Synthèse de traces temporisées à coût optimal pour l'ordonnancement de systèmes embarqués intermittents

Antoine Bernabeu, Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, and Olivier Henri Roux

Université de Nantes, Centrale Nantes, CNRS, LS2N, F-44000 Nantes, France.

Résumé

Le calcul intermittent est un paradigme émergent pour les systèmes sans batterie, alimentés à l'aide de sources d'énergie elles-mêmes intermittentes. Ce paradigme promet une conception plus sobre des systèmes informatiques. Il semble particulièrement adapté au domaine des capteurs connectés qui forment le premier niveau de l'Internet des Objets (IdO). Ce domaine d'application nécessite de disposer d'un modèle de calcul réactif. La définition d'un modèle intermittent et réactif est un problème encore peu exploré dans la littérature. Dans cet article, nous nous intéressons à la modélisation et l'analyse de systèmes réactifs intermittents. Nous utilisons des réseaux de Petri de haut niveau temporel à coût pour modéliser les différentes dimensions du système : concurrence, temps réel, et énergie. Nous synthétisons ensuite des traces à coût optimal, afin d'obtenir des informations exploitables en ligne pour la gestion conjointe du temps de calcul et de l'énergie.

1 Introduction

Aujourd'hui, les capteurs connectés autonomes sont équipés de batteries lithium-ion, éventuellement alimentées par un ou plusieurs dispositifs permettant de récolter l'énergie présente dans l'environnement (p. ex. solaire ou thermique). Ces batteries ayant un nombre de cycles limités, leur utilisation induit des actions de maintenance qui peuvent s'avérer délicates voire impossibles à réaliser lorsque les capteurs sont déployés dans des environnements difficiles d'accès ou dangereux. Par ailleurs, les batteries lithium-ion sont sources de pollution aussi bien dans les phases de production que de recyclage ou d'élimination.

Tous ces arguments plaident en faveur du retrait des batteries de ces systèmes lorsque le domaine d'application le permet. Or, dans un grand nombre de cas, ceci est possible en utilisant des technologies existantes. Un système sans batterie est typiquement alimenté par l'énergie récoltée dans son environnement et intègre un super-condensateur qui, utilisée comme tampon d'énergie, permet de maintenir le système en opération quelques instants après une perte d'alimentation afin d'éviter les coupures brutales. Ces quelques instants peuvent permettre de sauver l'avancement du calcul si le système est également équipé d'une mémoire non volatile suffisamment performante et efficace. C'est par exemple le cas des RAM ferro–électriques, ou FRAM [14], qui offrent des performances, une efficacité énergétique, et une durée de vie accrues de plusieurs ordres de grandeur par rapport aux mémoires Flash. Cette technologie est sur le marché depuis plusieurs années. Cependant, sa densité d'intégration étant limitée, les composants disponibles ont une capacité de quelques centaines de KiO, au mieux quelques MiO pour des mémoires externes. Il existe un cadre d'utilisation bien adapté à ces contraintes, où cette technologie est aujourd'hui largement utilisée : celui des petits microcontrôleurs, où elle remplace la mémoire Flash. Les technologies citées (super-condensateur, FRAM) ne souffrent pas des limitations des batteries en terme de maintenance, ce qui doit donc permettre en théorie de prolonger la durée de vie des systèmes.

Du point de vue logiciel, la difficulté principale consiste à définir et supporter un modèle de calcul intermittent, c'est-à-dire un modèle au sein duquel les pertes d'alimentation sont un événement normal. Les premiers travaux se sont intéressés au calcul intermittent séquentiel, mais il est apparu rapidement que le domaine d'application visé, les capteurs intelligents, nécessite un modèle de calcul réactif et concurrent. Plusieurs équipes ont alors commencé à étudier de tels modèles. Nos travaux s'inscrivent dans cette direction.

Plus précisément, nous étudions le problème de la modélisation d'un système intermittent réactif et concurrent à l'aide de réseaux de Petri de haut niveau temporels à coût. Dans ce modèle, la consommation énergétique est approchée par un coût linéaire, qui s'appuie sur un ensemble de mesures réalisées sur cible. Le coût est également utilisé pour mesurer le gain du système en terme d'objectifs fonctionnels atteints. L'analyse du modèle nous permet ensuite de déterminer des traces à coût optimal pour atteindre certains marquages d'intérêt, le caractère optimal mesurant ici un compromis entre efficacité énergétique et objectifs fonctionnels. Ces traces ont vocation à fournir des informations pertinentes pour la gestion en ligne des ressources, en particulier l'énergie et le temps de calcul, en fonction de l'état du système. D'autres applications sont visées, par exemple le positionnement optimal de mesure de l'énergie récoltée, ou encore le placement des points de sauvegarde.

2 État de l'art

2.1 Sur les modèles de calcul intermittent

Sur le plan des modèles de calcul, les premiers travaux sur le calcul intermittent se sont concentrés sur l'insertion dynamique de point de sauvegardes par le biais d'un support d'exécution dédié [20, 4]. Cette approche souffre de plusieurs limitations bien identifiées qui la rendent sous optimale en termes de taille des sauvegardes et d'efficacité énergétique. Pour résoudre ces problèmes, une seconde direction a été explorée, dans laquelle le programme est décomposé en séquences atomiques terminées par des points de sauvegarde statiques. Les séquences sont construites soit par le programmeur [18], soit par le compilateur [12, 25], avec éventuellement une phase d'adaptation durant l'exécution [17]. Avec cette approche, après une perte d'alimentation, il suffit de reprendre l'exécution au début de la séquence interrompue. Cela permet de limiter la taille des sauvegardes aux informations utiles, et facilite le raisonnement statique sur le comportement du système. En revanche, cela induit l'exécution de sauvegardes inutiles.

Les travaux mentionnés ci-dessus supposent que le programme peut toujours être retardé, interrompu, ou rejoué plusieurs fois. Or, ce n'est pas toujours le cas. Certaines fonctions dépendent d'un état extérieur au contexte d'exécution. Ainsi, une fonction qui pilote un périphérique interagit avec l'état de ce dernier, qui est externe au système mémoire et ne peut pas être sauvegardé simplement. Ce problème est traité par Sytare [7], une couche logicielle qui enregistre les séquences d'opérations appliquées aux périphériques par les programmes afin de les rejouer, et ainsi restaurer l'état des périphériques après une perte d'alimentation.

