Rapport PPAR : Lab 1

Antoine LEVAL

March 29, 2018

Abstract

Vous pouvez retrouver la version finale du TP sur ce dépôt Github : https://github.com/Viinyard/PPAR_TP5 ou sur ppar12@parawell.irisa.fr

1 Parallelization

1.1 Question 1

```
float log2_series(int n) {
   float res = 0;
   for(int i = 0; i < n; i++) {
      res += ((i % 2 == 0) ? 1 : -1) / (i + 1.0f);
   }
   return res;
}
```

1.2 Question 2

	CPU result	log	time
Increasing	0.693138	0.693147	1.253156s
Decreasing	0.693147	0.693147	1.265730s

Les temps sont à peu près similaire mais la précision du résultat diffère : quand on additionne de la plus petite vers la plus grande valeur, le résultat est plus précis est tend vers log(2). Cela est dû au fait que l'addition de nombre à virgule flotante n'est ni associative, ni commutative, car ce ne sont pas des nombres exactes mais des approximations.

1.3 Question 3

Il y a deux possibilités pour répartir les n éléments équitablement parmis m threads :

- Avec m/n blocks.
- Avec x blocks, et chaque thread itérerais alors ((m / n) / x) fois.

Pour nous la meilleure solution serait la seconde, afin de limiter plus aisément le nombre de thread total à 32 768 maximum, tout en ayant un multiple de 32 threads par block.

Avec (4 x 32) 2^7 threads par block, et un maximum de 2^{15} threads cela nous donne 2^8 blocks dans la grille. Avec un nombre total d'élément de (2^{10} * 2^{10} * 2^7) 2^{27}, divisé par le nombre de thread (2^{27} / 2^7 = 2^{20}), divisé par le nombre de block (2^{20} / 2^8) 2^{12} = 4096 itération par thread.

1.4 Question 6

Limiter le nombre de thread à 2^{15} est bien plus rapide que l'autre solution.

1.5 Question 7

Le mieux est d'avoir au maximum 2^{15} threads, une taille de block multiple de 32. Un nombre de thread par block de 64 ou 128 semble être le plus adapté, on observe une augmentation du temps de calcul au dessus et en dessous de ces valeurs.

2 Reduction

2.1 Question 8

Grâce à une fonction de réduction, en effet dans un même block les threads peuvent avoir une mémoire partagée.

```
__global__ void reduce(float data_size, float * data_out, float * data_block) {
   extern __shared__ float sdata[];
   unsigned int tid = threadIdx.x;
   unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
   sdata[tid] = (i < data_size) ? data_out[i] : 0;
   __syncthreads();

   for(unsigned int s = blockDim.x/2; s > 0; s >>= 1) {
      if(tid < s) {
         sdata[tid] += sdata[tid + s];
      }
      __syncthreads();
   }
   if(tid == 0) data_block[blockIdx.x] = sdata[0];
}
```

2.2 Question 9

```
int smemSize = threads_per_block * sizeof(float);
while(1) {
   reduce<<<blocks_in_grid, threads_per_block, smemSize>>>(data_size, data_out_gpu,
       data_block);
   data_out_gpu = data_block;
   if(blocks_in_grid == 1) break;
   if(blocks_in_grid < threads_per_block) threads_per_block = blocks_in_grid;
   blocks_in_grid /= (blocks_in_grid >= threads_per_block) ? threads_per_block :
       blocks_in_grid;
   printf("BLOCKS IN GRID = %d\n", blocks_in_grid);
}
```

Résultat :

```
CPU result: 0.693138
 log(2)=0.693147
 time=1.266734s
BLOCKS IN GRID = 256
BLOCKS IN GRID = 2
BLOCKS IN GRID = 1
0>0.693145
GPU results:
 Sum: 0.693145
 Total time: 0.0014984 s,
 Per iteration: 0.0111639 ns
 Throughput: 358.296 GB/s
```