Sorbonne Université
Faculté des Sciences et de l'ingénierie
Licence Informatique

Rapport
Projet Vision LO3IN401

Par :
ElHadi CHITER
Mehdi ZENINE

Encadrants :
LEMAITRE Florian
MAURICE Nathan

Année Universitaire : 2022/2023

Table des matières

1 Chaîne de traitements et algorithmes **2**
1.1 Chaîne de traitements . . . 2
1.1.1 Détection de mouvements . . . 2
1.1.2 Traitement du Bruit . . . 2
1.2 Algorithmes . . . 3
1.2.1 Algorithme ΣΔ . . . 3
1.2.2 Algorithmes de Morphologie . . . 5

2 Description des optimisations **5**
2.1 Scalarisation . . . 5
2.2 Rotation de registres . . . 5
2.3 Déroulage de boucles . . . 6
2.4 Réduction de colonnes . . . 6
2.5 Factorisation . . . 6
2.6 Software Pipelining . . . 6
2.7 SWP (Sub Word Parallelism) . . . 6
2.8 Fusion . . . 7

3 Analyse des résultats visuels **7**

4 Mesure des performances (Benchmarking) **8**
4.1 Optimisations sans SWP . . . 8
4.1.1 Pipeline . . . 8
4.1.2 Fusion . . . 9
4.1.3 Conclusion (Comparaison) . . . 9
4.2 Optimisations avec SWP8 . . . 9
4.2.1 Pipeline SWP8 . . . 10
4.2.2 Fusion SWP8 . . . 11
4.2.3 Conclusion (Comparaison SWP8) . . . 11
4.3 Optimisations avec SWP64 . . . 11
4.3.1 Pipeline SWP64 . . . 12
4.3.2 Fusion SWP64 . . . 12
4.3.3 Conclusion (Comparaison SWP64) . . . 13
4.4 Conclusion générale sur les performances . . . 13

1 Chaîne de traitements et algorithmes

1.1 Chaîne de traitements

1.1.1 Détection de mouvements

La chaîne de traitement commence par le passage par un algorithme de détection de mouvements. Un algorithme de détection de mouvements permet à partir d'une séquence d'images en niveaux de gris permet de fournir des images en noir et blanc qui permettent de connaître les objets en mouvement. Il existe de nombreux algorithmes de détection de mouvements, mais pour le projet on nous a demandé d'utiliser *Sigma-Delta (*$\Sigma\Delta$)

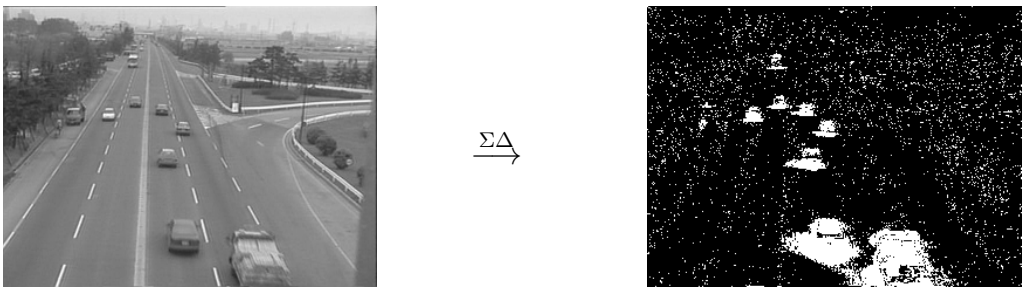

FIGURE 1 – Résultat du traitement avec $\Sigma\Delta$

1.1.2 Traitement du Bruit

Comme l'algorithme $\Sigma\Delta$ est un algorithme de détection de mouvements, alors il permet de détecter toute sorte de mouvements, y compris le bruit. Par là, on veut dire l'agitation de l'air, de la poussière et de la caméra. Ainsi, le résultat obtenu est brouillé et ne permet pas de voir de façon efficace le mouvement des objets étudiés. C'est pour cette raison, qu'une fois qu'on a appliqué notre algorithme de détection de mouvements, on applique par la suite nos algorithmes de morphologie pour enlever toute sorte de bruit. Dans ce qui suit on verra de façon détaillée la morphologie et ses principales opérations et leurs réalisations.

