Méta-morphosis – rapport 3

Owl's team

Pierre Bourdon <bourdo_p@epita.fr>
Tiphaine Petit <petit_t@epita.fr>
David Stavaux <stavau_d@epita.fr>

Table des matières

Introduction

Après deux soutenances qui firent office de démonstration technologique de la possibilité d'un projet comme celui que nous envisagions, nous avons décidé de terminer ce projet et cette année par quelque chose de concret.

Pour un programme comme Méta-morphosis, quelque chose de concret, c'est par exemple de prendre un vrai jeu de PlayStation pour le traduire. Cela demande un fonctionnement exemplaire et sans bug du traducteur de code ainsi qu'un comportement reproduisant à l'identique celui de la console pour laquelle le jeu était à la base prévu.

Cet objectif, nous l'avons en partie réalisé. En travaillant d'arrache-pied ces derniers mois, la Owl's Team vous présente ici un prototype de Méta-morphosis capable de faire fonctionner en partie un jeu commercial, plus précisément le cultissime *Final Fantasy 9*. En partie, car malgré un début de fonctionnement, des bugs subsistent. Nous parlerons de tout cela dans les parties suivantes.

Commençons ce rapport par une présentation du groupe, du projet et de l'avancement réalisé depuis la deuxième soutenance par rapport à ce qui était prévu.

1 Présentation du projet

1.1 Méta-morphosis

Méta-morphosis est un programme dont le but est de permettre la traduction d'un jeu de PlayStation en un binaire natif pour l'architecture habituelle de nos micro-ordinateurs : x86. Étant donné que les architectures x86 et MIPS32 LE (architecture utilisée par la PlayStation et son processeur, le MIPS R3000A) sont totalement différentes, cela implique une décompilation partielle du code exécutable du jeu PlayStation en une forme abstraite, puis une recompilation de cette forme abstraite en de l'assembleur pour x86.

La « forme abstraite » dont nous parlons est en fait pour nous de l'assembleur pour LLVM. LLVM, la Low Level Virtual Machine, est un projet de machine virtuelle et de compilateur pour un langage assembleur portable (ou presque) et haut niveau (par rapport à de l'assembleur pour un processeur). C'est un projet soutenu par Apple et qui peut être considéré comme mature, utilisé par de nombreux projets de compilateurs, de machines virtuelles, d'analyseur statiques, etc. C'est donc naturellement que nous nous sommes tournés vers ce choix comme langage résultant de la décompilation du binaire PlayStation : suffisamment portable, suffisamment bas niveau et traduisible vers de l'assembleur natif.

FIG. 1 – Le logo de LLVM, donné gracieusement par Apple à la communauté.

Même si le problème du processeur est réglé, une PlayStation ne contient évidemment pas qu'un CPU. De nombreux composants gravitent autour :

RAM, ROM, cartes mémoires, lecteur de CD-ROM, GPU, SPU, MDEC (décodeur d'images et de vidéos dont nous parlerons plus dans la dernière partie), GTE (Graphics Transformation Engine, pallie le manque d'un processeur de calculs flottants et réalise toutes les opérations utiles à la 3D dans un jeu vidéo : multiplication de matrices, produit scalaire, calcul d'éclairage, etc.) et quelques autres composants mineurs. Il aurait été possible de reproduire tout cela via des bibliothèques d'émulation du matériel, mais cela nous aurait pris un temps phénoménal. Ainsi, à la fin de notre deuxième soutenance, nous avons décidé de nous tourner vers l'intégration de Méta-morphosis à un émulateur fournissant déjà ces bibliothèques d'émulation : PCSX. Nous parlerons de cela dans la prochaine section.

1.2 Le groupe

Notre groupe fut assez fluctuant durant la période de développement et de conception du projet. Partant à 4, revenant à 3, retrouvant un membre pour ensuite redescendre à 3 à cause de différentes expulsions de l'école, la Owl's team est donc au final constituée des trois membres suivants :

- Pierre Bourdon, chef de projet, qui a travaillé pendant cette soutenance sur le traducteur de binaires ;
- Tiphaine Petit, à qui est due l'intégration de Méta-morphosis dans PCSX, l'émulateur PlayStation le plus utilisé ;
- David Stavaux, qui a réalisé l'implémentation actuelle du décodeur de vidéos et d'images.

