Rapport Rendu 3

Kristo DHIMA
Garion GOUBARD

08 Avril 2022

4.5 AVX implementation

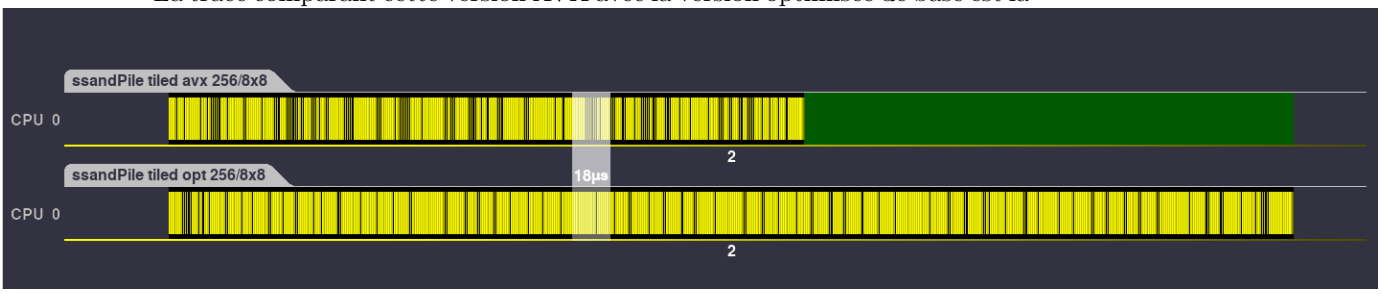
4.5.1 The synchronous case

Le code de la fonction ssandPile_do_tile_avx est le suivant:

```
unsigned ssandPile_do_tile_avx(int x, int y, int width, int
↪  height)
{
  if (x == 1 || y == 1 || x + width == DIM - 1 || y + height ==
  ↪  DIM - 1)
    return ssandPile_do_tile_opt(x, y, width, height);

  unsigned diff = 0;
  __m512i set3 = _mm512_set1_epi32(3);
  for (int i = y; i < y + height; i++)
  {
    for (int j = x; j < x + width; j += AVX512_VEC_SIZE_INT)
    {
      // optimal version using AVX
      __m512i base_table = _mm512_loadu_epi32(&table(in, i, j));

      base_table = _mm512_and_epi32(base_table, set3);
      __m512i right = _mm512_loadu_epi32(&table(in, i, j + 1));
      base_table = _mm512_add_epi32(base_table,
      ↪  _mm512_srli_epi32(right, 2));
      __m512i left = _mm512_loadu_epi32(&table(in, i, j - 1));
      base_table = _mm512_add_epi32(base_table,
      ↪  _mm512_srli_epi32(left, 2));
      __m512i down = _mm512_loadu_epi32(&table(in, i + 1, j));
      base_table = _mm512_add_epi32(base_table,
      ↪  _mm512_srli_epi32(down, 2));
      __m512i up = _mm512_loadu_epi32(&table(in, i - 1, j));
```

