Préface

Depuis la première édition de ce livre, en 1986, le monde de la compilation a grandement changé. Les langages de programmation ont évolué, présentant de nouveaux problèmes de compilation. Les architectures des ordinateurs offrent des ressources diverses, que le concepteur d'un compilateur doit mettre à profit le mieux possible. Le plus intéressant est peut-être que l'optimisation de code, technique vénérable, a trouvé un emploi dans d'autres domaines. On l'utilise maintenant dans des outils qui recherchent les erreurs dans les logiciels et, mieux encore, qui mettent au jour des trous de sécurité dans des codes existants. Et on utilise toujours largement une grande part de la technologie de la partie frontale — les grammaires, les expressions régulières, les analyseurs syntaxiques et les traducteurs s'appuyant sur la syntaxe.

C'est pourquoi notre philosophie reste celle des versions précédentes de ce livre. Peu de nos lecteurs auront un jour à construire, ou même à maintenir, un compilateur pour un langage de programmation majeur. Pourtant les modèles, la théorie et les algorithmes associés aux compilateurs peuvent être mis à profit pour un grand nombre de problèmes liés à la conception et au développement logiciels. Nous mettrons donc l'accent sur les problèmes les plus fréquemment rencontrés lors de la conception des logiciels de traitement des langages, quels que soient le langage source ou la machine cible.

Comment utiliser ce livre

Pour couvrir tout le contenu de ce livre ou presque, un cours doit durer au moins deux trimestres, voire deux semestres. La première moitié du livre sera généralement utilisée dans un cours d'introduction en premier cycle. La deuxième moitié — traitant plus particulièrement de l'optimisation de code — servira de matière à un cours plus avancé, de second cycle ou au-delà.

Le premier chapitre expose les motivations ainsi que quelques notions fondamentales de l'architecture des ordinateurs et des principes des langages de programmation.

Le chapitre 2 présente un compilateur jouet et introduit de nombreux concepts importants, développés dans les chapitres suivants. Le compilateur lui-même fait l'objet d'une annexe.

Le chapitre 3 traite de l'analyse lexicale, des expressions régulières, des machines à états finis et des outils de production d'analyseurs lexicaux. Tout cela est fondamental pour tout outil de traitement automatique de texte.

Le chapitre 4 couvre les principales techniques d'analyse syntaxique, aussi bien descendantes (descente récursive, LL) qu'ascendantes (LR et ses variantes).

Le chapitre 5 introduit les principes fondamentaux des définitions et de la traduction dirigées par la syntaxe.

Le chapitre 6 montre comment utiliser les idées du chapitre 5 pour produire un code intermédiaire pour un langage de programmation typique.

Le chapitre 7 étudie l'environnement d'exécution des programmes, en particulier la gestion de la pile et la récupération du rebut.

Le chapitre 8 est consacré à la production de code cible. On y expose notamment la construction des blocs de base, la production de code pour les expressions et les blocs de base, ainsi que les techniques d'allocation de registres.

Le chapitre 9 introduit les techniques d'optimisation de code, notamment les méthodes d'analyse des flots de données et les algorithmes d'optimisation s'appuyant sur cette analyse.

Le chapitre 10 traite de l'optimisation au niveau des instructions cibles. On s'y intéresse à l'extraction du parallélisme possible entre courtes séquences d'instructions et à leur ordonnancement sur des processeurs capables de faire plusieurs choses à la fois.

Le chapitre 11 s'intéresse à la détection et à l'exploitation du parallélisme à plus grande échelle. On se concentre ici sur les programmes numériques exploitant de nombreuses boucles travaillant sur des tableaux multidimensionnels.

Le chapitre 12 traite de l'analyse interprocédurale. On y étudie l'analyse de pointeur, la synonymie et l'analyse de flot de données prenant en compte les suites d'appels de procédures permettant d'atteindre un point déterminé dans le code.

