Comptage de personnes dans un contexte simple

Projet parallélisme

Réalisée par :

BEN HAMOUDA Amel

Université Gustave Eiffel
Master 2 Science de l'image
2019/2020

Table des matières

I. Description du projet

L'objectif de ce projet est de se mettre dans le contexte d'une application simple afin d'appliquer par la suite la méthodologie d'optimisation des calculs / parallélisation à l'aide d'OpenMP.

L'application réalisée est un model de comptage de personnes dans un contexte simple, qui est développé en C++ à l'aide de la librairie OpenCV.

Ce rapport a pour but d'expliquer à l'utilisateur le mode de fonctionnement du projet et de décrire les différentes optimisations effectuées.

II. Mode d'emploi

Au lancement du programme, on doit faire un choix selon le rendu que l'on souhaite obtenir à la fin du programme, puis une fois le choix effectué on obtient ces fenêtres suivantes :

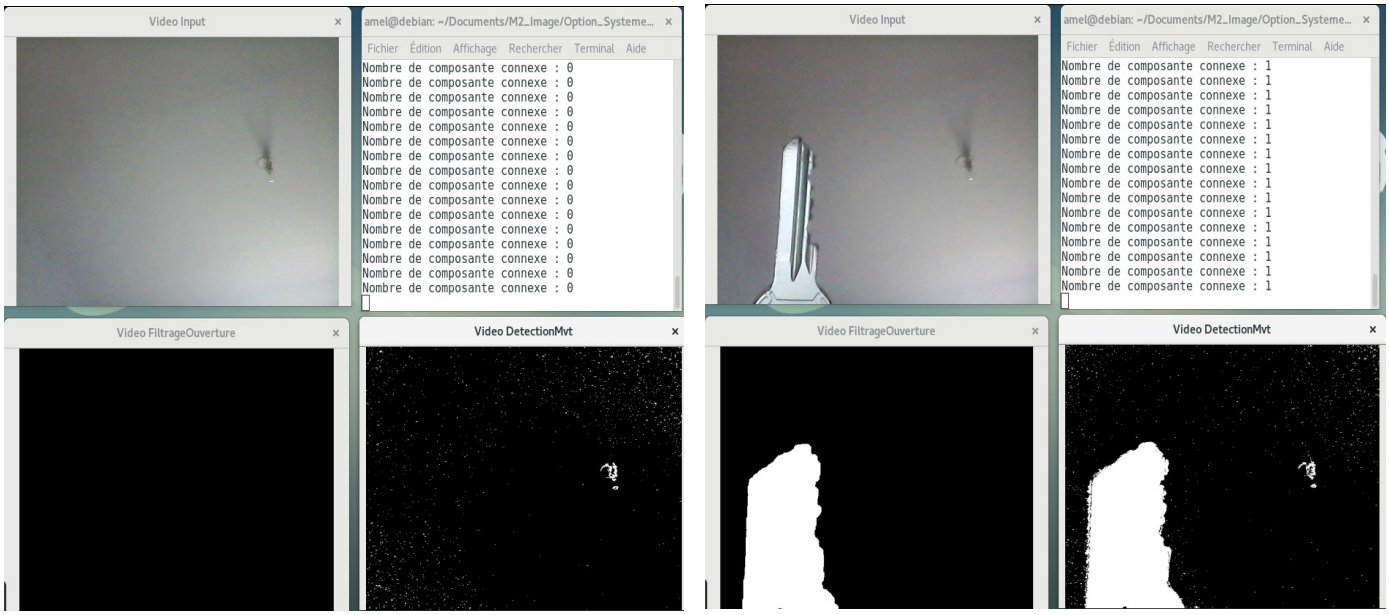


Fig-1. *Image gauche 1min après lancement du projet et image droite quand une clé bouge devant la caméra.*

→ La fenêtre en haut à gauche correspond à l'image de la caméra initiale sans aucune modification.
→ La fenêtre en bas à droite correspond à l'image de la caméra après avoir effectué les étapes de "calcul de l'arrière plan" et "calcul de la différence entre image arrière plan et acquise".
→ La fenêtre en bas à gauche correspond à l'image de la caméra après avoir effectué les étapes de "calcul de l'arrière plan", "calcul de la différence entre image arrière plan et acquise" et "filtrage par ouverture".

