Rapport CA

shiyao chen 28707756

May 2024

Contents

1 Introduction **3**

2 Architecture de la VM **3**
2.1 Fonctionnalités fonctionnelles 3
2.2 Fonctionnalités non fonctionnelles 5

3 Comparaison avec la version C (NativeVM) **6**

4 Extensions et jeu de tests **6**

5 Optimisations potentielles **6**

6 Conclusion **7**

1 Introduction

Dans le cadre du projet 2 de CA pour l'année académique 2023-2024, nous avons été chargés de réaliser une implémentation en langage Eclat d'une machine virtuelle appelée VM3I018. Cette machine est destinée à exécuter du bytecode généré à partir du langage MicroJS sur un circuit FPGA. Notre objectif était de reproduire une expérience similaire à celle réalisée précédemment avec la machine virtuelle OCaml en Eclat, permettant ainsi l'exécution de programmes MicroJS sur un circuit FPGA.

Ce rapport décrit notre travail sur ce projet, l'architecture de la machine virtuelle VM3I018, les défis rencontrés, les fonctionnalités implémentées avec succès ainsi que celles qui restent à finaliser en langage, ainsi que les résultats obtenus. Nous discuterons également des comparaisons avec la version C de la machine virtuelle (NativeVM), des extensions potentielles, des optimisations envisagées et des conclusions tirées de ce projet.

2 Architecture de la VM

La machine VM3I018 est relativement simple. Son architecture repose sur plusieurs registres principaux qui jouent des rôles cruciaux dans son fonctionnement. Parmi ces registres, on retrouve le pointeur de pile (sp), l'environnement lexical courant (env), le pointeur de code (pc), le pointeur vers le bloc d'activation suivant (fp), le pointeur sur la dernière variable globale allouée (gp), le pointeur vers la fin du tas (hp), le buffer d'écriture postée (write buf), et le registre finished.

Les instructions de la machine sont conçues pour permettre des opérations de manipulation de la pile, de gestion des variables globales, de contrôle du flux d'exécution, et d'appels de fonctions. Ces instructions incluent des opérations telles que PUSH, POP, GALLOC, GSTORE, GFETCH, JUMP, JTRUE, JFALSE, CALL, PUSH FUN, STORE, GFETCH, et RETURN. Chaque instruction joue un rôle spécifique dans le fonctionnement global de la machine virtuelle.

Nous avons choisi d'implémenter la machine VM3I018 en utilisant le langage Eclat, qui offre une combinaison unique de fonctionnalités fonctionnelles et impératives pour la programmation de circuits FPGA. Dans notre implémentation, le bytecode est représenté comme un tableau d'instructions de taille fixe.

2.1 Fonctionnalités fonctionnelles

Nous avons réussi à implémenter les principales fonctionnalités de la VM3I018 en Eclat, notamment des instructions de base telles que PUSH, POP, JUMP, JTRUE, JFALSE, CALL, RETURN. Chacune de ces instructions est exécutée séquentiellement en appelant la fonction vm_run_instr. Elles permettent de manipuler la pile, les variables globales, d'effectuer des sauts conditionnels et inconditionnels, ainsi que des appels de fonction. En outre, les autres fonctionnalités

telles que l'initialisation du bytecode, le compilateur Eclat et la simulation logicielle sont déjà terminées.

Exemple de compilation de MicroJS: Nous avons fourni un exemple de compilation d'un programme MicroJS en bytecode utilisant le compilateur MicroJS. Le bytecode généré est ensuite utilisé comme entrée pour notre machine virtuelle VM3I018 en Eclat

```
let load_bytecode() =
  code.(0) <- I_GALLOC ();      (* N := 10 *)
  code.(1) <- I_PUSH (Int 10);
  code.(2) <- I_GSTORE(0);

  code.(3) <- I_GALLOC ();      (* ACC := 0 *)
  code.(4) <- I_PUSH (Int 0);
  code.(5) <- I_GSTORE(1);

  (* "debut boucle:" *)
  code.(6) <- I_GFETCH(0);      (* ACC := ACC + N *)
  code.(7) <- I_GFETCH(1);
  code.(8) <- I_PUSH(Prim (P_ADD()));
  code.(9) <- I_CALL(2);
  code.(10) <- I_GSTORE(1);

  code.(11) <- I_GFETCH(0);     (* goto "fin boucle:" si (N == 0) *)
  code.(12) <- I_PUSH (Int 0);
  code.(13) <- I_PUSH (Prim (P_EQ()));
  code.(14) <- I_CALL(2);
  code.(15) <- I_JTRUE 22;

  code.(16) <- I_GFETCH(0);     (* N := N - 1 *)
  code.(17) <- I_PUSH(Int 1);
  code.(18) <- I_PUSH(Prim (P_SUB()));
  code.(19) <- I_CALL(2);
  code.(20) <- I_GSTORE(0);

  code.(21) <- I_JUMP(6);      (* goto "debut boucle:" *)

  (* "fin boucle:" *)
  code.(22) <- I_GFETCH(1);
  code.(23) <- I_POP() ;;

  (* devrait afficher 55 *)
```

