Report : Hill – Climbing implementation – Alexandre Temperville

All this work is in the folder 'Hill_Climbing'. Then, the two first programs are located in folder 'Version_CPU', the final program in the folder 'Version_GPU', and the old version of the final program I keep (because it is the first idea I had to make the program and it is interesting) in folder 'Old_Version_GPU'. I do not detail so much my programs as I put comments inside which help to understand how it works. I explain in this report the main idea.

Hill – Climbing procedure on CPU

We do that with the program *HC_CPU.cpp*. While a neighbor of a solution makes a small score than the previous one, we permut the two elements to make a new solution. When there are no best neighbor solution, we stop the computation. Here is what we obtain when we compile and then run the program with the file *tai20a.dat*.

```
temperville@temperville-alex:~/Part4/10_1$ g++ HC_CPU.cpp -o HC
temperville@temperville-alex:~/Part4/10_1$ ./HC tai20a.dat
Solution initiale : 19 18 12 13 4 14 15 7 1 10 16 17 2 6 8 5 9 11 0 3
Score initial = 886034

Score temporaire = 859222
Solution temporaire = 19 18 12 11 4 14 15 7 1 10 16 17 2 6 8 5 9 13 0 3
Score temporaire = 835056
Solution temporaire = 19 18 12 11 4 14 15 0 1 10 16 17 2 6 8 5 9 13 7 3
Score temporaire = 819110
Solution temporaire = 14 18 12 11 4 19 15 0 1 10 16 17 2 6 8 5 9 13 7 3
Score temporaire = 803252
Solution temporaire = 14 18 5 11 4 19 15 0 1 10 16 17 2 6 8 12 9 13 7 3
Score temporaire = 788678
Solution temporaire = 15 18 5 11 4 19 14 0 1 10 16 17 2 6 8 12 9 13 7 3
Score temporaire = 780806
Solution temporaire = 15 18 5 11 4 19 14 0 1 10 16 17 2 9 8 12 6 13 7 3
Score temporaire = 773418
Solution temporaire = 15 18 5 11 4 19 14 0 1 2 16 17 10 9 8 12 6 13 7 3
Score temporaire = 771908
Solution temporaire = 15 18 5 11 4 19 14 0 1 2 16 13 10 9 8 12 6 17 7 3
Score temporaire = 765472
Solution temporaire = 3 18 5 11 4 19 14 0 1 2 16 13 10 9 8 12 6 17 7 15
Score temporaire = 757442
Solution temporaire = 3 18 5 11 4 19 14 0 8 2 16 13 10 9 1 12 6 17 7 15
Score temporaire = 754016
Solution temporaire = 3 18 5 11 4 19 14 0 8 2 16 13 7 9 1 12 6 17 10 15
Score temporaire = 748702
Solution temporaire = 3 18 5 11 4 19 14 0 8 2 12 13 7 9 1 16 6 17 10 15
Score temporaire = 748702
Solution temporaire = 3 18 5 11 4 19 14 0 8 2 12 13 7 9 1 16 6 17 10 15

Solution finale : 3 18 5 11 4 19 14 0 8 2 12 13 7 9 1 16 6 17 10 15
Score finale = 748702
temperville@temperville-alex:~/Part4/10_1$
```

Multistart Hill – Climbing procedure on CPU

The program *multistartHC_CPU.cpp* do the same that the program *HC_CPU.cpp* but looks for several random initial solutions (in my program there are 5 iterations so 5 random initial solutions). The programs keeps the best solution and the best score in each iteration to give a solution at the end.

```
temperville@temperville-alex:~/Part4/10_1$ g++ multistartHC_CPU.cpp -o multistar
t
temperville@temperville-alex:~/Part4/10_1$ ./multistart tai20a.dat
Solution du 1 -ieme Hill Climbing : 8 7 15 11 19 17 16 2 12 3 10 6 9 1 18 4 13 5
 0 14
score = 756214

Solution du 2 -ieme Hill Climbing : 6 14 12 11 17 1 2 19 10 7 18 9 3 15 8 0 16 5
 13 4
score = 734514

Solution du 3 -ieme Hill Climbing : 10 17 6 16 1 7 0 2 12 9 18 3 15 19 4 13 5 14
 8 11
score = 731206

Solution du 4 -ieme Hill Climbing : 4 8 17 19 1 11 10 0 13 6 16 14 15 5 7 3 9 2
18 12
score = 729450

Solution du 5 -ieme Hill Climbing : 3 19 5 18 2 15 9 14 17 10 11 13 0 12 4 16 7
6 8 1
score = 762584

Solution du meilleur Hill Climbing : 4 8 17 19 1 11 10 0 13 6 16 14 15 5 7 3 9 2
 18 12
Meilleur score : 729450

temperville@temperville-alex:~/Part4/10_1$
```

GPU implementation

I choose to implement the first scheme proposed. One of the difficulties is to find a mapping between a CUDA thread and a neighbor.