D'autres fonctions nécessitent d'être exécutée de façon réactive ou atomique pour assurer la cohérence des calculs. On peut citer les fonctions qui assurent l'échantillonnage de signaux provenant de capteurs, ou encore la mise en œuvre des protocoles de communication. Pour répondre à ces besoins, des travaux complémentaires ont été réalisés pour étendre le modèle de programme séquentiel considéré initialement et définir un modèle de calcul réactif et concurrent pour les systèmes intermittents. Coati [21] permet ainsi l'intégration de routines d'interruption, qui peuvent partager des données avec des tâches transactionnelles. InK [26] est une extension plus complète qui s'apparente à un noyau de système d'exploitation pour les systèmes orientés

événements. Enfin, Catnap [19] adopte un modèle événementiel similaire et ajoute la possibilité de réserver une part du tampon d'énergie pour fournir une qualité de service minimale aux fonctions périodiques.

Une caractéristique commune à ces travaux est l'adoption d'une approche *best effort* dans laquelle les pertes d'énergie sont considérées comme des événements aléatoires. Certains utilisent une boucle de rétroaction sans modèle pour ajuster la distance entre les points de sauvegarde (p. ex. [4]), mais la plupart fonctionnent en boucle ouverte. Enfin, concernant la gestion de l'énergie, ils se concentrent tous sur les activités du cœur d'exécution, en supposant qu'il dispose d'une alimentation distincte du reste du système, négligeant ainsi la consommation du système mémoire et des périphériques. Cette hypothèse n'est pas vérifiée sur la majorité des architectures matérielles disponibles sur le marché.

2.2 Sur la modélisation des systèmes intermittents

Si l'on s'intéresse en premier lieu aux modèles formels, les travaux de l'état de l'art se concentrent sur les aspects fonctionnels des systèmes intermittents avec l'objectif d'établir des preuves de correction [6, 23]. Les ressources physiques (p. ex. temps de calcul, énergie) sont modélisées de façon très abstraites puisque les objectifs des travaux cités n'intègrent pas leur gestion.

En dehors du cadre des modèles formels, plusieurs travaux proposent des solutions pour simuler l'exécution d'un système intermittent, avec un intérêt particulier pour la modélisation de la consommation énergétique. Certains sont orientés vers la prédiction statique du pire cas de consommation énergétique [24], d'autres, comme le *framework* Fused [22], vers la simulation. Les modèles utilisés dans ces travaux sont très fins mais ont une complexité importante, ne les rendant pas adaptés à une phase d'exploration de l'espace de conception.

Pour permettre la mise au point de modèles de plus haut niveau, des plateformes de mesure de la consommation ont été développées [13, 8]. Ces plateformes se basent principalement sur la mesure du courant afin de mesurer la consommation énergétique du système et de ses différents modules. De plus elles font l'hypothèse que la tension d'alimentation est constante, ce qui suppose l'intégration d'un régulateur entre la source d'énergie et le microcontrôleur.

Enfin, concernant les modèles conçu pour estimer l'énergie récoltée, la granulométrie est très importante : les fenêtres temporelles considérées sont longues de plusieurs minutes, voire plusieurs heures. EWMA [15] est certainement l'algorithme le plus utilisé en raison de sa simplicité, malheureusement il montre des erreurs de prédiction allant jusqu'à 20%. Il existe des algorithmes plus précis comme Pro-Energy-VLT [11], mais celui-ci nécessite la définition de nombreux profils de récolte et requiert donc un espace mémoire trop important pour être utilisé en ligne sur les architectures visées.

3 Contribution

Dans cet article, nous nous intéressons à la modélisation et à l'analyse de systèmes intermittents réactifs. Pour cela, nous utilisons un formalisme riche permettant d'exprimer la concurrence, l'écoulement du temps, mais également la consommation énergétique des différentes activités du système (p. ex. le calcul, mais aussi l'écriture dans une mémoire externe, ou encore l'émission d'un message sur une liaison sans-fil). Le formalisme sélectionné est celui des réseaux de Petri de haut niveau temporel à coût, tel qu'il est supporté par l'outil Romeo [16]. À partir de ces modèles, nous synthétisons des traces de coût optimal. Le coût optimisé intègre

deux dimensions : d'une part la consommation énergétique, et d'autre part l'atteinte d'objectifs fonctionnels. À terme, les informations extraites de ces traces seront utilisées en ligne au sein d'une stratégie d'ordonnancement intelligente capable de tenir compte simultanément des contraintes de réactivité et de consommation énergétique. À notre connaissance, ce travail est le premier à proposer ce type de modélisation.

Le travail est présenté selon le plan suivant. Puisque nous considérons un modèle de consommation linéaire en tension, nous exposons en section 4 pourquoi ce modèle est une approximation raisonnable de la dynamique réelle des systèmes considérés (qui n'intègrent pas de régulateur de tension). Puis, en section 5, nous rappelons brièvement les caractéristiques du formalisme de modélisation sélectionné. La contribution principale vient en section 6 : nous y présentons la démarche de modélisation et d'analyse des systèmes intermittents réactifs, que nous illustrons à l'aide d'une étude de cas. Enfin, pour conclure, nous donnons quelques pistes concernant les travaux futurs sur ce thème ainsi que les principales leçons tirées de notre travail.

4 Modèle de consommation d'énergie

4.1 Introduction

Notre objectif à terme est de concevoir des stratégies en ligne d'allocation de ressources (temps de calcul, énergie) pour les systèmes intermittents réactifs. Nous visons des stratégies en boucle fermée, qui profitent donc d'une estimation de l'énergie disponible pour optimiser leurs décisions. Nous savons qu'il existe une relation entre la consommation d'énergie d'un système et les variations de sa tension d'alimentation [2]. Ainsi, si aucun régulateur de tension n'est installée entre la source d'énergie (le super-condensateur dans notre cas) et le microcontrôleur, une mesure de la tension d'entrée du système peut être utilisée pour nourrir un modèle d'estimation de l'énergie disponible. Notre objectif est qu'il soit assez simple, pour pouvoir être utilisé en ligne ou dans un modèle formel plus complet permettant d'analyser conjointement différents aspects du système, tout en offrant une précision suffisante. Dans cette section, nous décrivons un tel modèle, et nous donnons brièvement les principaux arguments théoriques et expérimentaux qui justifient son bien fondé. Soulignons cependant que la contribution principale de ce travail est la démarche de modélisation et d'analyse présentée en section 6, qui suppose simplement l'existence d'un tel modèle.