Et on peut voir sur le terminal le nombre de composantes connexes (personnes) détectées, à chaque prise de caméra.

III. Description de code

1. Naïf

A – Lecture de la vidéo

Pour la partie lecture de vidéo en continu on utilise la classe "VideoCapture" de la librairie OpenCv. On met la valeur 0 en paramètre de son constructeur pour lui indiquer que le point d'accès du flux est la caméra (/dev/video0).
Et une fois que le flux vidéo est ouvert on récupère chaque trame de la vidéo pour pouvoir la traiter dans la suite du programme, jusqu'à ce que l'utilisateur quitte le programme.

C.f. ici pour voir le code qui permet la lecture de vidéo.

B – Calcul de l'arrière plan

Pour la partie du calcul de l'arrière plan qui s'actualise à chaque image, on utilise cette formule qui calcule l'écart-type de l'illuminance de chaque pixel :

$$\mu(x,y) = \frac{1}{N}\sum_{i=1}^{N} I_i(x,y)$$

C.f. ici pour voir le code qui permet le calcul de l'arrière plan.

C – Calcul de la différence entre l'image d'arrière plan et acquise

Pour la partie du calcul de la différence entre l'image de l'arrière plan et l'image acquise, qui sera seuillée pour détecter le mouvement significatif, on utilise cette formule :

$$\sigma(x,y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(I_i(x,y) - \mu(x,y)\right)^2}$$

Avec un seuil qui est réglé à la main pour que cela marche bien. Dans mon cas le seuil est à 3.

C.f. ici pour voir le code qui permet le calcul de la différence entre image arrière plan et acquise.

D – Filtrage par ouverture

Pour la partie du filtrage par ouverture, on utilise la méthode *morphologyEx()* de la librairie OpenCV, avec l'opération *MORPH_OPEN* qui effectue l'opération d'ouverture.

C.f. ici pour voir le code qui permet le filtrage par ouverture.

E – Comptage des composantes connexes

Pour la partie de comptage des composantes connexes, on utilise la méthode findContours() de la libraire OpenCV, qui remplie un vecteur de vecteurs de points. Et le nombre de composantes connexes (personnes) est la taille de ce vecteur.

C.f. ici pour voir le code qui permet le comptage des composantes connexes.

2. Temps d'exécution naïf

Les temps d'exécution disponible dans cette partie sont donnés en micro-secondes.

Fonction / étape du traitement	Temps d'exécution-A	Temps d'exécution-B
Lecture de la vidéo	14 596,9	1 822
Calcul de l'arrière plan	**726 904,14**	**720 039**
Calcul de la différence entre image arrière plan et acquise		
Filtrage par ouverture	4 369,6889	4 573
Comptage des composantes connexes	339,92	308
Total	*746 210,63*	*726 742*

Le temps d'exécution de la colonne du milieu (Temps d'exécution-A) est donné sur un moyenne de 100 acquisitions pour la lecture de la vidéo et 90 acquisitions pour le reste.

Et pour la dernière colonne (Temps d'exécution-B) c'est le temps d'exécution à la 50éme exécution.

3. Optimisé

La fonction / étape qui sera optimisée dans la suite de ce document est l'étape de "calcul de l'arrière plan" et "calcul de la différence entre image arrière plan et acquise". Puisque que c'est celle qui prend le plus de temps lors de l'exécution du programme.