Il affiche comme:

```
START (cy=0)
[sp:0|env:0|pc:0|fp:0]|gp:0|hp:0
```

```
[sp:0|env:0|pc:1|fp:0]|gp:1|hp:1
[sp:1|env:0|pc:2|fp:0]|gp:1|hp:1
[sp:0|env:0|pc:3|fp:0]|gp:1|hp:1
[sp:0|env:0|pc:4|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:5|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:6|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:7|fp:0]|gp:2|hp:2
[sp:2|env:0|pc:8|fp:0]|gp:2|hp:2
[sp:3|env:0|pc:9|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:10|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:11|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:12|fp:0]|gp:2|hp:2
[sp:2|env:0|pc:13|fp:0]|gp:2|hp:2
[sp:3|env:0|pc:14|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:15|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:16|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:17|fp:0]|gp:2|hp:2
[sp:2|env:0|pc:18|fp:0]|gp:2|hp:2
[sp:3|env:0|pc:19|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:20|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:21|fp:0]|gp:2|hp:2
          ...
[sp:0|env:0|pc:6|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:7|fp:0]|gp:2|hp:2
[sp:2|env:0|pc:8|fp:0]|gp:2|hp:2
[sp:3|env:0|pc:9|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:10|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:11|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:12|fp:0]|gp:2|hp:2
[sp:2|env:0|pc:13|fp:0]|gp:2|hp:2
[sp:3|env:0|pc:14|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:15|fp:0]|gp:2|hp:2
[sp:0|env:0|pc:22|fp:0]|gp:2|hp:2
[sp:1|env:0|pc:23|fp:0]|gp:2|hp:2
Popped value: 55
END (cy=1192)
```

2.2 Fonctionnalités non fonctionnelles

Cependant, nous avons rencontré quelques difficultés dans l'implémentation de la fermeture pour l'instruction I_CALL. La fermeture est un concept crucial dans de nombreuses machines virtuelles, y compris VM3I018, car elle permet de représenter des fonctions avec leur environnement lexical. Dans le cas de l'instruction I_CALL, si la valeur dépilée est une fermeture, la machine VM3I018 doit allouer un nouveau bloc de mémoire pour stocker l'environnement et les

valeurs des arguments dépilés, sauvegarder les valeurs des registres dans le bloc d'activation courant, allouer un nouveau bloc d'activation, puis sauter à l'adresse du code de la fermeture avec l'environnement courant de la fermeture. Cependant, je suis bloqué dans la gestion du tas (heap), notamment lors de l'allocation de nouveaux blocs pour les fermetures.

À cause de cela, des erreurs se produisent lors de l'exécution des programmes qui utilisent des fonctions ou des fermetures.

3 Comparaison avec la version C (NativeVM)

Tout d'abord, en ce qui concerne la structure et les fonctionnalités de base, les deux versions partagent des concepts similaires tels que les registres principaux, les instructions de manipulation de la pile et des variables, ainsi que les mécanismes de contrôle du flux d'exécution. Cependant, les détails de mise en œuvre peuvent différer en raison des différences entre les langages C et Eclat.

Nous avons constaté que l'implémentation en Eclat présentait des avantages et des inconvénients par rapport à la version C de la machine virtuelle. D'une part, l'utilisation de types somme en Eclat nous a permis de structurer notre code de manière similaire à la version C, facilitant ainsi la compréhension et la maintenance du code. Cependant, la complexité de certains aspects de la machine, tels que la gestion des fermetures, a rendu l'implémentation en Eclat plus difficile par rapport à la version C.

4 Extensions et jeu de tests

Pour étendre notre implémentation, nous pourrions envisager d'ajouter des fonctionnalités supplémentaires telles que la gestion dynamique de la mémoire, la gestion des exceptions ou la prise en charge de primitives supplémentaires. Ces extensions permettraient d'améliorer les capacités de la machine virtuelle et d'adresser un plus large éventail de scénarios d'utilisation.

De plus, la création d'un ensemble de tests exhaustif serait essentielle pour garantir la fiabilité et la robustesse de notre implémentation. Ces tests devraient couvrir différents cas d'utilisation, notamment des cas de bord, des scénarios de fonctionnement normal et des situations d'erreur. Un jeu de tests complet permettrait de vérifier le bon fonctionnement de chaque fonctionnalité de la machine virtuelle et d'identifier rapidement les éventuels problèmes ou bogues.

5 Optimisations potentielles

Une optimisation potentielle que nous pourrions envisager est l'utilisation de l'accès en parallèle aux tableaux, comme suggéré dans l'Annexe D. En exploitant cette fonctionnalité, nous pourrions améliorer les performances de notre implémentation en réduisant le temps d'accès aux tableaux, notamment lors de l'accès à des tableaux différents en même temps.

Cette optimisation pourrait être particulièrement bénéfique dans les cas où plusieurs accès mémoire sont effectués simultanément, ce qui est fréquent dans le fonctionnement de la machine virtuelle. En permettant à ces accès mémoire de se faire en parallèle, nous pourrions réduire les temps d'attente et améliorer l'efficacité globale de l'exécution des programmes.

6 Conclusion

En conclusion, malgré quelques difficultés rencontrées lors de l'implémentation de certains aspects complexes de la machine virtuelle, nous avons réussi à reproduire une partie des fonctionnalités de la VM3I018 en Eclat. Toutefois, des efforts supplémentaires sont nécessaires pour finaliser certaines fonctionnalités et pour explorer des possibilités d'optimisation afin d'améliorer les performances de notre implémentation.