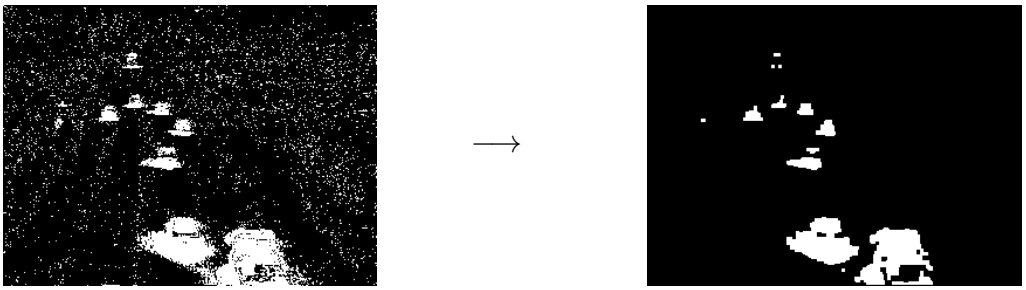

FIGURE 2 – Résultat de l'application de la morphologie

1.2 Algorithmes

1.2.1 Algorithme $\Sigma\Delta$

Nous allons voir dans cette partie, les différentes variables de l'algorithme $\Sigma\Delta$ et leurs correspondances, et essayer de comprendre comment elles sont mit à jour.

Variables
- V_{min}, V_{max} constantes.
- I_t est la matrice qui correspond à l'arrière plan de l'image à l'instant t.
- M_t est la matrice qui correspond à la moyenne de l'arrière plan de l'image à l'instant t.
- O_t est la matrice qui correspond à la valeur de l'observation entre les deux matrices M_t et I_t à l'instant t.
- V_t est la matrice qui correspond à la variance de l'arrière plan de l'image à l'instant t.
- $\hat{E}_t$ est la matrice qui correspond à l'estimation du mouvement de l'image à l'instant t.

Initialisations
- $M_0 = I_0$
- $V_0 = V_{min}$

Mises à jour

- La mise à jour de la matrice Moyenne M_t à l'instant de t dépend de la matrice M_{t-1} à l'instant t-1 et de la matrice I_t à l'instant t. Pour chaque pixel x, on compare $M_{t-1}(x)$ avec $I_t(x)$, et selon les trois cas, on mit à jour $M_t(x)$ différemment.
- La mise à jour de la matrice Observation O_t à l'instant de t dépend de la matrice M_t à l'instant t et de la matrice I_t à l'instant t. En effet, l'observation de chaque pixel $O_t(x)$ est la valeur absolue de la différence entre $M_t(x)$ et $I_t(x)$.
- La mise à jour de la matrice Variance V_t à l'instant t dépend de la matrice V_{t-1} à l'instant t, de la matrice O_t à l'instant t, et d'une variable N.
- La mise à jour de la matrice Estimation $\hat{E}_t$ à l'instant t dépend de la matrice O_{t-1} à l'instant t-1 et de la matrice V_t à l'instant t. En effet, pour chaque pixel x, $\hat{E}_t = 0$ si la variance du pixel à l'instant t est supérieure à son observation à l'instant t-1 ($V_t(x) > O_{t-1}(x)$) et $\hat{E}_t = 255$ sinon.

Vous trouvez dans la page suivante l'algorithme Sigma-Delta utilisé dans le cadre de ce projet.
Les calculs sont faits avec les constantes suivantes :

- $N = 3$.
- $V_{min} = 2$.
- $V_{max} = 253$.

Algorithm 1 $\Sigma\Delta$

$M_0(x) \leftarrow I_0(x)$
$V_0(x) \leftarrow V_{min}$ ▷ V_{min}, V_{max} constantes
for all $x \in$ pixels **do** ▷ Estimation de M_t(x)
  if $M_{t-1}(x) < I_t(x)$ **then**
    $M_t(x) \leftarrow M_{t-1}(x) + 1$
  else if $M_{t-1}(x) > I_t(x)$ **then**
    $M_t(x) \leftarrow M_{t-1}(x) - 1$
  else
    $M_t(x) \leftarrow M_{t-1}(x)$
  end if
end for
for all $x \in$ pixels **do** ▷ Calcul de la différence
  $O_t(x) \leftarrow |M_t(x) - I_t(x)|$
end for
for all $x \in$ pixels **do** ▷ Mise a jour de V_t(x)
  if $V_{t-1}(x) < N * O_t(x)$ **then**
    $V_t(x) \leftarrow V_{t-1}(x) + 1$
  else if $V_{t-1}(x) < N * O_t(x)$ **then**
    $V_t(x) \leftarrow V_{t-1}(x) - 1$
  else
    $V_t(x) \leftarrow V_{t-1}(x)$
  end if
  $V_t(x) \leftarrow max(min(V_t(x), V_{max}), V_{min})$
end for
for all $x \in$ pixels **do** ▷ Estimation de $\hat{E}_t$(x)
  if $O_{t-1}(x) < V_t(x)$ **then**
    $\hat{E}_t(x) \leftarrow 0$
  else
    $\hat{E}_t(x) \leftarrow 255$
  end if
end for

1.2.2 Algorithmes de Morphologie

Les opérateurs de morphologie permettent de restreindre l'image (Érosion) ou bien de l'étendre (Dilatation). Ces deux opérations sont implémentées en utilisant le opérateurs binaires OR et AND.

