StenC

Cyrille Muller, Jean-Romain Luttringer

10 décembre 2017

Résumé

Ce projet avait pour objectif de créer un compilateur pour le langage StenC.

1 Présentation rapide de l'outil

1.1 Présentation Rapide

Le projet se compile via la commande *make*. L'exécutable généré se situe dans le dossier *bin/* du projet. L'analyse d'un fichier source se fait ensuite via la commande "*./bin/stenc* ***file***". Le fichier MIPS généré se nomme *out.s* et se situe à la racine du projet.

1.2 Fonctionnalités

1.2.1 Résumé

Hormis les fonctions et quelques restrictions détaillées ci-dessus, les structures de contrôles demandées (if, if else , for et while) ont été implémentées, ainsi que les tableaux et les stencils, dont des exemples de résultats sont disponibles ci-dessous.

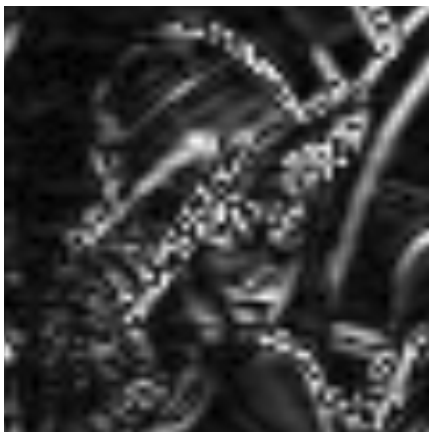
1.2.2 Détail

Certaines instructions ne sont pas supportées. Les incrémentations/décrémentation –i et ++i ne sont pas reconnues, seules i++ et i– le sont. De surcroît, ces dernières se comportent comme les précédentes : si elles se trouvent à l'intérieur d'une instruction, l'incrémentation/la décrémentation s'opérera à la fin de cette dernière. Les déclarations de variables peuvent être écrites *"int x ;"* , *"int x=0 ;"*. *"int x,y,z ;"* n'est pas une instruction valide. Cependant, *x=y=z=5 ;* l'est, si ces trois variables ont été déclarées précédemment.
Nous pouvons faire des #define CONSTANT int en tête de fichier pour définir des constantes entières.
Les printi(expression) ainsi que les printf(string) sont implémentés. Les déclarations de tableaux sont possibles, cependant elle ne peuvent s'opérer que de la manière suivante : *int tableau[2] ;*, *int tableau[CONSTANT] ;* si la constante à été #define, ou encore *int tableau[2]={1,2} ;* . Cependant, *int tableau[2] ; tableau={1,2} ;* ne sera pas supportée. Les accès aux variables des tableaux peuvent effectuées via des expressions (i.e *tab[5+6]* ou encore *tab[i]* sont valides). Les tailles de dimensions d'un tableau renseignée lors de la déclaration doivent être soit des entiers, soit des constantes définies à l'aide d'un #define (Le compilateur doit connaître la taille du tableau lors de la compilation pour allouer l'espace nécessaire.).
Les structures "if" et " if else" sont implémentées. Les structures de type "else if" doivent être écrites "else { if { ... }}".
Les fonctions n'ont pas été implémentées, il aurait fallu implémenter les variables locales et donc les définir à la volée dans la pile. Nous avons donc jugé que ce n'était pas interessant d'implémenter les fonctions sans cette fonctionnalité et nous nous sommes concentrés sur les stencils.
Outre ces restrictions, les déclarations, les structures "for", "while" , "if", les booléens, les tableaux et les stencils sont fonctionnels. Le dossier *tests/* contient des exemples compilables par notre programme. La plupart de ces exemples contiennent chaque structures de contrôles demandées. Les trois fichiers sources "gaussianblur.c", "sharpen.c", "sobel.c" contiennent des exemples d'application de stencil sur une image 64x64. La conversion de l'image en tableau C, ainsi que la conversion inverse ont été faite avec l'aide d'un script tiers.
Le fichier "sobel.c" contient le code source du sujet du projet , très légèrement modifié afin de respecter nos restrictions.
Quelques optimisations ont été apportées. Les calculs effectués sur des constantes sont opérés directement, sans charger les valeurs de ces dernières dans des temporaires, afin de limiter la taille de la table des symboles. L'initialisation des tableaux, si cette dernière est faite lors de la déclaration de ce dernier, est effectué directement dans le .data du code mips. Bien

que cette modification empêcher d'initialiser les valeurs du tableau via des expression, elle permet de corriger un problème survenant lorsque la section .text devient trop importante, en limitant la taille de ce dernier. En effet, des exceptions MIPS survenaient lorsque la taille de cette section dépassait un certain seuil.
Ci-dessous sont affichées les images résultats de nos application de stencil sur l'image lena.jpg. Le scripts ayant servis à convertir l'image en tableau C et vice versa sont dans le dossier *img2array*. img2array.py traduit l'image img_in en une déclaration de tableau C (output sur stdout). Array2img.c prend un fichier contenant une suite de chiffres séparés par des virgules en argument , et créer une image au format ppm . Seules les images en niveaux de gris sont acceptées.

2 Exemple de résultat d'application de stencils

(a) Image originale



(a) Application d'un filtre sobel via stencil (tests/sobel.c)

(b) Application d'un flou gaussien 5x5 via stencil (tests/gaussianblur.c)

(c) Application d'un filtre "sharpen" via stencil (tests/sharpen.c)