Malgré une baisse claire de motivation pour la 2ème soutenance qui a amené à une réduction de nos objectifs, le travail à faire pour cette 3ème soutenance nous semble à tous satisfaisant et indique une grande remontée de l'intérêt du groupe dans les tâches à réaliser. Cela est notamment du à la présence d'un résultat visible de notre travail : faire fonctionner un jeu commercial est un accomplissement non négligeable.

1.3 État actuel du projet

L'accomplissement de cette soutenance est, avant tout, le fonctionnement partiel d'un jeu commercial traduit par Méta-morphosis. Nous nous sommes pour cette soutenance concentré sur un jeu qui nous semble conforme à ce que nous souhaitons faire fonctionner. Ce jeu, c'est le RPG japonais culte *Final Fantasy 9* de SquareSoft (maintenant SquareEnix). Les différents critères qui nous ont fait choisir ce jeu sont, premièrement, notre avis très positif sur le jeu en lui-même, qui nous a poussé à travailler dur pour le faire fonctionner. Ensuite, ce jeu respecte l'ABI standard de l'architecture MIPS32 LE, ce qui est loin d'être le cas de tous les jeux de PlayStation. Encore sur un plan technique, tout le code du jeu est regroupé en un binaire, et aucun code n'est chargé

depuis un autre emplacement du CD. Enfin, c'est un jeu complexe, utilisant toutes les fonctionnalités de la PlayStation, de la 3D aux cinématiques en passant par, bien entendu, le lecteur de CD-ROM.

FIG. 2 – Le menu principal de FF9 affiché par Méta-morphosis.

2 Intégration à PCSX

Pendant la période de travail entre la deuxième et la dernière soutenance, Tiphaine s'est chargée d'intégrer le code traduit par Méta-morphosis à l'émulateur PCSX pour générer des binaires natifs tout en profitant des bibliothèques d'émulation de PCSX. Cependant, même si l'on parle de bibliothèques d'émulation, il ne faut pas se méprendre : ces bibliothèques contiennent uniquement du code natif et ne réalisent aucune interpétation du code. Elles ne font qu'exporter des fonctions utiles à l'interfaçage entre Méta-morphosis et le système.

Commençons par parler de l'historique du projet, puis de la bibliothèque principale de PCSX : `libpcsxcore`, avant de terminer par le travail d'intégration qui a été réalisé et son état actuel.

2.1 Historique du projet

PCSX est un projet qui a un historique assez long et qui a été forké de nombreuses fois suite au désintéressement de ses mainteneurs originaux qui travaillent maintenant sur PCSX2, un émulateur PlayStation 2.

Il existe actuellement trois émulateurs PlayStation majeurs sur PC : PCSX, epsxe et pSX. Alors que les deux premiers sont libres et se basent sur des standards, pSX est non libre et réimplémente les bibliothèques graphiques, sonores, etc. partagées par PCSX et epsxe. Le choix devait donc être fait entre PCSX et epsxe, étant donné qu'il est impossible d'intégrer Méta-morphosis à un projet non libre (pour des raisons techniques évidentes).

Étant donné son inactivité (la dernière news de PCSX date de 2003 et annonce l'abandon du projet, considéré comme fonctionnant parfaitement), il aurait été facile d'éliminer PCSX et de partir sur epsxe. Cependant, le code de PCSX est bien mieux construit que celui d'epsxe, avec notamment un support pluggable pour des plugins de CPU (exactement ce que nous souhaitions ajouter). De plus, deux forks de PCSX existent, tous deux en collaboration : PCSX-df, nommé ainsi pour « Debian Fork », qui est la version de PCSX du projet Debian, et PCSXR, pour « PCSX Reloaded », qui est une version améliorée, accélérée et avec plus de fonctionnalités que le PCSX original.

Il a donc été décidé par Tiphaine de partir sur la base de PCSX-R pour y intégrer Méta-morphosis.

2.2 libpcsxcore

La `libpcsxcore` est la bibliothèque principale sur laquelle repose PCSX. Elle fournit une interface simple à l'émulation de la PlayStation pour utiliser

dans un projet d'émulateur. La plupart des ajouts que Tiphaine a réalisé sont donc dans cette partie de la codebase de PCSX.