— **Pour la dilatation**, il suffit d'avoir dans son voisinage un pixel à 255 pour que le pixel soit mis à 255 dans le résultat (C'est pour cela qu'on utilise l'opérateur binaire OR).

— **Pour l'érosion**, il suffit d'avoir dans son voisinage un pixel à 0 pour que le pixel soit mis à 0 dans le résultat (C'est pour cela qu'on utilise l'opérateur binaire AND).

Après avoir défini l'érosion et la dilatation, on va passer a ce qui va servir dans le projet, l'ouverture et la fermeture.

— **L'ouverture** est l'application d'une érosion puis d'une dilatation sur l'image. Elle permet d'agrandir les trous qui sont à l'intérieur des composants.

— **La fermeture** est l'inverse de l'ouverture, c'est l'application d'une érosion puis d'une dilatation sur l'image. Ce procédé permet de combler les trous qui sont à l'intérieur des composants.

Durant le projet, on a essayé de coder les algorithmes d'érosion, de dilatation, d'ouverture et de fermeture avec différentes versions correspondantes aux optimisations.

2 Description des optimisations

2.1 Scalarisation

C'est le fait d'éviter d'utiliser directement les valeurs chargées (tableaux), mais les charger d'abord dans des variables, les manipuler et puis les sauvegarder en mémoire.

Impact attendu :

— Économiser des accès mémoire.
— Réutiliser les variables, par exemple si on doit faire un calcul sur une case d'un tableau, on va éviter de la recharger deux fois et donc on va réutiliser la variable.
— Une meilleure lisibilité du code.

2.2 Rotation de registres

La rotation de registres est la réutilisation de valeurs chargées au cours d'itérations précédentes. Utilisées beaucoup dans des boucles.

Impact attendu :

- Faire moins d'accès mémoire.
- Eviter de recalculer.

2.3 Déroulage de boucles

C'est l'une des optimisations les plus importantes. C'est l'augmentation du pas de la boucle, par exemple au lieu d'incrémenter de 1, on incrémente de 3.

Impact attendu :

- Minimiser le temps passé dans les conditions des boucles.
- Faire moins de sauts, qui sont coûteux.
- Un déroulage de boucle combiné à une rotation de registres a un grand impact sur le code, car elle permet de faire moins d'accès mémoire.

2.4 Réduction de colonnes

Cette optimisation se base sur le principe que beaucoup d'opérateurs sont séparables. Par exemple, une matrice 2×2 peut être vue comme le produit de deux vecteurs 1×2

Impact attendu :

- Réduire la taille des données sur lesquelles on fait les calculs.

2.5 Factorisation

Dans le cas de calculs similaires, on factorise les calculs pour éviter de les refaire.

2.6 Software Pipelining

C'est la parallélisation des taches faites dans les boucles. Il est mis en oeuvre en découpant les expressions complexes en expressions simples et puis faire le lien entre les expressions au cours des itérations.

Impact attendu :

- Paralléliser les calculs.
- Gagner en temps d'exécution.
- Réutiliser des calculs précédents, donc éviter de recalculer.

2.7 SWP (Sub Word Parallelism)

Il se base sur le principe qu'une donnée peut contenir un nombre de données (Par exemple si notre donnée est de taille 8 bits et que le registre où on fait nos calculs sont de taille 64 bits, alors on peut paralléliser les calculs en chargeant dans notre registre 8 données de 8 bits, au lieu d'une seule donnée). Ce principe peut être utilisé pour paralléliser les calculs.

Impact attendu :

- Paralléliser les calculs.
- Gagner en temps d'exécution.

— utilisation entière de nos ressources.

L'optimisation SWP a été combinée avec les autres optimisations : Déroulage de boucles, rotation de registres, réduction de colonnes.

2.8 Fusion

On a vu précédemment, que dans l'ouverture et la fermeture on utilisait une matrice intermédiaire dans laquelle on sauvegardait le résultat de la première opération (Érosion ou Dilatation). La fusion consiste en la suppression de cette matrice intermédiaire.

Impact attendu :

— Réduire le nombre d'accès mémoire.
— Réduire la complexité en mémoire.
— Et donc, accélérer les calculs.