4.2 Un modèle linéaire

L'énergie fournie par le dispositif de récolte est emmagasinée dans le super-condensateur. Cette énergie, E, est fonction de la tension conformément à l'équation (1), où C est la capacité du super-condensateur et V_{cc} est la tension à ces bornes.

$$E = \frac{1}{2} \times C \times V_{cc}^2 \tag{1}$$

La puissance P consommée par le système est la somme de la puissance statique P_{stat}, quasiment constante, et de la puissance dynamique P_{dyn}, qui varie en fonction de l'activité du système : $P = P_{stat} + P_{dyn}$. Sur les circuits considérés (technologie CMOS à fréquence faible, quelques MHz à quelques dizaines de MHz), la puissance statique est très faible. Elle peut être estimée en mesurant la consommation d'énergie lorsque le système est en mode basse consommation avec son horloge arrêtée mais avec le CPU toujours alimenté.

La puissance dynamique consommée par un circuit CMOS est régi par l'équation (2), où f_{clk} est la fréquence du circuit (dans notre cas le microcontrôleur), C_L est la somme des capacités

des condensateurs chargés et déchargés pendant les différentes opérations, et V_{dd} est la tension d'alimentation.

$$P_{dyn} = f_{clk} \times C_L \times V_{dd}^2 \quad (2)$$

Cette équation dépend, tout comme l'énergie, du carré de la tension d'alimentation. En considérant un système alimenté directement par le super-condensateur sans l'intermédiaire d'un régulateur de tension qui maintiendrait la tension d'alimentation constante, la consommation est non linéaire et suit une loi en V^2. Le résultat est que la tension varie linéairement en fonction du temps au fur et à mesure que le tampon d'énergie se vide. En utilisant ce type d'alimentation, l'énergie disponible est donnée par la tension d'alimentation et le modèle de consommation se ramène à un modèle linéaire en fonction de cette tension. En caractérisant la pente de tension de chaque sous-système de notre plate-forme intermittente, il est possible de calculer la pente de tension globale. Plus précisément, en accumulant les pentes de tension des différents sous-systèmes actifs à un instant donné, on estime la consommation instantanée de notre système. Cette estimation peut être corrigée en ligne en se basant sur une mesure de la tension du super-condensateur, ce qui en pratique se fait simplement à l'aide d'un convertisseur analogique–numérique (ADC), qui est un composant standard sur ce type de plate-forme.

4.3 Modes de fonctionnement

Nous utilisons une notion de *mode* de fonctionnement d'un système intermittent similaire à ce qui est introduit dans les travaux précédents [5]. Un *mode* est un état du système caractérisé par la liste des sous-systèmes actifs, ainsi que par leur pente de tension (un même sous-système peut avoir différentes pentes de tension selon son état). Ainsi pour chaque *mode*, on associe une pente de tension du système qui s'obtient en additionnant les pentes de tension des sous-systèmes actifs, et qui nous renseigne sur la puissance dynamique du système dans ce *mode*. Nous pouvons alors modéliser l'exécution d'un système intermittent comme une machine à état, où chaque transition marque l'activation ou la dés-activation d'un ou plusieurs sous-systèmes. Les sous-systèmes considérés sont les composants du microcontrôleur, ainsi que les périphériques internes et externes.

4.4 Utilisation du modèle en ligne

La notion de *mode* décrite précédemment ne permet pas de modéliser le système avec un grain très fin. En effet, si l'on souhaite pouvoir utiliser le modèle en ligne ou bien dans un modèle formel plus complet, il n'est pas envisageable d'utiliser la notion de mode au niveau des modes atomiques du système, par exemple l'exécution d'une addition entre deux registres alors qu'une écriture est en cours sur la FRAM externe. En adoptant un grain plus gros mais compatible avec nos objectifs, le calcul de la tension aux bornes du super-condensateur, et donc l'estimation de l'énergie restante du système, ne sera qu'une valeur approchée. De plus, le modèle présenté n'intègre pas le fait que le système intermittent continue de récolter de l'énergie lorsqu'il exécute l'application.

Nous pouvons gérer ces erreurs en effectuant de manière régulière des mesures de la tension d'alimentation. Comme décrit précédemment, cela nous renseigne directement sur l'énergie disponible. En ligne, cette mesure permet d'effacer les erreurs accumulées à la fois du fait de la granularité du modèle, et de la non prise en considération de la récolte d'énergie. Elle devra cependant être effectuée avec modération car elle est elle-même consommatrice d'énergie.

Il est à noter qu'on pourra s'assurer que l'estimation de la quantité d'énergie restante est toujours pessimiste : il suffit de maximiser les pentes de consommation des modes.

4.5 Validation expérimentale.

Afin de mettre en œuvre le modèle de consommation présenté ci-dessus, nous avons caractérisé la pente de tension des différents sous-systèmes d'une plate-forme intermittente. La cible est une carte sur étagère de Texas Instrument, dont la référence est MSP430FR5994-launchpad. Elle est équipée d'un microcontrôleur ultra-basse consommation MSP430, une mémoire non volatile en technologie FRAM de 256KiO, une mémoire de travail en technologie SRAM de 8KiO, et d'un super-condensateur de 0,22F.

Pour modéliser ce système, nous avons tout d'abord identifié un ensemble de *modes*. Pour chaque *mode* identifié, nous avons ensuite conçu un *benchmark*, ce qui nous a permis de mesurer la tension aux bornes du super-condensateur lorsque le système est dans ce *mode*. Chaque exécution partait d'un état initial où le super-condensateur est complètement chargé et s'arrêtait lorsque la tension passait sous 1.9V, limite basse de fonctionnement du microcontrôleur. Pour piloter ces expérimentations et réaliser les mesures de façon non intrusive, nous avons réalisé une carte *ad-hoc*. En tout, nous avons caractérisé 38 *modes*, mais d'autres *modes* peuvent être facilement ajoutés notamment des *modes* liés à des périphériques extérieurs. Pour chaque *mode*, nous avons réalisé 5 mesures de tension au cours du temps. Les résultats présentés sont moyennés afin d'obtenir une pente de tension pour le *mode* en question.