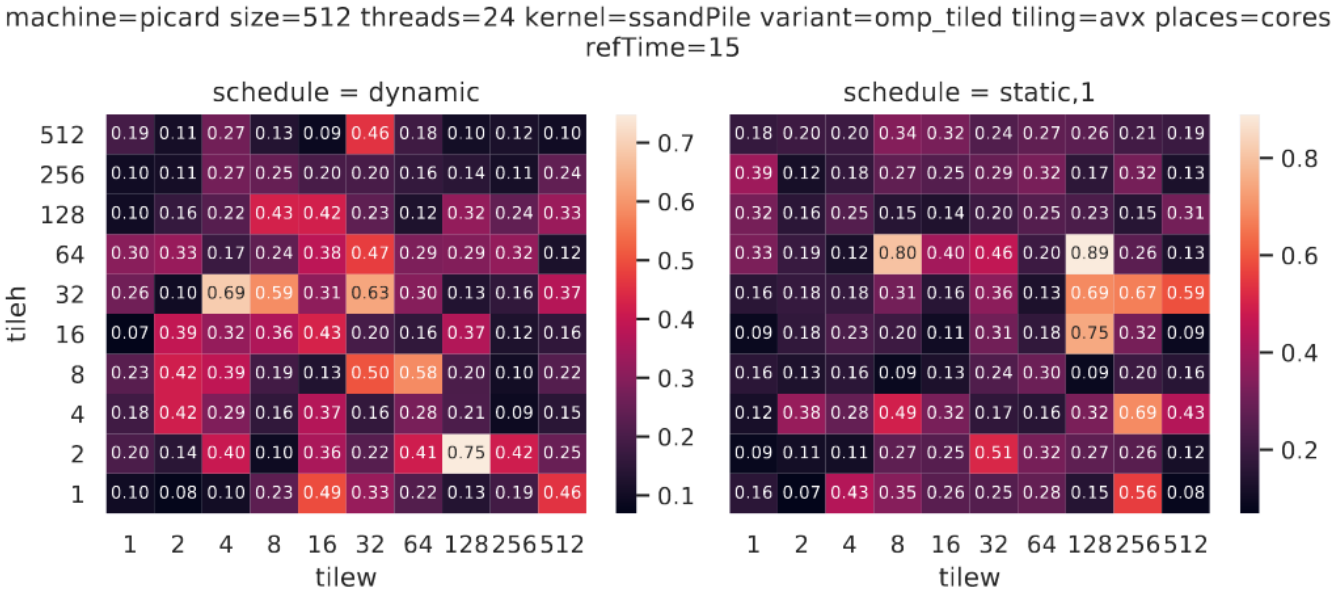
```
            base_table = _mm512_add_epi32(base_table,
            ↪  _mm512_srli_epi32(up, 2));
            _mm512_storeu_epi32(&table(out, i, j), base_table);

            diff += _mm512_cmpgt_epi32_mask(base_table, set3);
          }
        }
        return diff;
}
```

