Optimisation de l'exécution d'une application de flot optique sur une architecture parallèle hétérogène

Enrique GALVEZ

Encadrants:
Adrien CASSAGNE
Lionel LACASSAGNE
Alix MUNIER
Maxime MILLET

Sommaire

Sommaire

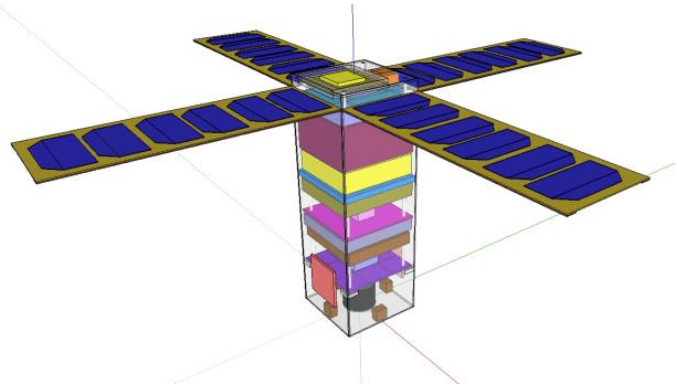
Motivations

METEORIX : mission universitaire dédiée à la détection et à la caractérisation des météores

Motivations

METEORIX : mission universitaire dédiée à la détection et à la caractérisation des météores

(a) Le logo



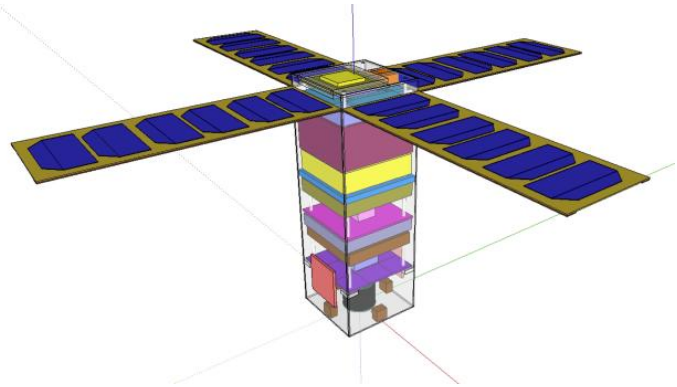
(b) Un CubeSat

FIGURE – METEORIX : faire du traitement d'image depuis un CubeSat

Motivations

METEORIX : mission universitaire dédiée à la détection et à la caractérisation des météores

(a) Le logo



(b) Un CubeSat

FIGURE – METEORIX : faire du traitement d'image depuis un CubeSat

Objectif du stage : Tester le déploiement de l'algorithme de traitement d'image sur une carte conçue pour l'embarqué

La carte NVIDIA Jetson TX2

FIGURE – La carte NVIDIA Jetson TX2

Performances constructeur :

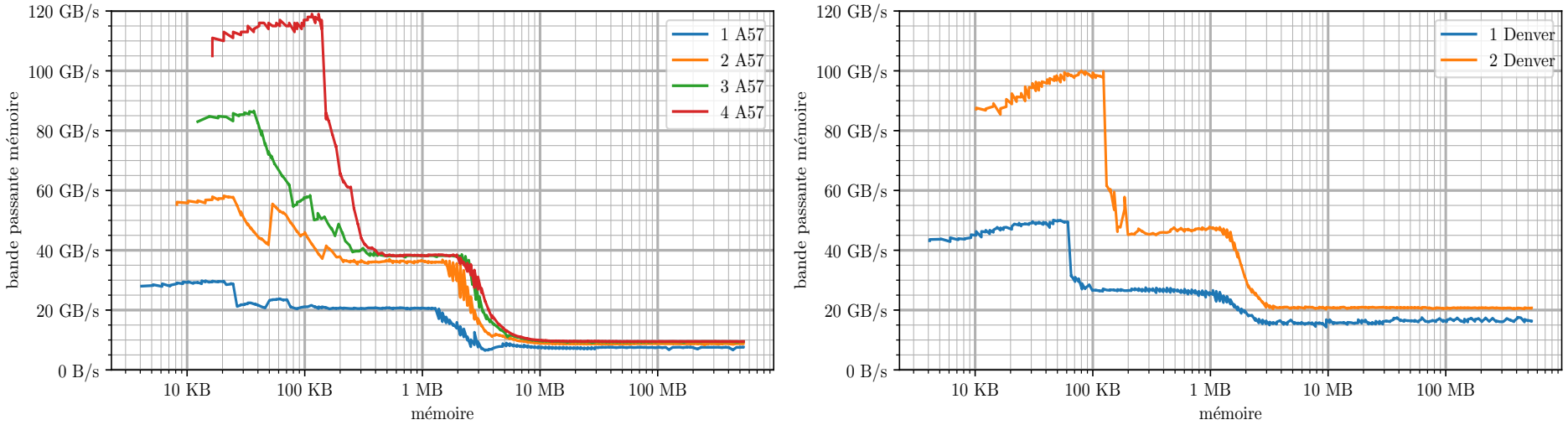
	Cœurs	Fréquence max	Cache L1	Cache L2
Denver 2	2	2.5 GHz	192 kiB	2 MiB
ARM Cortex A57	4	2 GHz	80 kiB	2 MiB

Bande passante mémoire

STREAM-TRIAD Benchmark qui évalue la *bande passante mémoire*

Bande passante mémoire

STREAM-TRIAD Benchmark qui évalue la *bande passante mémoire*



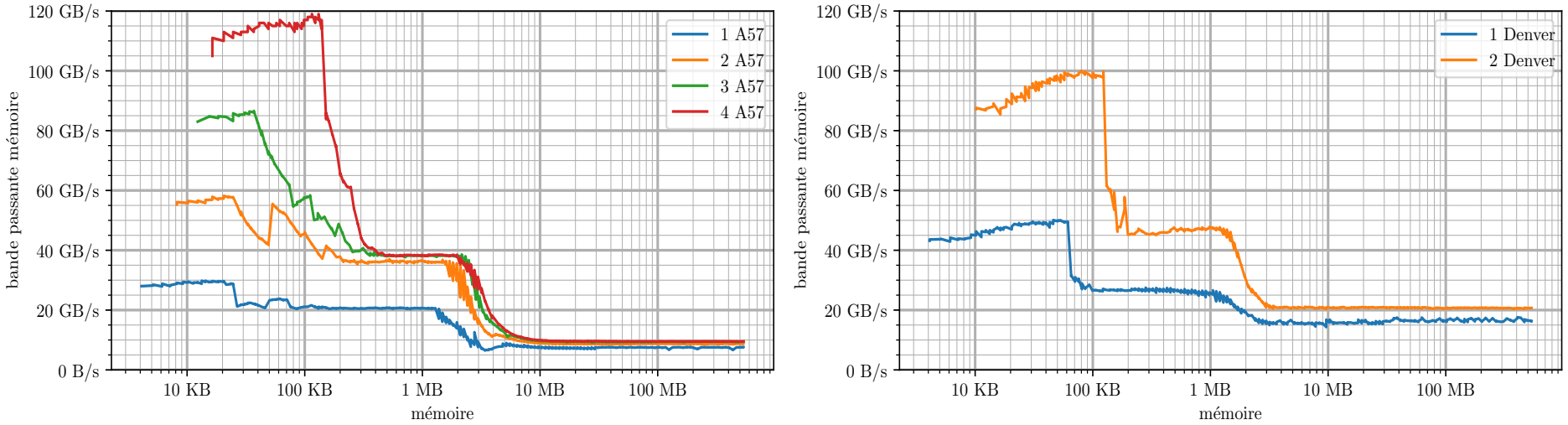
(a) Cœurs A57

(b) Cœurs Denver 2

FIGURE – Bande passante mémoire selon le type de cœur

Bande passante mémoire

STREAM-TRIAD Benchmark qui évalue la *bande passante mémoire*



(a) Cœurs A57

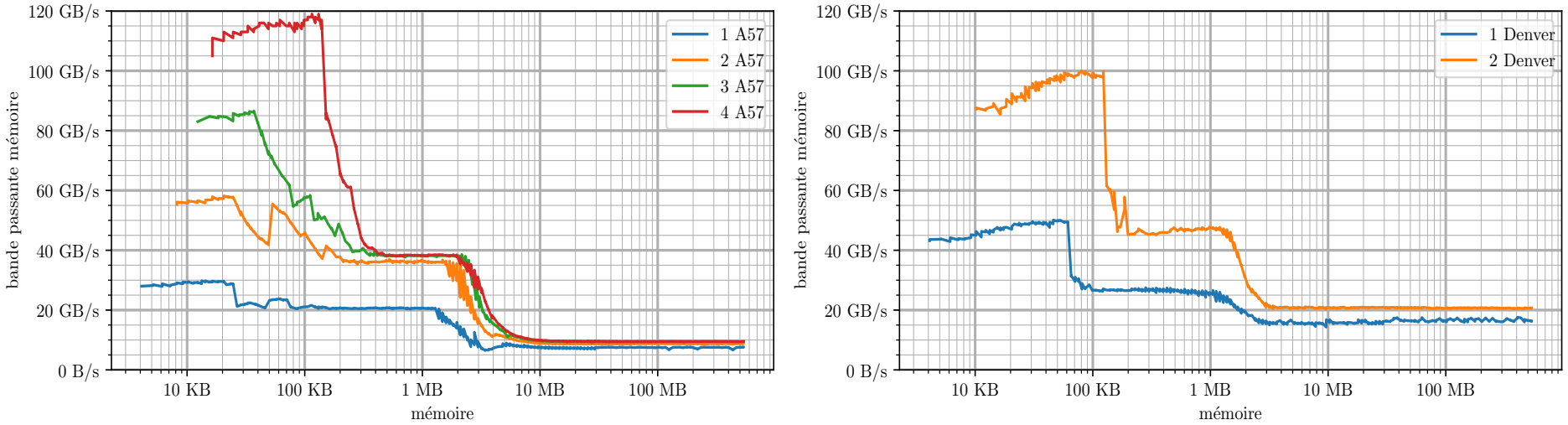
(b) Cœurs Denver 2

FIGURE – Bande passante mémoire selon le type de cœur

- Les plateaux correspondent aux caches des processeurs

Bande passante mémoire

STREAM-TRIAD Benchmark qui évalue la *bande passante mémoire*



(a) Cœurs A57

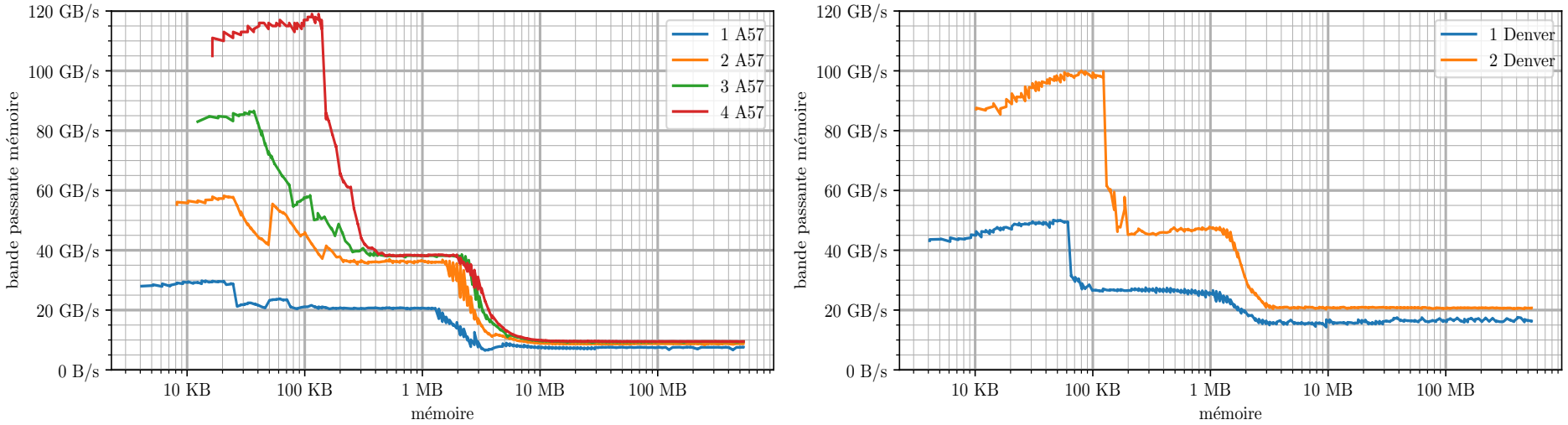
(b) Cœurs Denver 2

FIGURE – Bande passante mémoire selon le type de cœur

- Les plateaux correspondent aux caches des processeurs
- Pour les A57, L2 est partagé entre 2 cœurs

Bande passante mémoire

STREAM-TRIAD Benchmark qui évalue la *bande passante mémoire*



(a) Cœurs A57

(b) Cœurs Denver 2

FIGURE – Bande passante mémoire selon le type de cœur

- Les plateaux correspondent aux caches des processeurs
- Pour les A57, L2 est partagé entre 2 cœurs
- Un parallélisme efficace nécessite une bonne localité de données.

