Ordonnancement et réflexion pour une prise en compte de la consommation énergétique

Maximilien Weinmann

Objectif de la recherche

Rendre un MCU + scheduler autonome à l’aide de la récupération d’énergie

Notre système

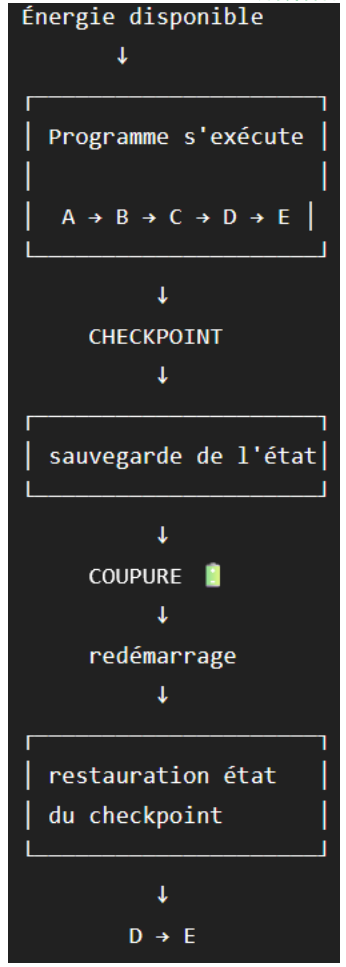
- MCU EDMS105N (ARM Cortex M0)
- PMIC
- Une cellule photovoltaïque
- Un supercondensateur
- Les divers éléments nécessaires à une application (capteur, transmetteur etc)

Checkpointer

- Le Checkpointing c’est quoi ?
  - c'est le fait de **sauvegarder l'état d'exécution du programme dans une mémoire non volatile avant que l'énergie ne disparaisse**

  - Les registres CPU, notamment le PC,
  - les variables présentes en RAM volatile
  - éventuellement certains états périphériques

- Des implémations réelles?
  - Mementos, Ratchet et Clank



Le problème de non-idempotence

- Pour illustrer un code **non idempotent**, imaginons qu'un checkpoint ait été effectué juste avant les lignes de code ci-dessous, et qu' un deuxième checkpoint, suivi d'une coupure de courant surviennent juste après la dernière ligne, entraînant ainsi la réexécution du code. Imaginons également que les valeurs des variables soient sauvegardées dans une mémoire non volatile.

- *a = *b; // READ de b
- *b = 42; // WRITE de b

Solution Ratchet

Title 2

- Pour éviter cela, le compilateur examine les opérations de **lecture et d'écriture** effectuées sur la mémoire non volatile et s'assure qu'aucune section ne contient une dépendance **write-after-read (WAR)** sur une même adresse mémoire.
- En divisant les programmes en ces sections réexécutables et en les reliant par des *checkpoints* contenant l'état de la mémoire volatile, **Ratchet** peut prendre en charge des programmes existants de n'importe quelle longueur, indépendamment de la source d'énergie utilisée.

- Les checkpointers plus récents se servent de ce mécanisme développé par les développeurs de Ratchet, ou par des versions améliorées telles que celle de Clank

Task Based Energy Awareness

- Chaque tâche doit être entièrement exécutée ou ne pas être exécutée du tout, les coupures de courant n'étant autorisées qu'entre les tâches.

- Ce modèle garantit la progression du calcul et la cohérence des données. Les résultats n'étant validés qu'une fois la tâche terminée avec succès, cette approche offre de fortes garanties d'atomicité tout en réduisant le recours au checkpointing et son overhead.

- Implémentations :
  - Alpaca, InK, AsTAR, Coala, CatNap, Culpeo...

Exemple

Coala

- Les développeurs de **Coala** sont arrivés à la réflexion suivante :
- Lorsque l'ordonnancement des tâches entraîne une transition d'une tâche à une autre, le système doit sauvegarder les modifications effectuées en mémoire non volatile afin de maintenir la cohérence du programme. Cette opération génère cependant un **overhead**.
- Plus les transitions entre les tâches sont fréquentes, plus le coût en termes d'overhead durant l'exécution est important. Cela incite donc les programmeurs à créer des **tâches plus grandes**, afin de réduire le nombre de transitions et, par conséquent, de minimiser cet overhead.
- **Nouveau problème: les tâches sont trop grandes, elles ne peuvent plus terminer**

Exemple

Coala

- Le runtime de **Coala** fusionne dynamiquement plusieurs petites tâches statiques afin de réduire l'overhead, et divise les tâches plus grandes lorsqu'elles dépassent la capacité énergétique fournie par une seule charge du buffer.
- Coala s'appuie sur **l'historique récent de l'exécution** comme métrique afin de réduire l'overhead lié aux transitions entre les tâches.
- Pour utiliser Coala, le programmeur doit transformer le code C classique en tâches en encapsulant le code dans un ensemble de fonctions de haut niveau (*top-level functions*), en définissant l'ordre d'exécution et le flux de contrôle entre ces tâches, et en annotant les accès mémoire qui manipulent des données partagées entre les tâches.

Stratégie de regroupement

Coala

- Energy-Oblivious: avec cette stratégie, le nombre de tâches à fusionner, augmente d'une constante **x** après la validation réussie d'une séquence de tâches fusionnées, et diminue de **x** lorsqu'une séquence n'arrive pas à se terminer en raison d'une coupure de courant.

- Weighed Energy-Guided: Si une tâche nécessite 10 secondes pour s'exécuter tandis qu'une autre n'en nécessite qu'une seule, considérer ces deux tâches de manière équivalente ne permettrait pas de représenter correctement la charge de travail réelle de la séquence fusionnée ni son historique d'exécution. Pour résoudre ce problème, **WEG** attribue un poids unique à chaque tâche, reflétant son **coût énergétique**.
- De plus, lorsqu'une coupure de courant survient pendant l'exécution d'un groupe de tâches fusionnées, celui-ci est entièrement séparé et les tâches qui le composent sont à nouveau considérées comme des **tâches individuelles**.

Benchy EAS et InK

State	Current Consumption					Execution Time				
	APP	EAS - FLASH	EAS + FLASH	InK - FLASH	InK + FLASH	APP	EAS - FLASH	EAS + FLASH	InK - FLASH	InK + FLASH
Temperature measurement	1.12 mA	1.4 mA	2.1 mA	1.2 mA	1.61 mA	1.22 ms	1.73 ms	12.83 ms	1.45 ms	11.81 ms
Store 1	1.9 mA	2.16 mA	2.16 mA	1.97 mA	1.97 mA	5.84 ms	7.69 ms	7.69 ms	7.26 ms	7.26 ms
Store 2	1.9 mA	2.16 mA	2.43 mA	1.97 mA	2.24 mA	5.84 ms	7.69 ms	12.23 ms	7.26 ms	11.2 ms
Compute	1.94 mA	2.18 mA	2.58 mA	2.02 mA	2.3 mA	6.08 ms	7.75 ms	12.4 ms	7.33 ms	11.2 ms
Transmit	0.47 mA	0.61 mA	0.84 mA	0.51 mA	0.72 mA	28.41 ms	34.31 ms	36.5 ms	32.65 ms	34.38 ms
Blinking LED	0.22 mA	0.48 mA	0.48 mA	0.29 mA	0.29 mA	502.6 ms	506.1 ms	506.1 ms	504.8 ms	504.8 ms

Qu’intégrer d’autre dans notre système?

- Prédiction de l’ensoleillement pour savoir quand l’énergie sera disponible ?
- Exécuter des versions dégradées mais moins consommatrices des tâches ?

maximilien.weinmann@unamur.be

Thanks