La trace comparant cette version AVX avec la version optimisée de base est la

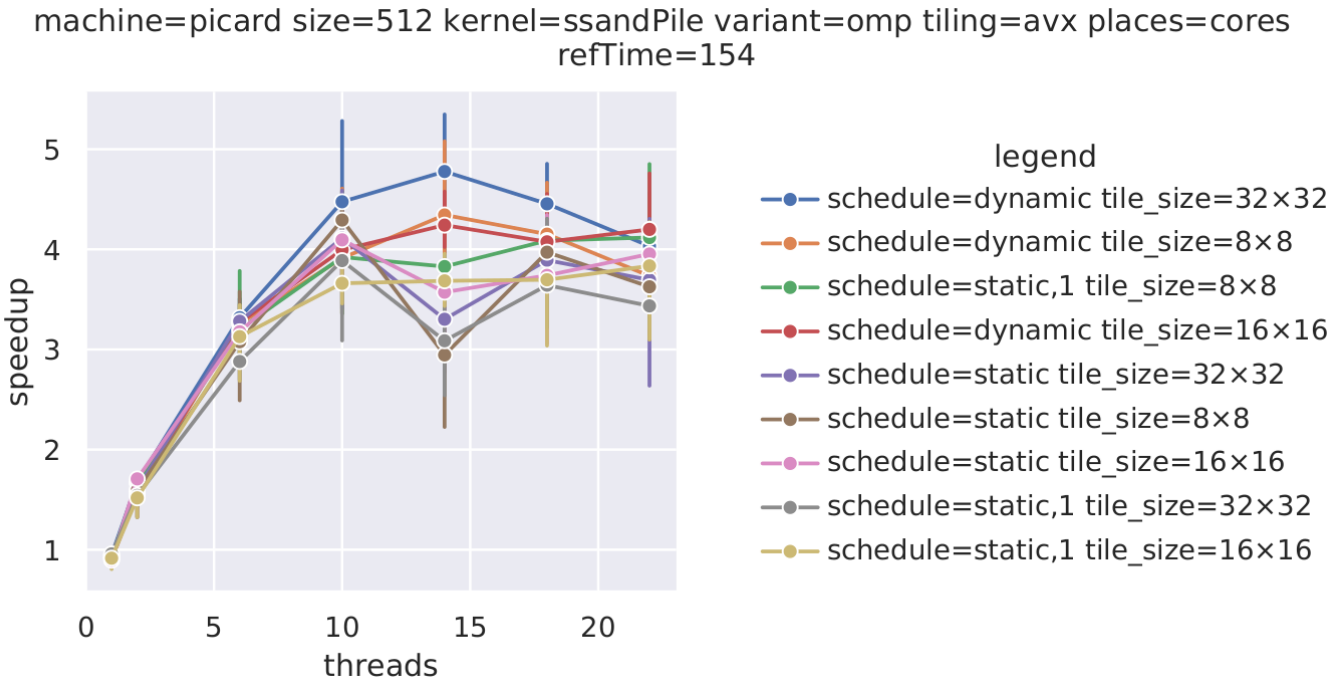


Nous pouvons facilement voir que la version AVX est beaucoup plus rapide que la version optimisée de base en sequentiel avec un seul thread.

Le heatmap de la version avx en utilisant le kernel `omp_tiled` est le suivant :



Nous pouvons remarquer que pour schedule = dynamic, l'ideal size est le 32, tandis que l'ideal height et width de la tuile sont respectivement 2 et 128. Pour schedule = static, height doit etre soit 64, tandis que width doit etre 8 ou 128 pour avoir des bons resultats.



Nous pouvons voir a partir de ce graph que le meilleur schedule a utiliser est le dynamic, avec tile_size = 32, en parallelisant en 14 threads.

Nous avons aussi essayé de faire une fonction qui n'utilise que les opérations SIMD, même pour les bordures. Cette version qui ne fonctionne pas totalement mais sur 80% de l'image est environ 4 fois plus rapide. Par manque de temps, nous n'avons pas pu terminer la fonction, mais nous avons bien compris la théorie. Il s'agit pour les cases qui n'ont pas de voisins de gauche ou de droite de copier le tableau de la case actuelle et de le décaler avec un 0 simulant la case qui n'existe pas pour que les calculs puissent s'effectuer. Celà est seulement nécessaire pour les x puisque le load s'effectue en largeur.

Le code de cette version est le suivant :

```
unsigned ssandPile_do_tile_avx2(int x, int y, int width, int
↪  height)
{
  unsigned diff = 0;
  __m512i set3 = _mm512_set1_epi32(3);
    for(int i = y; i < y + height; i++)
    {
      for(int j = x; j < x + width; j+=AVX512_VEC_SIZE_INT)
      {
        // optimal version using AVX
        __m512i base_table = _mm512_loadu_epi32(&table(in, i,
        ↪  j));
```

```
        base_table = _mm512_and_epi32(base_table, set3);
        if(j  != DIM - width - 1)
        {
        __m512i right = _mm512_loadu_epi32(&table(in, i, j + 1));
        base_table = _mm512_add_epi32(base_table,
        ↪  _mm512_srli_epi32(right, 2));
        }
        else {
          __m512i right = _mm512_alignr_epi32(base_table,
          ↪  _mm512_setzero_si512() , AVX512_VEC_SIZE_INT - 1);
          base_table = _mm512_add_epi32(base_table,
          ↪  _mm512_srli_epi32(right, 2));
        }
        if(j!=1)
        {
          __m512i left = _mm512_loadu_epi32(&table(in, i, j - 1
          ↪  ));
          base_table = _mm512_add_epi32(base_table,
          ↪  _mm512_srli_epi32(left, 2));
        }
        else{
          __m512i left = _mm512_alignr_epi32(
          ↪  _mm512_setzero_si512() , base_table ,1);
          base_table = _mm512_add_epi32(base_table,
          ↪  _mm512_srli_epi32(left, 2));
        }
        __m512i down = _mm512_loadu_epi32(&table(in, i + 1, j));
        base_table = _mm512_add_epi32(base_table,
        ↪  _mm512_srli_epi32(down, 2));
        __m512i up = _mm512_loadu_epi32(&table(in, i - 1, j));
        base_table = _mm512_add_epi32(base_table,
        ↪  _mm512_srli_epi32(up, 2));
        _mm512_storeu_epi32(&table(out, i, j), base_table);

        diff += _mm512_cmpgt_epi32_mask(base_table, set3);
      }
    }
  return diff;

}
```