A – Calcul coûteux

Pour optimiser cette étape on peut commencer par retirer les calculs coûteux sans modifier pour autant le rendu final du programme.
Du coup dans notre cas on modifie la fonction *detectionMouvement()* qui devient *detectionMouvementOpti()* en modifiant la ligne :

```
somme += (images[i].at<uint8_t>(y, x) - soustFond) * (images[i].at<uint8_t>(y, x) - soustFond);
```

par

```
calcul = images[i].at<uint8_t>(y, x) - soustFond;
somme += (calcul << calcul);
```

C.f. ici *pour voir le temps d'exécution.*

B – OpenMP

En plus de l'optimisation déjà effectuée ci-dessus, on rajoute l'optimisation en modifiant les fonctions pour effectuer des calculs parallèles sur une architecture à mémoire partagée. Cette optimisation est faite à l'aide de l'interface de programmation OpenMP.
Dans notre cas on a modifié la fonction *calculDetectionMouvement()* qui devient *calculDetectionMouvementOpti()* en ajoutant cette ligne :

```
#pragma omp collapse(3) private(i, j)
```

au-dessus de la double boucle. Ce qui a permis au programme d'aller un petit peu plus vite.

C.f. ici *pour voir le temps d'exécution.*

C – Optimisation compilateur

Passons maintenant à la dernière optimisation qui est effectuée par le compilateur, en ajoutant les options nécessaires à la compilation du programme.

Le compilateur possède trois niveaux d'optimisation importants, **-O1**, **-O2** et **-O3**, du moins optimisé au plus rapide. Les programmes générés seront en général de plus en plus gros et, surtout, la compilation devient de plus en plus longue au fur et à mesure que l'on monte dans les niveaux. Mais comme l'optimisation n'est pas une science exacte, il arrive souvent que le meilleur niveau soit -O2, le dernier pouvant parfois introduire des erreurs dans le code généré ou simplement ne pas être le plus rapide. -O2 est donc très souvent le meilleur choix à faire. Mais cela dépend énormément des algorithmes utilisés et de la manière dont on accède à la mémoire.

→ **-O1** c'est le niveau d'optimisation le plus basique. Le compilateur va essayer de produire un code plus rapide et plus compact sans prendre trop de temps de compilation. C'est très basique mais ça fait toujours le travail.
→ **-O2** c'est le niveau *recommandé* d'optimisation si vous n'avez pas de besoin spécifique. Il active quelques options de plus que -O1 . Avec -O2 , le compilateur va essayer d'augmenter la performance sans compromettre la taille et sans prendre trop de temps en compilation.
→ **-O3** c'est le plus haut niveau d'optimisation possible. Il active des optimisations qui sont coûteuses en terme de temps de compilation et d'usage de la mémoire. Compiler tous vos paquets avec cette option ne garantit pas une amélioration de la performance. En réalité, dans de nombreuses situations, cela ralentit le système à cause des binaires plus volumineux qui réclament plus de mémoire. De plus cette option est réputée pour casser de nombreux paquets. C'est pourquoi utiliser -O3 n'est pas recommandé.

Mais dans notre cas la meilleur optimisation du code est avec l'option -O3, comme il est dit ci-dessus la meilleur option dépend de l'algorithme.

C.f. ici *pour voir le temps d'exécution.*

Cette optimisation a aussi était effectué directement sur la compilation de l'algorithme naïf, pour comparer les résultats à la compilation du programme entre l'algorithme naïf et celui optimisé.