La figure 1 montre les mesures réalisées pour différents *modes* (les échelles de temps sont différentes). On peut ainsi remarquer que le système peut fonctionner plus de 20 minutes en mode de basse consommation LPM0 (*Low Power Mode 0* : le système est sous tension, la RAM et le CPU conservent l'état des données mais l'horloge est arrêtée), ou près de 90s lors de l'utilisation d'un module de communication radio (LoRa). Pour chaque expérimentation, la droite associée au modèle est affichée en rouge et elle se superpose aux mesures, à l'exception d'une chute de tension à $t = 0s$, due à la résistance interne du super-condensateur.

Toutes ces expérimentations confirment qu'une estimation d'énergie basée sur une approximation de la tension dans les différents *modes* de fonctionnement par un modèle linéaire est pertinente. Cette approche est adaptée pour une utilisation dans un modèle de plus haut niveau pour modéliser un système complet, dont les modes de fonctionnement vont évoluer dans le temps.

5 Modélisation par réseaux de Petri

Réseaux de Petri Un réseau de Petri, également connu sous le nom de réseau de place/transition (PT), est l'un des nombreux langages de modélisation mathématique pour la description des systèmes concurrents distribués. Une place peut contenir un nombre quelconque de jetons. Un marquage M d'un réseau de Petri est un vecteur représentant le nombre de jetons de chaque place. Une transition est sensibilisée (elle peut être tirée) à partir de M s'il y a suffisamment de jetons dans ses places d'entrée pour que les consommations soient possibles. Le tir d'une transition t à partir d'un marquage M consomme un jeton dans chacune de ses places d'entrée, et produit un jeton dans chacune de ses places de sortie.

Réseaux de Petri de haut niveau . Les réseaux de Petri peuvent être classés en deux catégories : les réseaux de Petri ordinaires et les réseaux de Petri de haut niveau. Les réseaux de Petri de haut niveau ont été proposés pour la modélisation de problèmes scientifiques présentant des structures complexes et manipulant différents types d'expressions composées de variables et écrites selon une syntaxe prédéfinie. Dans les réseaux de haut niveau, chaque jeton peut porter

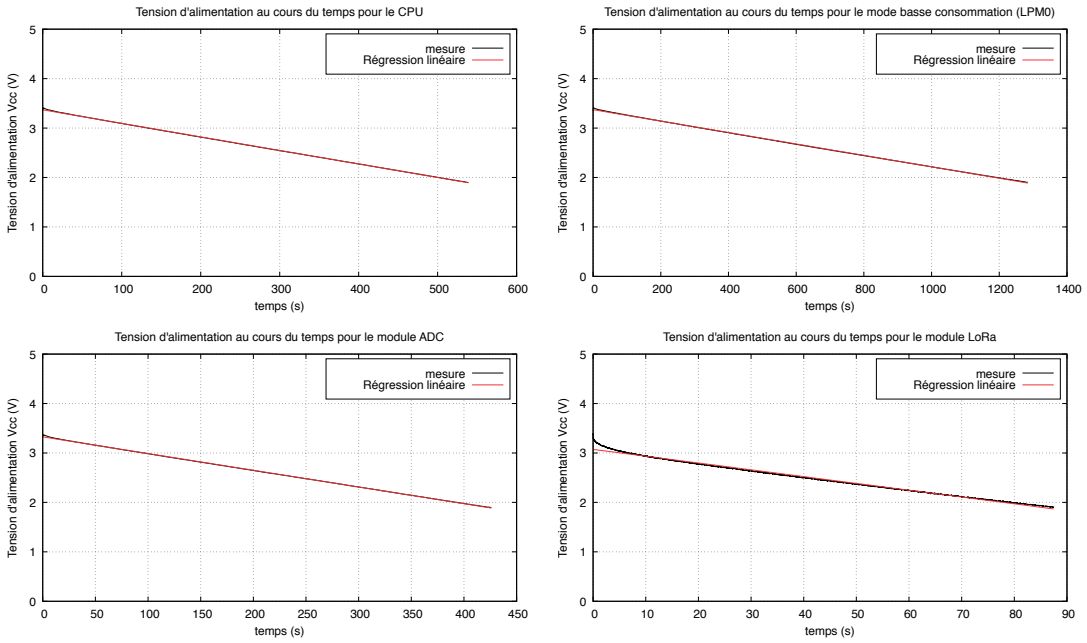

FIGURE 1 – Mesure de la chute de tension du super-condensateur en fonction du temps pour différents modes de fonctionnement (CPU alimenté, basse consommation, en utilisant l'ADC ou un module de communication radio LoRa).

des informations complexes qui, par exemple, peuvent décrire l'état complet d'un processus ou d'une base de données.

Dans cet article, nous considérons qu'à chaque transition est associée une précondition (garde) et une postcondition (mise à jour) sur un ensemble de variables (X). Une transition est sensibilisée (elle peut être tirée) s'il y a suffisamment de jetons dans ses places d'entrée et si la garde est vraie. Lorsque la transition est tirée, les mises à jour correspondantes sont exécutées en modifiant les valeurs des variables. Les variables prennent leurs valeurs dans un ensemble fini (comme un entier borné ou un type énuméré...), les gardes sont des expressions booléennes sur X et les mises à jour peuvent être décrites comme une séquence de code impératif exprimée dans un langage de programmation mais dont l'exécution est atomique du point de vue du déclenchement de la transition.

Réseaux de Petri temporels Les réseaux de Petri temporels (TPN) étendent les réseaux de Petri avec des intervalles temporels associés à des transitions. En supposant que la transition t a été sensibilisée pour la dernière fois au moment d et que les bornes de son intervalle de tir sont α et β, alors t ne peut pas tirer avant $d + \alpha$ et doit tirer au plus tard à $d + \beta$ à moins d'être désensibilisée par le tir d'une autre transition. L'exécution d'une transition ne prend pas de temps.

Réseaux de Petri temporels à coût Les modèles dits tarifés (priced) ou à coût temporisé sont adaptés à la représentation de systèmes temps réel dont le comportement est contraint par une certaine consommation de ressources (qu'il s'agisse d'énergie ou de temps CPU, par

exemple) et pour lesquels nous devons évaluer le coût total accumulé pendant leur exécution. De tels modèles peuvent décrire si l'évolution du coût au cours de l'exécution est causée par le maintien dans un état donné (coût continu) ou par l'exécution d'une action donnée (coût discret). Ainsi, les problèmes consistant à déterminer si le modèle peut atteindre certains états « bons » tout en maintenant le coût global sous une limite donnée ou à trouver le coût minimum pour atteindre un état objectif sont intéressants dans des applications telles que l'ordonnancement optimal ou la planification.