Cette bibliothèque émule toute l'architecture autour du CPU de la PlayStation (le R3000A) et fournit une API standard à respecter pour implémenter son propre CPU. Deux CPU sont fournis : un interprétant instruction par instruction l'exécutable PlayStation, et l'autre réalisant de la recompilation à la volée (JIT compiling, donc non statique, contrairement à Méta-morphosis).

Parmi les composants de l'architecture émulée, on trouve notamment les choses suivantes :

- Le contrôleur de mémoire
- Le contrôleur d'interruptions
- Le Graphics Transformation Engine
- Le Movie DECoder
- La gestion de la DMA
- Le lecteur CD-ROM
- Le contrôleur de manettes
- L'accès aux cartes mémoires

2.3 Travail d'intégration

Le code généré par Méta-morphosis est linké avec la `libpcsxcore` sous la forme d'un fichier objet classique exportant les quelques fonctions nécessaires : exécution d'une instruction, exécution jusqu'à un breakpoint, exécution jusqu'à un saut ou exécution jusqu'à la fermeture du programme. Ces fonctions sont celles nécessaire pour implémenter un CPU dans PCSX.

En plus de cela, le code généré par Méta-morphosis réalise des appels aux fonctions de la `libpcsxcore` afin de profiter de l'interface d'émulation exportée par la bibliothèque. Par exemple, les accès aux registres mappés en mémoire sont convertis en un appel à la fonction adéquate de `libpcsxcore`.

À partir de là, intégrer un nouveau CPU dans la `libpcsxcore` n'est qu'une affaire de remplissage de structure `psxCpu` et la modification d'une valeur par défaut pour utiliser le CPU émulé par Méta-morphosis par défaut.

2.4 État du travail

À l'heure actuelle, l'intégration de Méta-morphosis à PCSX est fonctionnelle et efficace pour les tests simples. Elle semble également fonctionner sur des tests plus conséquents tels que Final Fantasy 9, mais des disfonctionnements empêchent de tester en profondeur cette intégration. Au fi-

nal, malgré la nécessité d'une refonte d'une grande partie du backend de notre traducteur pour s'adapter à PCSX, ce travail a énormément payé : cela nous évite de maintenir notre propre interface graphique, notre propre bibliothèque graphique, notre propre gestion des interruptions, etc.

3 Travail réalisé sur le traducteur

Le traducteur est la pièce centrale de Méta-morphosis. Son rôle est simple : il prend en entrée un CD-ROM de PlayStation et convertit le code qui y est présent en un fichier objet à linker avec PCSX.

Il réalise un travail algorithmiquement complexe : il s'agit de déterminer dans la mémoire les zones remplies avec du code et celles remplies avec des données afin de compiler uniquement les zones nécessaires et de résoudre les sauts et les appels de fonction.

De nombreuses améliorations ont été faites sur ce traducteur pour cette dernière soutenance : tout d'abord, le travail de David sur le MDEC y a été intégré afin de permettre le décodage de vidéos rapide et optimisé. Ensuite, une meilleure détection du code mort y est réalisé afin d'éviter de compiler du code qui n'est pas utilisé. Enfin, un travail a été réalisé sur l'allocation des registres et l'optimisation via propagation de constantes.

3.1 Intégration du MDEC

Le MDEC est la puce de décodage matérielle MJPEG de la PlayStation, sur laquelle plus d'informations sont données dans la section suivante du rapport consacrée au décodage de vidéos.

Le travail qui a été réalisé dans ce domaine est surtout au niveau de l'intégration entre les procédures de décodage et la propagation de constantes. En effet, avec un MDEC décrit entièrement par le traducteur, il est possible d'optimiser bien plus efficacement la lecture de vidéos qu'avec des simples appels de fonctions C qui se comportent comme des boites noires pour notre traducteur au niveau des accès mémoire et registres.

Ainsi, en ajoutant la reconnaissance des patterns du MDEC au traducteur, on a pu observer des gains d'environ 75% sur de la lecture simple de vidéos dans un jeu de PlayStation. Ce gain est non négligeable dès que l'on est sur des configurations minimales tels que des netbooks.

