSEK/VDX and AUTOSAR compliant RTOS are static, i.e. the RTOS does not allow dynamic creation of objects at runtime; these are all known at compile time. The RTOS configuration, the objects that make up the application like tasks, interrupt service routines (ISR), ...

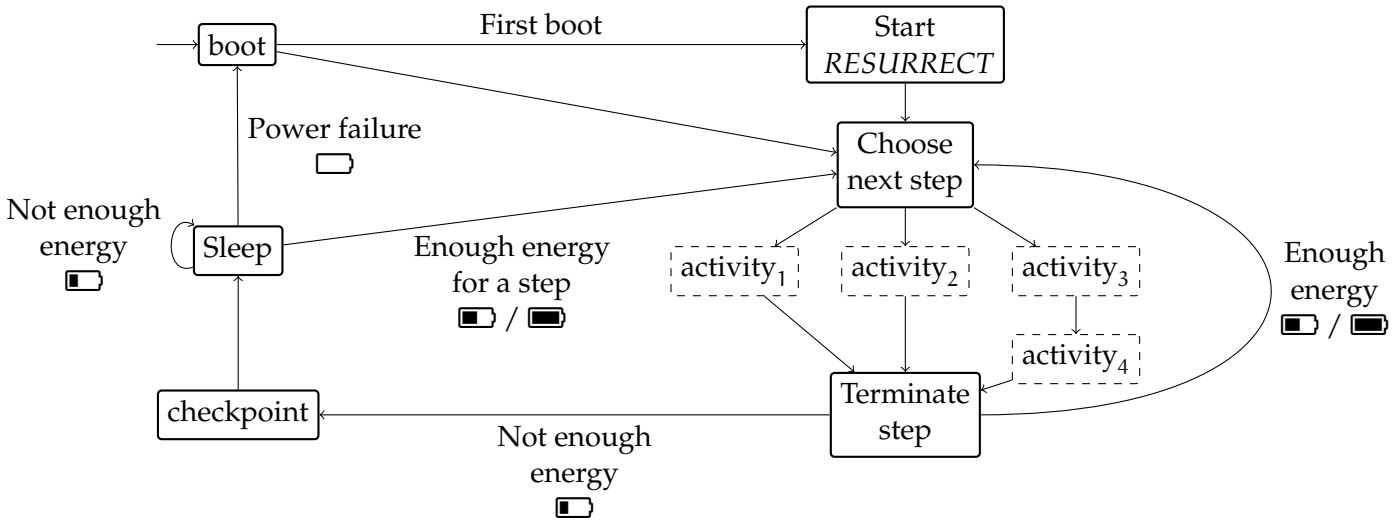

FIGURE 1 – Steps management in *RESURRECT* design.

as well as their relationships, are therefore described using a dedicated language called OIL for OSEK/VDX and using arXML for AUTOSAR. A static RTOS offers several advantages. First, the RTOS can be configured based on the objects present in the application description so that only the necessary code is embedded. For example, if an application does not use ISRs, all the RTOS services for handling ISRs can be omitted and thus allow to save memory space. For an intermittent system, a static configuration allows to checkpoint the RTOS more easily because the memory area is known at compile time.
The current implementation of *RESURRECT* targets the MSP430FR5994 microcontroller of Texas Instrument as it is widely used in similar works [1, 11, 12].
We have implemented the step management as services of the RTOS. Two specific system calls are used to start a step and terminate activities. When an activity terminates, instead of the traditional *TerminateTask* service from RTOS, we designed a custom service which terminate the activity and, in addition checks if the current step is finished or not.

4. *RESURRECT* : evaluation

We proceed with the experimental evaluation of *RESURRECT*. We compared *RESURRECT*with state-of-the-art intermittent runtime such as InK and ImmortalThread.

4.1. Experimental setup

Eembedded platform and tools : We used the MSP430FR5994 launchpad evaluation board from Texas Instrument. This board include 256 kB FRAM and 8 kB SRAM and can operate up to 16 MHz. Experiments have been done with a 1 MHz frequency to be compatible with other studies [11, 12]. Applications have been compiled using the GNU GCC v9.3.1.11 toolchain [1]. A Saleae logic analyzer has been used to get timing measurements of the execution of applications.