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

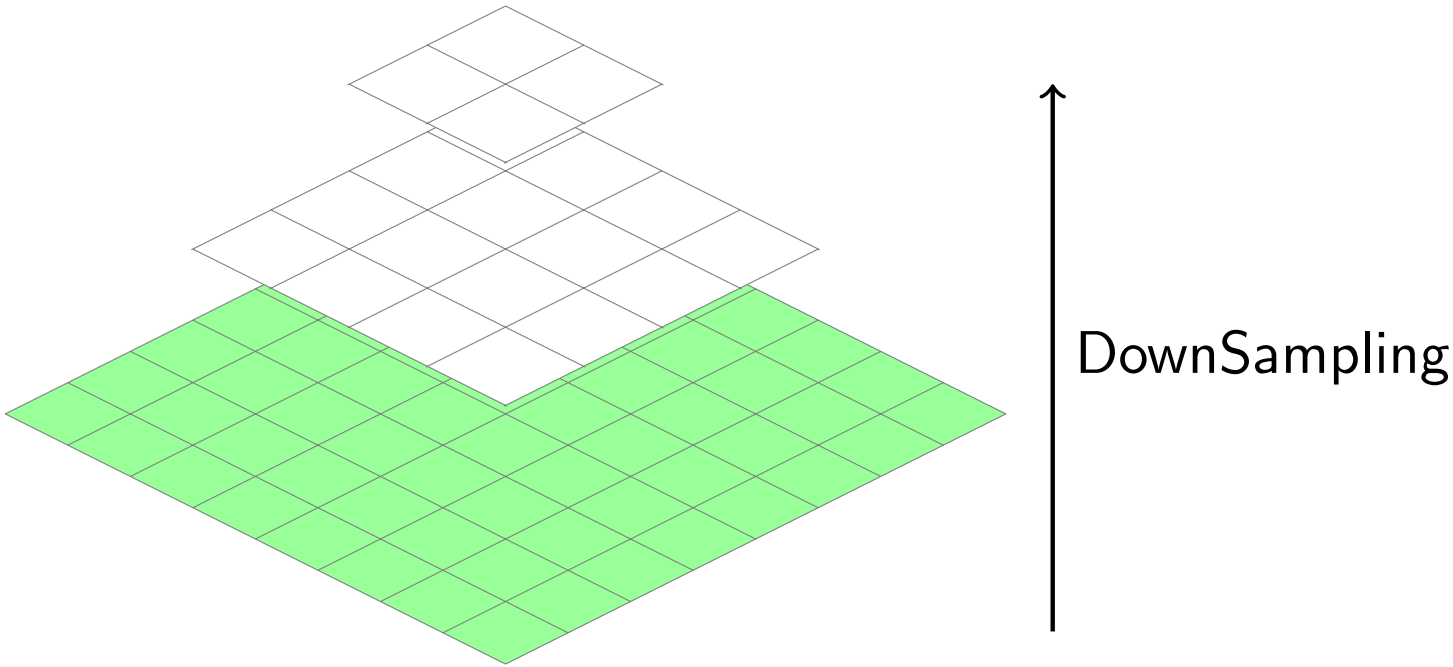


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

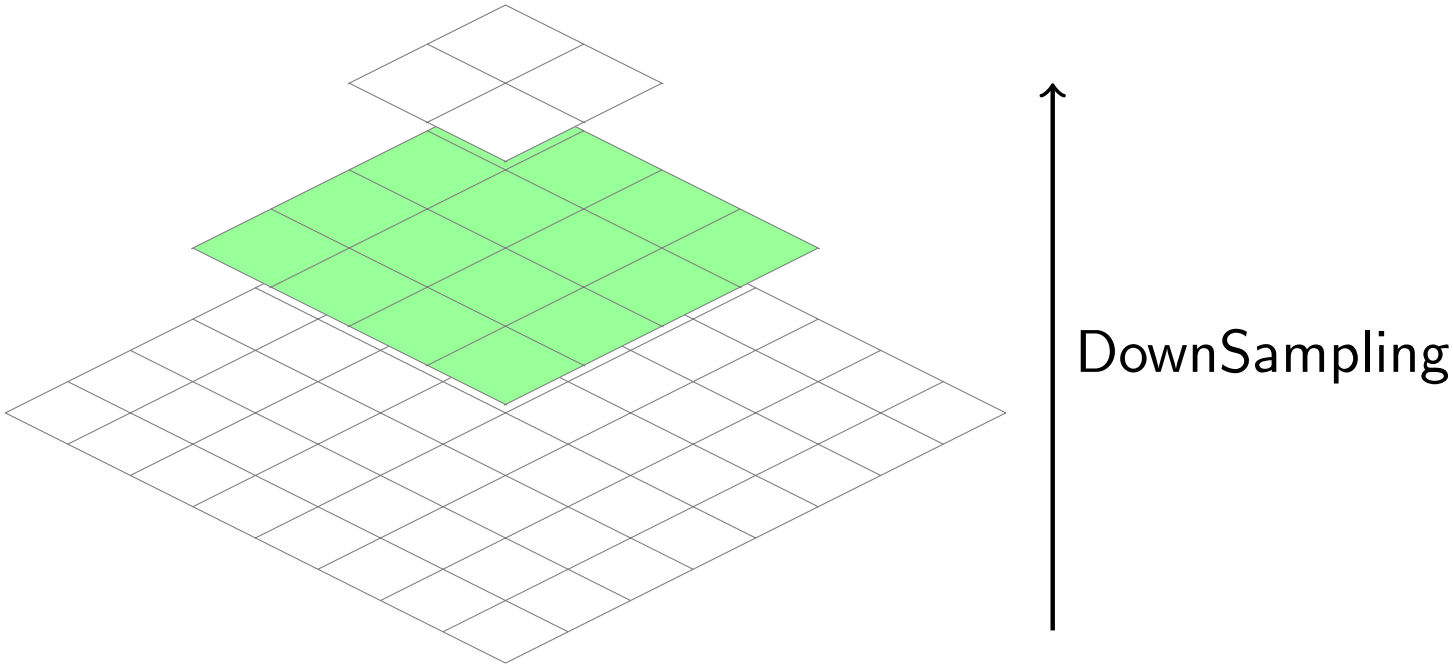


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

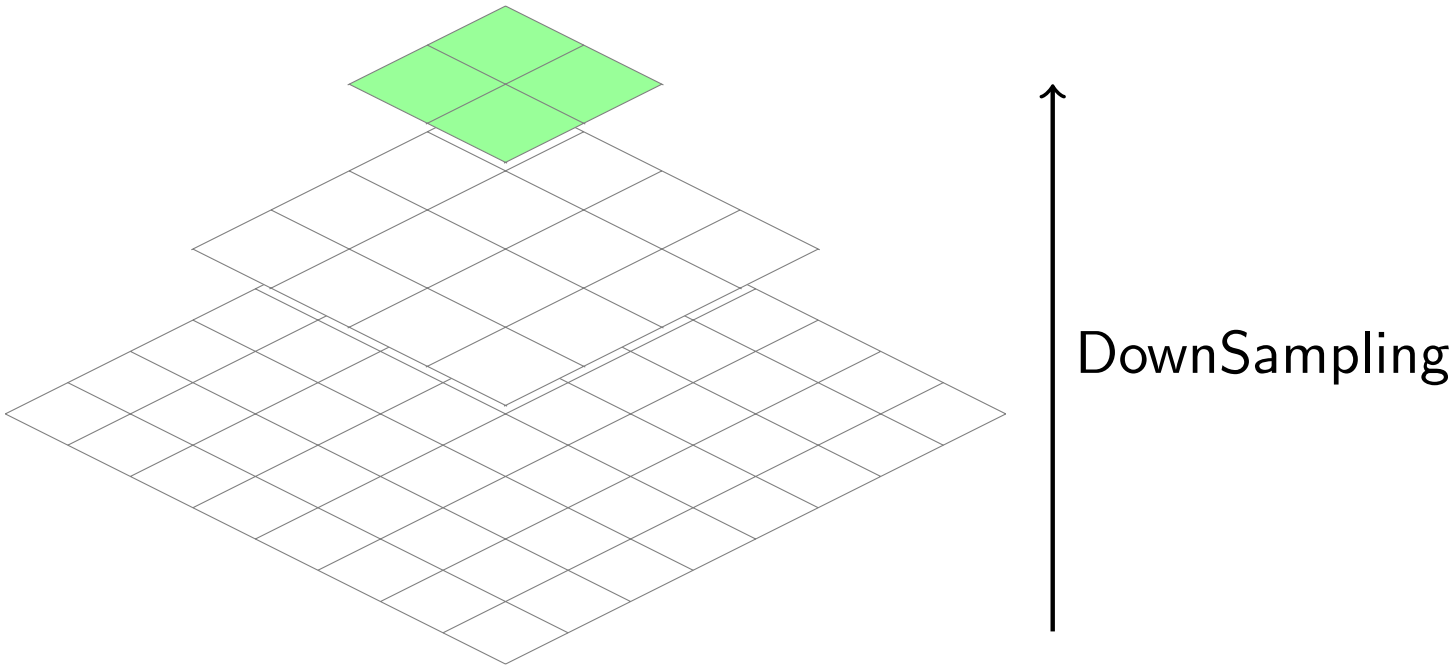


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

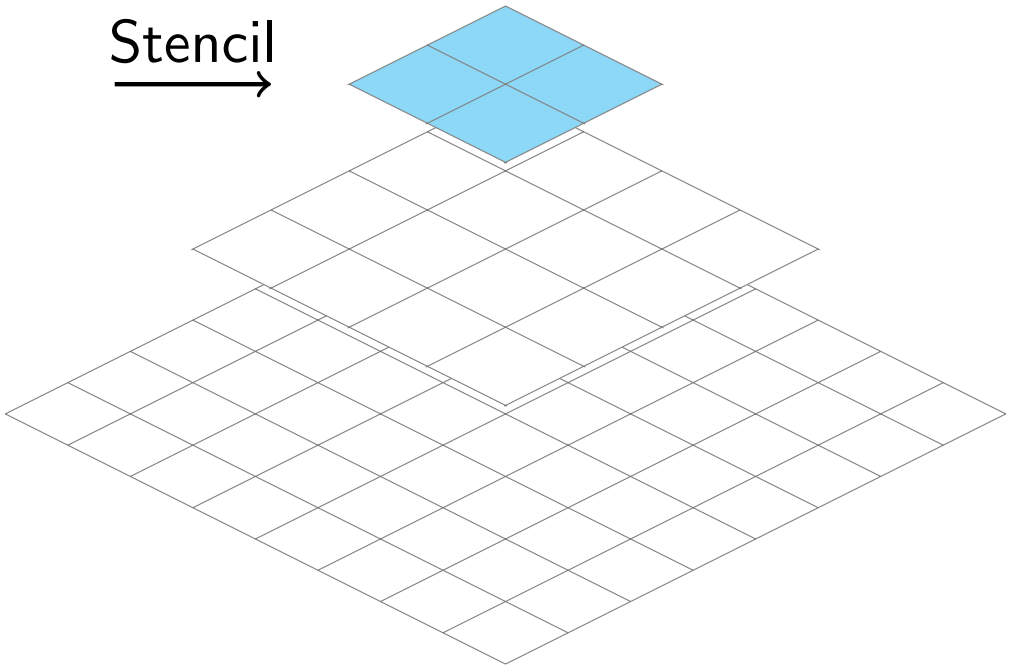


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

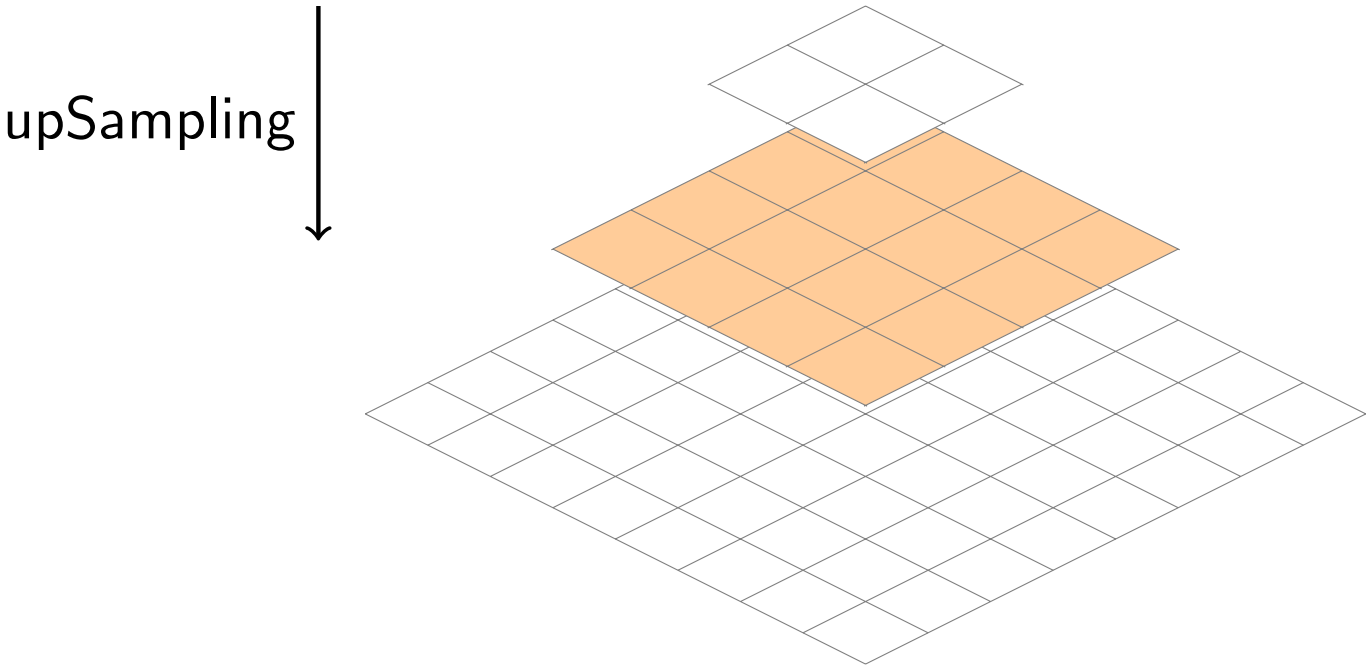


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

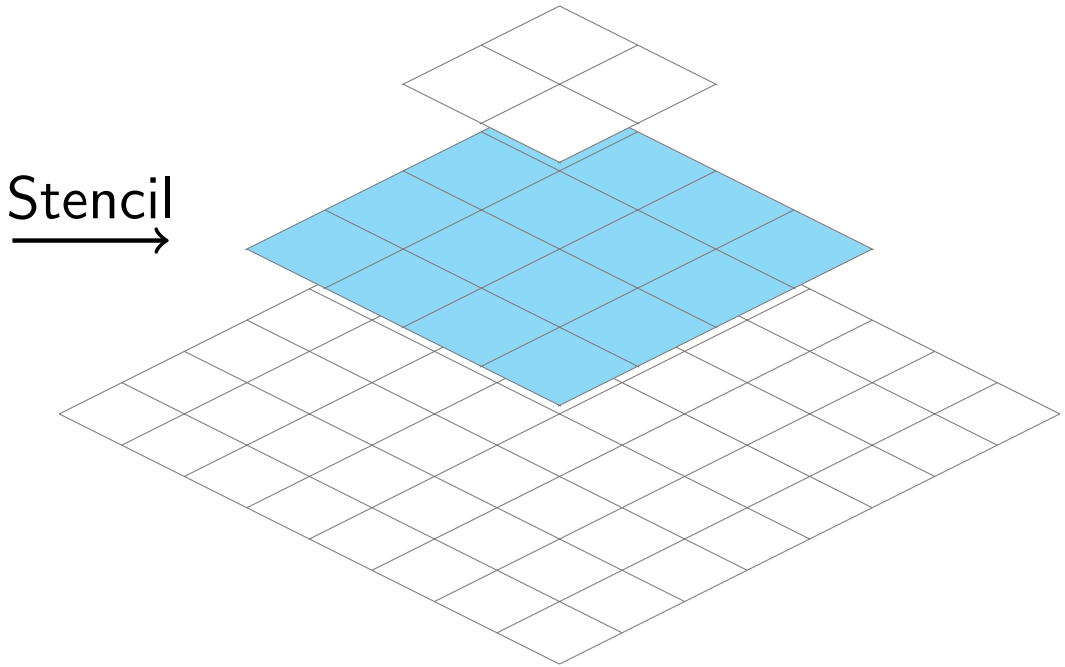


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

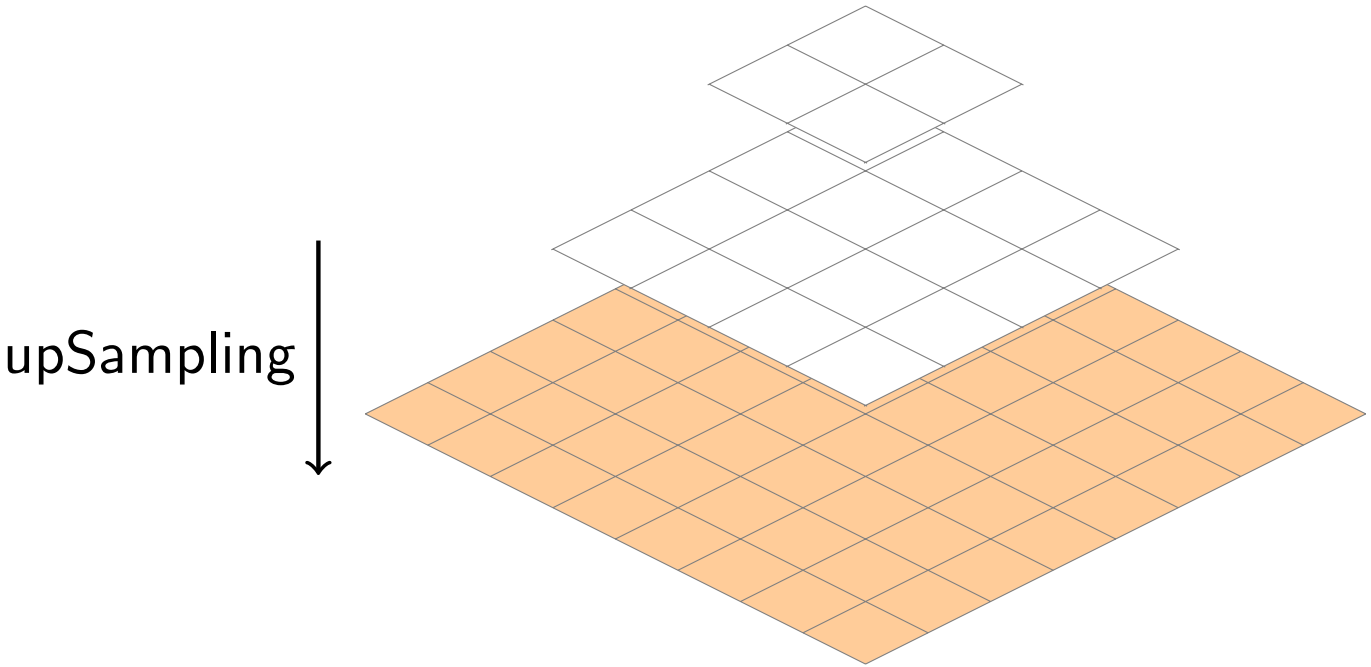


FIGURE – Fonctionnement de l'algorithme

Application de flot optique