Dans la littérature, le problème du coût optimal a été étudié pour les automates temporisés pondérés, tarifés ou à coût [3, 10], les réseaux de Petri temporisés tarifés [1] et les réseaux de Petri Temporels à coût [9].

Le problème d'accessibilité à coût optimal est décidable pour tous ces modèles si tous les coûts sont négatifs et pour des coûts quelconques sous condition d'exemption de cycles de coûts négatifs.

L'approche par réseau de Petri présente plusieurs avantages majeurs. D'une part, les réseaux de Petri capturent intrinsèquement la concurrence et d'autre part la technique d'abstraction symbolique, appelé classes d'états, des réseaux de Petri Temporels ne nécessite aucune extrapolation, ni aucune hypothèse de bornage des variables d'horloge. Ansi, contrairement aux zones utilisées pour les automates temporisés, les classes d'états, même tarifées (c.-à-d. étendues avec des coûts), ne nécessitent aucune approximation (comme dans la k-extrapolation) pour garantir leur nombre fini et le coût optimal d'une classe tarifée est obtenu en minimisant sa fonction de coût sous les contraintes de la classe.

6 Modélisation d'un système intermittent réactif

6.1 Modélisation de la consommation d'énergie

Pour modéliser la consommation d'énergie par le système, nous utilisons la variable de coût. Nous utilisons pour cela le modèle de consommation décrit en section 4.2. À chaque *mode* du système, nous associons une place. Un jeton est positionné dans cette place lorsque le mode devient actif, et retiré lorsqu'il devient inactif. Ces mouvements de jeton sont contrôlés par la partie du modèle qui décrit l'exécution du programme et le comportement des périphériques. Lorsqu'un jeton est positionné dans une place de *mode*, la dynamique de la variable de coût est modifiée : elle augmente avec une pente correspondant à la pente de tension du mode. Il est bien sûr possible que plusieurs modes soient actifs simultanément, la dynamique globale étant alors obtenue par addition.

Considérons l'exemple de la figure 2 qui modélise une application composée de 2 tâches. Dans un souci de simplicité, chaque tâche est associée à un *mode*. Lorsque la transition $ChooseTask_1$ est tirée, le système commence à exécuter la tâche 1 et le *mode* $Mode_1$ devient actif. Pour plus de lisibilité, le marquage de la place $Mode_1$ est modifiée directement à l'aide d'une action (en bleu sur la figure), plutôt qu'avec un arc. De la même façon, lorsque la transition $ChooseTask_2$ est tirée, le système commence à exécuter la tâche 2 et le *mode* $Mode_2$ devient actif. Le taux de variation de la variable de coût c est alors définie par :

$$\dot{c} = \sum_m penteMode_m \times Mode_m$$

La variable de coût augmente donc, à chaque unité de temps, de la valeur de la pente de tension des différents modes actifs. Dans cet exemple, les *modes* permettent de modéliser la

consommation d'énergie induite par l'exécution d'une tâche en fonction de son temps d'exécution.

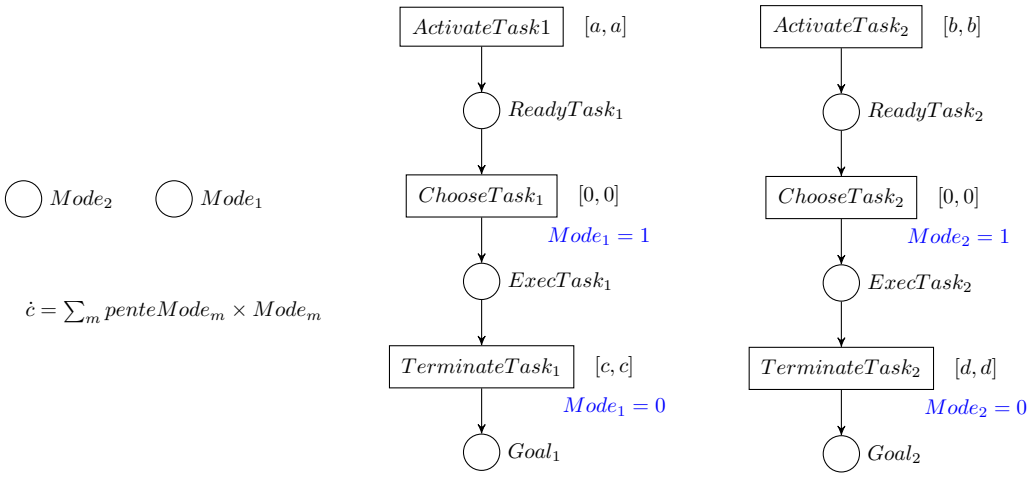

FIGURE 2 – Modélisation de 2 tâches simples associées à un mode, en utilisant un réseau de Pétri temporisé à coût

6.2 Prise en considération de l'erreur sur le modèle

Comme décrit dans la section 4.4, le modèle d'énergie utilisé est approché, mais il peut être corrigé en ligne à l'aide de mesures. Il faut cependant espacer les mesures autant que possible car celles-ci consomment de l'énergie pour des activités qui ne font pas progresser les objectifs fonctionnels du système. Ce problème peut être résolu à l'aide d'un modèle complet du système.

En procédant d'une manière similaire au modèle de consommation de l'énergie, à l'aide de mesures expérimentales sur la dynamique de l'erreur pour les différents *modes* du système, il est possible de borner l'erreur accumulée au cours du temps par le modèle de consommation d'énergie (à ce stade, la récolte d'énergie n'est pas associée à un prédicteur, et elle ne peut donc pas être prise en considération). Cette dynamique peut ensuite être intégrée au modèle complet. Elle ne peut cependant pas être intégrée de la même façon que la consommation d'énergie, sous la forme d'une variable de coût, car dans notre formalisme, une seule variable de coût peut être utilisée. Elle doit donc être intégrée sous la forme d'une variable discrète, comme illustré par la figure 3. Dans ce réseau, le déclenchement de la mesure est associé au franchissement d'un seuil par une variable modélisant l'erreur accumulée ($Error$). La variable $Error$ est incrémentée lorsque la transition $TerminateTask_1$ (resp. $TerminateTask_2$) est tirée, modélisant ainsi l'erreur maximale accumulée lors de l'estimation de l'énergie consommée par la tâche 1 (resp. 2). Lorsque la valeur de $Error$ dépasse le seuil visé, la transition $Mesure$ est tirée. L'intervalle de tir de cette transition est $[0, 0]$ afin qu'elle soit tirée dès qu'elle est sensibilisée. Lorsqu'elle est tirée, la variable $Error$ est remise à 0. On pourrait également appliquer au tir de cette transition un coût fixe, qui modélise l'énergie consommée pendant la mesure de la tension d'alimentation. C'est une approche alternative à la modélisation de la consommation par mode, qui semble pertinente pour les activités qui ne durent pas dans le temps et n'ont pas d'effet sur le flot de contrôle général du système.

