

PAP

Projet : rapport2

4TIN804U

BERASATEGUY Tanguy, GOEDEFROIT Charles

2021-2022

Table des matières

1	ILP optimization (4.1)	2
2	OpenMP implementation of the synchronous version (4.2)	5
3	OpenMP implementation of the asynchronous version (4.3)	5
4	Lazy OpenMP implementations (4.4)	9

1 ILP optimization (4.1)

On a fait les modifications :

Pour `ssandPile_do_tile_opt()` on a retiré les appels à `table(out, i, j)` pour passer par une variable intermédiaire `result`. Cette modification permet au compilateur de vectoriser car il peut maintenant facilement voir que les différentes lignes peuvent être calculés en parallèles.

Nous avons modifié ces lignes :

```
1  int ssandPile_do_tile_opt(int x, int y, int width, int height)
2  {
3      int diff = 0;
4
5      for (int i = y; i < y + height; i++)
6          for (int j = x; j < x + width; j++)
7              {
8                  - table(out, i, j) = table(in, i, j) % 4;
9                  + int result = table(in, i, j) % 4;
10                 - table(out, i, j) += table(in, i + 1, j) / 4;
11                 + result += table(in, i + 1, j) / 4;
12                 - table(out, i, j) += table(in, i - 1, j) / 4;
13                 + result += table(in, i - 1, j) / 4;
14                 - table(out, i, j) += table(in, i, j + 1) / 4;
15                 + result += table(in, i, j + 1) / 4;
16                 - table(out, i, j) += table(in, i, j - 1) / 4;
17                 + result += table(in, i, j - 1) / 4;
18                 + table(out, i, j) = result;
19                 - if (table(out, i, j) >= 4)
20                 -     diff = 1;
21                 + diff |= result >= 4;
22             }
23
24     return diff;
25 }
```

Le code de la fonction final :

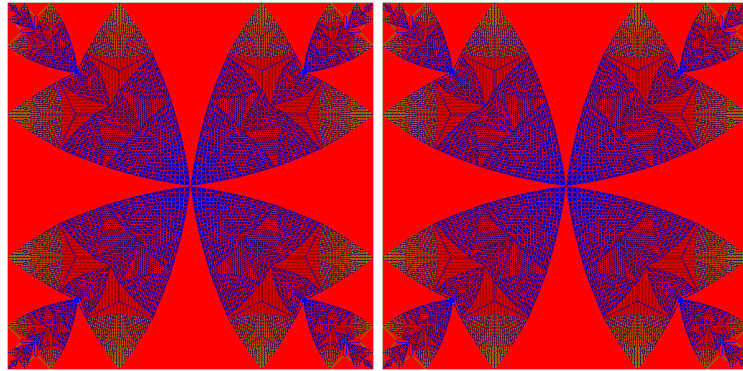
```
27 int ssandPile_do_tile_opt(int x, int y, int width, int height)
28 {
29     int diff = 0;
30
31     for (int i = y; i < y + height; i++)
32         for (int j = x; j < x + width; j++)
33             {
34                 int result = table(in, i, j) % 4;
35                 result += table(in, i + 1, j) / 4;
36                 result += table(in, i - 1, j) / 4;
37                 result += table(in, i, j + 1) / 4;
38                 result += table(in, i, j - 1) / 4;
```

```

39     table(out, i, j) = result;
40     diff /= result >= 4;
41 }
42
43 return diff;
44 }

```

Nous avons vérifié et on obtient le même résultats et le même nombre d'itérations (69190) avec la version par défaut et la version opt.



(a) avec do_tile_default

(b) avec do_tile_opt

FIGURE 1 – Verification du résultats pour ssandPile

Le gain de performance est de 2.37 car $\frac{178812}{75408}$

Pour asandPile_do_tile_opt() on a retiré les appels à atable(i, j) pour passer par une variable intermédiaire result. Cette modification permet au compilateur de vectoriser car il peut maintenant facilement voir que les différentes lignes peuvent être calculés en parallèle.