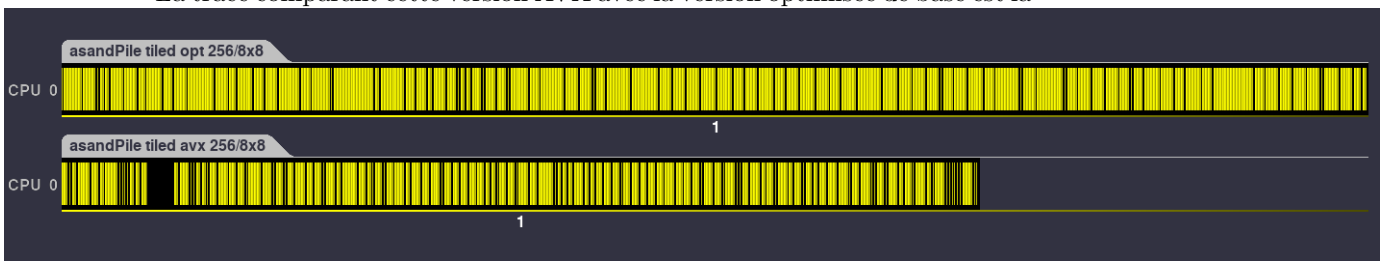
4.5.2 The asynchronous case

Nous avons implémenté la version asynchrone avec les instructions `mm512` :

```
int asandPile_do_tile_avx(int x, int y, int width, int height)
{
  if (x == 1 || y == 1 || x == DIM - width - 1 || y == DIM -
  ↪  height - 1)
  {
    return asandPile_do_tile_opt(x, y, width, height);
  }
  int change = 0;
  __m512i set3 = _mm512_set1_epi32(3);
  for (int i = y; i < y + height; i++)
    for (int j = x; j < x + width; j += AVX512_VEC_SIZE_INT)
    {
      int unstable = atable(i, j) >= 4;

      //  ------------ load
      __m512i up = _mm512_loadu_epi32(&atable(i - 1, j));
      __m512i base_table = _mm512_loadu_epi32(&atable(i, j));
      __m512i down = _mm512_loadu_epi32(&atable(i + 1, j));

      __mmask16 mask = 0xffff /* >> (j == DIM -
      ↪  AVX512_VEC_SIZE_INT) << (j == 0) */;

      // ------------ let the sand fall synchronously
      // D <- Tj,i div 4
      __m512i D = _mm512_srli_epi32(base_table, 2);
      __m512i mod4 = _mm512_and_epi32(base_table, set3);
      __m512i dright =
      ↪  _mm512_alignr_epi32(_mm512_setzero_si512(), D, 1);
      __m512i dleft = _mm512_alignr_epi32(D,
      ↪  _mm512_setzero_si512(), AVX512_VEC_SIZE_INT - 1);
      base_table = _mm512_add_epi32(mod4, dright);
      base_table = _mm512_add_epi32(base_table, dleft);
      up = _mm512_add_epi32(up, D);
      down = _mm512_add_epi32(down, D);
      // ------------ finally store
      // put D in a array
      TYPE save[AVX512_VEC_SIZE_INT];
      _mm512_storeu_epi32(&save, D);

      atable(i, j - 1) += save[0];
      atable(i, j + AVX512_VEC_SIZE_INT) +=
      ↪  save[AVX512_VEC_SIZE_INT - 1];

      _mm512_storeu_epi32(&atable(i, j),
      ↪  _mm512_maskz_abs_epi32(mask, base_table));
```