3 Analyse des résultats visuels

On remarque que le résultat donnée par $\Sigma\Delta$ contient beaucoup de bruit. On peut voir ça dans la figure 1, en effet même la route et les arbres sont considérés comme des objets en mouvement. Et c'est la que la morphologie rentre en jeu. Elle permet de minimiser ou bien même de supprimer le bruit d'une image et de garder que les objets qui sont réellement en mouvement (voir figure 2).

4 Mesure des performances (Benchmarking)

4.1 Optimisations sans SWP

Les figures suivantes montrent les résultats des benchmarking (testé sur les algorithmes de l'ouverture).

	basic	fusion basic	fusion red	fusion ilu5_r	fusion elu2_r	fusion i5_e2_r	fusion i5_e2_rf	fusion ilu15_r	pipeline basic	pipeline red	pipeline ilu3_red	pipeline elu2_r	pipeline elu2_r	pipeline i3_e2_r	pipeline i3_e2_rf
i = 128	14.87	39.93	11.03	14.46	9.11	6.72	6.78	9.56	10.10	5.52	5.52	9.07	5.15	6.99	4.67
i = 136	10.17	25.68	10.95	8.04	8.23	6.47	6.53	15.22	13.97	8.66	7.69	13.24	9.36	10.72	4.63
i = 144	9.85	24.92	10.61	8.09	7.86	6.50	6.53	9.12	10.89	5.34	5.19	8.77	4.98	6.69	4.43
i = 152	9.83	24.90	10.59	7.81	7.85	6.31	6.32	8.11	9.73	5.33	5.32	8.79	4.99	6.76	4.53
i = 160	9.82	24.88	10.59	7.31	7.83	5.87	5.88	8.95	9.73	5.32	5.27	8.77	4.97	6.73	4.49
i = 168	9.83	24.90	10.58	7.97	7.96	6.36	6.37	8.14	9.72	5.31	5.18	8.73	4.96	6.67	4.46
i = 176	9.83	24.48	10.27	7.52	7.83	7.24	6.24	8.99	14.20	6.63	5.15	8.81	4.96	6.72	4.63
i = 184	9.50	24.64	10.29	7.74	7.59	6.03	5.87	7.72	9.15	5.00	4.95	8.22	4.67	6.31	4.23
i = 192	8.74	22.24	9.44	6.93	6.98	5.57	5.44	7.74	8.44	4.61	4.50	7.61	4.31	5.78	3.87
i = 200	8.51	21.13	8.96	6.18	6.63	5.00	5.02	7.20	8.44	4.60	4.58	7.60	4.30	5.82	3.91
i = 208	8.50	21.66	9.19	6.71	6.79	5.28	5.29	7.52	8.22	4.48	4.44	7.42	4.19	5.65	3.78
i = 216	8.28	21.12	8.96	6.38	6.62	5.17	5.18	6.99	8.37	4.47	4.39	7.42	4.19	5.61	3.76
i = 224	8.27	21.11	8.95	6.57	6.62	5.31	5.32	7.49	8.21	4.47	4.45	7.41	4.18	5.68	3.85
i = 232	8.27	21.11	8.94	6.42	6.62	5.20	5.21	6.99	8.21	4.46	4.43	7.39	4.18	5.69	3.83
i = 240	8.26	21.10	8.94	6.09	6.61	4.96	4.97	6.60	8.60	4.58	4.49	7.57	4.28	5.83	3.92
i = 248	8.25	21.47	9.17	6.62	6.78	5.36	5.36	7.01	8.19	4.45	4.44	7.39	4.17	5.70	3.83
i = 256	8.25	21.10	8.94	6.32	6.61	5.14	5.14	6.58	8.19	4.45	4.41	7.40	4.17	5.68	3.81
i = 264	8.25	21.09	8.93	6.49	6.60	5.26	5.26	7.02	8.19	4.44	4.37	7.38	4.17	5.63	3.74
i = 272	8.74	21.83	9.20	6.72	7.08	5.71	6.06	7.57	9.36	5.08	5.08	8.45	4.76	6.42	4.08
i = 280	8.91	22.20	9.16	6.24	6.77	5.10	5.10	7.21	8.39	4.55	4.52	7.58	4.27	5.76	3.85
i = 288	8.44	21.63	9.16	6.55	6.76	5.33	5.33	6.82	8.39	4.55	4.53	7.57	4.27	5.79	3.89
i = 296	8.44	21.64	9.15	6.44	6.78	5.25	5.27	7.23	8.39	4.55	4.59	7.56	4.27	5.81	3.90
i = 304	8.44	21.62	9.15	6.59	6.76	5.36	5.36	6.85	8.38	4.54	4.58	7.56	4.26	5.79	3.90
i = 312	8.65	21.88	9.15	6.49	6.76	5.28	5.29	7.24	8.37	4.54	4.52	7.56	4.26	5.78	3.88
i = 320	8.43	21.62	9.14	6.09	6.60	4.98	4.98	6.74	8.38	4.53	4.57	7.57	4.26	5.80	3.90
i = 328	8.42	21.07	8.91	6.42	6.59	5.18	5.18	7.08	8.17	4.42	4.45	7.38	4.15	5.64	3.78
i = 336	8.21	21.07	8.91	6.27	6.61	5.11	5.12	6.75	8.16	4.42	4.40	7.62	4.26	5.91	3.96
i = 344	8.64	21.66	9.19	6.58	6.79	5.35	5.35	7.29	8.37	4.53	4.56	7.55	4.26	5.80	3.89
i = 352	8.42	21.65	9.24	6.56	7.01	5.45	5.45	7.18	8.77	4.86	5.15	8.20	4.61	6.10	3.97
i = 360	8.41	21.64	8.95	6.11	6.61	5.00	5.00	6.38	8.16	4.41	4.39	7.37	4.15	5.79	3.87
i = 368	8.41	21.64	9.13	6.56	6.78	5.32	5.19	6.82	8.15	4.41	4.44	7.39	4.15	5.64	3.78
i = 376	8.19	21.10	8.95	6.32	6.81	5.27	5.24	6.70	8.16	4.58	4.43	7.39	4.15	5.63	3.76
i = 384	8.20	21.10	8.95	6.42	6.61	5.20	5.20	6.96	8.15	4.41	4.39	7.38	4.16	5.62	3.76

