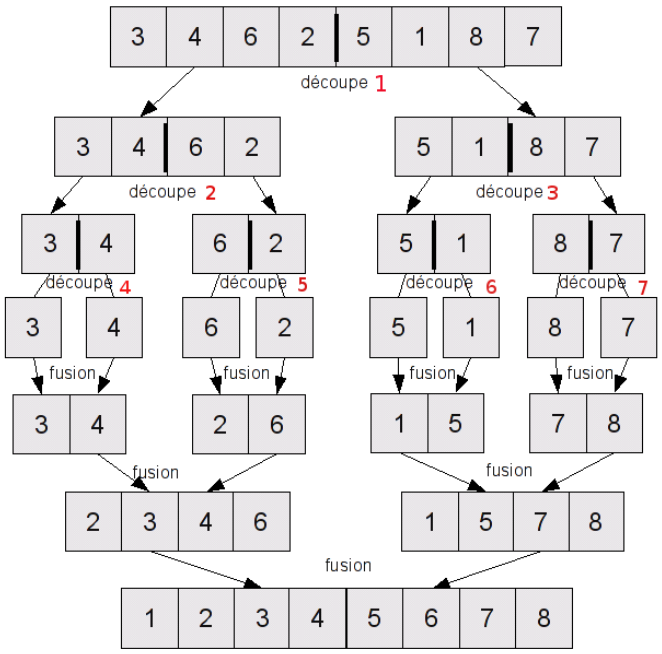
TP No1: Thread

January 27, 2025

Le **Tri fusion** est un tri efficace **O(nLog(n))**. Son efficacité est basée sur une opération **fusion** de deux listes triées cette opération étant réalisée en temps linéaire. Cet algorithme est optimal pour un type liste bien implémenté.

Nous allons étudier si la **parallélisation** de cet algorithme peut apporter des gains de performance.



1 tri à bulle

Pour commencer vous allez implémenter le **tri à bulle** dans le fichier **bubble_sort.c**.

La fonction **main** est implémentée pour réaliser une série de tests sur des tableaux aléatoires dont la taille est **doublée** à chaque itération.

La fonction de tri aura toujours la même signature pour pouvoir être exécuté dans des Threads.

```
void* sort(void* arg)
```

Le pointeur générique **void*** est utilisé pour passer les paramètres d'entré de la fonction exécutée dans un Thread.

```
struct timespec start, end;
        for(int i=2; i <= (1 << 24); i*=2 ){

                int* T=random_array(i);
                Data d= {T,0,i-1} ;

                clock_gettime(CLOCK_MONOTONIC, &start);
                // Code a mesurer ici
                bubble_sort(&d) ;

                clock_gettime(CLOCK_MONOTONIC, &end);
                // Temps d'execution
                double elapsed = (end.tv_sec - start.tv_sec) +
                     (end.tv_nsec - start.tv_nsec) / 1e9;

                // Affiche le temps d'execution sur la console
                printf("%d\t%.9f\n", i,elapsed);
        }
```

Question 1 Implémentez le tri à bulle et exécutez pour obtenir les temps cpu en fonction de la taille du tableau. Vous pouvez interrompre le traitement de votre processus avec en lui envoyant le signal **SIGINT (Ctrl+c)**.

Vous allez utiliser **gnuplot** pour tracer la courbe des temps d'exécution en fonction de la taille des tableaux à trier.

Question 2 Copiez les valeurs dans un fichier **00_bubble_sort.txt**. Exécutez gnuplot :

```
gnuplot> plot "00_bubble_sort.txt" using 1:2 with lines title "Bubble sort"
```

Question 3 Le résultat est-il conforme à la complexité de l'algorithme ?

Question 4 Pourrait-on paralléliser cet algorithme. Y-aurait-il un impacte significatif ?

2 tri fusion

Le tri fusion est un bien meilleur algorithme, il traite récursivement la partie droite et gauche d'un tableau et fusionne les deux tableaux triés obtenus.

2.1 l'algorithme

L'algorithme s'implémente de façon récursive :

```
entree : un tableau T
sortie : T triee

fonction triFusion(T, deb, fin)
  Si deb < fin
    mil = deb + (fin - deb) / 2

    #Trier les deux moitier du tableau
    triFusion(T, deb, mid)
    triFusion(T, mid, fin)

    # Fusion des sous-tableaux triees
    fusion(T,deb,mil,fin)
```

2.2 L'opération de fusion

L'opération principale de l'algorithme est **la fusion**, qui consiste à réunir deux tableaux triées en une seule en produisant un tableau(ou liste) unique.

Dans ce Tp on utilisera uniquement des pointeurs pour identifier les sous-tableaux à traiter.