PYRAMIDE D'IMAGE représentation multi-résolution d'une image

3 opérations : UpSampling, DownSampling et Stencil

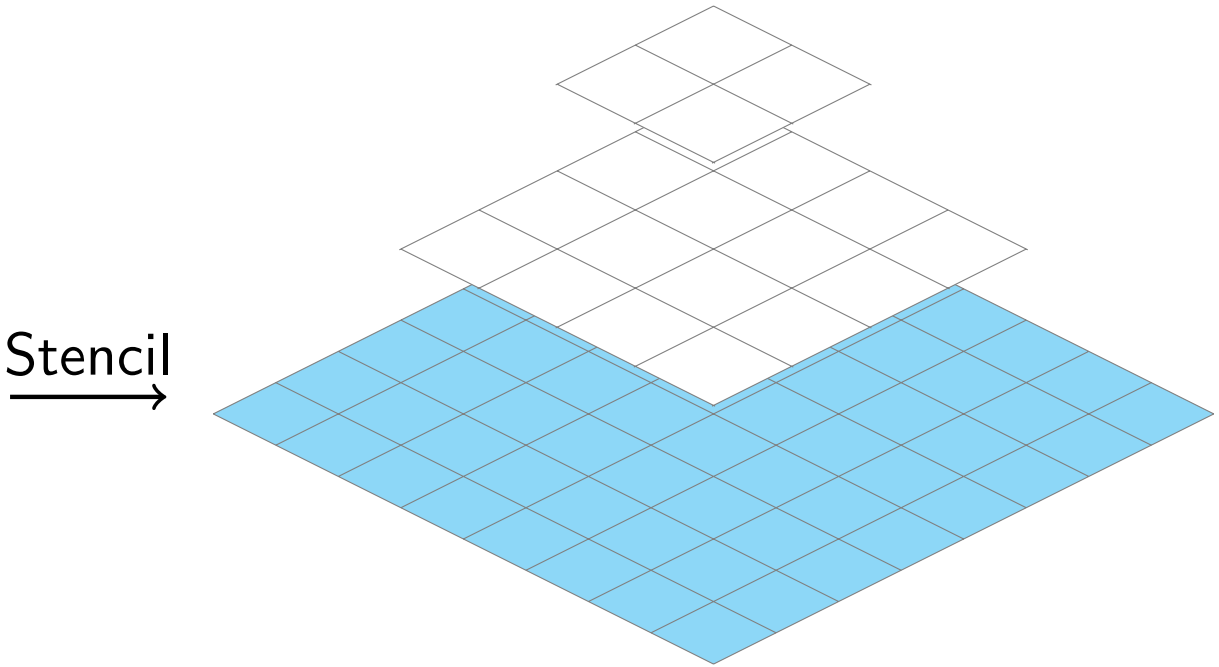


FIGURE – Fonctionnement de l'algorithme

Sommaire

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

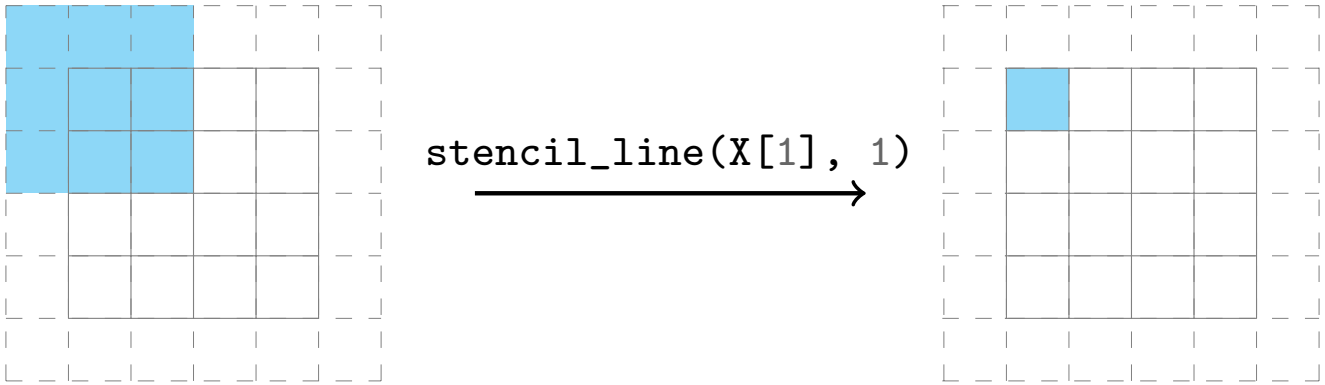


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

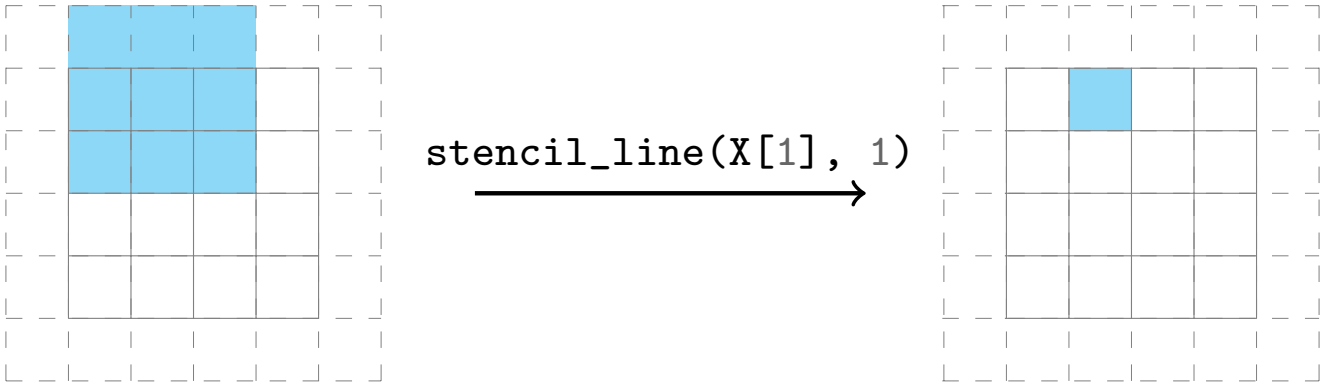


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

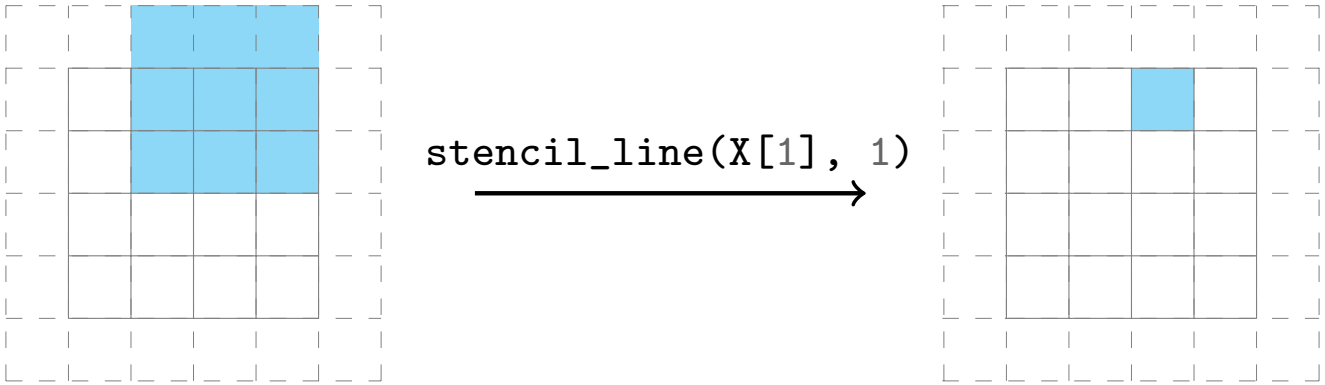


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

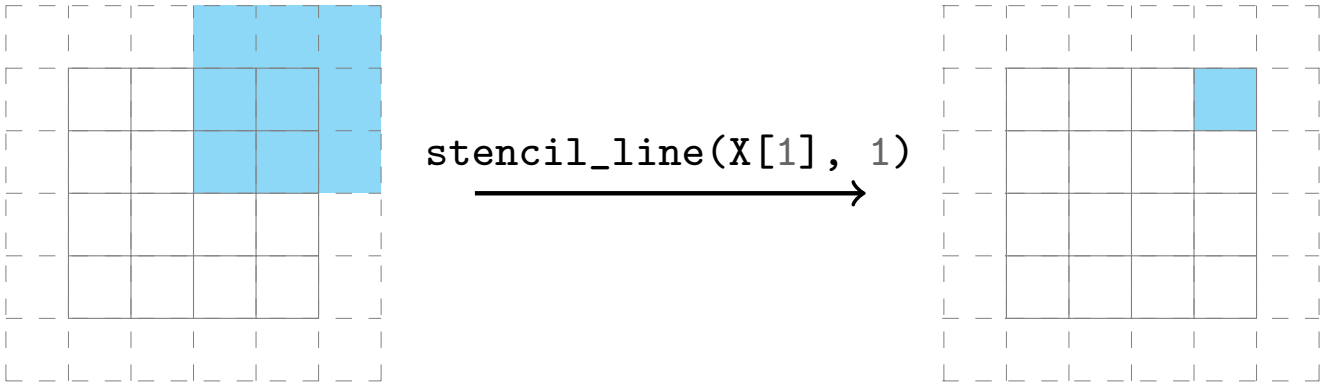


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

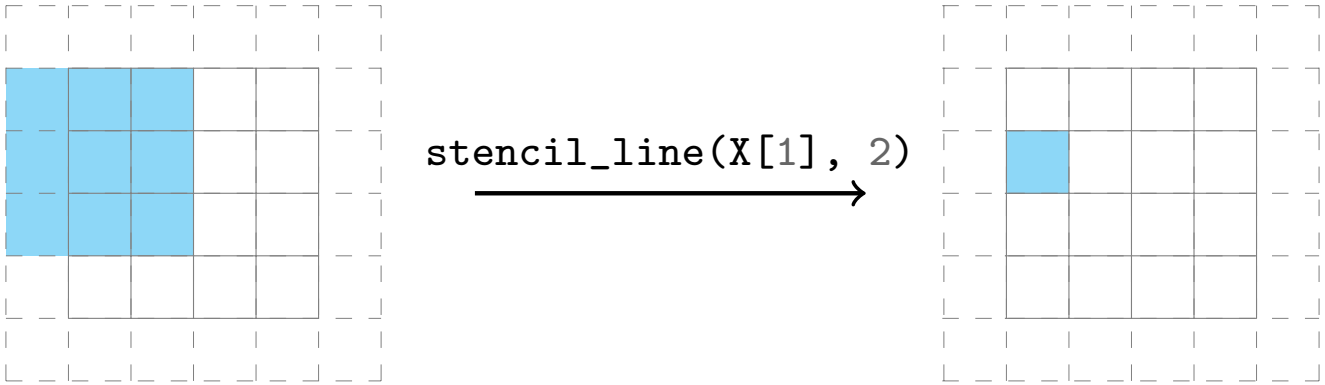


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

