Émulateur

Felix-Antoine Ouellet
Départment d'informatique
Université de Sherbrooke
felix-antoine.ouellet@usherbrooke.ca

Abstract

1 Introduction

blah

Le présent article à pour but de faire connaître les techniques utilisées dans le développement d'un émulateur pour une architecture virtuelle. La section 2 s'attardera justement à décrire l'architecture virtuelle ciblée. La section 3 mettra en relief diverses techniques venant du milieu des interpréteurs pouvant servir à mettre en place un émulateur. La section 4 décrira les principaux problèmes liés à la mise en place d'un recompilateur dynamique et les approches classiques utilisées pour les résoudre. Finalement, la section 5 résumera le contenu de l'article pour quiconque serait tenté de programmer un émulateur et ouvrira des portes vers d'autres techniques n'ayant pas été abordée dans le cadre du présent article.

2 Chip16

2.1 Historique

2.2 Caractéristiques

3 Interpréteur

3.1 Switch Dispatch

3.2 Direct Threading

3.3 Context Threading

4 Recompilateur dynamique

4.1 Définition et motivation

La meilleure définition de ce qu'est un recompilateur dynamique passe par la description du travail qu'il accomplit. Tout d'abord, il lit une partie d'un fichier exécutable. Par partie, on peut signifier une instruction du langage assembleur dans lequel l'exécutable a été écrit[REF], un bloc de base[REF] ou bien une trace[REF]. Par la suite, la partie de l'exécutable lue est analysée pour y extraire de l'information sur l'activité des registres virtuels. Il est aussi possible d'extraire d'autres informations pour aider à optimiser le nouveau code assembleur à produire, mais ceci dépasse le cadre du présent article. Finalement, à l'aide des informations d'activités recueillies à la phase

précédente, le recompilateur dynamique émet des instructions dans le language assembleur compris par la machine hôte. Généralement, un recompilateur dynamique est couplé à un interpréteur qui aura pour rôle d'exécuter le code assembleur natif généré.

L'aspect dynamique du recompilateur est nécessaire dans le contexte de l'émulation. Ainsi, le code produit par un recompilateur statique risque fortement de ne pas être correct. La raison derrière cette affirmation est qu'il est impossible, tout du moins avec l'état des connaissances au moment de la rédation de cet article, pour un recompilateur de déduire certains comportements dynamique du code à recompiler. De fait, le code original d'un exécutable peut contenir des branchements indirects dont la cible ne peut être déduite de façon statique. De plus, compte tenu qu'il n'existe pas de distinction entre le code et les données au niveau auquel opère un recompilateur, il n'est pas exclu que le code de l'exécutable que l'on désire recompiler contienne du code auto-modifiant.

La raison majeure pour laquelle on peut désirer mettre en place un recompilateur dynamique est qu'il permet des gains de vitesse énorme au niveau de la vitesse d'exécution d'un émulateur. Ce désir ce manifestera surtout lorsque l'on tente d'émuler des consoles de jeux vidéos appartenant à la 6e (GameCube, PlayStation2, XBox) et à la 7e (Wii, PlayStation3, XBox 360) génération. En effet, ces systèmes ont des architectures assez particulières leur permettant des performances remarquables au niveau de la vitesse d'exécution. La seule manière de s'approcher de ces performances est d'utiliser un recompilateur dynamique au sein d'un émulateur.

Les raisons pour lesquelles on voudrait éviter de programmer un tel système sont par contre nombreuses. La première est que programmer un recompilateur est une tâche très ardue. Il s'agit, grosso modo, de mettre en place un "backend" de compilateur. On rappelle au passage que les trois grandes étapes mise en place par cette partie d'un compilateur, la sélection d'instructions, l'ordonnancement d'instructions et l'allocation de registres, sont toutes des problèmes NP-complet. Qui plus est, ces trois problèmes interagissent ensemble ce qui complique encore plus la chose.

La seconde raison pouvant rendre un recompilateur dynamique peut attrayant est la difficulté de tester un tel système. De fait, l'analyse des binaires produits de façon dynamique pour déterminer de leur qualité requiert une expertise dans le langage assembleur ciblé. **TODO : Finish**

4.2 Émission d'instructions x86

4.3 Analyse d'activité

L'analyse d'activité dans un compilateur se définit comme un problème d'analyse de flux de données. Le but recherché dans cette analyse est la découverte des endroits dans un programme où une variable est active, c'est-à-dire où elle potentiellement lue avant sa prochaine écriture.

L'analyse d'activité dans un recompilateur, dynamique ou statique, diffère de celle effectuée dans un compilateur traditionnelle. Tel que mentionné précédemment, l'approche classique consiste à évaluer l'activité des variables symboliques contenues dans un sous-ensemble de la représentation intermédiaire résultant de transformations appliqués au programme original. Or, dans le cas d'un recompilateur, l'analyse d'activité se concentrera plutôt sur l'utilisation des registres virtuelles dans une partie de l'exécutable traité. Ce problème d'analyse peut être autant approché à partir d'une perspective locale que globale.

Une approche locale à l'analyse d'activité des registres virtuels consiste à se limiter à analyser le flux de données dans une sous-partie bien définie du programme, soit un bloc de base ou une trace. **TODO : Finish**

TODO : Approche globale

4.4 Allocation de registres

TODO : Heuristiques

TODO : Linkage

5 Conclusion

6 Références

TODO : Ref Chip16 spec

A. V. Aho, R. Sethi, J. D. Ullman, M. S. Lam, *Compilers: Principles, Techniques and Tools, Second Edition*

K. Cooper and L. Toczon, *Engineering a Compiler*

M. Zaleski, A. D. Brown, K. Stoodley, *YETI: a gradually Extensible Trace Interpreter*

N. Topham and D. Jones, *High Speed CPU Simulation using JIT Binary Translation*

M. Berndl, B. Vitale, M. Zaleski, A. D. Brown, *Context Threading: A flexible and efficient dispatch technique for virtual machine interpreters*

M. Anton Ertl and D. Gregg, *The Structure and Performance of Efficient Interpreters*

V. Chipounov and G. Candea, *Dynamically Translating x86 to LLVM using QEMU*

G. S. M. de Paula and H. D. H. O. Gomes, *Analysis and extension of PCSX2, a PlayStation2 emulator*

M. Probst, A. Krall, B. Scholz, *Register Liveness Analysis for Optimizing Dynamic Binary Translation*