Anne-Elisabeth Lelièvre 16 084 130

Olivier Beauséjour 16 099 883

Devoir en Processus concurrents et parallélisme

Projet de session

Rapport de travail

Dans le cadre du cours IFT630

Présenté à :

Gilles Brunet

Université de Sherbrooke

10 août 2018

**Table des matières**

[1. Présentation de l’application 2](#_Toc520724982)

[2. Démarche et recherches 2](#_Toc520724983)

[3. Description du mode de déploiement 2](#_Toc520724984)

[4. Références 3](#_Toc520724985)

# Présentation de l’application

Nous avons décidé de faire un projet pratique basé sur un traitement parallèle pour la recherche d’images par le contenu. Le but principal de notre projet est de faire un analyseur d’image de pixel. Le cœur du programme est que nous avons des images de références d’une table de cartes dans le dossier ressources et notre analyseur doit déterminer la nature de nos 2 cartes dans notre main. Notre analyseur détermine seulement que la valeur de la carte.

# Démarche et recherches

Pour nos recherches, nous avons un intérêt commun pour les jeux de cartes. De plus, après avoir complété le cours d’intelligence artificielle, nous avons acquis des connaissances sur la méthode des k plus proches voisins. Cette approche est une méthode d’apprentissage supervisé. Dans notre cas, nous avons préalablement sélectionné des images de références que nous comparons ensuite aux images testées. Nous considérons la distance euclidienne entre chaque pixel de l’image. Ainsi, en considérant l’image de référence ayant la moindre distance avec l’image testée, nous pouvons déterminer la valeur de cette dernière. Un peu comme le devoir 2 avec le *BruteForceAttaque*, nous avons fait une version *SingleThread* et une version *MultiThread.* Nous comparons aussi la différence de temps d’exécution entre les deux méthodes. Nous avons également fait une version utilisant OpenCL.

# Description du mode de déploiement

Au sujet du découpage de notre code, nous avons tout d’abord fait une classe *Analyseur.cs* qui s’occupe de la logique de l’analyse de nos images. Dans cette classe, nous parallélisons le calcul des distances des pixels. La classe *Carte.cs* contient toutes les caractéristiques propres à une carte (sa valeur et son type). La classe *Constantes.cs* contient les coordonnées exactes en pixel des valeurs des cartes de notre main. La classe *Position.cs* sert à représenter des coordonnés. Les classes *Program.cs* et *frmMain.cs* contiennent l’ouverture de l’application et l’interface. Notre application contient 7 boutons au total. Les boutons *SingleThread* et *MultiThread* exécutent l’analyse des échantillons et retourne le temps d’exécution selon leur stratégie respective. Le bouton *OpenCL* exécute l’analyse des mêmes samples en utilisant la librairie *OpenCL*. Les boutons *sampleX* exécutent les tests des échantillons avec chaque stratégie (*MultiThread* et *SingleThread*) et affichent les résultats. Toutes les images sont répertoriées sous *Ressources*.

Au sujet du multi fils, nous avons utilisé un *task.run()* pour que chaque processus exécute le calcul des distance de pixel. Par contre, nous avons rencontré un problème lors d’une modification d’une de nos variables lors des calculs des lignes. Il nous a donc fallu corriger ce problème en ajoutant un lock pour garantir qu’un thread n’entre pas dans cette section critique du code. Alors si un autre thread tenterait d’entrer dans un code verrouillé, il attend, en restant bloqué jusqu’à ce que l’objet soit libéré.

En comparant la stratégie MultiThread avec la stratégie SingleThread, on constate que la stratégie MutiThread est systématiquement plus lente que la stratégie SingleThread. En effet, le temps d’exécution de la méthode MultiThread varie d’environ 200 à 240 alors que la version SingleThread varie d’environ 170 à 210ms. Cette différence est dû au fait que le calcul parallélisé nécessite la création de threads et de support de synchronisation supplémentaire. Le calcul étant relativement petit, ces temps sont considérables. La version OpenCL prend plus de 200ms lors de la première exécution, mais il est possible de sauvegarder le contexte une fois créer et de réduire le temps à 100ms. Encore une fois, la majeure partie du temps est consacré à cadré l’image des *samples*. Sans cette étape, l’analyse des images prend moins de 20ms.

# Références

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/lock-statement>

<https://www.ggpoker.com/poker-card-games/special-features/>

<https://msdn.microsoft.com/en-us/library/hh195051(v=vs.110).aspx>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/unsafe-code-pointers/pointer-types>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>

<https://msdn.microsoft.com/fr-ca/library/dd492132.aspx>

<https://msdn.microsoft.com/en-us/library/ms182351%28VS.80%29.aspx>

<https://msdn.microsoft.com/en-us/library/system.drawing.bitmap(v=vs.110).aspx>

<https://msdn.microsoft.com/en-us/library/5ey6h79d(v=vs.110).aspx>