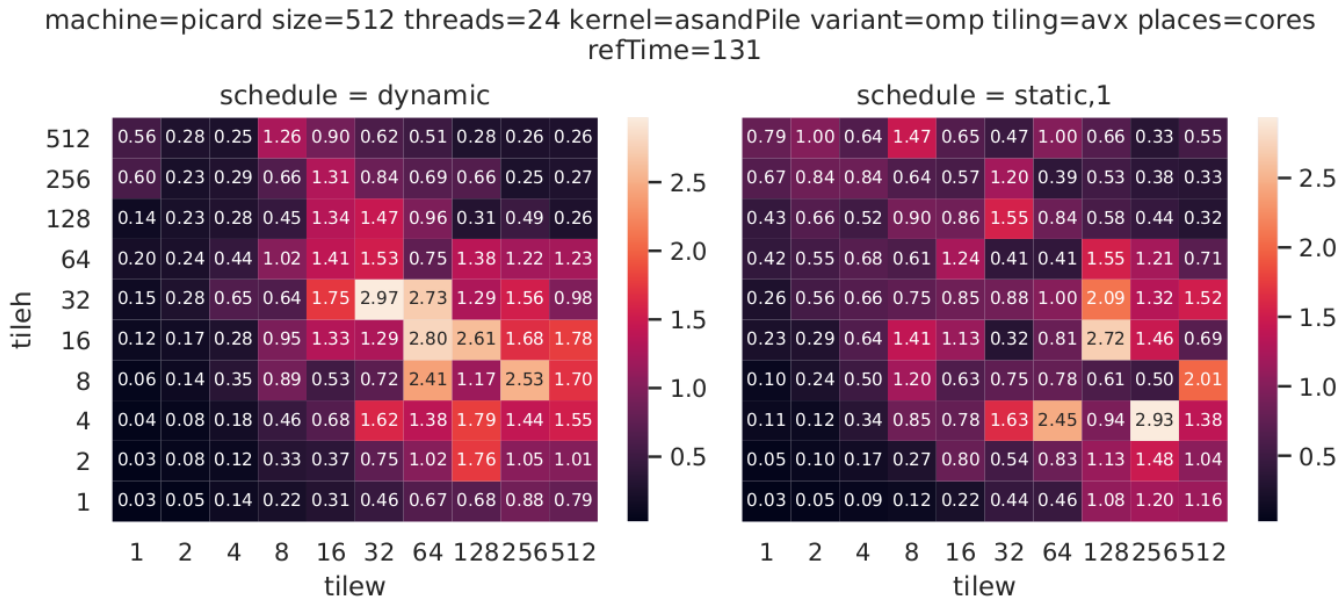
```
            _mm512_storeu_epi32(&atable(i - 1, j),
            ↪  _mm512_maskz_abs_epi32(mask, up));
            _mm512_storeu_epi32(&atable(i + 1, j),
            ↪  _mm512_maskz_abs_epi32(mask, down));
            change += unstable;
          }
        return change;
      }
```

