Rapport Matchine Universelle - Compilation avancée

Par Adrien MISS et Alexandre GASPARD CILIA.

Introduction

Dans un premier temps, nous aborderons l'**architecture** de notre implémentation de la Machine Universelle. Ensuite nous vous expliquerons nos **choix d'implémentation**. Et enfin nous finirons par faire un **bilan** de l'état actuel de notre programme.

Architecture

La machine virtuelle se compose d'une class principale *sobrement* nommée **MachinaMagnifica**, **PlateauDeSable** (et aussi **Registre**, nous y reviendrons plus tard) qui représente les données de taille 32 bits utilisés par la machine, de **Memory** qui comme son nom l'indique symbolise la mémoire utilisée par la machine.

MachinaMagnifica

La **MachinaMagnifica** regroupe tout les éléments nécessaires au fonctionnement de la machine universelle : les registres, la mémoire, les opérateurs et enfin la boucle principale d'interpretation des programmes.

PlateauDeSable

Cette class stocke les données sous forme de tableau de booléens. Nous reviendrons plus tard sur ce regrettable choix d'implémentation. Elle encaplsule aussi les operations basiques pouvant être éffectués sur les **PlateauDeSable** comme l'affectation de valeurs, la traduction vers des formats plus conventionnels comme l'*int* ou le *long* et des tests basiques comme par exemple *le plateau est-il égal à 0 ?*. Une class **Registre** qui étend **PlateauDeSable** n'est là que pour s'assurer que l'on utilise bien des registres lors de certaines opérations. Elle n'ajoute aucune fonction supplémentaire.

Memory

La mémoire encaplsule la *RAM* utilisée pour le fonctionnement de la machine virtuelle. Cette mémoire est sous forme d'un tableau à deux dimensions de **PlateauDeSable** (*PlateauDeSable[][] memory*). Elle dispose aussi d'opération basique pour l'utiliser :

- *alloc* pour allouer un tableau.
- *cpy* pour copier un tableau.
- *getData* pour recupérer un tableau. Cette méthode existe aussi avec un *offset* pour simplifier la réalisation des opérandes.
- *setData* pour affecter un tableau à une case. Existe aussi avec une option *offset*.
- *free* pour libérer une adresse mémoire.

Les choix d'implémentation

Commençons par l'implémentation de **PlateauDeSable**. Nous avons fait le regrettable choix de stocker les données sous forme d'un tableau de booléen. Ce choix a été fait tout d'abord pour des raisons de simplicité. Nous avons naïvement cru que cela aurait facilité le traitement des données. *Que nenni*! Par exemple :

```
public void add(Registre a, Registre b, Registre c) {
    a.setData(b.toLong() + c.toLong());
}

---
```

```
public long toLong() {
    long result = 0;

    for (int i = 0; i < data.length; i++) {
        if (data[i])
            result += Math.pow(2, i);
    }
    return result;
}
```

On peut voir que pour effectuer une simple addition il nous faut convertir deux registres, effectuer l'addition et enfin reconvertir la donnée vers notre tableau de booléen. De plus cette conversion est plus que gourmande. Alors que si nous avions simplement utilisé un *int*, nous aurions pu tout simplement utiliser l'addition de Java.

Bilan

À l'heure actuelle, les tests basiques de *sandmark.umz* fonctionne. Mais lorsqu'il lance le *stress test*, le chargement du programme ne fonctionne pas.

```
trying to Allocate array of size 0..
trying to Abandon size 0 allocation..
trying to Allocate size 11..
trying Array Index on allocated array..
trying Amendment of allocated array..
checking Amendment of allocated array..
trying Alloc(a,a) and amending it..
comparing multiple allocations..
pointer arithmetic..
check old allocation..
simple tests ok!
about to load program from some allocated array..
success.
verifying that the array and its copy are the same...
success.
testing aliasing..
success.
free after loadprog..
success.
loadprog ok.
ERROR: compressed data are corrupt?
```

Il doit y avoir un problème lors de la conversion d'un *int* vers tableau de booléen. Mais comme la structure du programme est trop lourde, le debugging requière plus de temps que nous n'avons à notre disposition.

D'autre part l'utilisation un peu trop soutenue d'objet rend la machine extrêmement lente.

Si nous devions recommencer ce projet, nous utiliserions tout simplement des *int* comme plateau de sable et limiterions au maximum l'utilisation de l'objet pour rendre l'exécution plus rapide. Une autre manière simple de rendre le programme plus rapide serait de faire la moins d'appel de fonctions possible. Par exemple au lieu de créer une fonction pour chaque opérateur, nous pourrions tout

simplement mettre le code de l’opération dans la boucle principale de l’interpréteur.

Donc en conclusion, la *machina* n’est pas si *magnifica*.