2024/25

Nom : DADA SIMEU CÉDRIC DAREL

Email : cedric-darel.dada@ensta-paris.fr

Titre : Compte rendu Examen

STIC

ENSTA Paris, Institut Polytechnique de Paris

Table des matières

Table des figures

1 Architecture matérielle

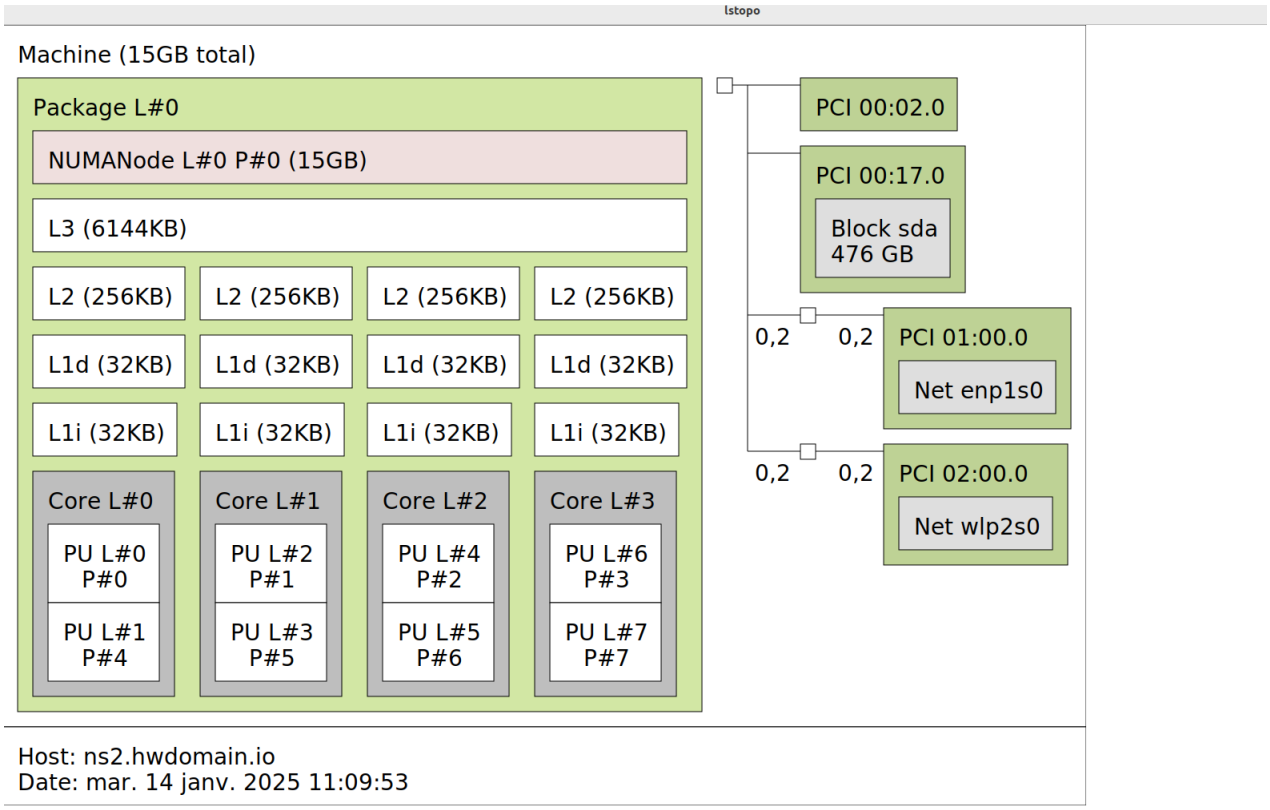

FIGURE 1 – Résultat de la commande lstopo : Nous pouvons visualiser les tailles des caches

```
cedric@ns2:~$ lscpu
Architecture :                          x86_64
  Mode(s) opératoire(s) des processeurs :   32-bit, 64-bit
  Address sizes:                        39 bits physical, 48 bits virtual
  Boutisme :                            Little Endian
Processeur(s) :                         8
  Liste de processeur(s) en ligne :     0-7
Identifiant constructeur :              GenuineIntel
  Nom de modèle :                       Intel(R) Core(TM) i5-10210U CPU @ 1.
                                        60GHz
    Famille de processeur :             6
    Modèle :                            142
    Thread(s) par cœur :                2
    Cœur(s) par socket :                4
    Socket(s) :                         1
    Révision :                          12
    Vitesse maximale du processeur en MHz : 4200,0000
    Vitesse minimale du processeur en MHz : 400,0000
    BogoMIPS :                          4199.88
    Drapaux :                           fpu vme de pse tsc msr pae mce cx8 a
                                        pic sep mtrr pge mca cmov pat pse36
                                        clflush dts acpi mmx fxsr sse sse2 s
                                        s ht tm pbe syscall nx pdpe1gb rdtsc
                                        p lm constant_tsc art arch_perfmon p
                                        ebs bts rep_good nopl xtopology nons
                                        top_tsc cpuid aperfmperf pni pclmulq
                                        dq dtes64 monitor ds_cpl vmx est tm2
                                         ssse3 sdbg fma cx16 xtpr pdcm pcid
                                        sse4_1 sse4_2 x2apic movbe popcnt ts
                                        c_deadline_timer aes xsave avx f16c
                                        rdrand lahf_lm abm 3dnowprefetch cpu
                                        id_fault epb ssbd ibrs ibpb stibp ib