Les auteurs ont enseigné le contenu de ce livre dans les universités de Columbia, Harvard et Stanford. À Columbia, les huit premiers chapitres sont la base d'un cours d'introduction et de perfectionnement sur les langages de programmation et les traducteurs. Ce cours comporte un projet d'un semestre, dans lequel les étudiants, travaillant en petites équipes, définissent et implémentent un petit langage de leur propre conception. Les étudiants ont créé des langages couvrant des domaines d'application très variés, par exemple le calcul quantique, la synthèse musicale, le dessin, le jeu vidéo, les opérations matricielles. Ils utilisent des outils de construction de compilateurs comme ANTLR, Lex ou Yacc, ainsi que les techniques de traduction dirigée par la syntaxe des chapitres 2 et 5. Les chapitres 9 à 12 sont utilisés dans un cours de troisième cycle sur la génération de code et son optimisation pour des machines actuelles, y compris les architectures utilisant des réseaux de processeurs.

À Stanford, un cours d'initiation d'un trimestre se fonde sur une partie du contenu des chapitres 1 à 8, et comprend également une introduction à l'optimisation de code globale issue du chapitre 9. Un second cours de compilation couvre les chapitres 9 à 12 ainsi que le contenu plus avancé du chapitre 7 concernant la récupération de rebut. Les étudiants utilisent un système développé localement en Java, appelé `Joeq`, pour l'implémentation des algorithmes d'analyse de flot de données.

Connaissances préalables

Pour pouvoir tirer profit de cet ouvrage, le lecteur devra avoir une certaine « culture informatique », incluant au moins un cours avancé de programmation et des cours sur les structures de données et les mathématiques discrètes. Il lui sera utile de connaître plusieurs langages de programmation.

Exercices

Ce livre propose des exercices nombreux et approfondis pour pratiquement chaque section. Un point d'exclamation signale les exercices ou les parties d'exercice plus difficiles. Les exercices les plus délicats sont signalés par un double point d'exclamation.

Suivi sur la Toile

La page d'accueil de la version originale du livre est

`dragonbook.stanford.edu`

On y trouve notamment des errata [1] et des supports de cours. Les auteurs ont également prévu d'y placer des descriptions de compilateurs importants, faites par leurs implémenteurs eux-mêmes.

Remerciements

L'image de couverture de la version originale est due à S. D. Ullman de la société Strange Tonic Productions.

Jon Bentley a fait de nombreux commentaires pertinents sur le premier jet de certains chapitres. Nous avons également reçu des commentaires et des corrections de Domenico Bianculli, Peter Bosch, Marcio Buss, Marc Eaddy, Stephen Edwards, Vibhav Garg, Kim Hazelwood, Gaurav Kc, Wei Li, Mike Smith, Art Stamness, Krysta Svore, Olivier Tardieu et Jia Zeng, que nous remercions tous très sincèrement. Les erreurs qui subsistent sont bien évidemment de notre fait.

Monica Lam souhaiterait également remercier, pour dix-huit années d'apprentissage de la compilation, ses collègues de l'équipe de compilation SUIF : Gerald Aigner, Dzintars Avots, Saman Amarasinghe, Jennifer Anderson, Michael Carbin, Gerald Cheong, Amer Diwan, Robert French, Anwar Ghuloum, Mary Hall, John Hennessy, David Heine, Shih-Wei Liao, Amy Lim, Benjamin Livshits, Michael Martin, Dror Maydan, Todd Mowry, Brian Murphy, Jeffrey Oplinger, Karen Pieper, Martin Rinard, Olatunji Ruwase, Constantine Sapuntzakis, Patrick Sathyanathan, Michael Smith, Steven Tjiang, Chau-Wen Tseng, Christopher Unkel, John Whaley, Robert Wilson, Christopher Wilson et Michael Wolf.

A. V. A., Chatham, New Jersey
M. S. L., Menlo Park, Californie
R. S., Far Hills, New Jersey
J. D. U., Stanford, Californie
Juin 2006

[1] Les errata se réfèrent à la version américaine du livre. Lire page xxv le mot des traducteurs. (N.d.T.)