Exercice “Automated discovery of interference patterns in multi-core architectures”

Les processeurs sont des systèmes complexes et l'exécution de code machine valide peut parfois engendrer des interférences diverses en son sein. Ici l'idée générale est de définir une boucle d'exploration qui itérativement explore la diversité de ces comportements, en jouant sur le code exécuté. (voir l’offre d’emploi pour plus d’information à ce sujet).

Le simulateur utilisé ici est simpliste, répliquant le fonctionnement du microprocesseur 8085. Les interférences potentielles sont donc très limitées par rapport à un processeur multi-cœur, mais la démarche ici est avant tout de créer un prototype de cette boucle d'exploration.

La première partie de l’exercice consiste à modifier le simulateur comme bon vous semble pour extraire des métriques potentiellement intéressantes à la fin de l'exécution d'un programme donné, servant à mesurer de potentielles interférences (par exemple certaines latences, temps d'exécution, compteurs internes etc). Il s'agit d'un simulateur qui n'a pas été créé pour l'exercice donc adaptez-le à vos besoins.
Disponible ici: https://github.com/wd400/8085simCli

Pour restreindre le problème, les programmes considérés seront uniquement des suites d'au plus 100 opérations **LHLD, SHLD et NOP**.

Exemple d'un programme:
SHLD 2024
NOP
LHLD 2024
SHLD 2024
NOP
LHLD 2024

Il n'est pas important de comprendre la programmation pour le microprocesseur 8085, voyez tout le code fourni simplement comme un système complexe capable de prendre en entrée une liste de LHLD, SHLD et NOP puis donne en sortie les métriques pertinentes que vous aurez définies.

Puis, dans une deuxième partie, l’exercice consiste à implémenter une boucle d’exploration simple basée sur le formalisme IMGEP (voir par ex. la section 3.1 de la thèse de M. Etcheverry sur les fondements de ce formalisme). L’objectif de cette boucle d’exploration est de chercher des programmes menant à une diversité de signatures comportementales selon des métriques de fonctionnement du microprocesseur que vous aurez extraites de la première partie.

Nous fournissons ci-dessous le pseudo-code d’un IMGEP canonique.

Pseudocode IMGEP simplifié:
Init:

1. un buffer des correspondances (code, signature comportementale), initialement vide ou initialisé avec quelques codes aléatoires et leurs signatures comportementales
2. Un espace de signatures comportementales à définir à votre convenance. Il peut par exemple s'agir d'un espace 2D avec une dimension correspondant à des latences et une autre au temps d'exécution
3. Une méthode pour tirer aléatoirement une signature comportementale dans cet espace
4. Une méthode permettant de déduire un code pouvant potentiellement induire une signature comportementale S désirée. Une solution courante à ce problème est de sélectionner l'élément du buffer existant dont la signature est la plus proche de S et de "muter" le code correspondant (par exemple en modifiant une des instructions de façon pseudo-aléatoire).
5. Un simulateur d'exécution capable de fournir la signature comportementale d'un code donné.

Boucle à répéter N fois:

1. Choisir une nouvelle signature comportementale aléatoirement
2. Identifier le code connu le plus proche de cette signature
3. Muter légèrement ce code (voir point 4 ci-dessus)
4. Exécuter ce code avec le simulateur et calculer la signature comportementale correspondante
5. ajouter au buffer (code, signature obtenue)

L'objectif de cette boucle d'exploration est d'explorer l'espace de codes afin de découvrir une diversité de signatures comportementales possibles. Cette méthode a prouvé son efficacité à résoudre ce problème dans de nombreux domaines, permettant de bien mieux couvrir l'espace des signatures comportementales qu'une méthode aléatoire. Dans le cadre du problème exposé ci-dessus, cette dernière consisterait à générer des codes aléatoires et à calculer leurs signatures comportementales, ce qui a peu de chance de mener à une grande diversité de signatures (voir figure 1.2 de la thèse de M. Etcheverry pour une illustration de ce problème).

L'objectif principal de cet exercice est de démontrer votre capacité à comprendre et à résoudre en partie le problème posé dans un temps limité. Nous vous conseillons d'y passer entre 2 heures et 5 heures en fonction de vos disponibilités. Nous sommes conscients du fait que réaliser "parfaitement" cet exercice nécessiterait beaucoup plus de temps. Il sera donc normal que vous ne réussissiez pas à tout faire, et nous évaluerons en particulier la façon dont vous saurez expliquer les problèmes rencontrés et les solutions que vous avez envisagées, même si celles-ci n'ont pas pu être implémentées par manque de temps. Vous pourrez aussi commenter sur de possibles limitations ou extensions, en vous limitant à une ou deux pages pour l'ensemble de vos commentaires.

Si des points de l'énoncé ne sont pas clairs, vous pouvez nous contacter par email à:

- "Zacharie Bugaud" <zacharie.bugaud@inria.fr>;
- "Clement Moulin-Frier" <clement.moulin-frier@inria.fr>;

Merci de bien contacter ces deux emails et de rester concis. Vos potentielles questions et nos réponses seront envoyées aux autres candidat.e.s concerné.e.s afin que tout le monde ait accès aux mêmes informations.

Vous pouvez écrire le code nécessaire dans le fichier exploration.ipynb du dépôt mentionné ci-dessus.
compiler.py montre comment compiler un programme puis python3 execute.py program.txt permet de l'exécuter avec le simulateur.
Vous pouvez également écrire vos commentaires et réflexions sur cet exercice dans le notebook, en vous limitant à une ou deux pages maximum.

Une fois terminé envoyez simplement un zip comprenant les modifications sur le simulateur, l'exploration dans exploration.ipynb puis quelques notes sur votre démarche.