Sur un tel modèle, il est possible de demander à Romeo d'engendrer une trace d'énergie

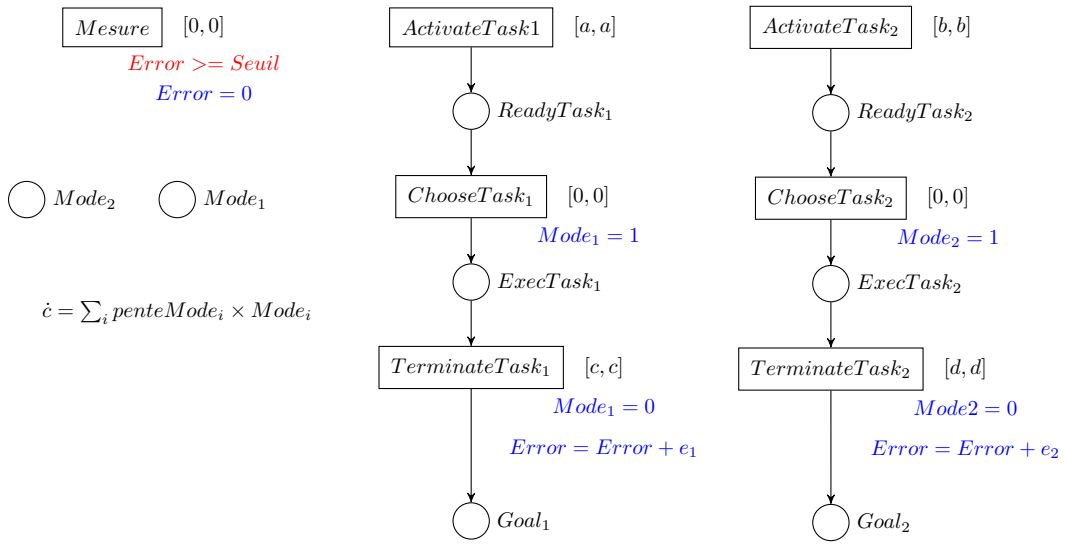

FIGURE 3 – Modélisation des 2 tâches simples de la figure 2, en rajoutant la gestion de l'erreur de mesure. La condition en rouge correspond à la garde.

minimale pour atteindre un certain marquage, par exemple 2 jetons dans $Goal_1$. Ici, le système n'a pas de décision d'allocation de ressource à prendre et donc la trace d'énergie minimale est la même que la trace d'énergie maximale. La trace permet en revanche de déterminer les instants auxquels déclencher des mesures pour garder l'erreur sous le seuil cible.

6.3 Modélisation d'un système intermittent

***Checkpointing* et exécution.** Les systèmes intermittents alternent des cycles d'exécution avec des cycles de recharge d'énergie et l'application doit pouvoir être arrêtée, puis reprendre son exécution lorsqu'il y a de nouveau de l'énergie. Le fait de sauver l'état dans une mémoire non volatile juste avant une perte d'énergie s'appelle le *checkpointing*, les points de sauvegarde (*checkpoints*) pouvant être placés statiquement dans le code, ou dynamiquement quand un seuil bas d'énergie est détecté.

Nous pouvons modéliser ce mécanisme avec les réseaux de Petri temporels. Une variable discrète représente l'énergie du système, et lorsque celle-ci atteint un seuil bas, on tire une transition modélisant l'opération de *checkpointing* et on pause l'exécution à l'aide d'arcs inhibiteurs temporisés. Le tir de cette transition va recharger le réservoir d'énergie et l'exécution pourra reprendre exactement là ou elle s'est arrêtée. La modélisation de ces points de sauvegarde permet de pouvoir modéliser le fonctionnement d'une application sur plusieurs cycles de charge/décharge propre aux systèmes intermittents.

La figure 4 montre la modélisation de l'opération de *checkpointing*. On peut également insérer un paramètre pour les bornes de l'intervalle de tir de la transition représentant le redémarrage du système après un *checkpoint* afin d'obtenir des contraintes en temps sur la recharge de l'énergie du système. Cela correspond à la transition *Recharge* de la figure 4. Il est vrai que les systèmes intermittents ne sont pas adaptés aux contraintes temporelles, mais avoir cette information peut aider au dimensionnement du tampon d'énergie dans les phases de conception.

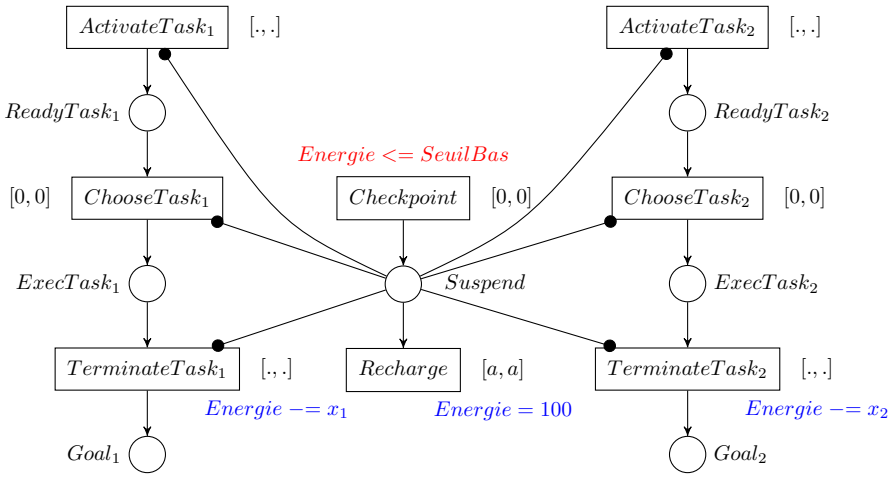

FIGURE 4 – Exemple de la modélisation du *checkpointing*

Dualité énergie/exécution. En premier lieu, on pourrait vouloir utiliser le coût pour modéliser l'énergie de notre système et minimiser sa consommation. Cependant pour un système intermittent, minimiser la consommation d'énergie n'est pas forcément la solution. D'une part, si l'énergie n'est pas utilisée, celle-ci est perdue et nous souhaitons plutôt utiliser au maximum l'énergie disponible. D'autre part, la minimisation de l'énergie utilisée peut conduire à un élagage trop agressif dans le modèle, des tâches trop gourmandes en énergie n'étant jamais appelées, bien qu'elles soient indispensable au bon fonctionnement du système complet.