Pour des raisons pratiques on dupliquera la partie droite et gauche du tableau en paramètre. On modifie ensuite le tableau à trier en ajoutant la valeur contenu dans le premier ou le second tableau.

```
Entree: T, start, mid, end
Entree/Sortie: T est modifie par la fonction

n1 <- mid - start + 1
n2 <- end - mid

Tg <- ... //recopie de la partie gauche
Td <- ... //recopie de la partie droite

Tant que i < n1 et j < n2
  Si Tg[i] <= Td[j]
    T[k] <- Tg[i]
    i<-i+1
  Sinon
    T[k] <- Td[j]
    j <- j + 1

  k <- k + 1

//copier le reste des elements
```

Copier le fichier **bubble_sort.c** dans **merge_sort_1.c**, pour chaque nouvelle implémentation pour créerez un nouveau fichier **merge_sort_2.c, merge_sort_2.c, ...**

Question 1 Implémentez le tri fusion.

Question 2 Tracez la courbe des temps d'exécution en fonction de n dans un fichier **01_merge_sort.txt**. Vérifiez que le résultat de son exécution correspond à sa complexité.

Question 3 Comment évolue la mémoire au cour de l'exécution de ce processus ?

3 Multithreading

Nous allons paralléliser l'exécution de cet algorithme en implémentant la fonction **mt_merge_sort**.

Les éléments à mettre en place sont :

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

int pthread_join(pthread_t thread, void **retval);
```

3.1 Utilisation de deux Threads

Implémentez deux Threads qui exécutent la fonction **merge_sort** tel quel.

Question 1 Testez l'impacte sur du multithreading sur le temps d'exécution. Conservez vos résultats dans un fichier **02_merge_sort_2T.txt**.

Question 2 Réalisez la même opération avec 4 thread.Conservez vos résultats dans un fichier **03_merge_sort_4T.txt**

3.2 Thread et récursivité

On peut se poser la question de **remplacer les appels récursifs par des Threads**

```
entree : un tableau T
sortie : T triee

fonction triFusion(T)
  n <- taille(T)
  si n = 1
    renvoyer T
  Sinon
    Lg <- decoupe(T,0,n/2)
    Ld <- decoupe(T,n/2+1,n-1)

    # Fusion des sous-tableaux triees
    Thread tg, td
        creer_thread(tg, Lg)
        creer_thread(td, Ld)

        attendre(tg)
        attendre(td)

    T <- fusion(tg, td)
```

Question 3 Combien de Thread seraient créés pour trier un tableau de 1024 valeurs ?

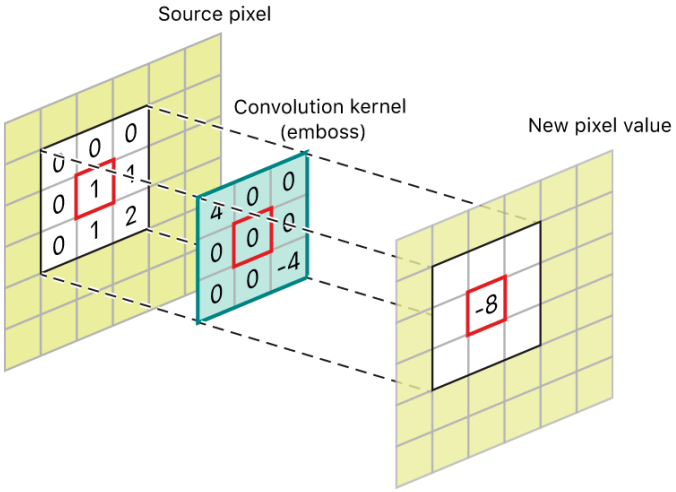
Question 4 Quel problème va se poser ? Tester ce programme et vérifiez l'impacte de cette solution.

Question 5 Modifiez la solution précédente pour prendre en compte la taille du tableau pour utiliser des Threads ou un appel récursif simple.

Question 6 Concluez sur les gains de temps possibles en utilisant la parallélisation du traitement ?

Question 7 Tracez les différents jeux de données sur le même graphique.

4 Flou gaussien



Le **flou gaussien** est une méthode de traitement d'image courant qui permet d'adoucir les images.

Le principe est d'appliquer **un noyau** qui contient une répartition gaussienne du poids de chaque pixel environnant d'un pixel dans le calcul du pixel dans la nouvelle image.

On fait donc une "moyenne" de la valeur des pixel de l'image source pour calculer les pixels de l'image cible.

Utilisez le fichier **blur_image.c** pour implémenter le flou gaussien sur le fichier **rose.pgm**

Question 7 Appliquez la même démarche que pour **le tri fusion** et montrez l'impacte de la parallélisation sur cet algorithme.