FIGURE 3 – Résultats des optimisations sans SWP

Comme vous pouvez le voir sur la figure 3, la toute première colonne correspond à une morphologie basique sans aucune optimisation, donc cette mesure (14.87) sera notre repère tout au long de l'analyse.

4.1.1 Pipeline

— Pour le pipeline, on voit que la meilleure mesure est 4.67 (3 fois plus petit que la mesure de référence) avec l'algorithme **ouverture3_pipeline_ilu3_elu2_red_factor**, et la pire mesure est 10.10 avec l'algorithme **ouverture3_pipeline_basic**. Ces résultats sont tout à fait logiques. En effet, dans le meilleur algorithme on fait deux déroulages de boucles, le premier est externe avec un pas de 2, et le deuxième est interne avec un pas de 3, rajouter à cela une factorisation de calcul (la rotation de registres est incluse aussi).

— Dans l'ensemble **toutes les performances des algorithmes Pipeline sont bien meilleurs que la performance de référence**, donc l'optimisation pipeline a bien récolté ses fruits. On constate aussi que certaines algorithmes sont meilleurs que d'autres, par exemple, l'algorithme **ouverture_pipeline_elu2_red** est plus performant que l'algorithme **ouver-**

ture_pipeline_ilu3_elu2_red, ce qui est compréhensible, parce que un tour de boucle dans le deuxième algorithme est plus coûteux par rapport à un tour de boucle dans le premier algorithme (dans le deuxième algorithme à chaque tour de boucle on fait 18 accès en mémoire (12 en lecture et 6 en écriture), ce qui est quand même énorme).

4.1.2 Fusion

- **petite remarque :** les deux algorithmes **ouverture_fusion_ilu5_elu2_red** et **ouverture_fusion_ilu5_elu2_red_factor** ont le même code, c'est pour cela qu'ils ont la même performance (sinon c'est impossible qu'ils aient la même performance).
- La première impression qu'on peut tirer c'est que les algorithmes de fusion ne sont pas si bons que ça en termes de complexité temporelle.
- On constate que la meilleure mesure est aux alentours de 6.72 (2 fois meilleure que la mesure de référence) avec l'algorithme **ouverture_fusion_ilu5_elu2_red_factor**. Tandis que la pire mesure est 39.93 (2.5 fois plus mauvaise que la mesure de référence).
- Les performances des algorithmes de Fusion ne sont pas convaincantes car la scalarisation était trop coûteuse (en moyenne 35 variables locales pour chaque algorithme), donc à un moment donné on aura plus de registres disponibles, donc on sera obligé de passer sur la pile. Ainsi, on aura plus d'accès mémoire que prévu, et donc plus de temps d'exécution.
- Quant à la complexité en mémoire, on peut être s'assurer sur le fait que la fusion est plus performante.