Nous avons modifié ces lignes :

```

1  int asandPile_do_tile_default(int x, int y, int width, int height)
2  {
3      int change = 0;
4
5      for (int i = y; i < y + height; i++)
6          for (int j = x; j < x + width; j++)
7              if (atable(i, j) >= 4)
8                  int result = atable(i, j);
9                  if (result >= 4)
10                     {
11                         result/=4;
12                         atable(i, j - 1) += atable(i, j) / 4;
13                         atable(i, j + 1) += result;
14                         atable(i, j - 1) += atable(i, j) / 4;

```

```

15 +         atable(i, j + 1) += result;
16 -         atable(i - 1, j) += atable(i, j) / 4;
17 +         atable(i - 1, j) += result;
18 -         atable(i + 1, j) += atable(i, j) / 4;
19 +         atable(i + 1, j) += result;
20         atable(i, j) %= 4;
21         change = 1;
22     }
23     return change;
24 }

```

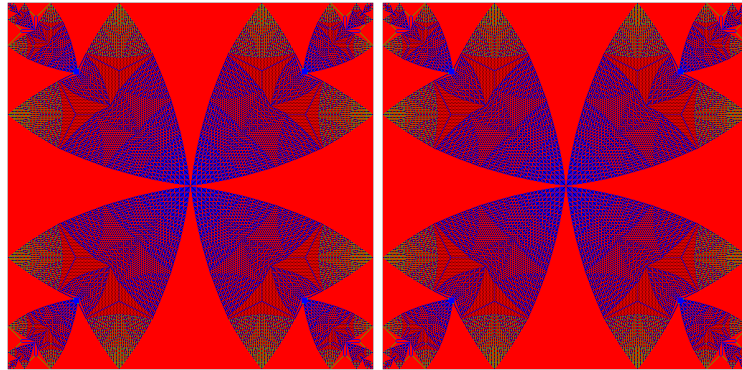
Le code de la fonction final :

```

26 int asandPile_do_tile_opt(int x, int y, int width, int height)
27 {
28     int change = 0;
29
30     for (int i = y; i < y + height; i++)
31         for (int j = x; j < x + width; j++)
32         {
33             int result = atable(i, j);
34             if (result >= 4)
35             {
36                 result/=4;
37                 atable(i, j - 1) += result;
38                 atable(i, j + 1) += result;
39                 atable(i - 1, j) += result;
40                 atable(i + 1, j) += result;
41                 atable(i, j) %= 4;
42                 change = 1;
43             }
44         }
45     return change;
46 }

```

Nous avons vérifié et on obtient le même résultat et le même nombre d'itérations (34938) avec la version par défaut et la version opt.



(a) avec do_tile_default

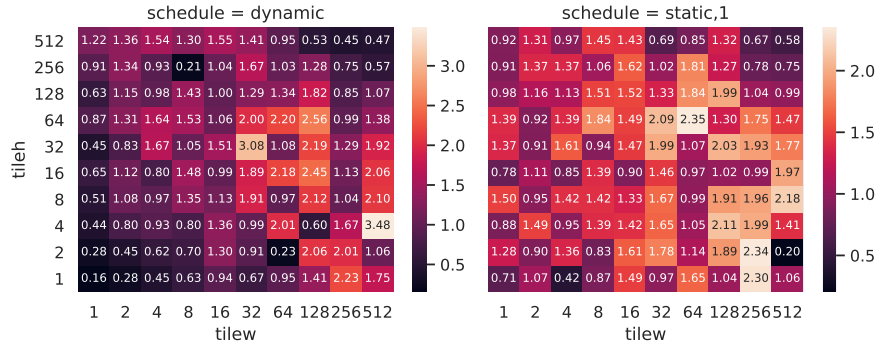
(b) avec do_tile_opt

FIGURE 2 – Verification du résultats pour asandPile

Le gain de performance est de 1.2 car $\frac{37990}{31405}$

2 OpenMP implementation of the synchronous version (4.2)

machine=data size=512 threads=24 kernel=ssandPile variant=omp_tiled tiling=default places=cores
refTime=28



3 OpenMP implementation of the asynchronous version (4.3)

Pour paralléliser avec *asandPile_compute_omp_tiled*, il a fallu créer 4 nids de boucles afin d'éviter les lectures/écritures concurrentes. On a donc un nid de boucle pour les lignes et colonnes impaires, un pour les lignes impaires et colonnes paires, un pour les lignes paires et colonnes impaires, et un pour les lignes et colonnes paires.