1. compile flag : -O0 to fit with result from InK[11] and ImmortalThread[12]

4.2. Preliminary Evaluation

We used three different benchmarks for our evaluation with different computational complexity. The bitcount benchmark (BC) consists in counting the number of 1 in a string of bits. The activity recognition benchmark consists in detecting using data processed from sensors whether sensors are moving or stationary. Data are faked using random generation. The DNN benchmark consists in classifying an image of a number using a deep neural network. The image input is stored as a constant in the memory. Figure 2 presents the execution time of applications and overhead due to the different runtimes. In *RESURRECT* the overhead is smaller thank to the offline computation of steps that lead to less checkpoints.

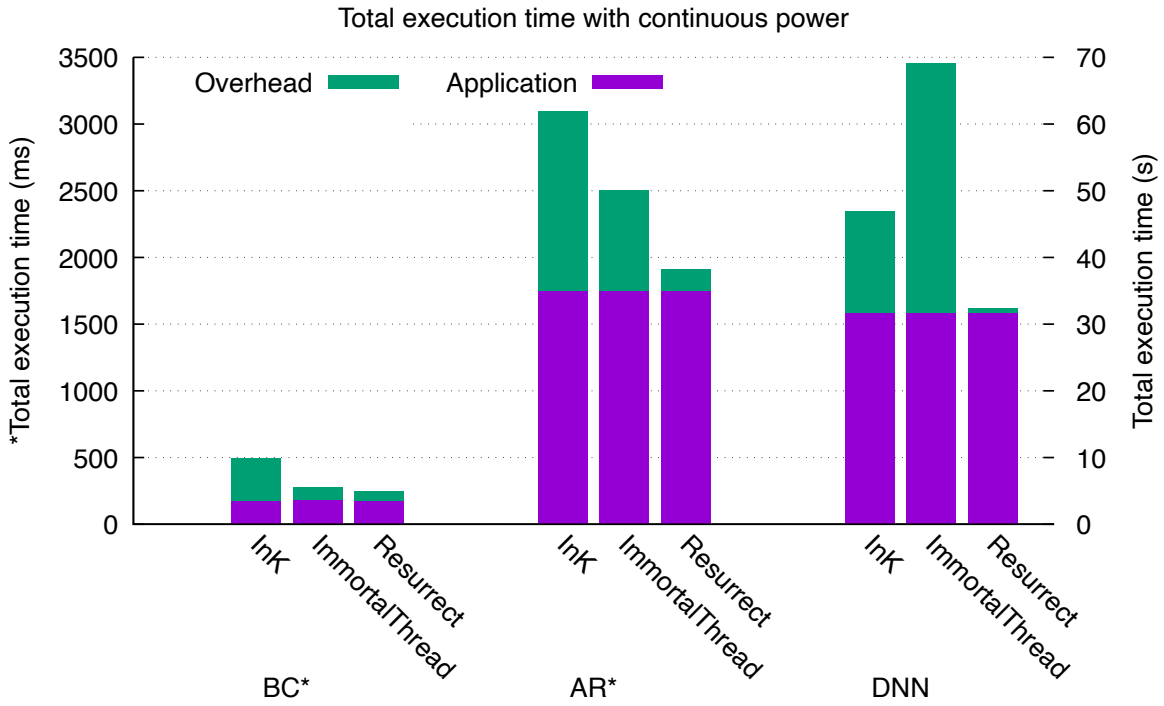

FIGURE 2 – Execution time of applications on continuous power running with different intermittent runtime. *RESURRECT* as a lower overhead compared to InK[11] and ImmortalThread[12]. The BC benchmark and AR benchmark have an execution time in order of hundreds of milliseconds (axis on the left) while the DNN benchmark have an execution in order of dozens of seconds (axis on the right).

We plan to evaluate the performance of *RESURRECT* on real intermittent scenario in the near future.