4.1.3 Conclusion (Comparaison)

Pour conclure cette analyse, si on veut gagner du temps, il est conseillé d'opter pour les optimisations Pipeline. Par contre, si on veut gagner de l'espace mémoire, il est conseillé de choisir les optimisations avec la fusion.

4.2 Optimisations avec SWP8

La première colonne de la fig.4 est la valeur de référence sur laquelle on va baser nos comparaisons par la suite.

swp8 basic	swp8 fusion basic	swp8 fusion red	swp8 fusion ilu3_r	swp8 fusion i3_e2_r	swp8 fusion i3_e2_rf	swp8 pipeline basic	swp8 pipeline ilu3_red	swp8 pipeline elu2_r_f	swp8 pipeline i3_e2_rf
1.90	7.05	3.95	4.20	2.96	2.95	1.88	1.15	0.99	1.27
1.89	6.98	3.91	4.23	2.91	2.92	1.82	1.12	0.96	1.22
1.83	6.79	3.78	2.93	2.01	2.01	1.81	0.95	0.96	1.08
1.82	6.80	3.80	3.93	2.73	2.72	1.82	1.06	0.95	1.19
1.82	6.85	3.78	4.04	2.83	2.87	1.81	1.07	0.95	1.19
1.82	6.82	3.81	2.94	2.00	2.02	1.82	0.93	0.95	1.07
1.84	6.77	3.77	3.72	2.49	2.50	1.74	0.99	0.91	1.13
1.70	6.39	3.46	3.54	2.36	2.38	1.61	0.92	0.84	1.04
1.57	5.90	3.28	2.52	1.73	1.74	1.56	0.79	0.81	0.92
1.56	5.89	3.27	3.14	2.17	2.17	1.56	0.87	0.81	0.99
1.56	5.89	3.19	3.16	2.18	2.19	1.52	0.85	0.79	0.96
1.52	5.74	3.19	2.44	1.68	1.69	1.51	0.76	0.79	0.89
1.52	5.74	3.18	2.99	2.08	2.06	1.51	0.82	0.79	0.95
1.51	5.72	3.18	3.08	2.12	2.13	1.51	0.83	0.79	0.95
1.55	5.89	3.26	2.50	1.72	1.72	1.55	0.77	0.81	0.90
1.51	5.73	3.18	2.93	2.03	2.03	1.51	0.81	0.78	0.93
1.51	5.74	3.26	3.01	2.07	2.08	1.51	0.82	0.78	0.94
1.61	5.78	3.18	2.43	1.67	1.68	1.51	0.75	0.78	0.89
1.63	6.20	3.43	3.11	2.16	2.11	1.58	0.84	0.82	0.97
1.54	5.88	3.26	3.03	2.09	2.09	1.54	0.83	0.80	0.95
1.54	5.87	3.25	2.49	1.71	1.71	1.54	0.76	0.80	0.89
1.54	5.87	3.29	2.91	2.00	2.01	1.54	0.81	0.80	0.93
1.64	5.94	3.25	2.99	3.23	2.05	1.54	0.85	0.82	0.96
1.54	5.87	3.29	2.48	1.71	1.71	1.54	0.75	0.80	0.88
1.62	5.91	3.25	2.87	1.99	1.99	1.53	0.80	0.80	0.93
1.50	5.72	3.23	2.87	1.98	1.98	1.50	0.79	0.78	0.90
1.66	5.91	3.25	2.48	1.71	1.72	1.53	0.75	0.80	0.88
1.54	5.87	3.24	2.84	1.97	1.97	1.53	0.79	0.80	0.92
1.58	6.02	3.33	2.98	2.05	2.06	1.57	0.82	0.82	0.95
1.53	5.87	3.25	2.48	1.76	1.75	1.57	0.76	0.81	0.89
1.49	5.72	3.16	2.74	1.90	1.90	1.49	0.77	0.77	0.89
1.49	5.72	3.16	2.80	1.93	1.93	1.49	0.77	0.77	0.89
1.49	5.71	3.16	2.41	1.66	1.66	1.49	0.72	0.77	0.85

FIGURE 4 – Résultats des optimisations avec SWP8

4.2.1 Pipeline SWP8

— On voit que les performances en pipeline restent assez proches de la performance de base (1.90) surtout le **pipeline_basic**.

— La version la plus optimisée de l'algorithme ***Pipeline SWP8*** qui est **pipeline_ilu3_elu2 _red_factor** est moins performante que **pipeline_elu2 _red_factor** car le déroulage de deux boucles nécessite beaucoup plus de variables locales, et donc l'utilisation de la pile (Accès en lecture et en écriture).