                                        rs_enhanced tpr_shadow flexpriority
                                        ept vpid ept_ad fsgsbase tsc_adjust
                                        bmi1 avx2 smep bmi2 erms invpcid mpx
                                         rdseed adx smap clflushopt intel_pt
                                         xsaveopt xsavec xgetbv1 xsaves dthe
                                        rm ida arat pln pts hwp hwp_notify h
                                        wp_act_window hwp_epp vnmi md_clear
                                        flush_l1d arch_capabilities
Virtualization features:
  Virtualisation :                      VT-x
Caches (sum of all):
  L1d:                                  128 KiB (4 instances)
  L1i:                                  128 KiB (4 instances)
  L2:                                   1 MiB (4 instances)
  L3:                                   6 MiB (1 instance)
NUMA:
  Nœud(s) NUMA :                        1
  Nœud NUMA 0 de processeur(s) :        0-7
Vulnerabilities:
  Gather data sampling:                 Mitigation; Microcode
  Itlb multihit:                        KVM: Mitigation: VMX disabled
  L1tf:                                 Not affected
  Mds:                                  Not affected
```

FIGURE 2 – Résultat de la commande lscpu

1.1 Informations clés de l'ordinateur

Contexte d'exécution

— **Système d'exploitation** : Ubuntu installé en dual boot (natif).

— **Runtime** : Exécution native sur le matériel, sans virtualisation intermédiaire.

— **Avantages du natif** : Meilleure performance grâce à l'absence de surcharge liée à une couche de virtualisation.

Architecture et processeur

— **Architecture** : x86_64 (64 bits).

— **Modèle du processeur** : Intel(R) Core(TM) i5-10210U.

— **Fréquence** :

  — Base : 1.60 GHz.

  — Turbo Boost : 4.20 GHz.

— **Cœurs et threads** : 4 cœurs physiques, 8 threads (Hyper-Threading).

— **Importance** : Permet une parallélisation efficace avec jusqu'à 8 threads.

Cache

— **L1d** : 128 KiB (4 instances).

— **L1i** : 128 KiB (4 instances).

— **L2** : 1 MiB (4 instances).

— **L3** : 6 MiB (1 instance).

— **Importance** : Réduit la latence d'accès à la mémoire, crucial pour les simulations intensives en calcul.

