Implantation distribuée de modèles concurrents

Franck Pommereau* Frédéric Gava†

Sujet de stage de M2 recherche
2014–2015

Contexte

Le formalisme *asynchronous box calculus with data* (ABCD) définit une notation simple permettant de spécifier des processus concurrents qui embarquent du calcul exprimé en Python. Ce formalisme est utilisé depuis plusieurs années pour modéliser des systèmes variés (protocoles de sécurité, systèmes pair-à-pair, ...) dans le but d'en faire l'analyse par model-checking ou par simulation. La sémantique d'un processus ABCD est un réseau de Petri coloré par du Python, c'est-à-dire que ses places contiennent des jetons qui sont des objets Python et ses transitions effectuent des calculs réalisés en Python.

Dans ce stage, on considère un processus ABCD comme un prototype d'une application réelle dont les calculs sont implantés en Python et dont les flots de contrôle et de données sont décrits à l'aide d'ABCD. Le but du stage est de fournir un schéma d'implantation des processus ABCD pour en permettre l'exécution efficace et concurrente.

D'une façon générale, l'implantation efficace de réseaux de Petri est un problème difficile n'ayant pas donné à ce jour de solution pleinement satisfaisante. Le verrou scientifique principal est qu'il n'est pas possible en général de décomposer correctement un réseau de Petri en composants concurrents de taille suffisante. En effet, une décomposition trop fine fait apparaître de nombreux composants concurrents qui ne peuvent pas s'implanter à base de *threads* ou de processus classiques, alors qu'une décomposition trop grossière limite la concurrence et ne permet pas l'efficacité. Dans notre cas, ce problème est résolu par la connaissance d'une décomposition par construction car la concurrence d'un système en ABCD est explicite et à deux niveaux : entre les processus et entre les processus. D'autre part, on propose une alternative à l'utilisation des *threads*.

Objectifs

L'objectif du stage est de compiler un tel système en ciblant une architecture à deux niveaux de concurrence correspondant à un réseaux de machines multicœurs. Ainsi, il faudra concevoir et développer deux stratégies d'implantation complémentaires correspondant au deux niveau de concurrence exprimables en ABCD.

Au sein des processus, la concurrence se traduira par du parallélisme sur une machine multicœurs à l'aide de *micro-threads* ou de *co-routines*, qui sont des alternatives extrêmement légères et efficaces aux threads classiques et réalisent du multi-tâche coopératif (par opposition au multi-tâche préemptif des threads). On les trouve dans divers langages de programmation : historiquement dans Erlang [1], plus tard dans Stackless Python [3], plus récemment encore dans le langage Go de Google [2] ou dans Continuation Passing C [4].

*IBISC, Université d'Évry, 23 boulevard de France, 91037 Évry. franck.pommereau@ibisc.univ-evry.fr
†LACL, Université Paris-Est Créteil, 61 avenue du gal de Gaulle, 94010 Créteil. frederic.gava@univ-paris-est.fr

Entre les processus, la concurrence se traduira par des communications entre machines, à l'aide de passage de messages en utilisant une bibliothèque dédiée telle que MPI ou OpenMP.

Cet objectif principal peut se décomposer sur plusieurs objectifs secondaires :

1. Définition d'un schéma d'implantation des réseaux de Petri et validation de cette approche par l'implantation manuelle d'exemples dans un langage supportant les micro-threads (de préférence Stackless Python ce qui limitera les développements nécessaires puisqu'ABCD est aussi implanté en Python).
2. Formalisation de l'ordonnancement ainsi obtenu et preuve de sa correction : (1) toute exécution de l'implantation doit être une exécution valide du réseau de Petri ; (2) réciproquement, toute exécution du réseau de Petri peut être obtenue par l'implantation ; (3) enfin, tout état de blocage est détecté par l'implantation qui peut s'arrêter (et non se bloquer).
3. Réalisation d'un prototype de compilateur automatisant l'implantation. Pour cela, on s'appuiera sur le compilateur ABCD et la bibliothèque SNAKES qui permettent de manipuler les spécifications ABCD et les réseaux de Petri correspondants.
4. Définition de stratégies d'ordonnancement permettant d'optimiser l'exécution selon différents critères (choix de certaines exécutions plutôt que d'autres), validation par des tests dans le prototype.
5. Validation de l'approche globale par quelques études de cas sur des exemples de la littérature, comparaison avec d'autres outils.

Résultats attendus et livrables

Le travail attendu est avant tout théorique mais sa finalité est d'être validé par la pratique puisqu'un objectif au delà du stage est d'intégrer cet outil à la plateforme SNAKES [5]. Il faudra donc fournir un schéma de traductions et la preuve de sa correction. De plus, un prototype de compilateur devra être fourni. Il n'est pas attendu du stagiaire qu'il fournisse un produit fini, mais bien un prototype. Cela signifie que l'implantation peut être partielle (sur des parties significatives) et laisser à l'utilisateur des manipulations manuelles. Cependant cette implantation doit être de bonne qualité car ce prototype doit pouvoir servir de base de travail et de modèle de développement pour compléter le compilateur. Cela implique aussi que cette implantation doit être correctement documentée.

Un rapport écrit doit détailler le travail réalisé au cours du stage en fournissant les définitions des modèles et des démarches retenus, avec des explications précises et des exemples, ainsi que les résultats des tests de performances et les preuves détaillées.

Perspectives de recherche

Le sujet proposé peut servir de point de départ à une thèse sur le même domaine, avec des problématiques similaires.

Informations pratiques

Le stage pourra se dérouler au laboratoire IBISC à Évry, ou au laboratoire LACL à Créteil. La rémunération sera de 436,05 euros par mois, sur une durée de 5 à 6 mois. Le stagiaire aura un bureau et un ordinateur ainsi que l'accès aux ressources du laboratoire.

Références

[1] Erlang programming language. `http://www.erlang.org`.

[2] The Go programming language. `http://golang.org`.

[3] Stackless Python. `http://www.stackless.com`.

[4] Gabriel Kerneis. Continuation passing C (CPC). `http://www.pps.jussieu.fr/~kerneis/software/cpc`.

[5] Franck Pommereau. SNAKES is the net algebra kit for editors and simulators. http ://www.ibisc.univ-evry.fr/ fpommereau/SNAKES/.