— On peut remarquer que les performances de **pipeline_ilu3_red** et **pipeline_elu2_red_factor** sont assez proches car les deux optimisations sont presque les mêmes, et sont efficaces.

— En gros, les performances algorithmes optimisés en **pipeline** sont bien meilleures que la performance de base, et donc on peut conclure que les optimisations marchent très bien et sont efficaces.

4.2.2 Fusion SWP8

— On peut vite remarquer que performances de la fusion sont faibles par rapport à la perf de référence, on justifie ça par l'utilisation de nombreuses variables (35 variables dans la variante basic) et donc le programme ne peut plus stocker dans ses registres et donc obligé d'allouer les variables dans la pile ce qui résulte des lectures et écritures en mémoire et donc des performances moins élevées.
— Quand on compare les optimisations de **fusion SWP8** entre elles, on peut voir que le déroulage de boucle externe et interne et plus efficace que les algorithmes sans déroulage de boucles, ce qui est logique car avec un déroulage de boucle on passe moins de temps dans le contrôle des boucles et les sauts.
— On peut conclure que l'optimisation en fusion ne sert presque à rien, car on perds vite en performances et on ne gagne rien à optimiser.

4.2.3 Conclusion (Comparaison SWP8)

Pour conclure cette analyse, on peut en déduire que la **Fusion SWP8** n'a pas servi à grand chose car ses performances sont plus mauvaise que la performance de référence. Contrairement au **Pipeline SWP8**, les optimisations sont très efficaces (sauf la basic car c'est la basique).

4.3 Optimisations avec SWP64

Les valeurs 0.0 sont présentes car les algorithmes n'ont pas été implémentés. On peut remarquer que d'ores et déjà que le **SWP64** et le plus performant de tous les algorithmes.

swp64 basic	swp64 fusion basic	swp64 fusion red	swp64 fusion ilu3_r	swp64 fusion elu2_r	swp64 fusion i3_e2_r	swp64 fusion i3_e2_rf	swp64 pipeline basic	swp64 pipeline red	swp64 pipeline ilu3_r	swp64 pipeline elu2_r	swp64 pipeline elu2_rf	swp64 pipeline i3_e2_rf
0.33	0.45	0.52	0.00	0.00	0.00	0.00	0.33	0.28	0.34	0.21	0.17	0.29
0.30	0.41	0.47	0.00	0.00	0.00	0.00	0.30	0.25	0.31	0.19	0.15	0.26
0.29	0.39	0.45	0.00	0.00	0.00	0.00	0.28	0.24	0.29	0.18	0.14	0.25
0.27	0.37	0.42	0.00	0.00	0.00	0.00	0.27	0.23	0.28	0.17	0.14	0.23
0.26	0.35	0.40	0.00	0.00	0.00	0.00	0.25	0.22	0.26	0.16	0.13	0.29
0.24	0.33	0.38	0.00	0.00	0.00	0.00	0.24	0.20	0.25	0.15	0.12	0.21
0.23	0.31	0.36	0.00	0.00	0.00	0.00	0.23	0.19	0.23	0.14	0.11	0.20
0.20	0.27	0.31	0.00	0.00	0.00	0.00	0.20	0.17	0.21	0.12	0.10	0.17
0.25	0.46	0.40	0.00	0.00	0.00	0.00	0.25	0.20	0.16	0.15	0.12	0.17
0.24	0.44	0.39	0.00	0.00	0.00	0.00	0.24	0.19	0.16	0.14	0.11	0.17
0.23	0.42	0.36	0.00	0.00	0.00	0.00	0.22	0.17	0.15	0.13	0.11	0.16
0.22	0.40	0.35	0.00	0.00	0.00	0.00	0.22	0.17	0.14	0.13	0.10	0.15
0.21	0.39	0.34	0.00	0.00	0.00	0.00	0.21	0.16	0.14	0.12	0.10	0.14
0.20	0.37	0.33	0.00	0.00	0.00	0.00	0.20	0.15	0.13	0.12	0.10	0.14
0.20	0.37	0.32	0.00	0.00	0.00	0.00	0.20	0.15	0.13	0.12	0.09	0.14
0.19	0.35	0.31	0.00	0.00	0.00	0.00	0.19	0.15	0.12	0.11	0.09	0.13
0.15	0.25	0.38	0.00	0.00	0.00	0.00	0.16	0.17	0.19	0.12	0.10	0.20
0.15	0.25	0.51	0.00	0.00	0.00	0.00	0.24	0.23	0.27	0.17	0.15	0.20
0.15	0.24	0.37	0.00	0.00	0.00	0.00	0.15	0.17	0.19	0.12	0.10	0.20
0.14	0.23	0.35	0.00	0.00	0.00	0.00	0.14	0.16	0.18	0.11	0.10	0.19
0.14	0.23	0.34	0.00	0.00	0.00	0.00	0.14	0.15	0.18	0.11	0.09	0.18
0.14	0.22	0.34	0.00	0.00	0.00	0.00	0.14	0.15	0.17	0.11	0.09	0.18
0.14	0.22	0.33	0.00	0.00	0.00	0.00	0.14	0.15	0.17	0.11	0.09	0.18
0.13	0.21	0.32	0.00	0.00	0.00	0.00	0.13	0.14	0.16	0.10	0.09	0.17
0.16	0.33	0.38	0.00	0.00	0.00	0.00	0.16	0.16	0.18	0.12	0.10	0.19
0.15	0.32	0.36	0.00	0.00	0.00	0.00	0.15	0.16	0.17	0.12	0.10	0.18
0.15	0.32	0.36	0.00	0.00	0.00	0.00	0.15	0.16	0.17	0.12	0.10	0.18
0.15	0.31	0.35	0.00	0.00	0.00	0.00	0.15	0.15	0.17	0.11	0.09	0.17
0.15	0.30	0.34	0.00	0.00	0.00	0.00	0.14	0.15	0.16	0.11	0.09	0.17
0.15	0.30	0.34	0.00	0.00	0.00	0.00	0.14	0.15	0.16	0.11	0.09	0.17
0.14	0.28	0.32	0.00	0.00	0.00	0.00	0.14	0.14	0.15	0.10	0.08	0.16
0.13	0.28	0.31	0.00	0.00	0.00	0.00	0.13	0.14	0.15	0.10	0.08	0.16
0.14	0.27	0.37	0.00	0.00	0.00	0.00	0.15	0.15	0.12	0.10	0.10	0.13

