Simulation d'exécution de programmes parallèles

TER. Rapport intermédiaire

Olga Pigareva
M1 SIL

1er mai 2019

Table des matières

1 La présentation de la problématique dans son contexte

1.1 Introduction

Le sujet de mon travail d'étude et de recherche aborde un domaine d'actualité dans le monde numérique, la programmation parallèle. La parallélisation de tâches et de processus permet d'augmenter la performance et faire l'exécution d'un programme beaucoup plus rapide. C'est pourquoi la programmation parallèle reste un sujet très actuel, même si elle était inventée il y a déjà plusieurs années.

La programmation parallèle comprend la résolution d'un problème simultanément par plusieurs processus. Cela implique l'utilisation d'une seule ressource par plusieurs processus, ce qui nécessite une synchronisation. De son côté, la synchronisation peut amener à un interblocage qui peut avoir les conséquences indésirables (catastrophiques) pour un programme.
Mon travail fait partie d'une recherche visant à caractériser ces interblocages.
Dans les parties suivantes, j'expliquerai ce que signifient ces termes.

1.2 Programmes parallèles

Les programmes parallèles permettent de créer les activités qui peuvent exécuter les instructions indépendamment. Souvent ces activités ont besoin d'acceder aux mêmes ressources. Pour ne pas avoir les résultats imprévus lors de modification de données par les processus différents, les ressources sont synchronisées. Cela veut dire que les processus peuvent acceder aux données chacun à son tour. Pour attendre la libération de données ou la fin d'instructions d'un processus, un programme parallèle utilise les horloges qui sont les barrières de synchronisation.

1.3 Interblocage

Le problème d'interblocage apparaît quand chaque processus attend un autre pour se terminer et finalement aucun peut acceder à la ressource. Par conséquent, le programme reste bloqué.

Par exemple, sur la Figure 1, le processus P1 attend la fin de processus P2 pour acceder à R1. Et P2, de son côté, attend la fin de P1 pour acceder à R2. Tous les deux processus restent bloqués.

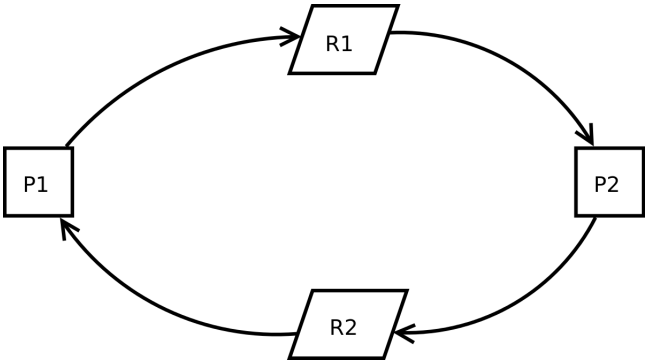

FIGURE 1 – Exemple d'interblocage

Souvent pour les programmes avec plusieurs processus et plusieurs ressources, il est difficile de contrôler l'absence d'interblocages, surtout avec la liberté et puissance offertes par la synchronisation. D'où l'intérêt de reconnaître ces interblocages automatiquement le plus tôt possible, à savoir, éventuellement, lors de la compilation du programme.

1.4 Problème et objectifs

Est-il possible de définir un interblocage pendant l'étape de compilation ?

Pour répondre à cette question, l'idée est de commencer par créer une version simplifiée d'un langage parallèle en prenant un langage X10 comme un exemple. Le compilateur doit reconnaître juste les instructions qu'on a besoin pour créer plusieurs activités parallèles (boucles, conditions, création de nouveaux processus et horloges).
L'exemple de tel programme est représenté par la Figure 2.

```
finish clk1 {
    async S0();
    for i in 0..3
        async clk1 {
            S1(i);
            advance clk1;
            S2(i)
        }
}
```

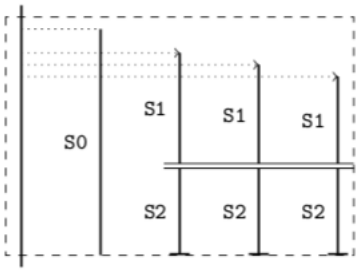

FIGURE 2 – Exemple d'un programme parallèle

Après avoir implémenté un langage qui permet de créer plusieurs activités parallèles avec la synchronisation, il doit être possible de visualiser l'état de toutes les activités au moment d'interblocages. Pour cela, il faut créer une machine virtuelle qui va être capable de simuler l'exécution de programmes parallèles et donner une liste d'activités.

2 La liste des tâches déjà effectuées

2.1 Le travail réalisé

— Installation de librairies et IDE pour le langage de programmation **X10** sur mon ordinateur.
Premièrement, il fallait préparer l'environnement de travail.
— Lecture de la documentation et d'autres ressources sur l'interblocage et programmes parallèles.
— Exécution de différents programmes test X10 pour comprendre le problème et voir comment il fonctionne.
— Implementation d'un compilateur inspiré par le langage X10 à l'aide de Lexer et parser (Lex et Yacc).
— Génération d'un arbre de la syntaxe abstraite (AST en anglais).

3 Le planning prévisionnel des tâches à réaliser

mars Implémentation de la simulation d'exécution du programme et captures de traces d'exécution.

avril Implémentation de visualisation des traces d'execution ou de systèmes mathématiques permettant de détecter un interblocage.

mai Rédiger un rapport avec le bilan de résultats atteints.

4 Choix d'implémentation et différentes étapes

4.1 Grammaire

```
program → list_stmt

stmt → instruction ';'
     |   FOR ID '=' range block
     |   IF '(' affine ')' block
     |   IF '(' affine ')' block ELSE block
     |   FINISH '(' clocks ')' block
     |   FINISH block
     |   ASYNC '(' clocks ')' block
     |   ASYNC block
     |   ADVANCE ID ';'

list_stmt → list_stmt stmt | stmt

instruction → ID

clocks → /*empty*/
       |   clocks ',' ID
       |   ID

block → \{' list_stmt '\}'
      |   stmt

range → affine RANGE affine

affine → affine '+' affine
  |   affine '-' affine
  |   affine '/' affine
  |   affine '*' affine
  |   affine COMP affine
  | '(' affine ')'
  | ID
  | NUMBER
```

4.2 Génération du code intermédiaire

EVAL

FINISH - créer une nouvelle structure finish - la rajouter dans une pile de finish de l'activité en cours

CLOCK$_{CREATE - crer une nouvelle structure clock - ajouter l'activite en cours dans la pile 'registe}$

Références

[1] Alain Ketterlin. *La présentation de sujet de TER.*

[2] Agarwal Shivali, Barik Rajkishore, Sarkar Vivek, Shyamasundar R.K. *May-happen-in-parallel analysis of X10 programs.* Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (2007), PPOPP.

[3] Vijay Saraswat, Bard Bloom, Igor Peshansky, Olivier Tardieu, and David Grove *X10 Language Specification.* Version 2.6.2

[4] Composing Programs : Parallel Computing, `https://composingprograms.com/pages/48-parallel-computing.html`