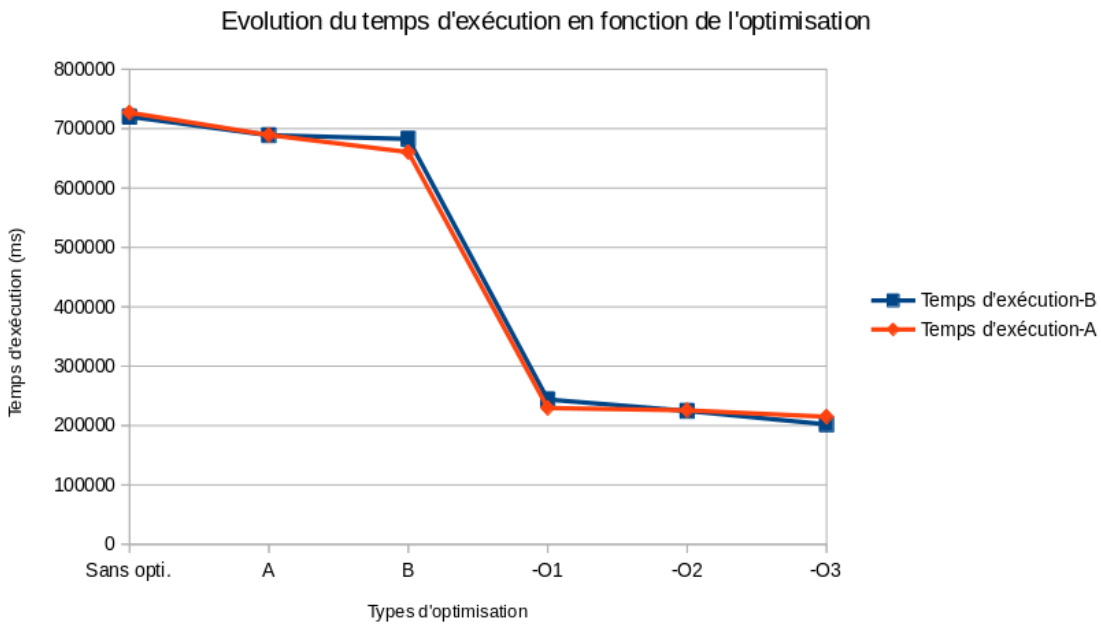
4. Temps d'exécution optimisé

Les temps d'exécution disponible dans cette partie sont sur l'algorithme optimisé et sont donnés en micro-secondes.

Optimisation	Sans opti.	A	B	C		
				-O1	-O2	-O3
Temps d'exécution-A	726 904,14	689 208,49	660 424,81	229 552,93	225 656,07	214 787,39
Temps d'exécution-B	720 039	688 909	682 706	243 836	224 578	201 899

Le temps d'exécution de la ligne du milieu (Temps d'exécution-A) est donné sur un moyenne de 90 acquisitions.

Et pour la dernière ligne (Temps d'exécution-B) c'est le temps d'exécution à la 50éme exécution.



Le graphique ci-dessus montre l'évolution du temps d'exécution en micro-seconde *(axe des ordonnées)* en fonction de l'optimisation effectuée *(axe des abscisses)*.
On remarque grâce au tableau et au graphique ci-dessus que plus on effectue des optimisations plus le temps d'exécution diminue.

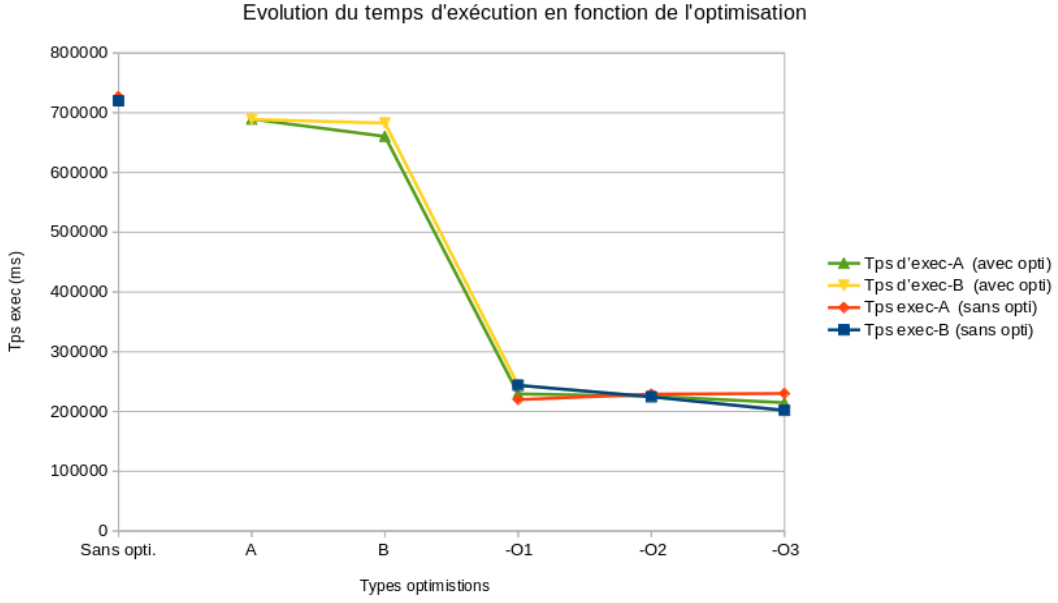
5. Optimisé vs Naïf

Puisque l'algorithme naïf donne un meilleur rendu visuel, j'ai donc décidé de tester l'optimisation du compilateur sur cette algorithme ce qui a donné ces résultats :

