os202 Examen

Thomas Varin

Mars 2024

Introduction

On peut voir sur le résultat de la commande "lscpu" ci-après que ma machine comporte 8 coeurs de calcul, chacun avec un cache L1d de 32KiB, un cache L1i de 32KiB et un cache L2 de 512KiB. De plus, il y a un cache L3 de 16MiB.

```
Architecture:          x86_64
  CPU op-mode(s):      32-bit, 64-bit
  Address sizes:       48 bits physical, 48 bits virtual
  Byte Order:          Little Endian
CPU(s):                16
  On-line CPU(s) list: 0-15
Vendor ID:             AuthenticAMD
  Model name:          AMD Ryzen 7 5800H with Radeon Graphics
    CPU family:        25
    Model:             80
    Thread(s) per core: 2
    Core(s) per socket: 8
    Socket(s):         1
    Stepping:          0
    Frequency boost:   enabled
    CPU max MHz:       3200,0000
    CPU min MHz:       1200,0000
    BogoMIPS:          6387.93
    Flags:             fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
        mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext
        fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc
        cpuid extd_apicid aperfmperf
                        rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1
                            sse4_2 movbe popcnt aes xsave avx f16c rdrand
                            lahf_lm cmp_legacy svm extapic cr8_legacy abm
                            sse4a misalignsse 3dnowprefetch osvw ibs skinit
                            wdt tce topoext perfctr
                        _core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3
                           cdp_l3 hw_pstate ssbd mba ibrs ibpb stibp vmmcall
                           fsgsbase bmi1 avx2 smep bmi2 erms invpcid cqm rdt_a
                           rdseed adx smap clflushopt clwb sha_ni xsaveopt
                           xsavec xge
                        tbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total
                           cqm_mbm_local clzero irperf xsaveerptr rdpru
                           wbnoinvd cppc arat npt lbrv svm_lock nrip_save
                           tsc_scale vmcb_clean flushbyasid decodeassists
                           pausefilter pfthreshold av
                        ic v_vmsave_vmload vgif v_spec_ctrl umip pku ospke
                           vaes vpclmulqdq rdpid overflow_recov succor smca
                           fsrm
Virtualization features:
  Virtualization:      AMD-V
Caches (sum of all):
  L1d:                 256 KiB (8 instances)
```

```
  L1i:                   256 KiB (8 instances)
  L2:                    4 MiB (8 instances)
  L3:                    16 MiB (1 instance)
NUMA:
  NUMA node(s):          1
  NUMA node0 CPU(s):     0-15
```

1ère étape

Pour cette étape, j'ai parallélisé selon les lignes, ce qui a permis de travailler avec des matrices plus petites en parallèles, pour ensuite les concaténer. Par chance (ou plutôt par anticipation du sujet), le code était déjà adapté à cette éventualité. Je me suis permis de renommer l'image de sortie "result.png" afin de pouvoir plus facilement la manipuler.
On a les speedups suivants :

Nombre de coeurs	Speedup
1	1.00
2	1.76
3	2.62
4	2.95
5	3.40
6	3.95
7	4.39
8	4.72

Commentaire

On remarque que, quand bien même l'exécution est plus rapide avec plus de coeurs de calcul, le résultat est de moins en moins satisfaisant. En effet, lorsque l'on parallélise sur les lignes (ou sur les colonnes), il est possible que la partie dessinée par l'utilisateur ne soit pas dans la sous-image traitée par un processus. Cela est du au fait qu'on essaie de colorer de la même couleur des "zones" de l'image, sans que les processus ne puisse connaître l'état des zones des autres. On se retrouve alors à appliquer un algorithme global sur des parties locales, ce qui ne fonctionne pas aussi bien. On remarque d'ailleurs assez bien avec 8 processus les lignes de découpage de l'algorithme.

2ème étape

J'ai eu un problème avec mpi reduce/scatter. Je vous envoie quand même l'essaie, avec la fonction parallelize_prod.