Le code de la fonction final :

```
1 unsigned asandPile_compute_omp_tiled(unsigned nb_iter)
2 {
```

```

3   for (unsigned it = 1; it <= nb_iter; it++)
4   {
5       int change = 0;
6
7       //BLEU
8       #pragma omp parallel for schedule(runtime) shared(change)
9       for(int y=0; y<DIM; y+=2*TILE_H)
10      {
11          for (int x = y%(TILE_H*2); x < DIM; x += TILE_W*2)
12          {
13              int localChange =
14                  do_tile(x + (x == 0), y + (y == 0),
15                          TILE_W - ((x + TILE_W == DIM) + (x == 0)),
16                          TILE_H - ((y + TILE_H == DIM) + (y == 0)), omp_get_thread_num());
17              if (change == 0 && localChange != 0)
18              {
19                  #pragma omp critical
20                  change |= localChange;
21              }
22          }
23      }

```

Le nid BLEU est répété 4 fois avec des initialisations de x et y qui diffèrent

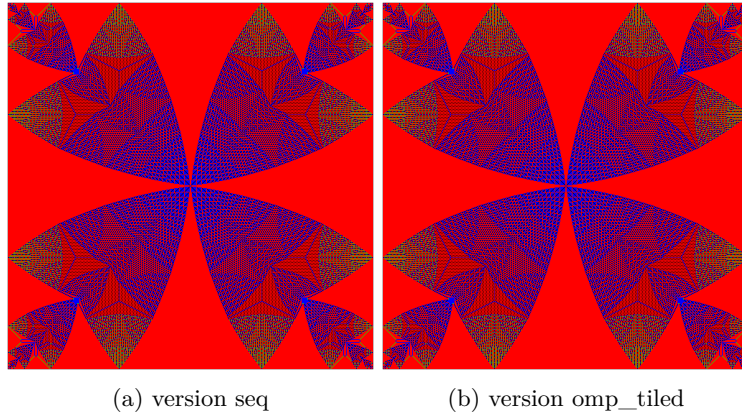


FIGURE 3 – Verification du résultats pour asandPile

Le gain de performance par rapport au séquentiel avec tuiles optimisées est de 1.98 car $\frac{30927.379}{15641.740}$

machine=kira size=512 threads=24 kernel=asandPile variant=omp_tiled tiling=default iterations=100
places=cores refTime=144

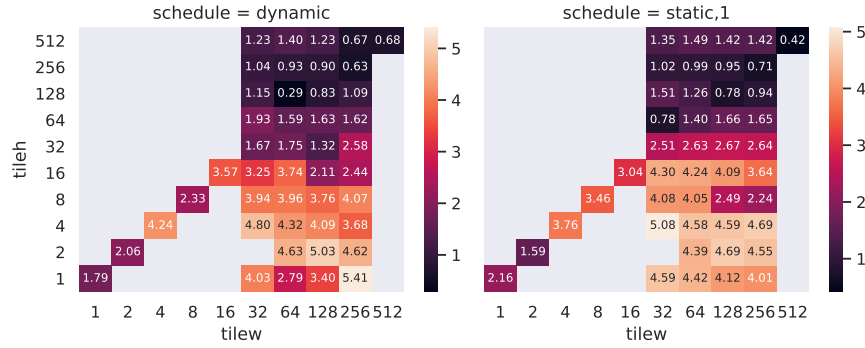
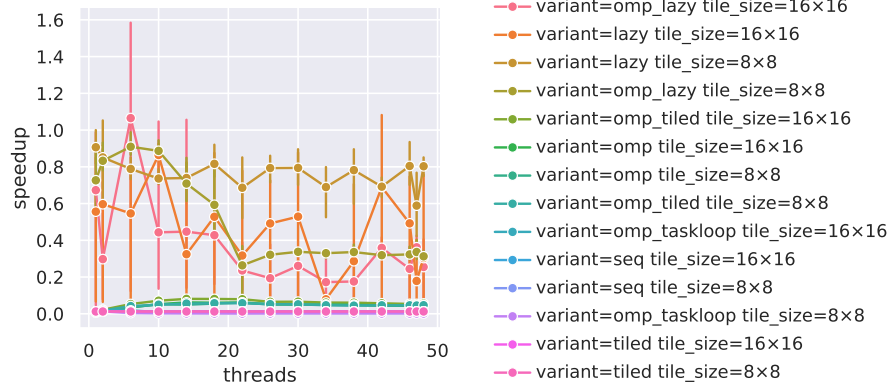