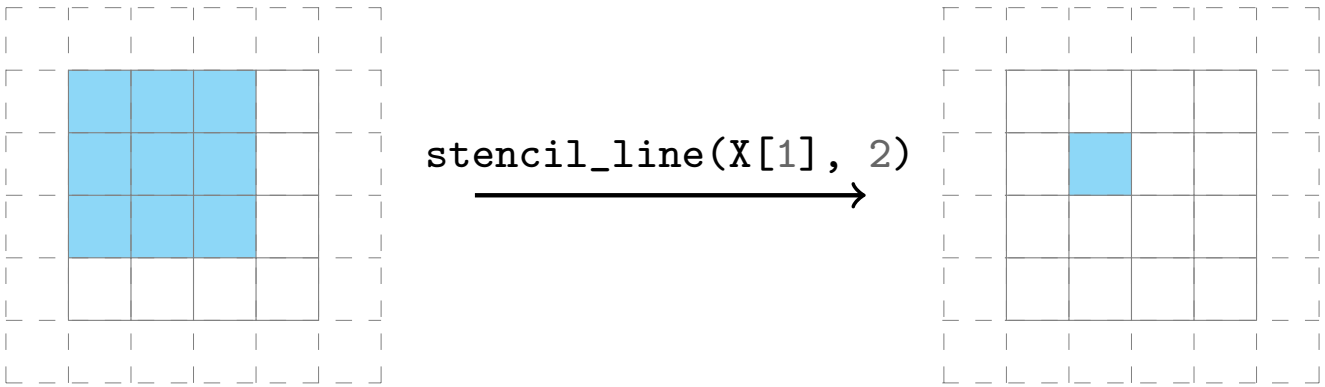


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

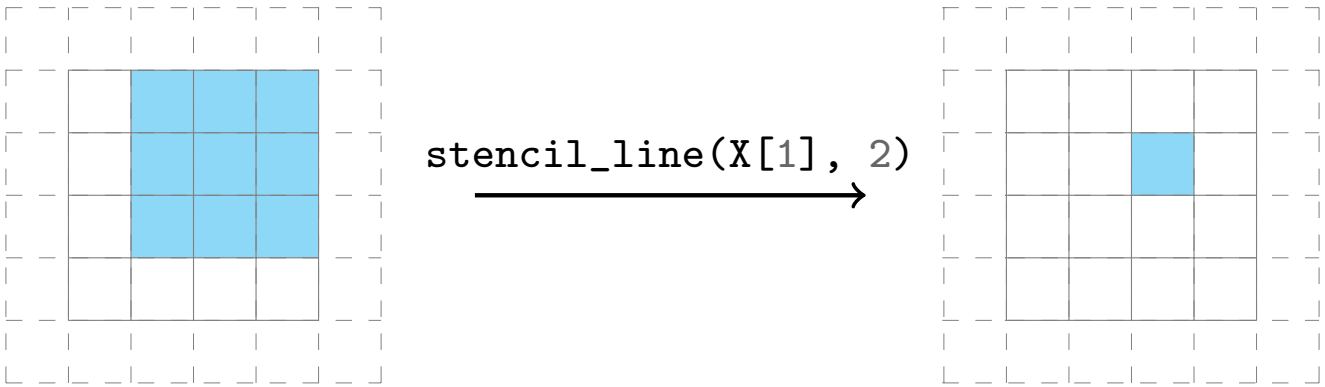


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

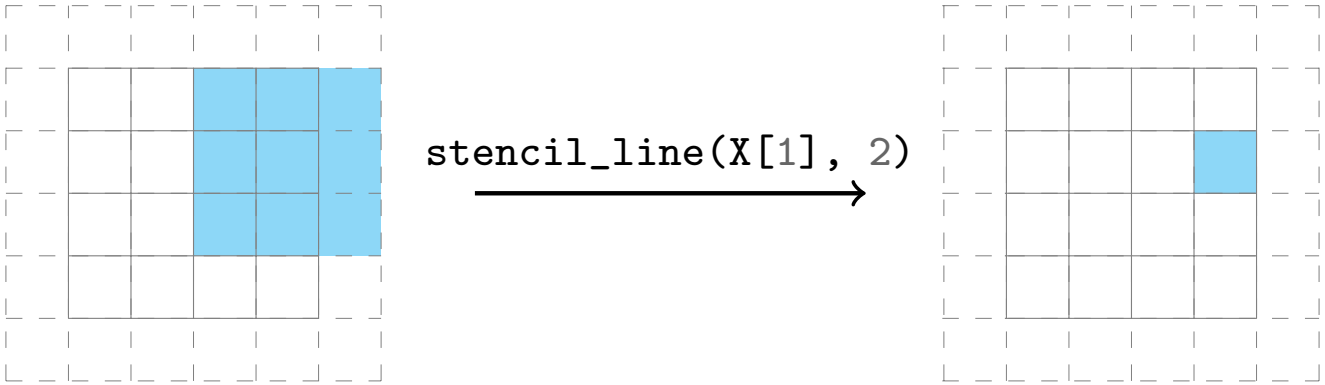


FIGURE – Application du stencil

Opérations à accélérer

```
1 void stencil_line(uint32** X, int i){
2     for (int j = 0; j < w; j++)
3         X[i][j]=    1*X[i-1][j-1] + 2*X[i-1][j] + 1*X[i-1][j+1]\
4                     2*X[i  ][j-1] + 4*X[i  ][j] + 2*X[i  ][j+1]\
5                     1*X[i+1][j-1] + 2*X[i+1][j] + 1*X[i+1][j+1]
6 }
```

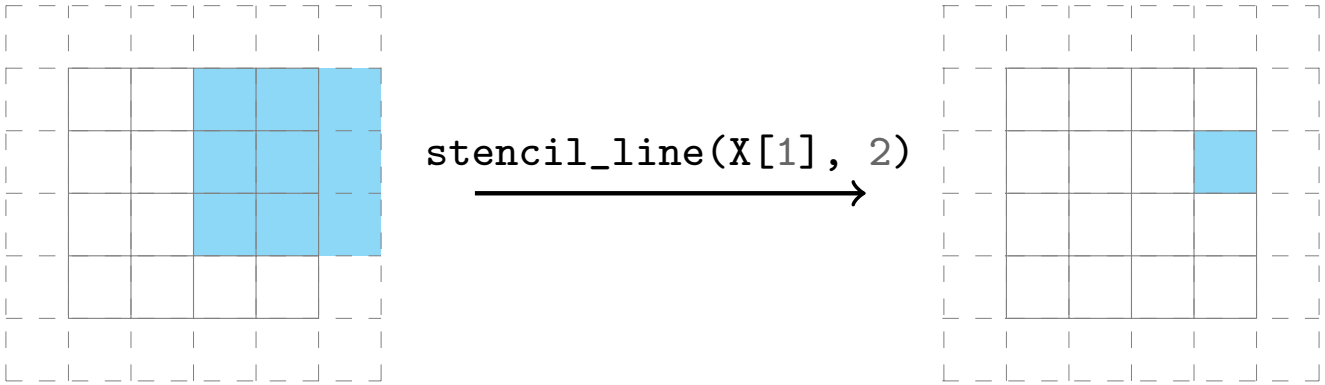


FIGURE – Application du stencil

L'application du stencil prend en moyenne 80% du temps de calcul.