2 Question 1

Nous avons attribué un processus à une portion des frames. Chaque processus traite un nombre égale de frames et sauvegarde les images correspondantes

TABLE 1 – Temps et speedup pour `movie_filter`

Nombre de processus	Temps (s)	Speedup	Commentaires
1	26.936	1.00	(version séquentielle)
2	35.674	0.76	
3	26.968	1.00	
4	23.744	1.13	

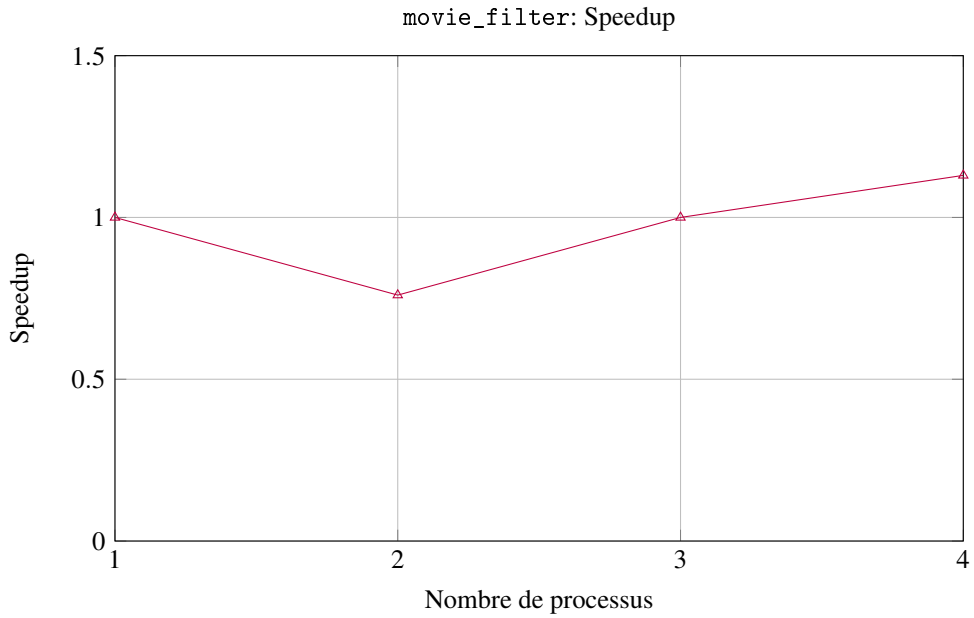

FIGURE 3 – Speedup de `movie_filter` en fonction du nombre de processus

3 Question 2

3.1 Stratégie de parallélisation

Le but est de doubler la taille des images d'une vidéo (un ensemble de N images) en leur appliquant différents filtres (filtre de flou gaussien, puis filtre de netteté).

3.2 Stratégie adoptée

Découper la liste d'images (les 37 images perroquet, par exemple) en blocs. Attribuer chaque bloc d'images à un processus. Chaque processus charge, traite et sauvegarde sa portion d'images.

Ce schéma (appelé parallélisation par distribution de tâches) est particulièrement bien adapté pour un ensemble d'images indépendant les unes des autres :

Peu de communication : chaque processus traite ses images localement, il n'y a pas besoin d'échanger des données entre processus. Charge relativement équilibrée : si le nombre d'images est suffisamment grand et que la répartition est équitable, chaque processus réalise une quantité de travail similaire.

3.3 Pourquoi c'est optimal pour ce problème ?

Les images sont indépendantes : on n'a pas besoin de transmettre les bords de l'image 1 à l'image 2, etc. Les filtres appliqués (flou gaussien, netteté) se font localement à l'échelle d'une seule image. On minimise les communications : la seule étape éventuellement commune est la création d'un répertoire de sortie et/ou la vérification des sommes MD5 (pour la validation).

4 Observations

Pour 2 processus, le temps total (bottleneck) se situe autour de 35s, ce qui est un peu plus long que la version séquentielle (26-27s), donc le speedup est < 1. Il peut y avoir plusieurs raisons : overhead de MPI, surcoût d'E/S disque, déséquilibre de charge, etc. Pour 3 processus, on retrouve un temps d'environ 26-27s, donc un speedup proche de 1. Pour 4 processus, le temps descend à 23-24s, soit un speedup autour de 1.13.