La trace comparant cette version AVX avec la version optimisée de base est la

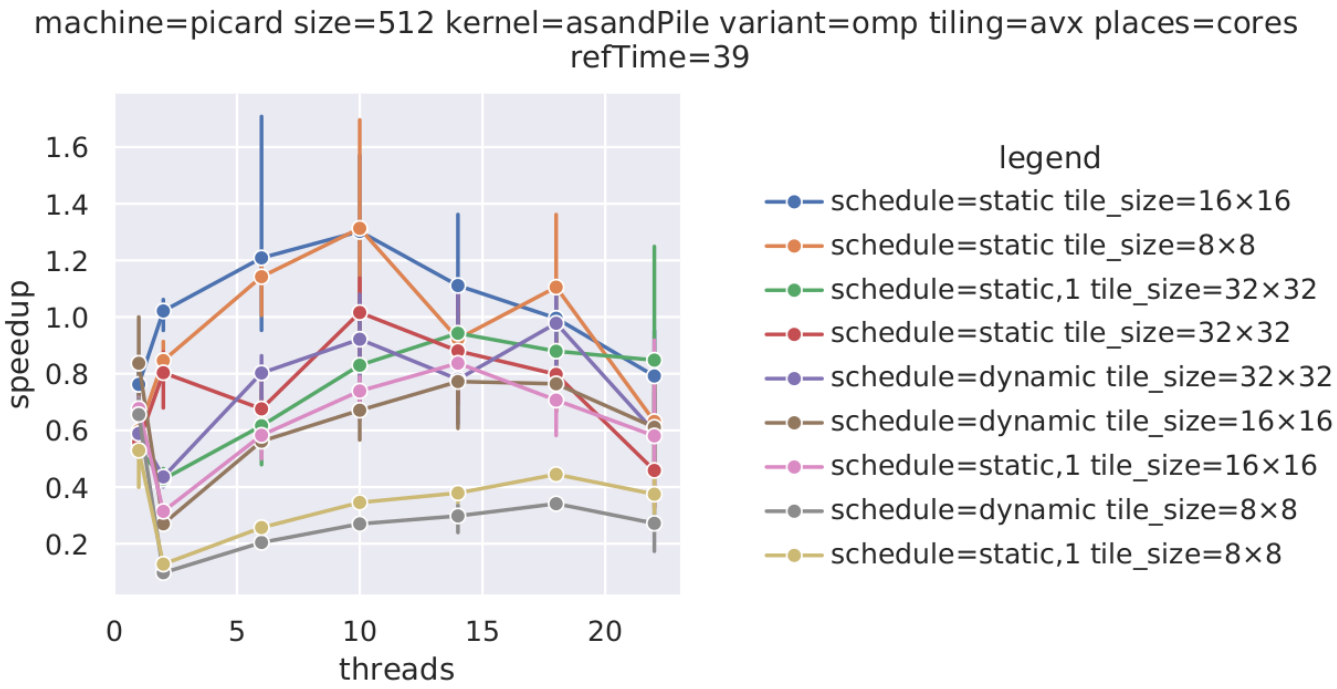


Nous pouvons facilement voir que la version AVX est beaucoup plus rapide que la version optimisée de base.

Le heatmap de la version avx en utilisant le kernel `omp_tiled` est le suivant :



Nous pouvons remarquer que pour schedule = dynamic, la taille ideale de tuile est 32x32, tandis que pour schedule = static,1 , la taille ideale de tuile est height = 16 et width = 128.



Nous pouvons voir que le meilleur schedule est le static, avec une taille de tuile de 16x16, en parallelisant en 10 threads.

4.7 OpenCL Implementation

4.7.1 Basic OpenCL Implementation

```
__kernel void ssandPile_ocl (__global unsigned *in, __global
↪  unsigned *out)
{
  int x = get_global_id (0);
  int y = get_global_id (1);

  if(x != 0 && y != 0 && x  != DIM  - 1 && y != DIM - 1)
  {
  out[y*DIM+x] = in[y*DIM+x] % 4 +  in[(y+1)*DIM+x] / 4 +
  ↪  in[y*DIM+x+1] / 4 + in[(y-1)*DIM+x] / 4 + in[y*DIM+x-1] /
  ↪  4;
  }


}
```

```
kdhima@picard:/autofs/unitytravail/travail/kdhima/Master/GL_S8/Programmation_Architectures_Parallelles/pap-3$ OMP_NUM_THREADS=1 ./run -k ssand
Pile -s 256 -n -i 17035
Using kernel [ssandPile], variant [seq], tiling [default]
Computation completed after 17035 iterations
10914.566
kdhima@picard:/autofs/unitytravail/travail/kdhima/Master/GL_S8/Programmation_Architectures_Parallelles/pap-3$ OMP_NUM_THREADS=1 ./run -k ssand
Pile -s 256 -n -o -i 17035
Using kernel [ssandPile], variant [ocl], tiling [default]
Using OpenCL Device: GPU [GeForce RTX 2070]
Using 256x256 workitems grouped in 16x16 tiles
Computation completed after 17035 iterations
63.657
```

La version OpenCL implementee est beaucoup plus rapide que la version sequentielle.