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

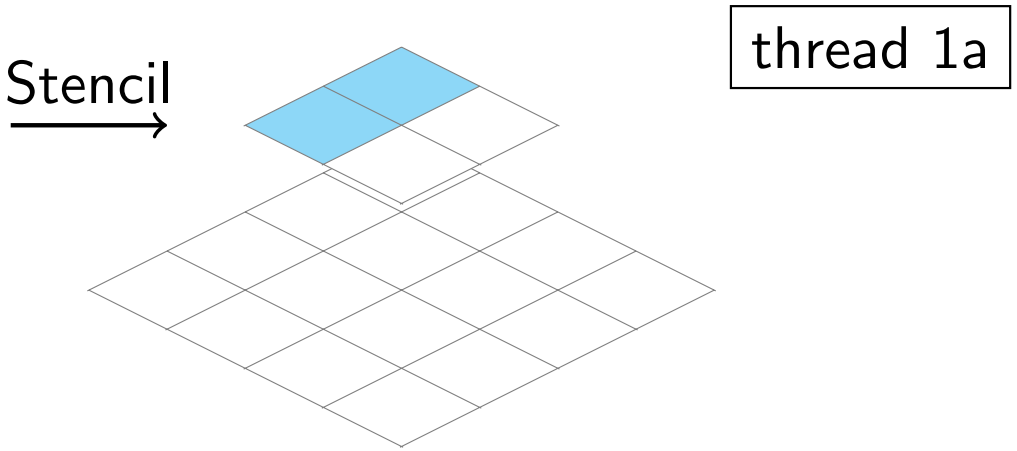


FIGURE – Fonctionnement de la version *classic*

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

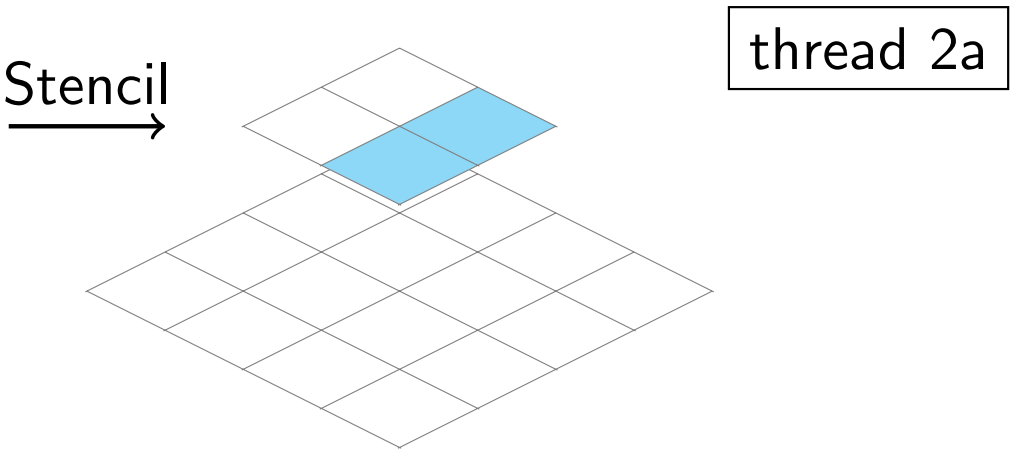


FIGURE – Fonctionnement de la version *classic*

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

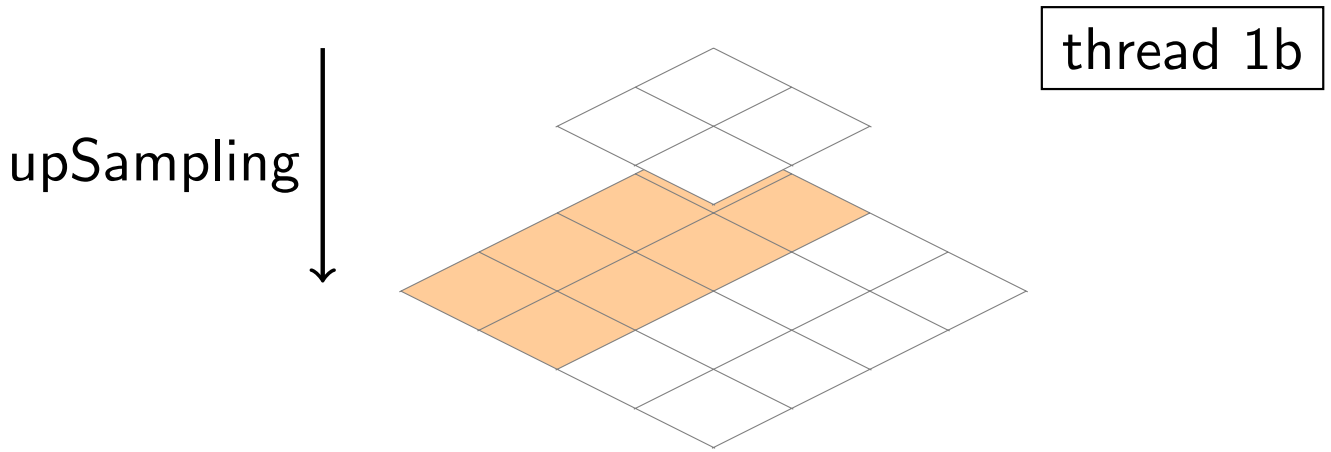


FIGURE – Fonctionnement de la version *classic*

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

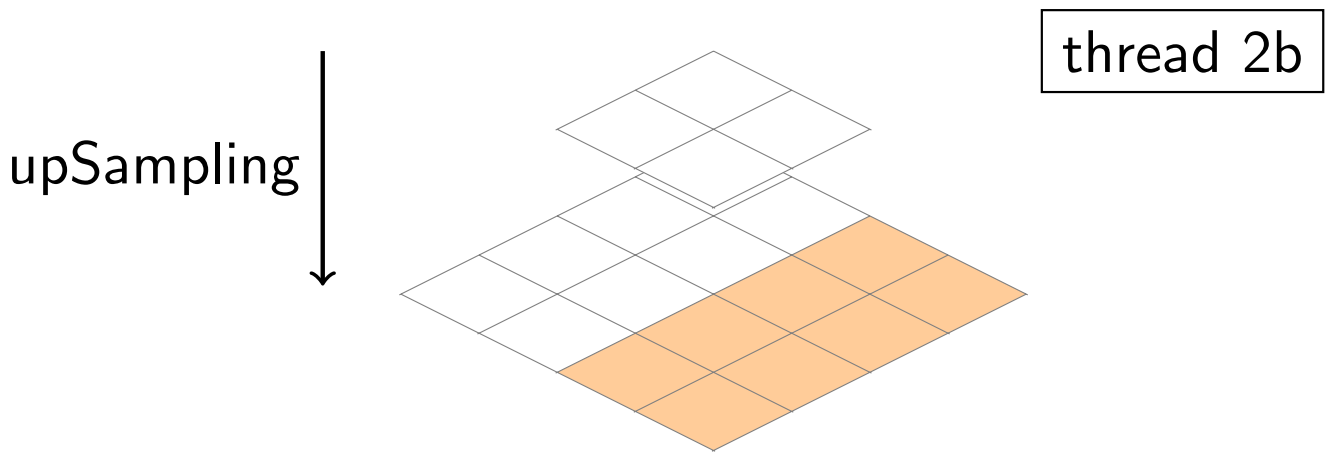


FIGURE – Fonctionnement de la version *classic*

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

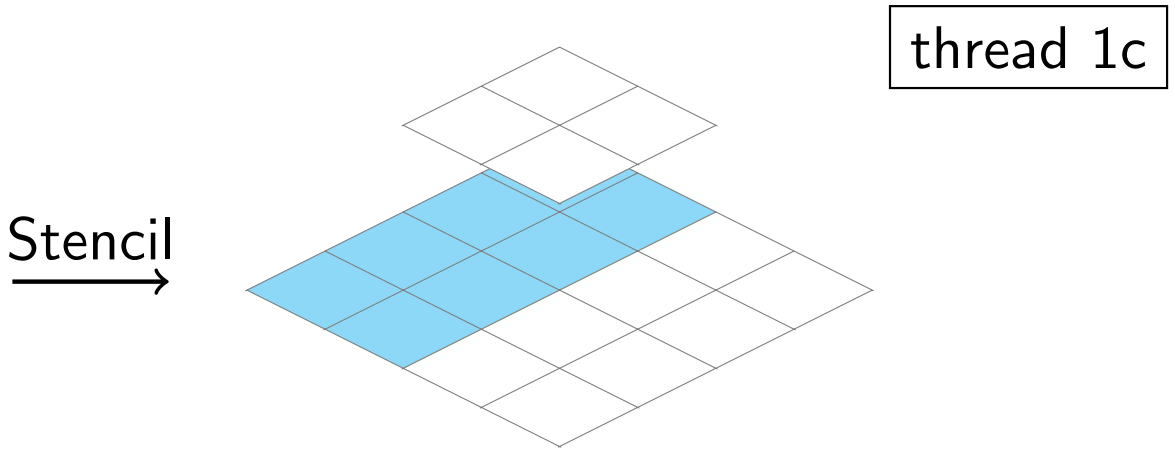


FIGURE – Fonctionnement de la version *classic*

Première version parallélisée de la phase asccendante

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

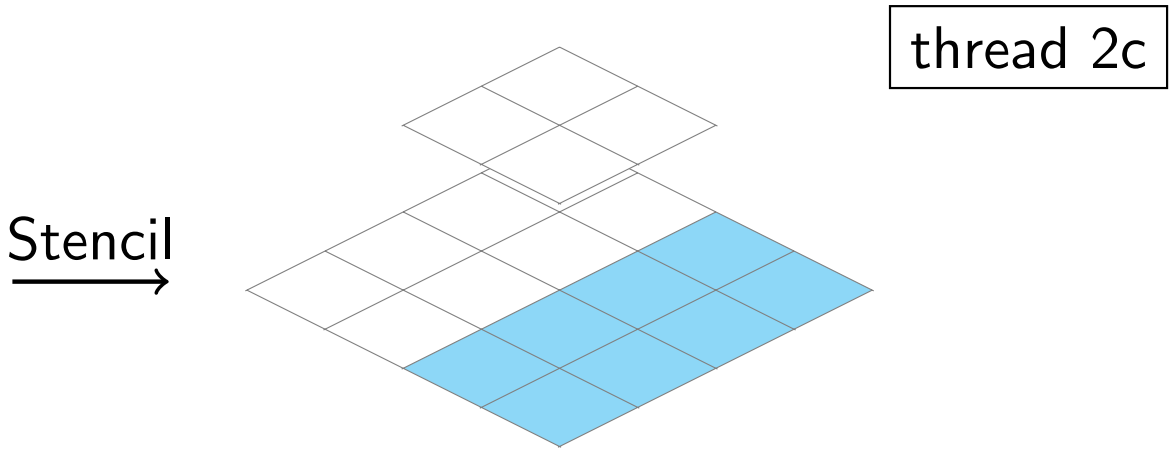


FIGURE – Fonctionnement de la version *classic*

Améliorer la localité des données : changer l'ordre de calcul

```
1  for(int level = nb_level - 1; level >= 0; level--) {
2      for(int iter = 0; iter < nb_iter; iter++){
3          #pragma omp parallel for
4          for(int i = 0; i < h; i++)
5              stencil_line(X[level], i);
6      } if(level) {
7          #pragma omp parallel for
8          for (int i = 0; i < h; i++)
9              upsample_line(X[level], i, X[level-1]);
10     }
11 }
```

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4
5          for(int iter = 0; iter < nb_iter; iter++){
6              for(int i = blocStart(k); i < blocEnd(k); i++)
7                  stencil_line(X[level], i);
8
9          } if(level) {
10             for (int i = blocStart(k); i < blocEnd(k); i++)
11                 upsample_line(X[level], i, X[level-1]);
12         }
13 }}
```

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

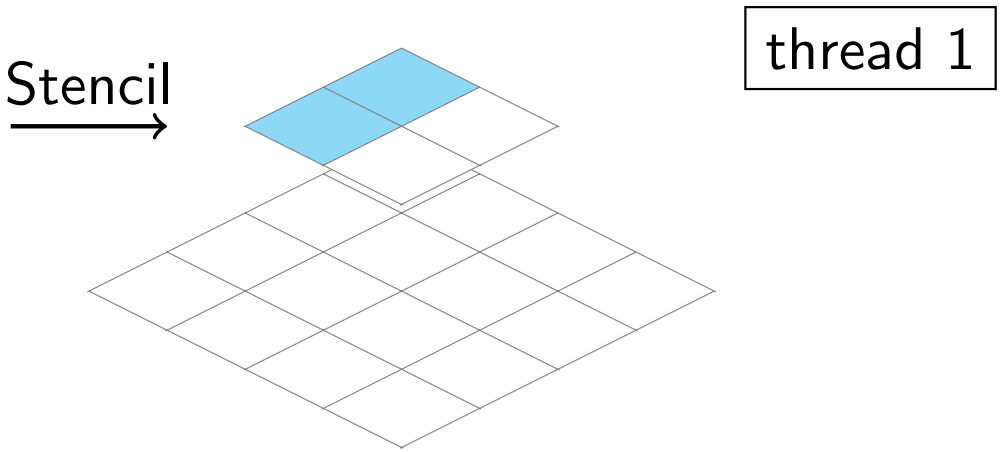


FIGURE – Fonctionnement de la version *ord*

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

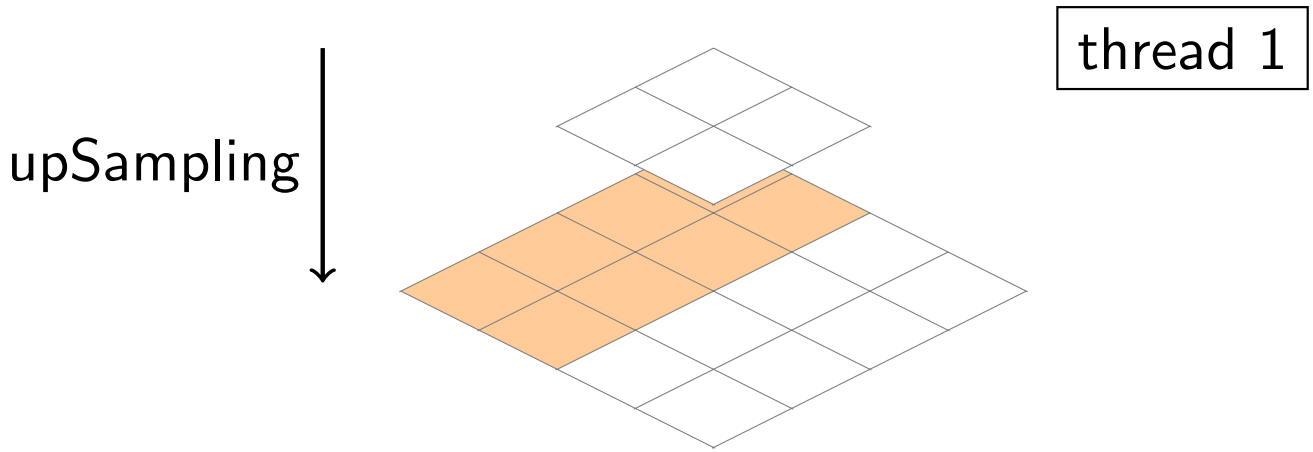


FIGURE – Fonctionnement de la version *ord*

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

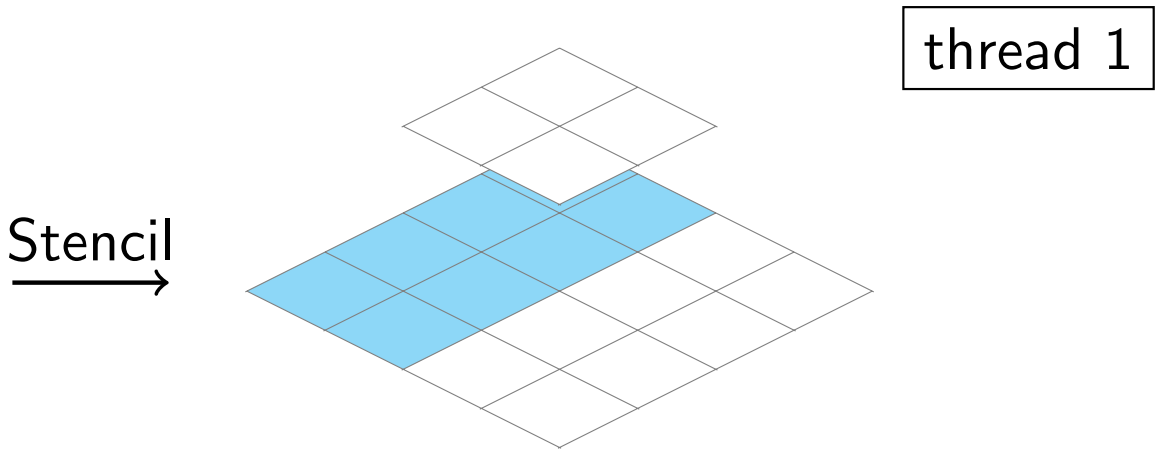


FIGURE – Fonctionnement de la version *ord*

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

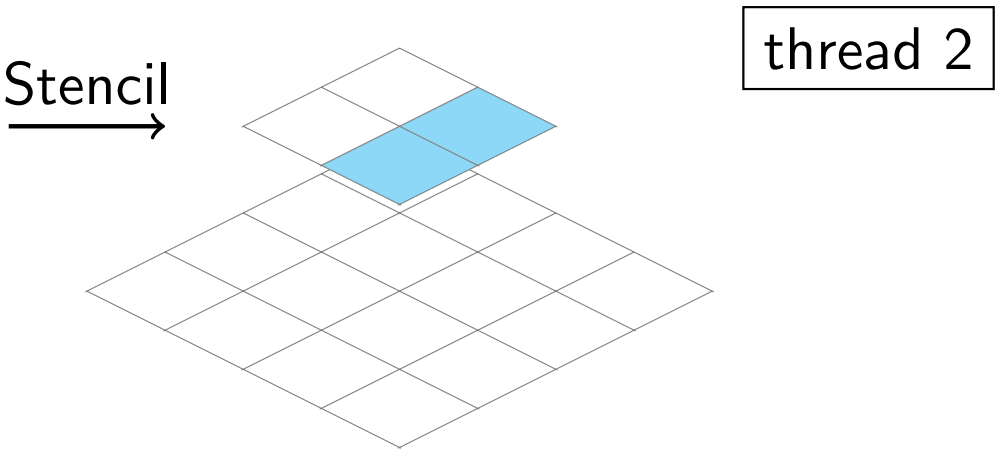


FIGURE – Fonctionnement de la version *ord*

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

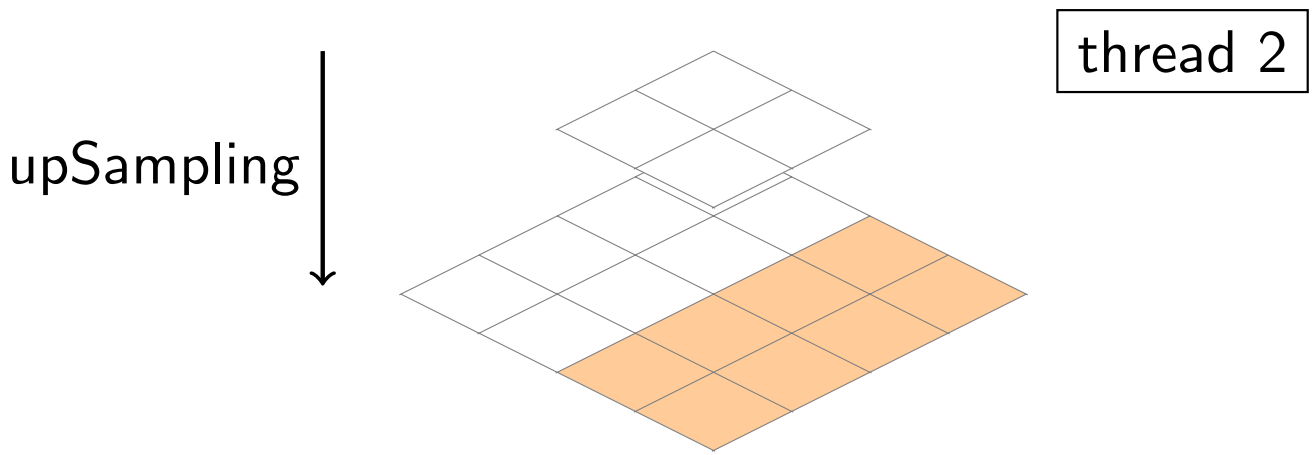


FIGURE – Fonctionnement de la version *ord*

Améliorer la localité des données : changer l'ordre de calcul