On constate donc que, dans ce cas précis, le gain n'est pas spectaculaire, voire peut parfois être inférieur à 1 si la charge E/S ou la synchronisation est trop importante. Cela dépend aussi de la taille des images, de la rapidité du disque, etc.

TABLE 2 – Temps et speedup pour `double_size`

Nombre de processus	Temps (s)	Speedup	Commentaires
1	26.600	1.00	(version séquentielle)
2	17.420	1.53	
3	17.176	1.55	
4	9.960	2.67	

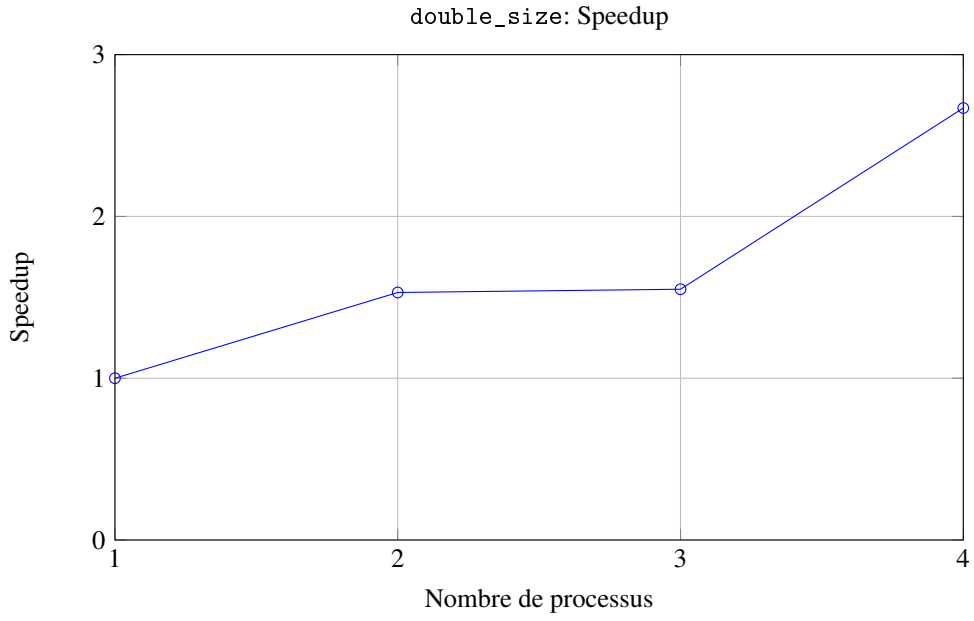

FIGURE 4 – Speedup de `double_size` en fonction du nombre de processus

5 Question 3

5.1 Stratégie de parallélisation

Ici, on cherche à doubler la taille d'une unique photo (en haute résolution) en appliquant des filtres. Le souci est d'éviter de surconsommer de la mémoire :

On ne veut pas que chaque processus charge l'intégralité de l'image, car celle-ci peut être volumineuse (plusieurs dizaines ou centaines de Mo). On découpe donc l'image en bandes (par lignes, par exemple) : chaque processus ne reçoit qu'une partie de l'image.

5.2 Pourquoi c'est adapté ?

On traite la même image, mais on peut appliquer les filtres localement sur la zone qui nous intéresse, en faisant attention aux bords (si besoin d'un recouvrement). Le volume de données manipulées par chaque processus est réduit, on reste dans la limite de la mémoire cache.

5.3 Optimalité par rapport à la question 2 ?

Dans la question 2, chaque processus traitait des images différentes. Ici, tous les processus travaillent sur différentes zones d'une seule et même image. L'approche en bandes (ou en tuiles) est nécessaire ici pour ne pas charger la totalité de l'image dans chaque processus. On peut dire qu'elle est adaptée (et souvent optimale) pour un traitement d'une seule image, tandis que la question 2 était une distribution par "liste d'images".

5.4 Parallélisation du programme double$_{size.py}$

On lit l'image sur le processus 0, on la convertit en un tableau. On découpe ce tableau en N blocs (un par processus). Chaque processus applique les filtres (flou, netteté) sur sa partie. On regroupe le résultat final (gather).