À noter qu'avec suffisamment de temps il aurait été possible de coupler cette amélioration avec une intégration du GPU et le décodage des vidéos MJPEG directement sur le GPU de l'ordinateur.

3.2 Détection du *dead code*

Ce que l'on appelle du *dead code* est soit des bouts non exécutables trouvables au milieu du code (par exemple, du padding) ou du code qui est présent dans le binaire mais jamais exécutable dans la configuration ou l'on

se trouve. Il est très utile de pouvoir détecter ce code mort afin de premièrement réduire la taille du code généré, puis deuxièmement réduire le temps de compilation en évitant de compiler de gros morceaux de code qui sont en fait des données.

La RAM de la PlayStation étant très petite, on peut très facilement la partitionner via une structure de donnée spécialisée afin d'avoir une représentation des zones analysées avec accès à la zone d'un élément en $O(logn)$ et séparation d'une zone en deux sous-zones en $O(logn)$. À partir de là, il suffit de partir de l'entry point et de tracer les appels et les sauts possibles.

Les sauts indirects et les pointeurs de fonctions font partie des exceptions à la règle : ils compliquent énormément cette détection de code mort et empêchent de savoir facilement quand du code sera exécuté et depuis quelle adresse. Cependant, on peut tirer partie de l'alignement obligatoire des instructions sur 4 octets et prédire la valeur des registres utilisés pour les sauts indirects, et ainsi effectivement repérer les zones de code.

3.3 Optimisation du code généré

Parmi les nombreuses optimisations réalisables sur le code généré, deux ont été implémentées depuis la deuxième soutenance de manière à augmenter de manière cruciale les performances et réduire énormément la taille du code.

Déjà, du fait de l'analyse de code, nous pouvons séparer le code binaire en blocs basiques d'exécution démarrant à une instruction qui est le point d'entrée et se terminant par un jump vers un autre bloc (qui peut être lui-même). Il est possible d'éviter le direct threading et d'exécuter toutes ces instructions en une fois étant donné que l'on sait que ces instructions seront toutes exécutées en une seule fois et toutes autant de fois. Cette optimisation a grandement réduit le temps de compilation du code, passant souvent de dizaines de secondes à une ou deux secondes dû à la réduction de la table de dispatch dans le code.

Ensuite, une peephole optimization avancée a été mise en place. Le principe de la peephole optimization est de suivre l'état de la mémoire et des registres durant l'exécution pour détecter les constantes et optimiser leurs valeurs. Un exemple classique est le chargement d'une valeur à une adresse mémoire : en assembleur MIPS, cela doit être effectué en deux instructions, la première qui charge les 16 bits de poids fort et la deuxième qui charge les 16 bits de poids faible. Il est possible de directement détecter cela dans notre peephole optimizer pour générer un seul chargement d'une valeur de 32 bits.

3.4 Allocation des registres

L'allocateur de registres est le sous-système qui sert à allouer des registres inutilisés pour stocker les valeurs intermédiaires et, si possible, dans notre cas, les registres du CPU de la PlayStation. Étant donné qu'un R3000A a bien plus de registres à usage général qu'un processeur x86, il nous est impossible de faire un mapping 1 :1 de ces registres. Il nous faut au fur et à mesure sélectionner ceux qui sont les plus appropriés à stocker en registres et les plus appropriés à swap-out.

Trouver une combinaison optimale est en fait un problème NP-complet se ramenant au problème de coloriage d'un graphe. Il nous faut donc trouver une approximation suffisamment correcte pour éviter de swap-in et swap-out tous les registres à chaque opération.

4 Décodage de vidéos

La possibilité de lire des vidéos et des cinématiques réalistes durant le jeu a été une des fonctionnalités qui ont démarqué la PlayStation par rapport à ses concurrents, et notamment la Nintendo 64.

Afin de permettre cela, la PlayStation inclut une puce de décodage vidéo nommée le MDEC (Movie DECoder) capable de décoder rapidement des blocs d'images ou de vidéos et de les uploader directement sur la puce graphique. Ainsi, le CPU est déchargé de cette tâche.

4.1 Décodage RLE

Le run-length encoding, appelé en français le codage par plages, est un algorithme de compression de données.

