PAP

Projet : rapport3

4TIN804U

BERASATEGUY Tanguy, GOEDEFROIT Charles

2021-2022

Table des matières

1 4.5 AVX implementation

1.1 4.5.1 The synchronous case

On fait le speedup avec *omp_tile* entre les tailing *opt* et *avx* sur la machine *UHURA* on obtien 4483 pour opt et 4764 pour avx un speedup de $0.94 = \frac{4483}{4764}$. Il n'y a pas une grande différence entre les 2 version car gcc a une très bonne vectorization.

1.2 4.5.2 The asynchronous case

On a implémenter en suivant la consigne su sujet mais on n'a par réussi a avoir autre chose qu'un fond noire.

Le code de la fonction :

```
1  int asandPile_do_tile_avx(int x, int y, int width, int height)
2  {
3    // $$\overrightarrow{X} == vecX$$
4    const __m256i vec3_i = _mm256_set1_epi32(3);
5
6    int diff = 0;
7    for (int j = y; j < y + height; j++)
8      for (int i = x; i < x + width; i += AVX_VEC_SIZE_INT)
9      {
10       // vecT_{j-1,i} <-- (t_{j-1,i+k}, ..., t_{j-1,i})
11       __m256i topVec_i = _mm256_loadu_si256((__m256i *) &table(in, j - 1, i));
12       // vecT_{j,i} <-- (t_{j,i+k}, ..., t_{j,i})
13       __m256i vec_i = _mm256_loadu_si256((__m256i *) &table(in, j, i));
14       // vecT_{j+1,i} <-- (t_{j+1,i+k}, ..., t_{j+1,i})
15       __m256i bottomVec_i = _mm256_loadu_si256((__m256i *) &table(in, j + 1, i));
16
17       // vecD <-- vec_i / 4
18       __m256i vecD = _mm256_srli_epi32(vec_i, 3);
19
20       // vec_i <-- vec_i % 4 + (vecD << 1) + (vecD >> 1)
21       vec_i = _mm256_add_epi32(_mm256_and_si256(vec_i, vec3_i),
22                                _mm256_add_epi32(_mm256_slli_epi32(vecD, 1), _mm256_srli_epi32(vecD, 1)));
23
24       // topVec_i <-- topVec_i + vecD
25       topVec_i = _mm256_add_epi32(topVec_i, vecD);
26
27       // bottomVec_i <-- bottomVec_i + vecD
28       bottomVec_i = _mm256_add_epi32(bottomVec_i, vecD);
29
30       // t_{j,i-1} <-- t_{j,i-1} + vecD[0]
31       __m256i leftVec_i = _mm256_loadu_si256((__m256i *) &table(in, j, i - 1));
32       leftVec_i         = _mm256_add_epi32(leftVec_i, vecD);
33       _mm256_storeu_si256((__m256i *) &table(in, j, i - 1), leftVec_i);
34
35       // t_{j,i+k+1} <-- t_{j,i+k+1} + vecD[k]  :  k = AVX_VEC_SIZE_INT-1
36       __m256i rightVec_i = _mm256_loadu_si256((__m256i *) &table(in, j, i + 1));
37       rightVec_i         = _mm256_add_epi32(rightVec_i, vecD);
```

```
38        _mm256_storeu_si256((__m256i *) &table(in, j, i + 1), rightVec_i);
39
40        // (t_{j-1,i+k}, ..., t_{j-1,i}) <-- vecT_{j-1,i}
41        _mm256_storeu_si256((__m256i *) &table(out, j - 1, i), topVec_i);
42        // (t_{j,i+k}, ..., t_{j,i}) <-- vecT_{j,i}
43        _mm256_storeu_si256((__m256i *) &table(out, j, i), vec_i);
44        // (t_{j+1,i+k}, ..., t_{j+1,i}) <-- vecT_{j+1,i}
45        _mm256_storeu_si256((__m256i *) &table(out, j + 1, i), bottomVec_i);
46
47
48        // __m256i mask = _mm256_xor_si256(result_i, currentPixelsRow_i);
49        // if (_mm256_testz_si256(mask, mask) != 1)
50          diff = 1;
51      }
52
53    return diff;
54  }
```

2 4.7 OpenCL Implementation

2.1 4.7.1 Basic OpenCL Implementation