```
1  #pragma omp parallel for
2  for(int k = 0; k < nb_blocs){
3      for(int level = nb_level - 1; level >= 0; level--) {
4          for(int iter = 0; iter < nb_iter; iter++){
5              for(int i = blocStart(k); i < blocEnd(k); i++)
6                  stencil_line(X[level], i);
7          } if(level) {
8              for (int i = blocStart(k); i < blocEnd(k); i++)
9                  upsample_line(X[level], i, X[level-1]);
10         }
11 }}
```

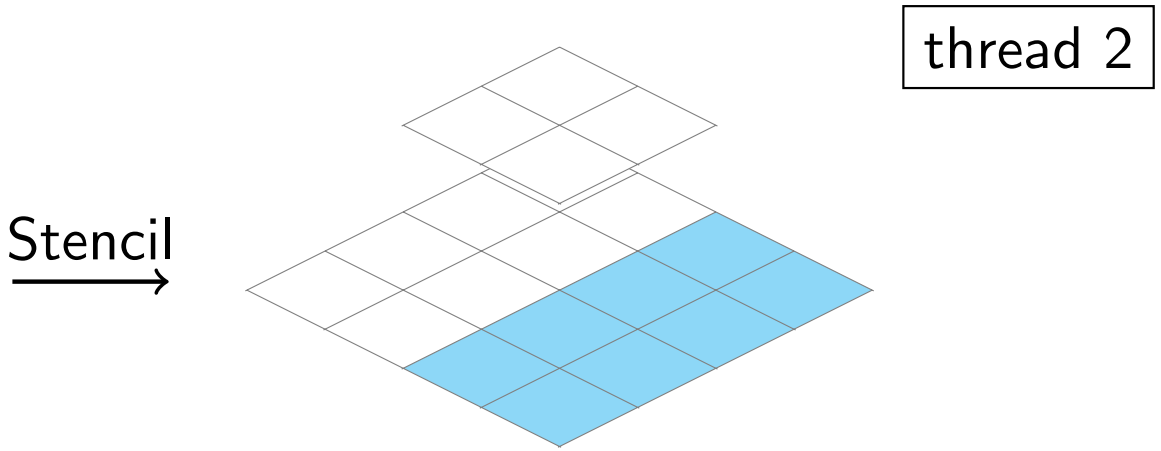


FIGURE – Fonctionnement de la version *ord*

Principe des shadow-zones

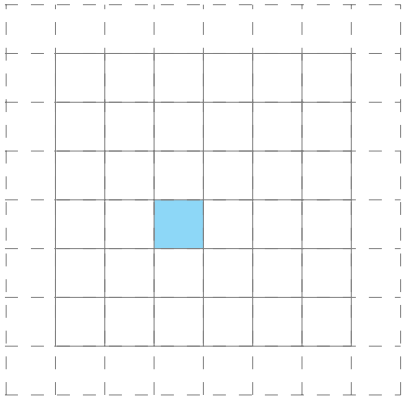


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

Principe des shadow-zones

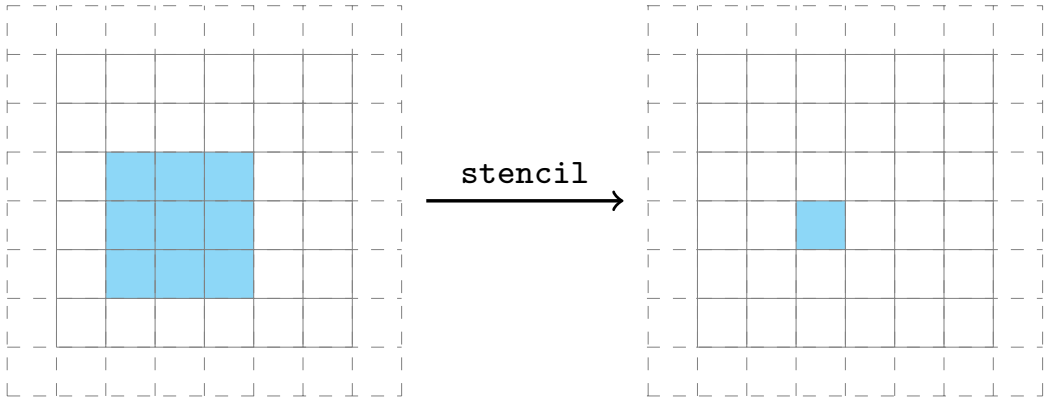


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

Principe des shadow-zones

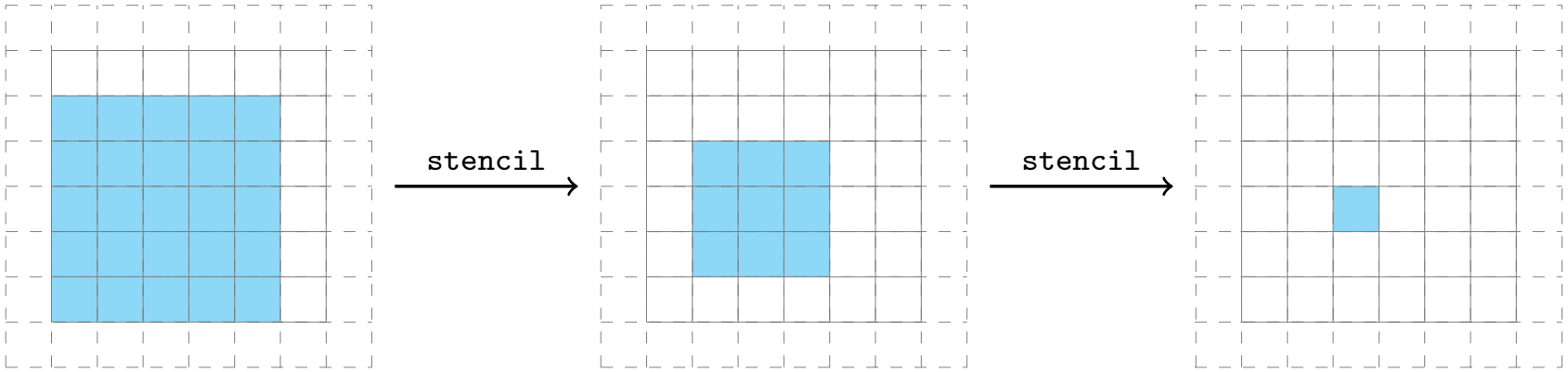


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

Principe des shadow-zones

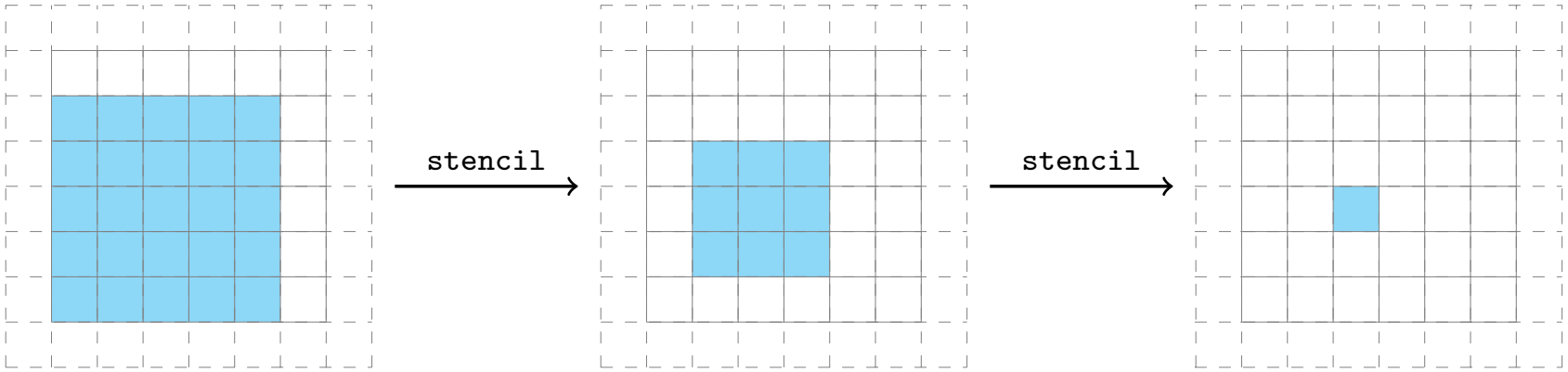


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

- Pour calculer un pixel sur une image il faut 9 pixels "à jour" sur l'image précédente

Principe des shadow-zones

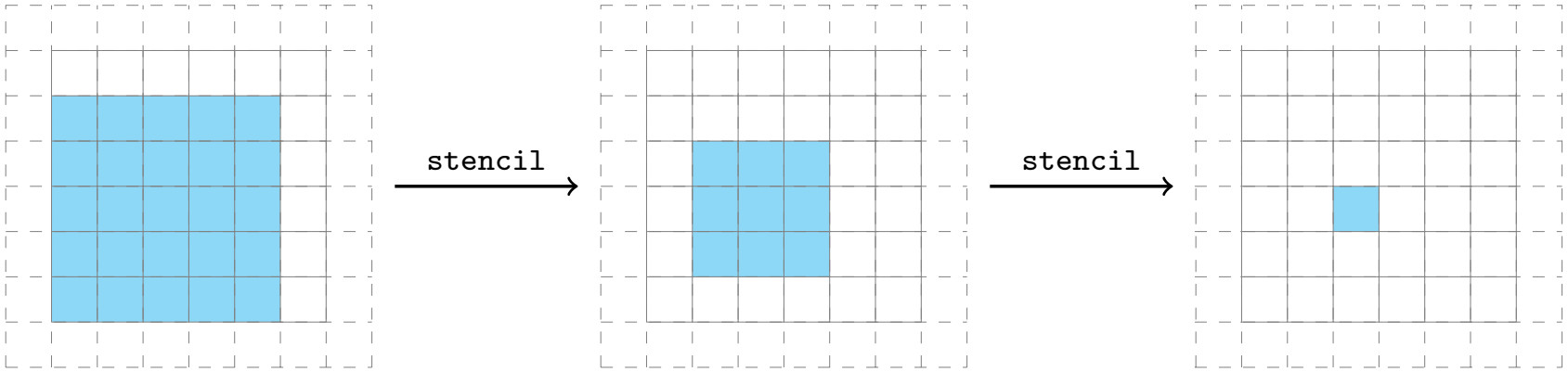


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

- Pour calculer un pixel sur une image il faut 9 pixels "à jour" sur l'image précédente
- Pour calculer le résultat de plusieurs itérations de Stencil sur une partie de l'image, il faut donc des calculs supplémentaires

Principe des shadow-zones

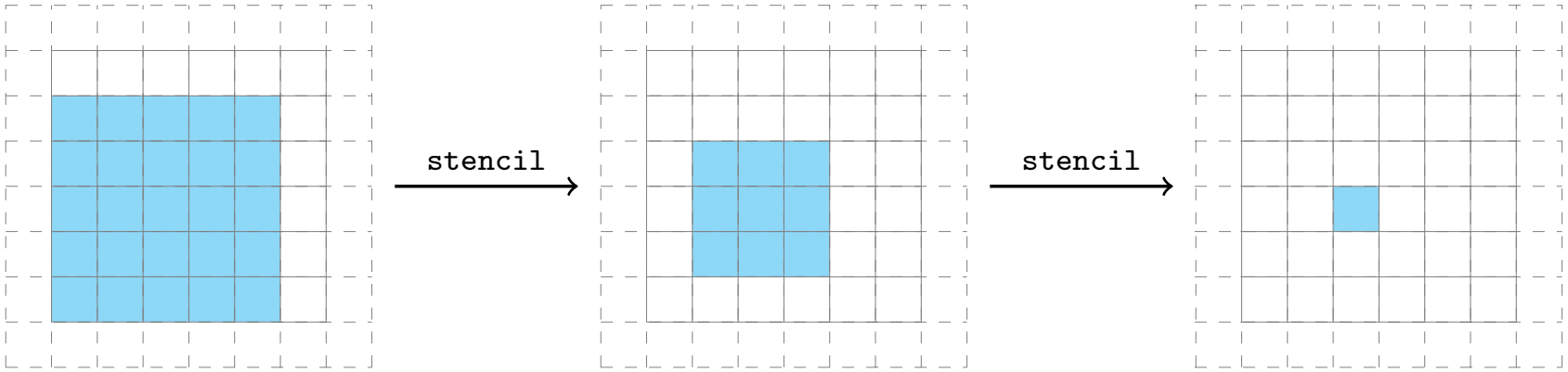


FIGURE – Calcul d'un pixel après 2 itérations de Stencil

- Pour calculer un pixel sur une image il faut 9 pixels "à jour" sur l'image précédente
- Pour calculer le résultat de plusieurs itérations de Stencil sur une partie de l'image, il faut donc des calculs supplémentaires

shadow-zone Ensemble de calculs redondants, nécessaire à la correction de la sortie de l'algorithme

Principe des shadow-zones

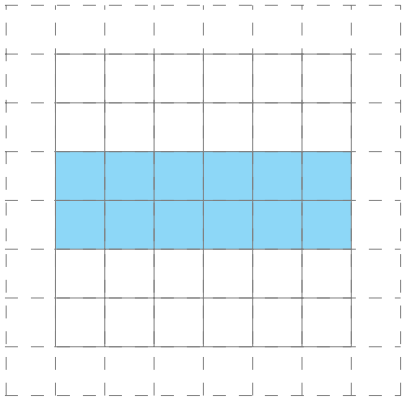


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

Principe des shadow-zones

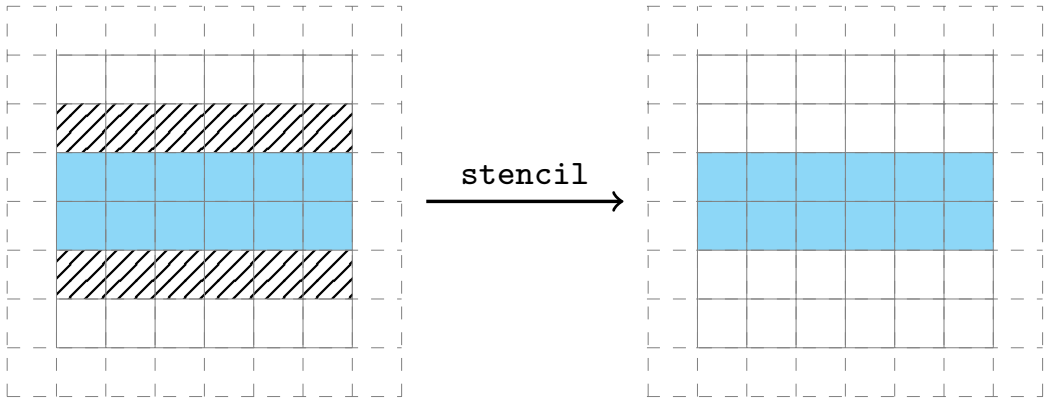


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

Principe des shadow-zones

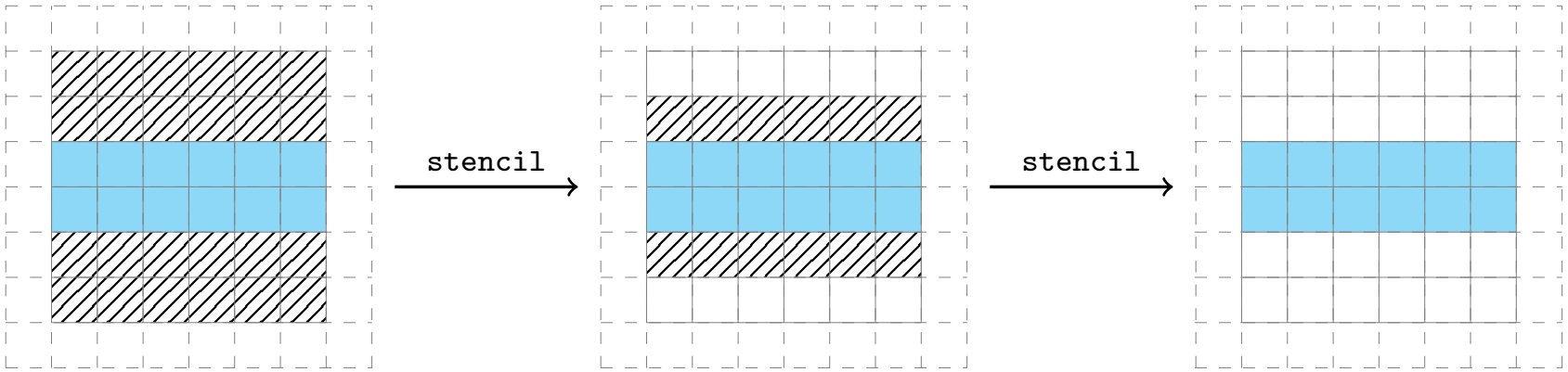


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

Principe des shadow-zones

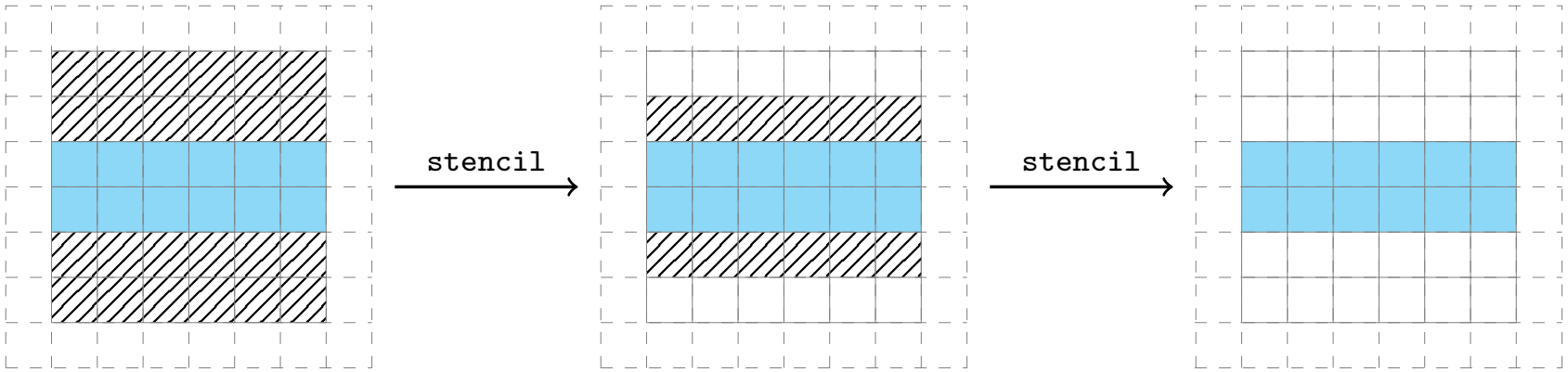


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

- En appliquant d'abord Stencil en profondeur sur les blocs, on introduit une redondance dans le calcul

Principe des shadow-zones

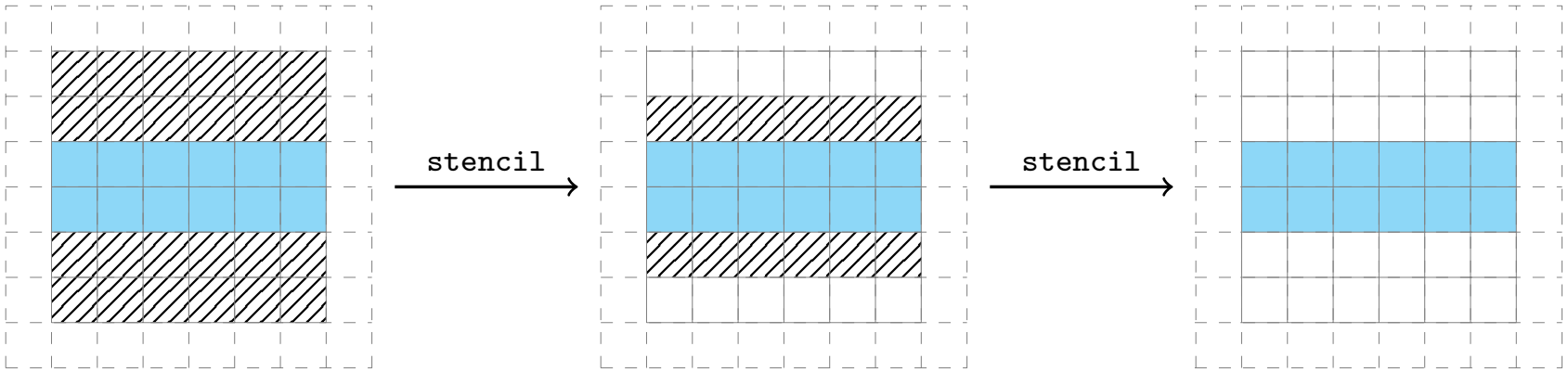


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

- En appliquant d'abord Stencil en profondeur sur les blocs, on introduit une redondance dans le calcul
- Plus le nombre d'opérations sur l'image est élevé, plus le calcul des shadow-zones est coûteux

