Vers l'exécution temps réel de la méthode MPS pour la détection automatique de fissures dans des images 2D de chaussées

Ph. Nicolle, V. Baltazart[1]

[1]IFSTTAR, LUNAM Université, CS4, 44340 Bouguenais, France

philippe.nicolle@ifsttar.fr, vincent.baltazart@ifsttar.fr

Résumé

Cet article présente les derniers avancements pour accélérer l'exécution de la méthode MPS, récemment proposée pour détecter automatiquement le squelette des fissures dans des images de chaussée. On considère successivement l'optimisation du code Matlab initial, le passage en programmation C, ainsi qu'une première tentative de parallélisation du code sous GPU. La réduction du temps d'exécution laisse espérer un traitement temps réel des images.

Mots Clef

Chemins minimaux, fissures, chaussée, optimisation, parallélisation, OpenCL, GPU.

Abstract

This paper presents the latest improvements to reduce the computer time of the MPS method, which accurately affords the automatic detection of cracks within grey level pavement images. Some optimization of the Matlab coding has been undergone at first, and then, the C-programming version has been used. Finally, parallel GPU programming has been tested (provided some adaptation of the coding). The performance makes possible the real time processing of the pavement images collected at traffic speed.

Keywords

Minimal path selection, cracking, pavement, optimization, parallelization, OpenCL, GPU

1 Introduction

La méthode MPS (Minimal Paths Selection) a été récemment proposée pour détecter automatiquement les fissures dans des images 2D de chaussée [1-2]. La méthode fournit de bonnes performances de segmentation, mais souffre d'un temps d'exécution important.

2 Travaux antérieurs

Le travail initial sur la méthode MPS (Minimal Paths Selection) a fait l'objet d'une thèse en collaboration avec l'IRCCyN et l'IRIT [1]. Supposant que les fissures sont représentées par des pixels sombres connectés entre eux, la méthode MPS permet de déterminer le squelette de la fissure en sélectionnant les chemins de coût minimal. Le calcul des chemins élémentaires est basé sur l'algorithme de Dijkstra.

MPS comporte 5 étapes, dont les trois premières permettent de segmenter le squelette de la fissure. Les 2 étapes de post-traitement permettent de « nettoyer » le squelette et d'estimer l'épaisseur le long de la fissure.

Les performances de segmentation de MPS se caractérisent par un taux de similitude (DICE) plus élevé que les autres méthodes de la littérature (Markov par exemple) [1,2]. En contrepartie, le temps d'exécution du code sous Matlab est important.

Le travail dans [3] a permis d'optimiser le code Matlab initial de [1], pour segmenter le squelette de la fissure. Ainsi, le nombre de points d'amorce a été significativement réduit, ainsi que le nombre de fausse alarme. L'implémentation Matlab du code de Dijkstra a fait appel à des fonctions plus économes en temps calcul. Enfin, la méthode de balayage de l'image a été optimisée en utilisant des blocs de 16×24 pixels. Globalement, le temps d'exécution s'est réduit d'un facteur 30.

4 Programmation en C

Le code Matlab optimisé dans [3] a servi de base pour une implémentation en langage C avec l'API Open CL [4]. A partir des points amorces sélectionnés, un seul processeur recherche les chemins minimaux dans des blocs de 16×24 pixels. Le passage de Matlab à C implique de gérer une file de priorité.

Le temps d'exécution s'est réduit d'un facteur 200, sans pour autant pouvoir atteindre une exécution temps réel.

La limite temps réel est définie par rapport à une vitesse d'auscultation de 90 km/h, soit une cadence de 25 images/seconde. La taille des images de dimension 4×1 m² varie de 1 à 4 MPx, selon le système imageur. La limite temps réel correspond alors à une vitesse de traitement variant de 40 à 10 ms/MPixels.

5 Parallélisation sous GPU

La parallélisation en utilisant les CPU spécialisés des cartes graphiques (Graphics Processing Unit ou GPU) permet de redonner de l'intérêt à l'utilisation d'algorithmes coûteux en temps de calcul. Elle implique en contrepartie d'adapter le principe des algorithmes. Plusieurs parallélisations de Dijkstra ou de calculs de chemins minimaux ont été proposées dans la littérature, avec OpenCL ou CUDA, e.g., [5].

En ce qui concerne MPS, le partitionnement de l'image de chaussée en blocs de 24×16 pixels permet la parallélisation de la recherche des chemins minimaux. Deux méthodes de mise en œuvre sont en cours de test [4]. La première reprend le principe de l'implémentation réalisée en C, en gérant une file de priorité. La seconde, utilise un balayage horizontal et vertical de blocs 24×16 pixels. Dans ce second cas, le balayage démarre sur la ligne ou la colonne d'un point amorce source, et un