Le système s'applique essentiellement à des documents scannés en noir et blanc : au lieu de coder un bit par point, on dispose d'un compteur — en général sur un octet — indiquant combien de points blancs ou noirs se suivent. Comme il est rare de ne pas avoir au moins 8 pixels noirs ou 8 pixels blancs qui se suivent, et que 256 ne sont pas rares sur les endroits vierges ou les à-plats noirs, le système a bien pour effet une compression. S'il y a plus de 256 bits de la même couleur, on peut placer ensuite un octet spécifiant 0 bit de la couleur opposée, puis coder le nombre de bits qui restent...

Par exemple, considérons un écran de texte noir sur fond blanc. Il sera constitué de longues séquences de pixels blancs pour le fond, et de courtes séquences de pixels noirs pour le texte. Représentons une ligne d'un tel écran, avec B pour les pixels noirs et W pour les pixels blancs :

```
WWWWWWWWWWWWBWWWWWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWW
```

Un encodage RLE consiste alors à indiquer pour chaque suite de pixels d'une même couleur, le nombre de pixels de cette séquence. Le résultat comporte en général moins de caractères, bien que ce ne soit pas une obligation. On obtient par exemple pour la ligne précédente :

```
12W1B14W3B23W1B11W
```

Tandis que :

```
WBWBWBWBWB
```

donnerait :

```
1W1B1W1B1W1B1W1B1W1B
```

Ce qui est deux fois plus long.

4.2 Inverse DCT

La transformée en cosinus discrète ou TCD (de l'anglais : DCT ou Discrete Cosine Transform) est une transformation proche de la transformée de Fourier discrète (DFT). Le noyau de projection est un cosinus et génère donc des coefficients réels.

La DCT est une fonction linéaire inversible de RN dans RN ou de manière équivalente une matrice carrée $N \times N$ inversible. Il existe plusieurs légères variantes de la DCT.

Le développement des algorithmes de calcul rapide des transformées DCT se basent sur la possibilité de décomposer la matrice de définition sous forme d'un produit de matrices dont le calcul est plus simple, et permet de réduire le nombre de multiplications scalaires, en profitant des identités remarquables de périodicité et symétries des fonctions sinusoïdales. Ainsi, on peut décomposer toute transformée DCT de RN en transformées plus simples en décomposant N en produit de facteurs premiers, et en composant des sous-transformées dans Rn où n est l'un de ces facteurs. En particulier, de nombreuses optimisations ont été développées quand N est une puissance de 2.

Cela revient à réécrire la matrice $N \times N$ sous forme de produit de sous-matrices identiques (disposées en pavage régulier et utilisant donc des coefficients réels communs ou différenciés uniquement par leur signe) et de matrices à coefficients unitaires ou nuls (-1, 0 ou 1), ces dernières ne nécessitant pas de multiplication.

4.3 Application au décodage MJPEG

Le format MJPEG est un des format vidéos les plus simples, et également celui utilisé sur la PlayStation pour stocker la plupart des cinématiques durant le jeu. Ce format consiste simplement en une suite d'images compressées au format JPEG.

Cela tombe plutôt bien étant donné que le format JPEG est basé sur les deux techniques énoncées avant : le RLE et l'iDCT. De ce fait, les vidéos MJPEG peuvent très facilement être décodées par la puce MDEC. Ainsi, pratiquement toutes les vidéos sur PlayStation sont stockées au format MJPEG.

Cependant, MJPEG est un très mauvais format par rapport à ce qu'il se fait de nos jours (h.264, Theora, VP8, etc.). En effet, aucune compression n'est faite entre les frames de la vidéo et on peut trouver énormément de redondances. Ce format a néanmoins l'avantage de la simplicité et du décodage rapide pour une plateforme lente comme la PlayStation.

En général, le MDEC peut décoder une vidéo MJPEG en 320x200 à une vitesse de 24 images par seconde, ce qui correspond à la vitesse de tournage des films et est donc tout à fait correct.

Conclusion

C'est donc sur une note plutôt agréable que se termine ce troisième rapport : en effet, la plupart de nos objectifs, même s'ils ne sont pas remplis entièrement, sont prouvés remplissables empiriquement par un exemple concret et pourraient être facilement terminés avec du temps supplémentaire que nous n'avons malheureusement pas.