Principe des shadow-zones

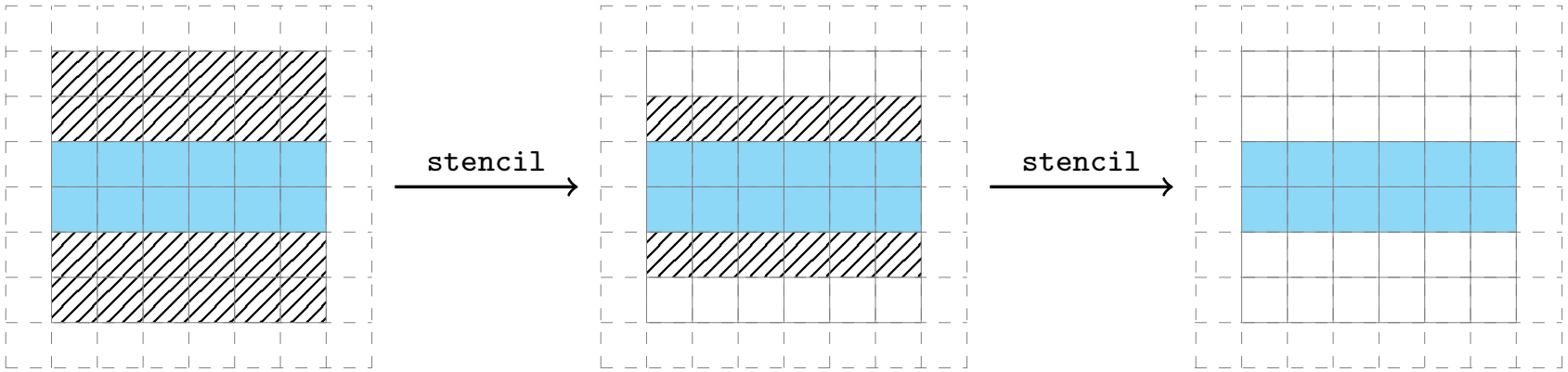
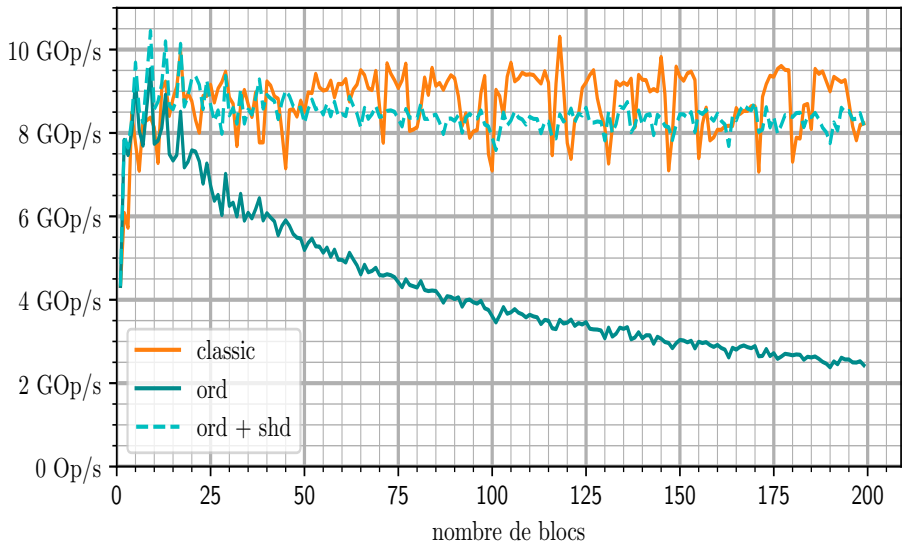


FIGURE – Shadow-zone pour le calcul de 2 itérations de Stencil

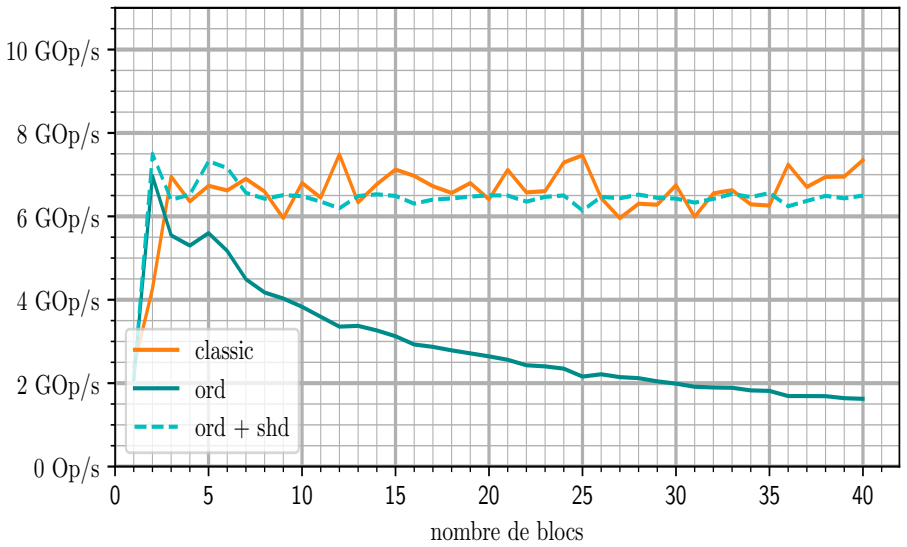
- En appliquant d'abord Stencil en profondeur sur les blocs, on introduit une redondance dans le calcul
- Plus le nombre d'opérations sur l'image est élevé, plus le calcul des shadow-zones est coûteux
- Lorsqu'une grande image est divisée en peu de blocs, le calcul rajouté est négligeable

Comment découper le calcul ?

Image de taille 2048, 4 cœurs utilisés



(a) 6 niveaux, 6 itérations ($\times 6$)

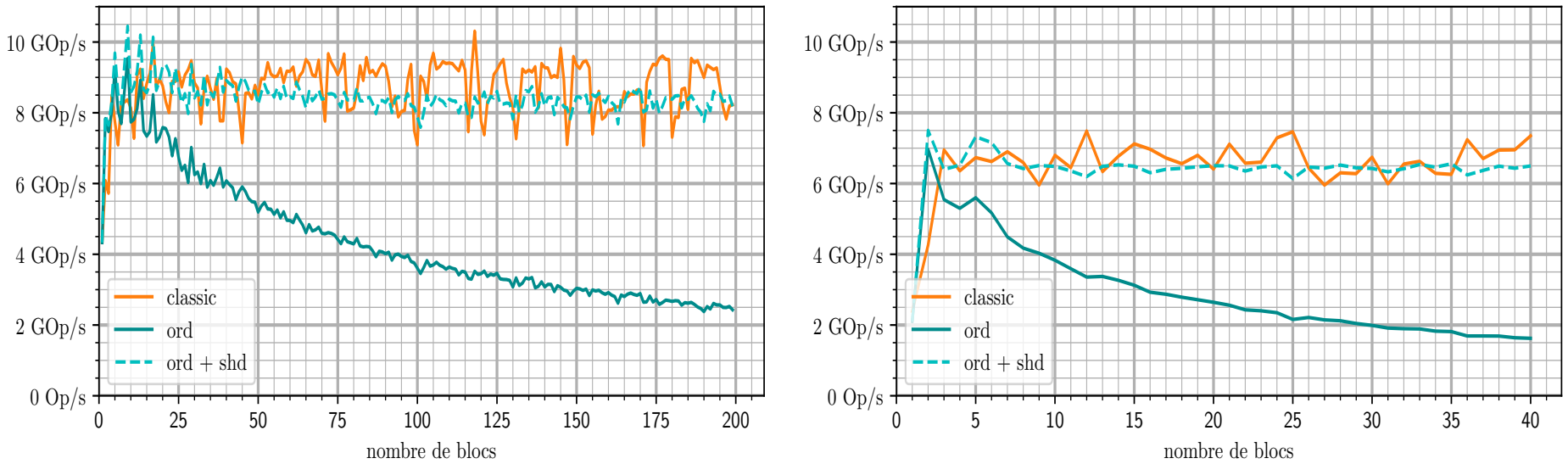


(b) 3 niveaux, 5, 10 puis 20 itérations

FIGURE – Influence du nombre de blocs sur la performance

Comment découper le calcul ?

Image de taille 2048, 4 cœurs utilisés



(a) 6 niveaux, 6 itérations ($\times 6$)

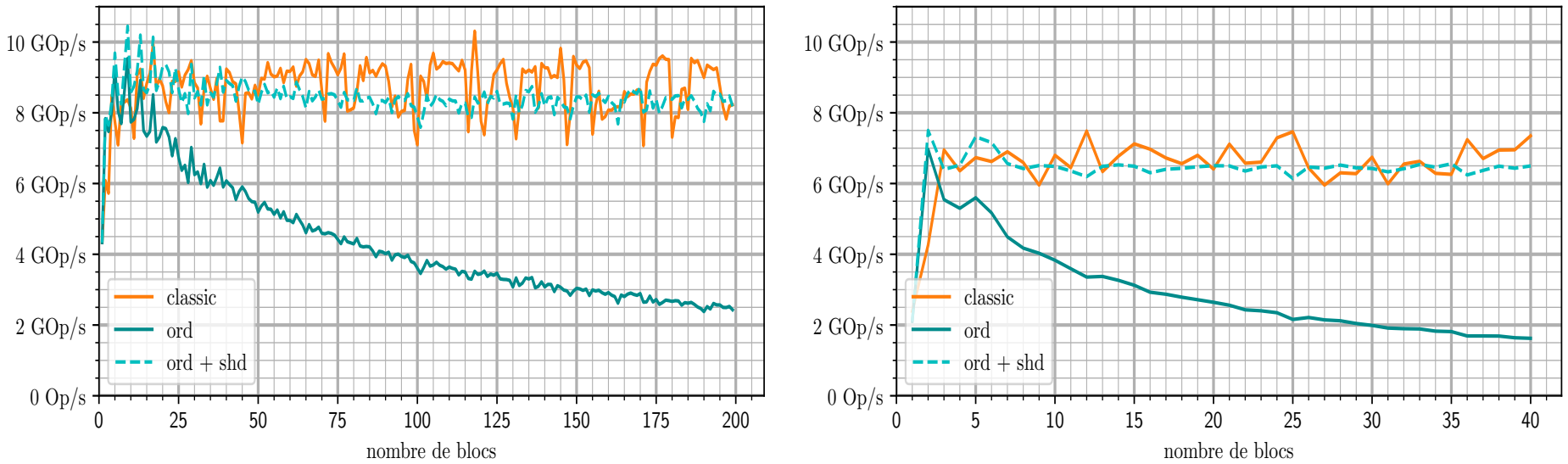
(b) 3 niveaux, 5, 10 puis 20 itérations

FIGURE – Influence du nombre de blocs sur la performance

- La performance de *ord* est équivalente à celle de *classic* si on prend en compte les calculs en plus.

Comment découper le calcul ?

Image de taille 2048, 4 cœurs utilisés



(a) 6 niveaux, 6 itérations ($\times 6$)

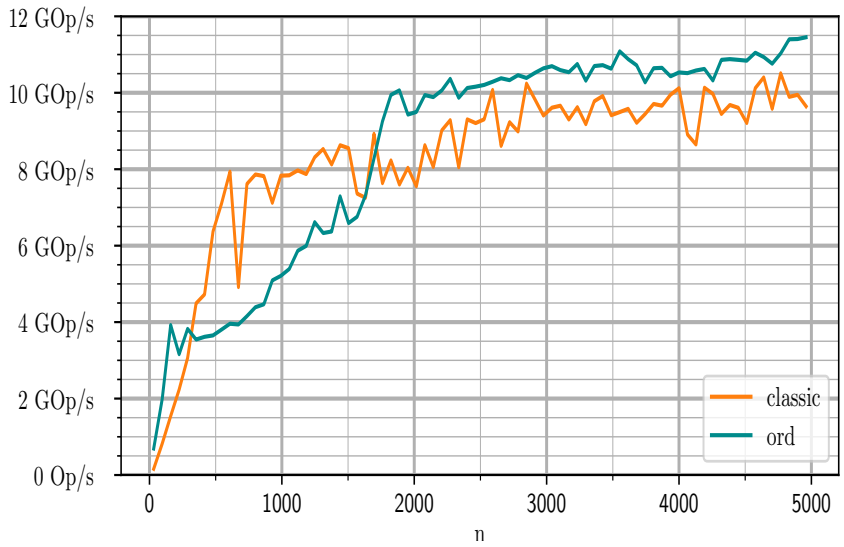
(b) 3 niveaux, 5, 10 puis 20 itérations

FIGURE – Influence du nombre de blocs sur la performance

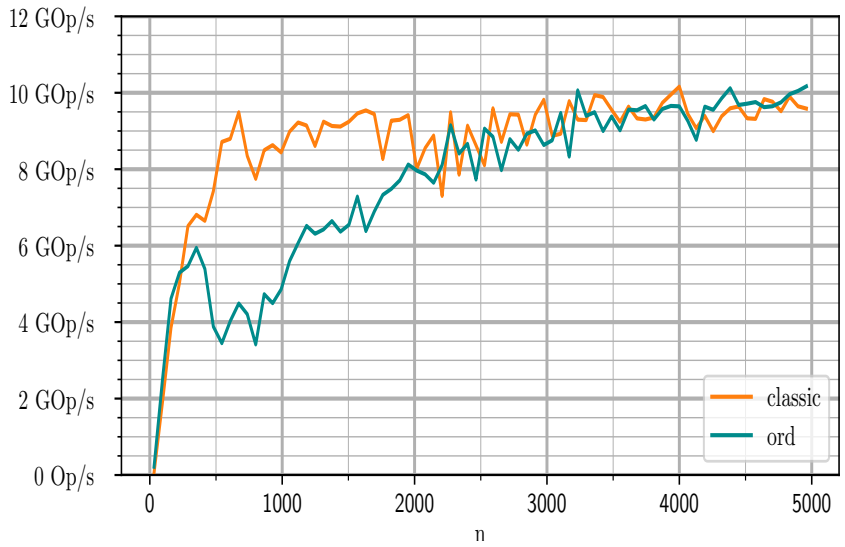
- La performance de *ord* est équivalente à celle de *classic* si on prend en compte les calculs en plus.
- En terme de GOp/s "utiles", *ord* devient bien moins bon dès que l'on a trop de blocs.

Performance des algorithmes

Performance des versions *ord* et *classic* sur des cas usuels, en utilisant 6 blocs pour *ord* et 4 cœurs de la carte.



(a) 6 niveaux, 6 itérations ($\times 6$)

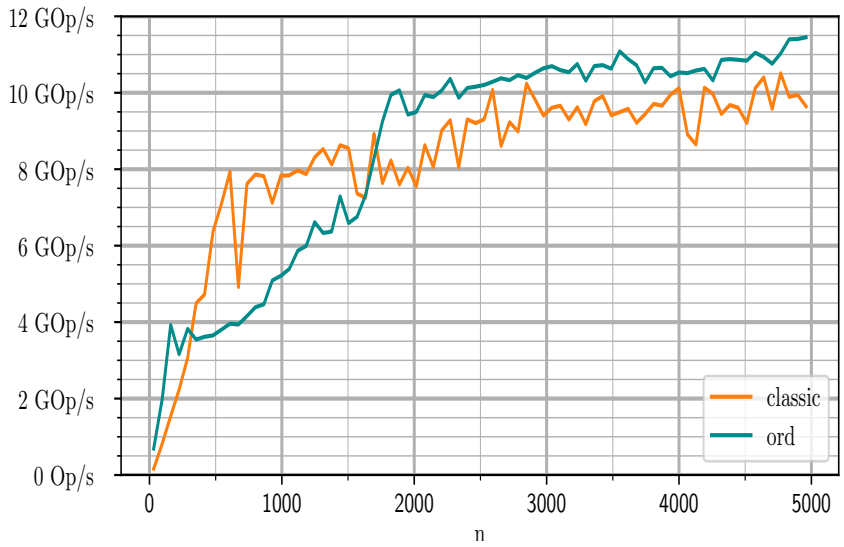


(b) 3 niveaux, 5, 10 puis 20 itérations

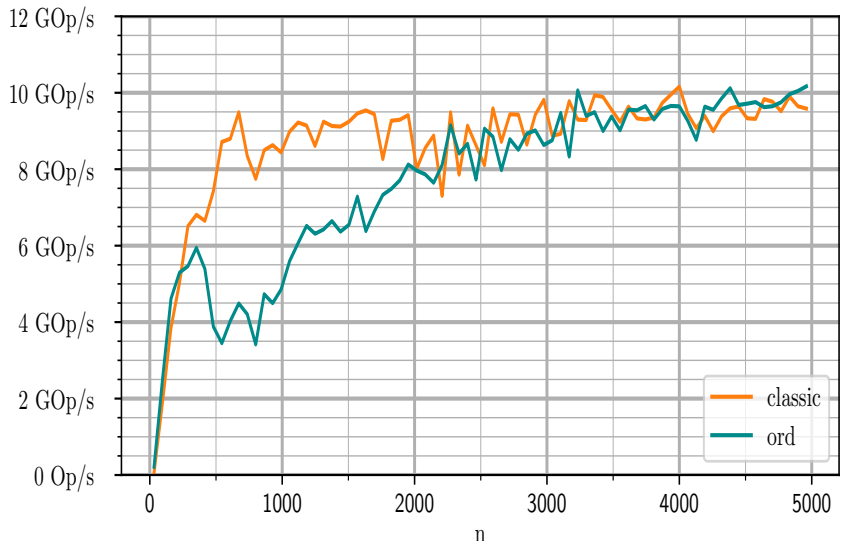
FIGURE – Performance selon la taille de l'image traitée

Performance des algorithmes

Performance des versions *ord* et *classic* sur des cas usuels, en utilisant 6 blocs pour *ord* et 4 cœurs de la carte.



(a) 6 niveaux, 6 itérations ($\times 6$)



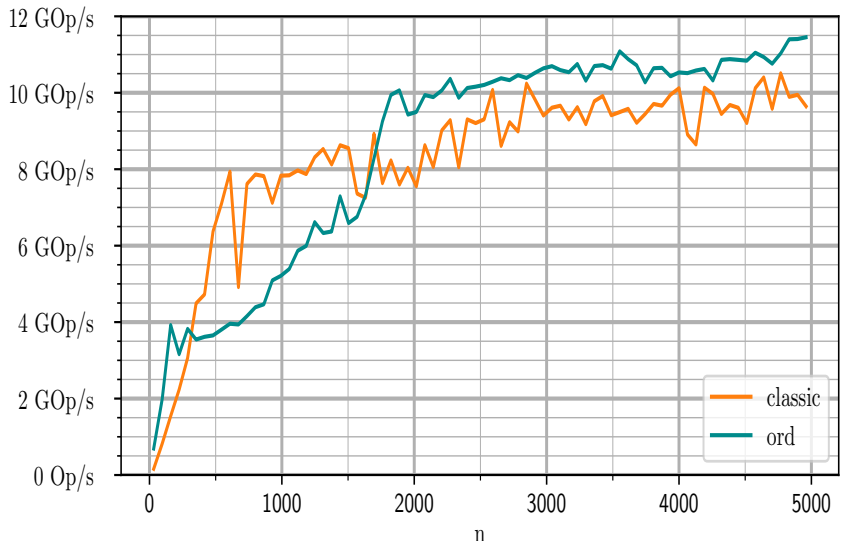
(b) 3 niveaux, 5, 10 puis 20 itérations

FIGURE – Performance selon la taille de l'image traitée

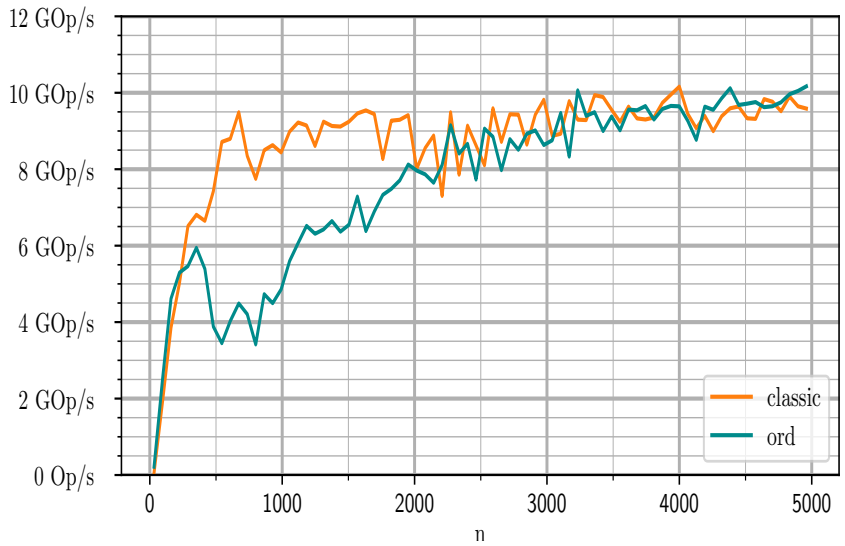
- Sur de grandes images, *ord* est souvent meilleur

Performance des algorithmes

Performance des versions *ord* et *classic* sur des cas usuels, en utilisant 6 blocs pour *ord* et 4 cœurs de la carte.



(a) 6 niveaux, 6 itérations ($\times 6$)



(b) 3 niveaux, 5, 10 puis 20 itérations

FIGURE – Performance selon la taille de l'image traitée

- Sur de grandes images, *ord* est souvent meilleur
- Sur de petites images, le calcul rajouté par *ord* est trop coûteux