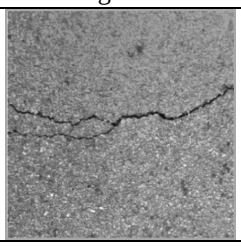
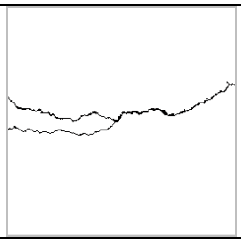
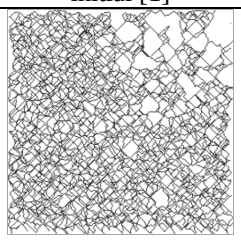
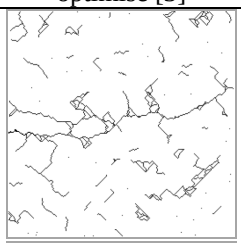
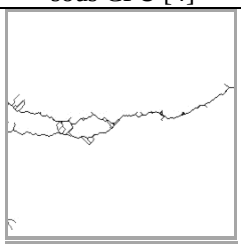
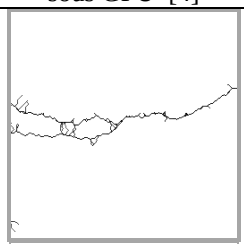
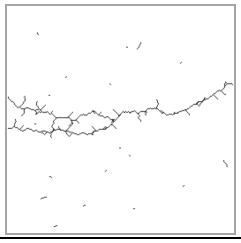
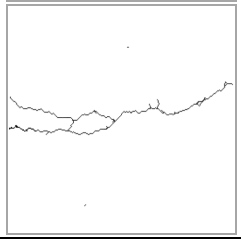
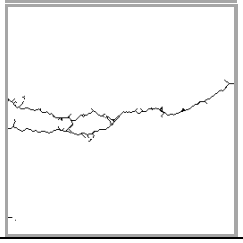
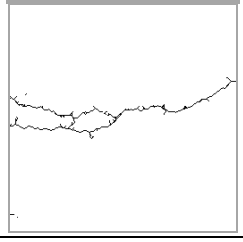
GPU est associé à un pixel de la ligne ou de la colonne balayée.
La réduction du temps de calcul laisse espérer un traitement temps réel des images de chaussées, à partir d'une carte graphique de 2800 cœurs environ.

6. Illustration

Le tableau suivant illustre le fonctionnement des différentes versions de MPS pour segmenter le squelette d'une fissure de chaussée sur une image de test 960×456 pixels (0.44 MPx). Pour l'illustration, une portion 256×256 de l'image initiale est représentée ; de même, l'étape 1 de l'algorithme n'est pas représentée. De même, l'image de pseudo-vérité terrain (PVT) permet d'évaluer qualitativement la segmentation.
On constate que le travail dans [3] et [4] a permis de réduire significativement le nombre de chemins calculés dans la texture (fausse alarme), réduisant le temps de calculs d'autant.
La vitesse d'exécution a été convertie en ms/MPx par une simple règle de proportionnalité ; ce chiffre est approximatif car le gain de temps est variable d'une image à l'autre selon le nombre d'éléments à détecter.
La programmation en C permet une accélération du traitement. La parallélisation sous GPU réduit le temps d'exécution d'un facteur supplémentaire. L'utilisation d'une carte GPU spécifique (cas b) permet d'atteindre le seuil de 10 ms/MPx, ce qui laisse espérer un traitement temps réel des images issues de capteurs récents.
Comme l'implémentation sous GPU a nécessité une modification de la méthode initiale, l'optimisation des 5 paramètres de MPS doit être reconsidérée, ainsi que la validation statistique de MPS à partir des bases d'images utilisées dans [1].

Bibliographie

[1] R. Amhaz, S. Chambon, J. Idier, V. Baltazart, Automatic crack detection on 2D pavement images : An algorithm based on minimal path selection, accepted to IEEE TITS, août 2015
[2] V. Baltazart, J-M. Moliard, R. Amhaz, L-M. Cottineau, A. Wright, M. Jethwa, Automatic crack detection on pavement images for monitoring road surface conditions – Some results from the collaborative FP7 TRIMM project, Rilem Conference, Nantes, France, Juin 2016.
[3] L. Yang, Contributions to improve and speed the existing MPS-based algorithm for both the semi-automated and the automated crack segmentation on 2D pavement images, rapport Master Aria, LUNAM Université, Ecole Centrale de Nantes, août 2015
[4] Ph. Nicolle, Rapport interne, IFSTTAR, Avril 2016.
[5] J. Kemp, All-pair shortest path algorithms using CUDA, Master thesis, Durham University, 2012.

	Image brute	Pseudo-Vérité Terrain		
Portion 256×256 pixels de l'image 960×456 qui a servi de test				
Segmentation du squelette	code Matlab initial [1]	Code Matlab optimisé [3]	Code C sous CPU [4]	Parallélisation sous GPU [4]
<u>Etape 2</u>: calcul des plus courts chemins entre points amorces (sélectionnés à l'étape 1)				
<u>Etape 3</u>: squelette « brut » de la fissure, après sélection des chemins minimaux.				
temps / MPx	~23 mn.	~46 sec.	(a) 220 ms ; (b) 105 ms	(a) 40 ms; (b) 9 ms
Matériel	HP Zbook14 Intel Core i7 @ 2.7 GHz	HP Zbook14 Intel Core i7 @ 2.7 GHz	a) Dell 3500 (Intel Xeon dual-core 2 GHz) b) Dell T1650 (Intel Xeon quad-core 3.2 GHz)	Dell 3500 Intel Xeon dual-core @2 GHz; a) Quadro 600-PCI Express 2.0 à 96 coeurs; b) Nvidia GTX 980-Ti PCI express 3.0 à 2816 coeurs