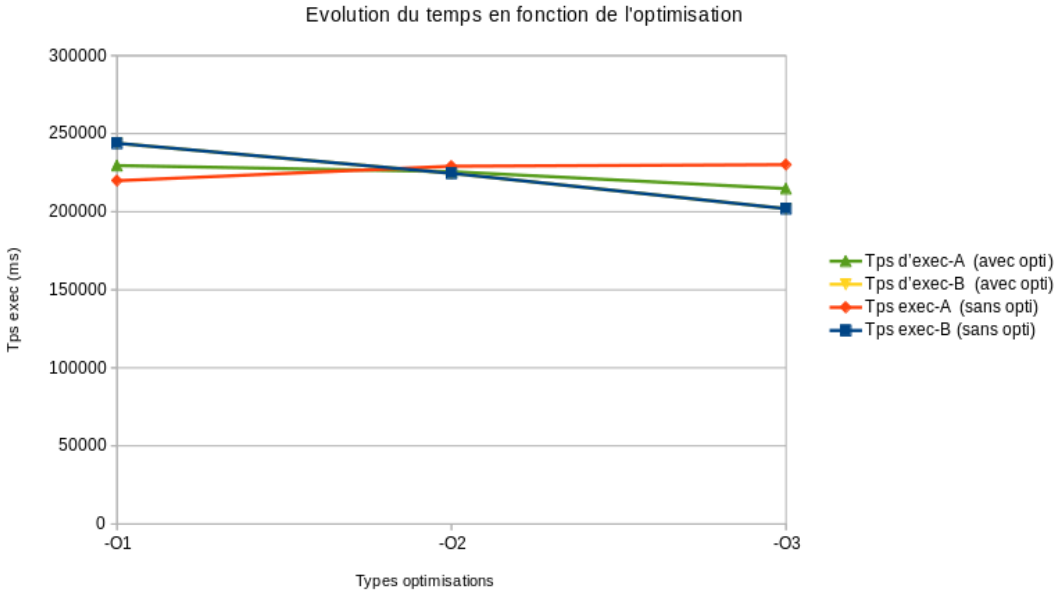
Optimisation	Sans opti.	-O1	-O2	-O3
Temps d'exécution-A	726 904,14	219 869,86	229 153,22	230 124,3
Temps d'exécution-B	720 039	243 836	224 578	201 899

On peut donc aussi garder l'algorithme naïf et juste lui appliquer l'option de compilation pour avoir un programme qui donne de bon résultat et qui est rapide.

Sur les graphiques ci-dessous on peut remarquer que sur les trois derniers points, le bleu et le jaune sont confondues.



Sur le graphique ci-dessus on observe le temps d'exécution du programme en fonction des optimisations effectués ou pas.



Sur le graphique ci-dessus on a un zoom des trois derniers points pour pouvoir bien voir qu'il n'y a pas de grande différence entre l'algorithme naïf et l'algorithme optimisé quand on leurs rajoutent à tous les deux l'option de compilation qui leur permet d'aller plus vite.

IV. Conclusion

Lors de ce projet j'ai pu mettre en application les connaissances liées à la programmation C++ et à la programmation parallèle (optimisée).
En développant un modèle de comptage de personnes dans un contexte simple.
Le modèle obtenu fonctionne plus ou moins bien.
Comme détaillé tout au long de ce rapport, plusieurs types d'optimisations ont était effectué pour baisser le temps d'exécution de l'étape de "calcul de l'arrière plan" et "calcul de la différence entre image arrière plan et acquise".
Au vu de ces modifications, la qualité de rendu final du programme est moyenne comparé à la qualité que donne l'algorithme naïf.
C'est pour cela que je laisse comme même le choix à l'utilisateur s'il veut un programme qui fonction bien (mais plus lent) ou s'il veut un programme qui fonctionne moyennement bien (mais plus rapide).