La figure 5 montre un exemple avec 2 tâches en concurrence, dont le temps d'exécution est identique (10 unités de temps). La tâche représentée par la séquence (t0, t2, t4) consomme beaucoup plus d'énergie que l'autre séquence (t1, t3, t5). Celà est représenté par le coût à travers la dynamique de la variable c qui va augmenter de 100 unités pour chaque unité de temps avec un jeton dans la place p_1 contre seulement 10 unités avec un jeton dans p_2.

Dans cette configuration, la trace minimisant la valeur du coût pour un nombre fixé de jetons dans la place p_5 passera à chaque fois par le chemin du bas.

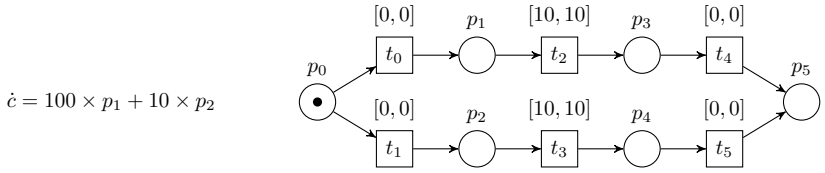

FIGURE 5 – Exemple de 2 exécutions concurrentes avec des coûts différents.

Nous proposons alors d'utiliser le coût pour modéliser la consommation d'énergie mais avec un système de récompense pour chaque tâche. La récompense est exprimée sous la forme d'une constante qui est déduite à la valeur du coût lorsqu'une tâche est exécutée. La valeur de cette constante est une donnée fonctionnelle qui mesure l'importance de la tâche pour l'application.

6.4 Synthèse de trace

Nous avons utilisé les éléments de modélisation présentées aux sections 6.1, 6.2 et 6.3 afin de modéliser une étude de cas complète. La plate-forme matérielle est le MSP430FR5994 présenté en section 4.5. L'application est un programme prototype d'analyse de son. Il comprend plusieurs tâches, de l'acquisition des échantillons à l'envoi des résultats par un protocole sans-fil. Nous avons modélisé le système puis l'avons analysé à l'aide de Romeo. L'analyse a consisté à synthétiser des traces d'exécution à coût optimal.

Dans l'état initial, le super-condensateur est plein. Chaque tâche est associée à un *mode*. L'exécution se déroule de la manière suivante :

1. Acquisition périodiques des données : récolte des sons via un microphone.
2. Traitement des données : utilisation d'un accélérateur matériel pour les opérations de FFT (Transformation de Fourier rapide).
3. Analyse des données : Algorithme d'analyse du son basé sur les résultats des FFT.
4. Finalisation selon l'énergie disponible :
   — Mise en mémoire non-volatile interne.
   — Mise en mémoire non-volatile externe.
   — Émission des données sur une liaison sans-fil.

Un arbre de décision comme décrit sur la figure 6 permet à l'outil d'explorer les différents ordonnancements de tâches possibles, notamment le choix entre la mise en mémoire non-volatile qui consomme peu d'énergie et l'émission de la donnée qui consomme beaucoup plus. Un jeton est remis dans la place *controller* à chaque fois qu'une tâche préalablement choisie est finie.

La mémoire non-volatile externe a une plus grande capacité que la mémoire non-volatile interne, cependant son utilisation coûte plus cher en énergie. La taille de la mémoire a été modélisée comme une variable discrète avec un certain seuil qui conditionne son accessibilité. Ainsi on alloue 10 blocs mémoire pour la mémoire non-volatile interne et 100 blocs pour la mémoire non-volatile externe. Chaque donnée traitée occupe un bloc mémoire. L'émission d'une donnée ne peut pas être interrompue. Il y a donc un seuil minimal d'énergie nécessaire avant de pouvoir exécuter la tâche correspondante. La taille d'un paquet LoRa (le protocole de communication utilisé par notre étude de cas) peut varier. Nous avons donc donné à l'outil le choix entre plusieurs tailles de données à transmettre (1, 2 ou 3 données traitées). Plus la taille est grande, plus l'émission consomme d'énergie mais plus cela est récompensé. Si les données utiles (*payload*) augmentent proportionnellement, ce n'est pas le cas de la trame complète de transmission, et donc du coût associé.

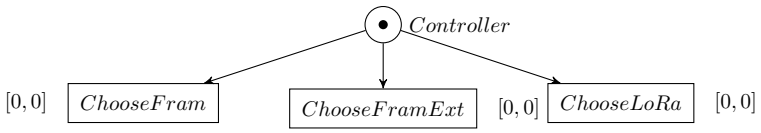

FIGURE 6 – Exemple d'arbre de décision

Le but de l'application étant d'émettre les données traitées, nous appliquons une grande récompense à l'émission de données et une plus faible récompense à la mise en mémoire des données traitées. Pour synthétiser la trace, nous utilisons la propriété de minimisation du coût pour un état, cet état étant atteint après un nombre de cycle exécution/recharge effectué comme présenté à la figure 4.

Ainsi pour un cycle d'exécution, nous obtenons une trace qui indique la mise en mémoire non-volatile externe puis l'envoi de deux fois une donnée et d'une fois 2 données par la liaison LoRa. Il faut tout de même prendre en compte que l'outil Roméo indique une unique solution. Si il existe plusieurs traces amenant à l'état désiré avec le même coût final, nous n'aurons accès qu'à une seule solution. De plus, la valeur de récompense pour une tâche est actuellement arbitraire, en modifiant les valeurs des récompenses la trace peut être profondément affectée.

7 Travaux futurs

La section précédente présente plusieurs modélisations relatives aux systèmes intermittents. Nos travaux futurs se focalisent sur l'extension des modèles présentés et la mise en pratique sur un système réel.