FIGURE 5 – Résultats des optimisations avec SWP64

4.3.1 Pipeline SWP64

— On voit que les performances en pipeline restent assez proches de la performance de base (0.33) surtout le **pipeline_basic**.

— La version la plus optimisée de l'algorithme ***SWP64*** qui est **pipeline_ilu3 _elu2_red_factor** est moins performante que **pipeline_elu2_red_factor** car le déroulage de deux boucles nécessite beaucoup plus de variables locales, et donc l'utilisation de la pile (Accès en lecture et en écriture).

— On peut remarquer que les performances de **pipeline_ilu3_red** et **pipeline_elu2_red_factor** sont assez proches car les deux optimisations sont presque les mêmes, et sont efficaces.

— En gros, les performances algorithmes optimisés en **pipeline** sont bien meilleures que la performance de base, et donc on peut conclure que les optimisations marchent très bien et sont très très efficaces.

4.3.2 Fusion SWP64

— On peut vite remarquer que performances de la fusion sont faibles par rapport à la perf de référence, on justifie ça par l'utilisation de nombreuses variables (35 variables dans la variante basic) et donc le programme ne peut plus stocker dans ses registres et donc obligé d'allouer les variables

dans la pile ce qui résulte des lectures et écritures en mémoire et donc des performances moins élevées.
— Quand on compare les optimisations de **fusion SWP64** entre elles, on peut voir que le déroulage de boucle externe et interne et plus efficace que les algorithmes sans déroulage de boucles, ce qui est logique car avec un déroulage de boucle on passe moins de temps dans le contrôle des boucles et les sauts.
— On peut conclure que l'optimisation en fusion ne sert presque à rien, car on perds vite en performances et on ne gagne rien à optimiser.

4.3.3 Conclusion (Comparaison SWP64)

Comme la conclusion précédente, on peut dire qu'un optimisation avec pipeline est plus efficace qu'une optimisation avec la fusion, après ça reste au choix du programmeur, si on veut une optimisation temporelle ou bien mémoire.

4.4 Conclusion générale sur les performances

— Le Pipeline est pour la complexité temporelle.
— La fusion est pour la complexité spatiale.
— SWP est très performant en termes de complexité temporelle et spatiale (on peut voir ceci avec les performances de SWP64).
— La scalarisation peut être coûteuse. En effet, lorsqu'on a plus de registres disponibles, on est obligé d'allouer sur la pile, qui dit allocation, dit plus d'accès mémoire.
— Le déroulage de boucle avec factorisation de calculs (rotation de registres aussi) peut être très intéressant à utiliser, dans le cas où on a pas de factorisation à effectuer dans la boucle, il est à voir.
— La réduction des colonnes est bonne aussi parfois. Elle est plus performante lorsque combiner avec la factorisation elle aussi.