The picture ***mapping2D.png*** explains why this is not a good idea to consider threads in two dimensions, but allows to see how we can do a mapping in one dimension. This is now the idea of the picture ***mapping1D.png***. We can order the position in the array voisin thanks to a function I called *f*.

We can define *f* like in the following page (we consider $f(k) = f(k,n)$). At the beginning, I use this function in my program on the CPU and on the GPU, but I saw that more *n* is bigger, more the computations take a long time as I define it recursively with a sum, next I give the expression of *f* after the following calculation, and now, I do not need this function any more. I keep it because it is the main idea of my reasoning and all the modification I did in my programs.

$$f(0,n) = 0$$

$$\forall k \in \mathbb{N},\ \ f(k,n) = \sum_{i=1}^{k}(n-i) = \sum_{i=1}^{k} n - \sum_{i=1}^{k} i$$

$$\forall k \in \mathbb{N},\ \ f(k,n) = k \times n - \frac{k \times (k+1)}{2}$$

$$\forall k \in \mathbb{N},\ \ f(k,n) = k \times \left(n - \frac{k+1}{2} \right)$$

We can find the old version of my final program in the folder 'Old_Version_GPU' and the new one in 'Version_GPU'. We can see that I used the function *f* I give previously too much. Indeed, we spend a lot of time to compute it, in particular in the last threads. So, the idea of my new program is to do computations inside my procedure without using a function *f*. I realise it using some variables *temp* ('temporary value of *f(k,n)*') and *prec* (value of *f(k*-1,n) at loop *k*). We can see the differences between my old program and the new one.

Concerning the new program, unless it does less operations than the old one, it takes approximatively the same execution time than the old one in the room 106. Maybe it should be less in *grid5000* but I do not have the time to verify it. I think that I could also optimize more my program. Indeed, in each loop, each thread computes its values *i* and *j*. We could create a GPU procedure which will compute these values before the loop.

Here we can see what the executable returns for 100 iterations to do the multistart in parallel on CPU and GPU. We should have a smaller time of execution for the GPU, it should be because of the capacity of the machines in the room 106.

```
[temperville@mathcalc11 version_GPU]$ nvcc part4.cu -o part4
[temperville@mathcalc11 version_GPU]$ part4 tai20a.dat 100
Le meilleur score trouvé par les Hill-climbing avec le CPU est :
z(pi) = 720728
pi = [ 12 0 17 9 2 16 10 1 13 18 19 15 11 8 7 3 6 14 4 5 ]
Temps d'exécution CPU : 0.100000 s

Le meilleur score trouvé par les Hill-climbing avec le GPU est :
z(pi) = 720728
pi = [ 12 0 17 9 2 16 10 1 13 18 19 15 11 8 7 3 6 14 4 5 ]
Temps d'exécution GPU : 0.160000 s

[temperville@mathcalc11 version_GPU]$ part4 tai100a.dat 100
Le meilleur score trouvé par les Hill-climbing avec le CPU est :
z(pi) = 21685674
pi = [ 40 73 88 19 2 68 22 79 36 66 37 14 95 41 29 75 59 0 11 18 21 67 27 81 72
4 76 43 39 62 98 25 23 97 4 63 7 99 96 77 42 86 45 30 28 47 ]
Temps d'exécution CPU : 72.879997 s

Le meilleur score trouvé par les Hill-climbing avec le GPU est :
z(pi) = 21685674
pi = [ 40 73 88 19 2 68 22 79 36 66 37 14 95 41 29 75 59 0 11 18 21 67 27 81 72
4 76 43 39 62 98 25 23 97 4 63 7 99 96 77 42 86 45 30 28 47 ]
Temps d'exécution GPU : 81.599998 s

[temperville@mathcalc11 version_GPU]$
```

Remarks : - I let the steps of my program in the folder 'Step', it is not very interesting to see them, but I keep it, it can still be used, even if there are not so good at the step I did them.

- I try to use a script (*HC.sh*) but it does not work and I do not find why.