Extension à plusieurs cycles. Nos travaux actuels se limite à un seul cycle d'exécution mais peuvent être étendues à plusieurs cycles. Cependant, il faut prendre en compte l'explosion possible de l'espace d'état. Nous souhaitons poursuivre ces travaux en réduisant l'espace d'état et ainsi atteindre une trace optimal sur plusieurs cycles. On peut également envisager un point de vue de proche en proche afin de synthétiser une trace d'exécution sur plusieurs cycles. On aurait ainsi une approche optimisant l'exécution sur tous les cycles et une autre approche optimisant l'exécution sur chaque cycle jusqu'à l'état visé. Ces deux approches pouvant donner des résultats opposés, il est intéressant de les comparer et de les analyser.

Gestion de l'erreur. Même si le modèle de consommation proposé n'est pas précis à l'instruction près, nous avons montré qu'il est possible de gérer et minimiser l'erreur induite par le modèle à l'aide de mesure de tension. Les travaux futurs se tourneront vers une modélisation plus précise de cette erreur. En effet, nous supposons dans un premier temps qu'il est possible de modéliser l'erreur induite par le modèle de manière convenable avec un modèle linéaire. Une étude approfondi de ces dynamiques nous permettra d'améliorer la finesse de notre modèle.

Utilisation du modèle en-ligne. Une utilisation du modèle de consommation d'énergie en ligne et hors ligne est à l'étude. Nous souhaitons mettre en pratique ce modèle sur une plateforme intermittente en développant l'application de reconnaissance du chant des oiseaux citée dans la section 6.4.

8 Conclusion

L'émergence des systèmes intermittents apporte son lot de défis et d'opportunités notamment dans l'étape d'exploration de l'espace de conception. Ces systèmes ne peuvent utiliser les méthodes et outils développés par les systèmes alimentés en continu car ceux-ci ne sont pas adaptées à la gestion des pertes d'alimentation. Nous introduisons dans ces travaux la synthèse de trace à coût optimal pour les systèmes intermittents réactifs. La synthèse de trace se base sur un modèle de consommation d'énergie simple et pertinent, utilisable à la fois en ligne et hors ligne. Les résultats obtenus peuvent servir de base afin de définir une politique d'ordonnancement qui tient compte simultanément des aspects réactifs et des contraintes énergétiques.

Références

[1] Parosh Aziz Abdulla and Richard Mayr. Priced timed Petri nets. *Logical Methods in Computer Science*, 9(4), 2013.

[2] Saad Ahmed, Abu Bakar, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. The betrayal of constant power × time : Finding the missing joules of transiently-powered computers. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES 2019, pages 97–109, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3) :297 – 322, 2004.

[4] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Hibernus++ : A self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 35(12), 2016.

[5] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus : Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embed. Syst. Letters*, 7(1), 2015.

[6] G. Berthou, P.-É. Dagand, D. Demange, R. Oudin, and T. Risset. Intermittent computing with peripherals, formally verified. In *Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2020.

[7] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. Sytare : A lightweight kernel for nvram-based transiently-powered systems. *IEEE Trans. Comput.*, 68(9), 2019.

[8] G. Berthou, K. Marquet, T. Risset, and G. Salagnac. Accurate power consumption evaluation for peripherals in ultra low-power embedded systems. In *Global Internet of Things Summit (GIoTS)*, 2020.

[9] Hanifa Boucheneb, Didier Lime, Olivier H. Roux, and Charlotte Seidner. Optimal-cost reachability analysis based on time Petri nets. In *18th International Conference on Application of Concurrency to System Design (ACSD'18)*, Bratislava, Slovakia, June 2018.

[10] Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In *Proceedings of the 28th International Conference on Computer Aided Verification (CAV'16)*, volume 9779 of *Lecture Notes in Computer Science*, pages 513–530, Toronto, Canada, July 2016. Springer.

[11] A. Cammarano, C. Petrioli, and D. Spenza. Online energy harvesting prediction in environmentally powered wireless sensor networks. *IEEE Sensors Journal*, 16(17), 2016.

[12] A. Colin and B. Lucia. Termination checking and task decomposition for task-based intermittent programs. In *Compiler Construction (CC)*, 2018.

[13] Behnam Dezfouli, Immanuel Amirtharaj, and Chia-Chi Li. EMPIOT : an energy measurement platform for wireless iot devices. *CoRR*, abs/1804.04794, 2018.

[14] Hiroshi Ishiwara. Ferroelectric random access memories. *J. Nanosc. Nanotech.*, 12(10) :7619–7627, October 2012.

[15] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *ACM Trans. Embed. Comput. Syst.*, 6(4), 2007.

[16] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo : A parametric model-checker for Petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom, March 2009. Springer.

[17] S. Liu, W. Zhang, M. Lv, Q. Chen, and N. Guan. LATICS : A low-overhead adaptive task-based intermittent computing system. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 39(11), 2020.

[18] K. Maeng, A. Colin, and B. Lucia. Alpaca : Intermittent execution without checkpoints. *Proc. ACM Program. Lang.*, 1, 2017.

[19] K. Maeng and B. Lucia. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Programming Language Design and Implementation (PLDI)*, 2020.

[20] B. Ransford, J. Sorber, and K. Fu. Mementos : system support for long-running computation on RFID-scale devices. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[21] E. Ruppel and B. Lucia. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Programming Language Design and Implementation (PLDI)*, 2019.

[22] Sivert T. Sliper, William Wang, Nikos Nikoleris, Alex S. Weddell, and Geoff V. Merrett. Fused : Closed-loop performance and energy simulation of embedded systems. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 263–272, Aug 2020.

[23] Milijana Surbatovich, Brandon Lucia, and Limin Jia. Towards a formal foundation of intermittent computing. *Proc. ACM Program. Lang.*, 4(OOPSLA), November 2020.

[24] P. Wägemann, C. Dietrich, T. Distler, P. Ulbrich, and W. Schröder-Preikschat. Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems. In *Euromicro Conf. on Real-Time Systems (ECRTS)*, volume 106, 2018.

[25] B. Yarahmadi and E. Rohou. Compiler Optimizations for Safe Insertion of Checkpoints in Intermittently Powered Systems. In *Embedded Computer Systems : Architectures, Modeling and Simulation (SAMOS)*, 2020.

[26] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester. InK : Reactive kernel for tiny batteryless sensors. In *Embedded Networked Sensor Systems (SenSys)*, 2018.