5.5 Courbe d'accélération

Version séquentielle : ≈26.600s 2 processus : ≈17.4s (max) => speedup 1.531.53 3 processus : ≈17.2s => speedup 1.551.55 4 processus : ≈9.96s => speedup 2.672.67

On observe un speedup significatif, surtout à 4 processus.

TABLE 3 – Temps et speedup pour `double_size2`

Nombre de processus	Temps (s)	Speedup	Commentaires
1	26.584	1.00	(version séquentielle)
2	23.372	1.14	
3	20.203	1.31	

5.6 Stratégie de parallélisation

Le principe est similaire (on veut doubler la taille de l'image, mais avec un filtre différent sur la composante V). On découpe toujours l'image en bandes ou en tuiles pour réduire la mémoire nécessaire par processus.

Ici, on applique le filtre F G (flou gaussien) sur H et S, et le filtre F D (différent) sur la composante V. Les canaux H, S, V peuvent être traités en parallèle, ou successivement, selon le code.

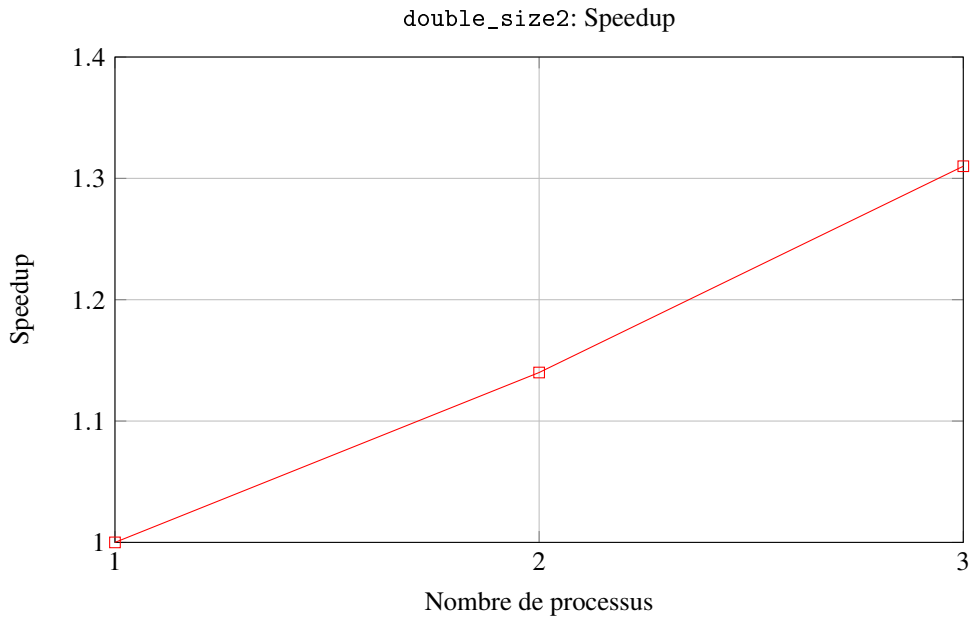

FIGURE 5 – Speedup de `double_size2` en fonction du nombre de processus

5.7 Différences par rapport à la question précédente

Différence de filtres : on n'applique plus le même masque sur la luminance, on utilise un masque F D (ou un masque 5×5 modifié). Surcoût éventuel : le masque 5X5 peut nécessiter plus de calcul qu'un 3×3, et parfois plus de lignes "fantômes" (bordures) pour la convolution. Avantage : le traitement H et S est plus léger (flou simple), le traitement V est plus précis. Inconvénient : si la communication des bords est nécessaire, on peut avoir un surcoût de synchronisation.

Version séquentielle : ≈26.584s 2 processus : ≈23.372s => speedup 1.141.14 3 processus : ≈20.203s => speedup 1.311.31

Le speedup est moins important que pour double$_{s}ize.py, cequipeuts'expliquerpar$:

Un overhead plus grand, Le masque plus grand sur la composante V (ou un autre type de filtre) qui ajoute du temps de calcul, Un possible coût plus important dans la distribution/assemblage.