V. Annexe

1. Référence PC

Les valeurs de ce document ont étaient obtenues sur un ordinateur portable a 4 core et une fréquence d'horloge de 2200 Mhz.

2. Instructions de compilation

Pour compiler le programme il suffit de taper sur le terminal :

→ pour avoir l'algorithme naïf :

g++ `pkg-config --cflags opencv` -fopenmp projet.cpp `pkg-config --libs opencv` -o projet

→ pour avoir l'algorithme naïf et optimisé avec option de compilation pour aller plus vite :

g++ `pkg-config --cflags opencv` -fopenmp -OX projet.cpp `pkg-config --libs opencv` -o projet

avec *X* un nombre allant de 1 à 3 pour le type d'optimisation.

Pour l'exécution il suffit de taper :

./projet

Puis de choisir le rendu final que l'on souhaite :

→ *1* pour un meilleur rendu, mais pas optimisé au niveau du temps d'exécution,

→ *2* pour un rendu moyen, mais optimisé au niveau du temps d'exécution.

3. Documentations / Instructions d'utilisation

Pour lancer le programme il suffit de taper *./projet* dans le terminal.
Ensuite voir Mode d'emploi pour la suite.

4. Codes naïfs

A – Lecture de la vidéo

```
VideoCapture videoCapture = VideoCapture(0);
if (!videoCapture.isOpened()) {
    cout << "Error opening video stream" << endl;
    return -1;
}

Mat3b frame; // couleur
unsigned char key = '0';
int f = 0;
// Création de la fenêtre pour affichage du résultat
cvNamedWindow("Video Input", WINDOW_NORMAL);
cvMoveWindow("Video Input", 10, 30);

// boucle infinie pour traiter la séquence vidéo
while (key != 'q' && key != 'Q') {
    // acquisition d'une trame video - librairie OpenCV
    videoCapture.read(frame);
    if (frame.empty()) {
        cout << "Frame is empty" << endl;
        break;
    }
    imshow("Video Input", frame);
    key = waitKey(5);
}
//Liberer l'objet capture video
videoCapture.release();
//Fermer toutes les fenetres
destroyAllWindows();
```

B – Calcul de l'arrière plan

```
// Calcul de l'arrière plan qui s'actualise à chaque image
double soustractionFond(const vector<Mat> images, int N, int x, int y) {
    double first = 1.0 / N;
    double somme = 0.0;
    for (int i = 0; i < N; i++) {
        somme += images[i].at<uint8_t>(y, x);
    }
    return first * somme;
}
```

C – Calcul de la différence entre l'image d'arrière plan et acquise

```
// Calcul de la différence entre l'image d'arrière plan et l'image acquise
double detectionMouvement(const vector<Mat> images, int N, int x, int y, double soustFond) {
    double first = 1.0 / N;
    double somme = 0.0;
    //double soustFond = soustractionFond(images, N, x, y);
    double calcul = 0.0;
    for (int i = 0; i < N; i++) {
        calcul = images[i].at<uint8_t>(y, x) - soustFond;
        somme += calcul * calcul;
    }
    return sqrt(first * somme);
}
```

D – Filtrage par ouverture

```
// Filtrage par ouverture
Mat filtrageOuverture(Mat background) {
    Mat result = background;
    Mat kernel = getStructuringElement(MORPH_ELLIPSE , Size(10, 10), Point(1,1));
    morphologyEx(background, result, MORPH_OPEN, kernel, Point(-1,-1), 2, BORDER_CONSTANT, morphologyDefaultBorderValue());
    return result;
}
```

E – Comptage des composantes connexes

```
// Comptage des composantes connexes restantes
int nbComposanteConnexe(Mat background) {
    vector<vector<Point>> contours;
    findContours(background, contours, CV_RETR_EXTERNAL, CHAIN_APPROX_NONE);
    return contours.size();
}
```

5. Références

→ Cours d'OpenMP de Mr. GRANDPIERRE

→ Cours de parallélisme et d'optimisation de Mme. DOKLADALOVA

→ Optimisation de compilation :
https://wiki.gentoo.org/wiki/GCC_optimization/fr