5. Conclusion

Developing autonomous edge-computing nodes either at the energetic, computational and operational level is a challenging research topic. All three aspects of intermittent and battery-free computing nodes must be taken into account in the design to ensure the best quality of service of any application running on it. To this end, we presented *RESURRECT*, an energy

consumption aware intermittent runtime. *RESURRECT* is designed to guarantee optimized quality of service on any application running on the intermittent platform based on a worst-case scenario. Results have highlighted how the integration of energy consumption of small set of the application into the runtime allow to decreased time execution overhead of intermittent runtime, thus optimizing quality of service of the application.

Bibliographie

1. Bakar (A.), Ross (A. G.), Yildirim (K. S.) et Hester (J.). – REHASH : A Flexible, Developer Focused, Heuristic Adaptation Platform for Intermittently Powered Computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, n3, septembre 2021, pp. 1–42.
2. Balsamo (D.), Weddell (A. S.), Merrett (G. V.), Al-Hashimi (B. M.), Brunelli (D.) et Benini (L.). – Hibernus : Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems. *IEEE Embedded Systems Letters*, vol. 7, n1, mars 2015, pp. 15–18. – Conference Name : IEEE Embedded Systems Letters.
3. Bechennec (J.-L.), Briday (M.), Faucou (S.) et Trinquet (Y.). – Trampoline An Open Source Implementation of the OSEK/VDX RTOS Specification. – In *2006 IEEE Conference on Emerging Technologies and Factory Automation*, pp. 62–69, septembre 2006. – ISSN : 1946-0759.
4. Bernabeu (A.), Béchennec (J.-L.), Briday (M.), Faucou (S.) et Roux (O.). – Cost-optimal timed trace synthesis for scheduling of intermittent embedded systems. *Discrete Event Dynamic Systems*, décembre 2022.
5. Bernardes (A. M.), Espinosa (D. C. R.) et Tenório (J. A. S.). – Recycling of batteries : a review of current processes and technologies. *Journal of Power Sources*, vol. 130, n1, mai 2004, pp. 291–298.
6. Colin (A.) et Lucia (B.). – Chain : Tasks and Channels for Reliable Intermittent Programs. p. 17.
7. Hester (J.), Storer (K.) et Sorber (J.). – Timely Execution on Intermittently Powered Batteryless Sensors. – In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pp. 1–13, Delft Netherlands, novembre 2017. ACM.
8. Maeng (K.), Colin (A.) et Lucia (B.). – Alpaca : intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, vol. 1, nOOPSLA, octobre 2017, pp. 1–30.
9. Maeng (K.) et Lucia (B.). – Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. – pp. 129–144, 2018.
10. Ransford (B.), Sorber (J.) et Fu (K.). – Mementos : System Support for Long-Running Computation on RFID-Scale Devices.
11. Yıldırım (K. S.), Majid (A. Y.), Patoukas (D.), Schaper (K.), Pawelczak (P.) et Hester (J.). – InK : Reactive Kernel for Tiny Batteryless Sensors. – In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pp. 41–53, Shenzhen China, novembre 2018. ACM.
12. Yıldız (E.), Chen (L.) et Yıldırım (K. S.). – Immortal Threads : Multithreaded Event-driven Intermittent Computing on Ultra-Low-Power Microcontrollers. p. 18.

A. Appendix

A.1. Step generation

FIGURE 3 – Generation of steps from a high level model of an application. On the left, the application is modeled using high level Petri net, on the middle, step are generated from the high level model and on the right, step are linked together through states of the application. For example, q_0 is the state with a token in P_1, q_1 is the state with a token in P_2, and q_2 is the state with a token in P_3. Note that the external event E is a condition for the activation of activity t_e. We suppose in this example that the energy buffer have a capacity of 3 and that each transition (t_1, t_2, t_3 and t_e) consume 1 energy. From each state of the system, for all energy level possible in the energy buffer (1 to 3) and for the detection or the non-detection of the event E, a step is generated maximizing the forward progress. For steps, the first number correspond to the starting state, the second to the energy available and the e or $\bar{e}$ correspond to the event detection. Thus S_{13_e} corresponds to the optimized step starting from state q_1, with 3 energy available and with the event E detected.