Équipe HP2, département d'informatique
Telecom SudParis

Gaël Thomas
Telecom SudParis, Bureau D306, 91011 Evry
Tel : 06 10 39 31 17
Email : gael.thomas@telecom-sudparis.eu

Paris, le 29 mai 2016

Rapport sur la thèse de Monsieur Etienne Brodu intitulée « Fluxional compiler : seamless shift from development productivity to performance efficiency, in the case of real-time web applications »

La thèse d'Etienne Brodu s'intéresse à rendre plus efficace des codes écrits dans le langage Javascript. Javascript est un langage maintenant utilisé pour développer des applications de type serveur internet car il permet de mettre rapidement en production une application. Toutefois, Javascript est intrinsèquement inefficace car il ne permet pas de paralléliser naturellement un code. Cette limitation le rend incapable d'absorber la charge générer par de nombreux utilisateurs. Après une phase de développement rapide pour mettre en production un logiciel, les développeurs doivent donc bien souvent entièrement reconcevoir leur programme pour le paralléliser. Ce problème de coût de réécriture du code pour paralléliser une application Javascript est parfaitement d'actualité : de nombreuses entreprises sont effectivement confrontées au compromis entre productivité et performance avec Javascript.

De façon à éviter une couteuse phase de réécriture du code, Etienne Brodu propose de paralléliser automatiquement un code Javascript. Etienne Brodu part d'un constat : un programme Javascript est écrit en suivant le modèle de programmation événementiel. Une application Javascript est composée de gestionnaires d'événements. Le moteur d'exécution Javascript exécute alors ces gestionnaires d'événements séquentiellement lorsque les événements arrivent. Ce modèle de programmation événementiel est relativement proche d'un modèle de programmation parallèle : le modèle de parallélisme de flux (aussi appelé parallélisme de tâches ou parallélisme pipeline). Dans ce modèle, des tâches s'activent les unes les autres en s'envoyant des messages. Toutefois, dans le parallélisme de flux, chaque tâche travaille de façon isolée dans sa propre mémoire, alors que dans le cas d'un langage événementiel, tous les gestionnaires d'événements travaillent dans une mémoire partagée. Éxécuter brutalement les gestionnaires d'événements Javascript en parallèle mènerait donc à des incohérences mémoires, ce qui rend la transformation du modèle événementiel vers le modèle de flux non trivial.

Etienne Brodu propose donc un compilateur pour identifier les dépendances de données dans un programme Javascript de façon à réécrire automatiquement un code Javascript en suivant le modèle de parallélisme de flux. L'idée de la thèse est d'isoler les dépendances de données de façon (i) à les transformer en messages et (ii) à identifier les gestionnaires d'événements qui travaillent sur les mêmes données. Si deux gestionnaires d'événements travaillent sur des données distinctes, il est ensuite possible de les exécuter en parallèle sans modifier la sémantique du programme. Bien que le compilateur proposé par Etienne Brodu possède quelques limitations, en particulier, le développeur doit manuellement vérifier que les transformations sont sémantiquement correctes. le compilateur proposé par Etienne Brodu forme une première étape majeure pour aider les développeurs Javascript à rendre automatiquement leurs codes plus efficaces.

Le document de thèse de 138 pages est rédigé en anglais. Il est très bien structuré et agréable à lire. Il s'articule autour de 6 chapitres principaux et d'une annexe présentant des évaluations et un exemple complet de réécriture de code.

Introduction. La courte introduction de 2,5 pages présente le problème, le principe de solution, c'est-à-dire l'équivalence entre les modèles événementiels et de parallélisme de flux, ainsi que l'organisation de la thèse. J'ai un peu regretté que ce chapitre soit si court : on comprend l'idée principale de la thèse, mais on ne peut pas connaître les principaux résultats de la thèse sans la lire entièrement.

Chapitre 2. Le chapitre 2 présente le contexte et les objectifs de la thèse. Ce chapitre présente le problème du développement logiciel en Javascript et montre la difficulté qu'ont aujourd'hui les développeurs lorsqu'ils

veulent rendre leurs applications efficaces. Ce chapitre présente aussi en détail les modèles événementiels et de parallélisme de flux. Ce chapitre est très agréable à lire et présente de façon convaincante le problème du compromis entre productivité et performance en Javascript.