FIGURE 4 – heatmap comparaison omp_tiled & tiled

Les expériences montrent que pour certaines tailles de tuiles, il n'y a pas de résultat d'accélération, et après vérification, le programme ne fonctionne pas sur ces tailles là. Néanmoins, avec ces résultats, on en déduit que le schedule static,1 avec une width de 32 et une height de 4 est la meilleure option.

machine=kira size=512 kernel=ssandPile tiling=default schedule=dynamic places=cores
refTime=3 legend



machine=kira size=512 kernel=ssandPile tiling=default schedule=dynamic places=cores
refTime=3 legend

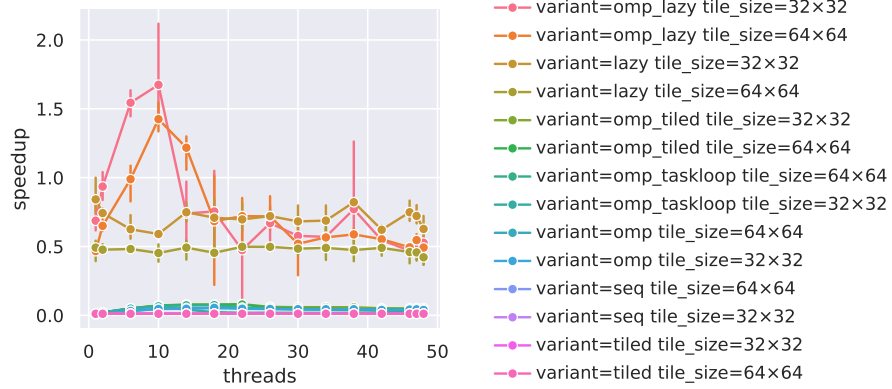
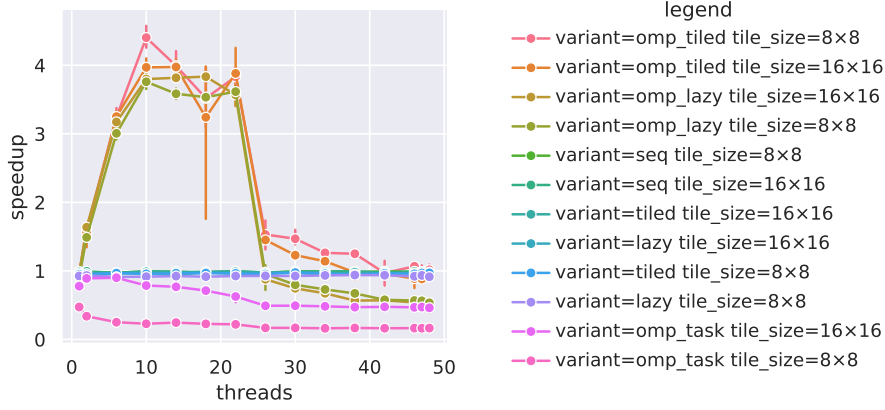


FIGURE 5 – ssandPile comparaison de toutes les variants

machine=kira size=512 kernel=asandPile tiling=default schedule=dynamic places=cores
refTime=143



machine=kira size=512 kernel=asandPile tiling=default schedule=dynamic places=cores
refTime=142