Performance des algorithmes

Pour quelles configurations la version *ord* est-elle optimale ?

Performance des algorithmes

Pour quelles configurations la version *ord* est-elle optimale ?

- Grandes images

Performance des algorithmes

Pour quelles configurations la version *ord* est-elle optimale ?

- Grandes images
- Peu d'itérations de Stencil

Performance des algorithmes

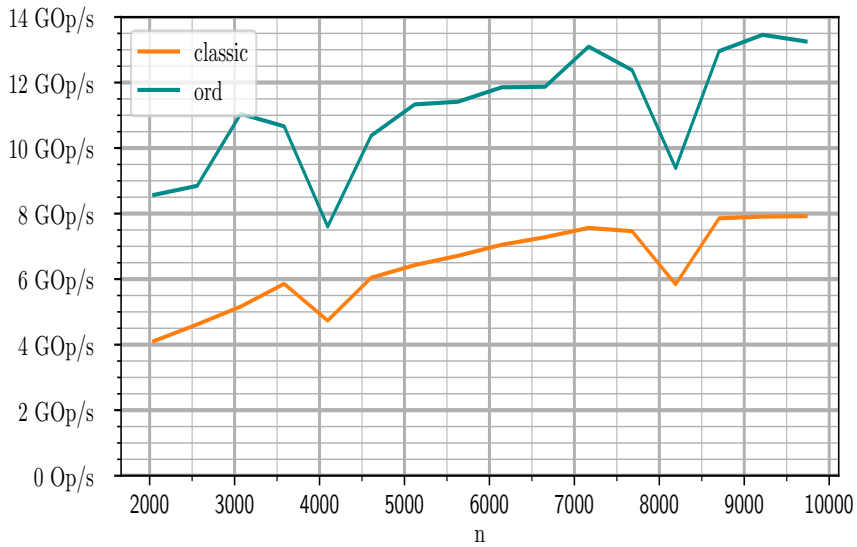
Pour quelles configurations la version *ord* est-elle optimale ?

- Grandes images
- Peu d'itérations de Stencil
- Beaucoup de niveaux

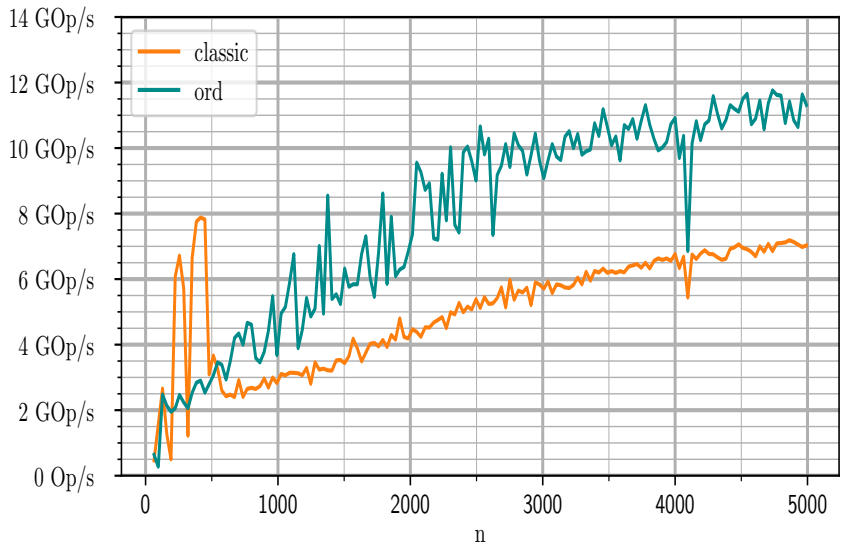
Performance des algorithmes

Pour quelles configurations la version *ord* est-elle optimale ?

- Grandes images
- Peu d'itérations de Stencil
- Beaucoup de niveaux



(a) 6 niveaux, 6 itérations par niveau



(b) 3 niveaux, 1 itération par niveau

FIGURE – Performance selon la taille de l'image traitée

Sommaire

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Optimisation de l'algorithme :

- *ord* parfois plus rapide que *classic* malgré plus de calculs

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Optimisation de l'algorithme :

- *ord* parfois plus rapide que *classic* malgré plus de calculs
- Cela montre l'importance de garantir une bonne localité de données

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Optimisation de l'algorithme :

- *ord* parfois plus rapide que *classic* malgré plus de calculs
- Cela montre l'importance de garantir une bonne localité de données
- Pour l'opérateur DownSampling, la version *ord* est bien plus performante car il n'y a pas besoin de shadow-zones

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Optimisation de l'algorithme :

- *ord* parfois plus rapide que *classic* malgré plus de calculs
- Cela montre l'importance de garantir une bonne localité de données
- Pour l'opérateur DownSampling, la version *ord* est bien plus performante car il n'y a pas besoin de shadow-zones

Pistes d'évolution :

- Prendre en compte l'aspect énergétique, enjeu clé des applications pour les systèmes embarqués

Conclusion

Objectifs de mon travail :

- *ord* et *classic* sont deux stratégies de déploiement de l'algorithme qui doit être embarqué sur Meteorix
- Selon les configurations du logiciel et le matériel embarqué, les concepteurs du logiciel choisiront *ord*, *classic* ou bien une toute autre approche

Optimisation de l'algorithme :

- *ord* parfois plus rapide que *classic* malgré plus de calculs
- Cela montre l'importance de garantir une bonne localité de données
- Pour l'opérateur DownSampling, la version *ord* est bien plus performante car il n'y a pas besoin de shadow-zones

Pistes d'évolution :

- Prendre en compte l'aspect énergétique, enjeu clé des applications pour les systèmes embarqués
- Tester une autre approche de parallélisme : un *pipeline temporel*