Chapitre 3. Le chapitre 3 dresse un état de l'art du domaine. Ce long chapitre de 30 pages classifie toute une série de langages/bibliothèques en fonction du niveau de productivité et de performance (en terme de parallélisme) qu'ils apportent. L'étude met aussi ces propriétés en perspective avec le succès commercial de ces langages. Le chapitre est relativement complet et montre l'étendu des connaissances d'Etienne Brodu dans le domaine des langages. La classification est pertinente et montre que plus un langage permet un haut niveau de parallélisme, plus il est difficile à utiliser, ce qui le rend d'autant plus inadéquat pour produire rapidement les premières versions d'un logiciel. J'ai beaucoup appris en lisant ce chapitre.

Chapitre 4. Le chapitre 4 présente le principe de la thèse : réécrire automatiquement un code événementiel Javascript en utilisant le parallélisme de flux. Le chapitre introduit un nouveau terme : le fluxion. Un fluxion est une tâche dans le modèle de parallélisme de flux, mais possède en plus une étiquette permettant d'identifier quels fluxions ne doivent pas être exécutés en parallèle. Le principe de la thèse est d'identifier automatiquement les fluxions d'un programme Javascript, c'est-à-dire les gestionnaires d'événements du programme original, d'identifier les étiquettes des fluxions, d'identifier les messages que s'échangent les fluxions, et de réécrire le code pour manipuler explicitement les fluxions. Le chapitre décrit clairement la démarche d'Etienne Brodu et présente avec des exemples comment il compte la mettre en œuvre. Comme Etienne Brodu a pris le parti d'utiliser des techniques de compilation statique pour identifier les fluxions, j'aurai bien aimé que le chapitre discute aussi les techniques dynamiques qui auraient, intuitivement, semblées plus simples à mettre en œuvre dans un langage dynamiquement typé. Ce point est discuté en conclusion mais une discussion plus approfondie aurait eu toute sa place en chapitre 4.

Chapitre 5. Le chapitre 5 présente le compilateur proposé par Etienne Brodu. Ce compilateur suit exactement les principes présentés dans le chapitre 4. Le chapitre 5 présente les techniques de compilations utilisées et comment le code est réécris sur de nombreux exemples. Ces descriptions sont très claires. Le chapitre présente aussi les limites de l'approche. J'ai particulièrement apprécié ces discussions. La thèse d'Etienne Brodu défriche le domaine et il est normal que son compilateur ait des limites. Les présenter montre les points durs que de futurs doctorants pourront attaquer. Le chapitre, et la thèse, aurait toutefois gagné à présenter quelques évaluations de performances. Pour commencer, il est en effet difficile de savoir si les transformations appliquées à un programme Javascript ont un impact en terme de performance sur une exécution séquentielle. Cette évaluation devrait être relativement facile à mener. Ensuite, le but des transformations est de pouvoir exécuter en parallèle une application Javascript qui a été conçue séquentiellement. Montrer que ces transformations permettent d'augmenter les performances en exécutant en parallèle une application aurait définitivement prouvé que la démarche d'Etienne Brodu est pertinente. J'imagine que de mener cette évaluation dans le cadre d'une thèse est très difficile : il faudrait probablement modifier en profondeur la machine virtuelle Javascript (Node.js) ce qui n'est pas raisonnable en trois ans. Toutefois, d'expliquer pourquoi ces évaluations n'ont pas été menées pourraient éclairer le lecteur.

Conclusion. La conclusion récapitule les différentes contributions d'Etienne Brodu et proposent plusieurs perspectives intéressantes aux travaux, en particulier l'utilisation de techniques de compilation dynamique pour isoler les fluxions. Le chapitre de conclusion présente aussi des réflexions fort intéressantes qu'Etienne Brodu partage avec le lecteur sur sa vision du futur de l'informatique.

Au cours de son travail, Etienne Brodu a fait preuve de très solides compétences en informatique dans le domaine des langages et de la compilation. Etienne Brodu a su réaliser des développements difficiles et des expérimentations sérieuses. La démarche d'Etienne Brodu visant à réconcilier productivité pour le développeur et performance à l'exécution est tout à fait pertinente et va dans le sens de l'histoire. Je ne peux que saluer la démarche d'Etienne Brodu visant à réunir ces deux propriétés. Les travaux d'Etienne Brodu ont été publiés à plusieurs reprises dans des ateliers internationaux, mais je l'encourage à les publier dans une conférence de premier plan en langage : les travaux d'Etienne Brodu sont de grandes qualités et mériterait une publication de premier plan.

Pour résumer, les travaux de thèse d'Etienne Brodu sont de toute première qualité et j'estime donc qu'Etienne Brodu est digne de l'obtention du grade de docteur de l'université de Lyon.

Gaël Thomas
Professeur, Telecom SudParis