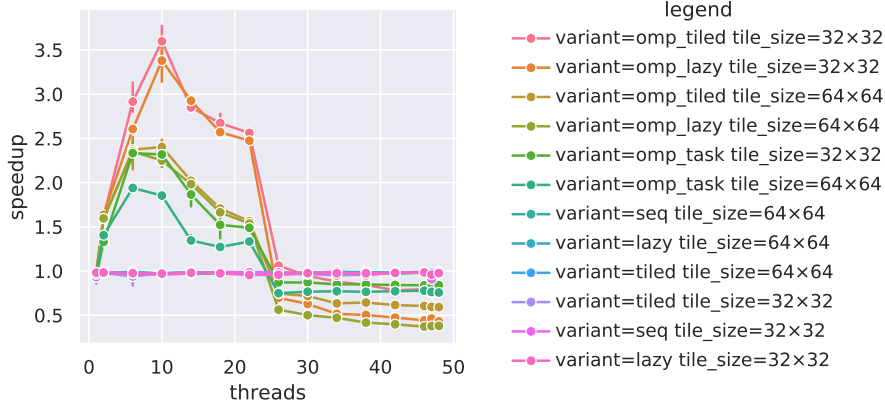
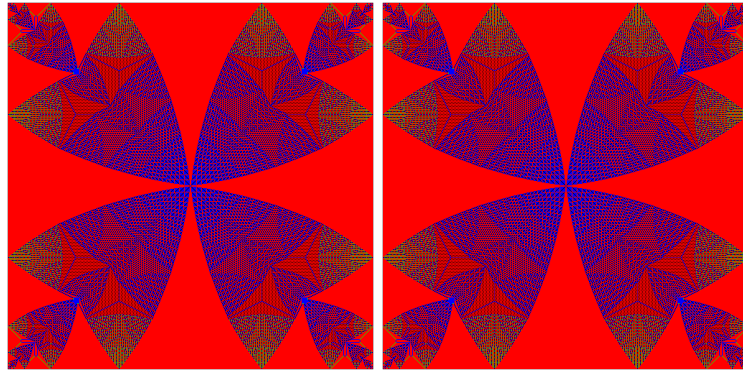


FIGURE 6 – asandPile comparaison de toutes les variants

4 Lazy OpenMP implementations (4.4)

Pour la version asynchrone, nous avons repris la version précédente et ajouté deux tableaux servant successivement de lecture et d'écriture pour tester si les tuiles autour ont été modifiés à l'itération précédente.



(a) avec do_tile_default

(b) version omp_lazy

FIGURE 7 – Verification du résultats pour asandPile

Le gain de performance par rapport au asynchrone lazy avec tuiles optimisées est de 1.35 car $\frac{32292.716}{23825.744}$

machine=kira size=512 threads=24 kernel=asandPile variant=omp_lazy tiling=default iterations=100
places=cores refTime=144

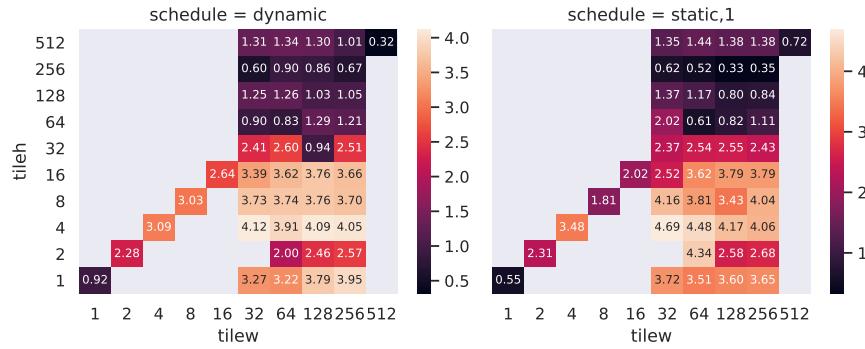


FIGURE 8 – heatmap comparaison omp_lazy & lazy

Les expériences montrent que pour certaines tailles de tuiles, il n'y a pas de résultat d'accélération, et après vérification, le programme ne fonctionne pas sur ces tailles là. Néanmoins, avec ces résultats, on en déduit que le schedule static,1 avec une width de 32 et une height de 4 est la meilleure option.