PHYSIQUE QUANTIQUE DU SOLIDE

UNIVERSITÉ DE BOURGOGNE

DÉPARTEMENT DE PHYSIQUE – L3PFA

Algorithmes de tri

Auteur :
THEODON Léo

Date : 2 avril 2019

Table des matières

Introduction

Le but de ce rapport est de comparer deux algorithmes de tri dans un contexte particulier. En effet, avant d'aller plus loin, il convient de rappeler le contexte dans lequel s'inscrit ce problème. L'objectif principal est de générer un réseau cristallin de type cubique face centrée (bien que tout autre type de réseau puisse convenir) à partir de la maille primitive de réseau de Bravais, puis de les trier par norme (distance) croissante par rapport à une origine arbitrairement fixée. C'est à ce moment qu'intervient notre algorithme de tri. Nous allons dans ce court rapport discuter de deux implémentations différentes d'un algorithme de type *tri par bulle* et proposerons éventuellement une alternative à ces deux propositions.

1 Définition du problème et méthode

Comme nous l'avons déjà mentionné, nous supposons que l'on a généré un réseau cristallin représenté par un tableau de dimension $n \times 4$ où n est le nombre d'éléments dans le réseau. Les trois premiers colonnes encodent les coordonnées spatiales de chaque nœud quand la quatrième contient la norme par rapport à l'origine arbitraire. Le calcul de cette dernière est effectué dès la génération du réseau de sorte que seule cette quatrième coordonnées est vue par notre algorithme de tri.

Notre programme sera codé en **FORTRAN** qui est un langage compilé. De fait, on limite l'influence que pourrait avoir un interpréteur ou une machine virtuelle sur les performances de notre programme. Dans note programme principal (Annexe C), on défini les constantes ainsi que les vecteurs de base du réseau de Bravais qui permettront de générer l'ensemble du réseau. On appelle ensuite la sous-routine de tri (Annexe B) au sein d'une boucle sur un entier n qui génère un réseau de taille $(2n+1)^3$ à chaque itération et effectue le tri selon l'algorithme choisi. On prendra soin de ne pas tenir compte du temps passé à générer le réseau et de ne compter que le temps passé à trier le réseau.

2 Comparaison des deux algorithmes

2.1 Tri par comparaison

Le premier algorithme mis en œuvre est un tri par comparaison. Il s'agit de partir du premier élément du tableau et de le comparer au suivant. Si le suivant est plus *grand*, alors on les permute, et on continue ainsi jusqu'à la fin du tableau en comparant toujours l'élément courant au premier élément. A la fin de la première boucle, le premier élément est le plus *petit* élément. On recommence alors du début en partant cette fois

ci du second élément, et ainsi de suite jusqu'au dernier élément du tableau.

Cet algorithme effectue par conséquent $n + (n-1) + (n-2) + ... + 2 + 1$ itérations, soit $n(n+1)/2$ itérations où n est la taille du tableau à trier. Ainsi, la complexité est en $O(n^2)$ en moyenne. Il ne s'agit par conséquent pas d'un algorithme optimal sachant que le tri par tas (*heapsort*) ou encore le tri rapide (*quicksort*) ont tous deux une complexité asymptotique en $n\ln(n)$.

2.2 Tri par peigne

Une alternative semblant plus performante est le tri par bulle. Il est possible de le mettre en œuvre (passer l'argument $tri = 2$ dans la fonction de tri) mais il s'est révélé moins efficace que le tri par comparaison dans le cas présent. Cela vient de la façon dont notre tableau est trié au départ, ce qui oblige le l'algorithme à atteindre sa complexité maximale et effectuer un grand nombre de comparaison pour une complexité temporelle en $O(n^2)$ également. Néanmoins, le tri par peigne est une alternative intéressant au tri par bulle, atteignant une complexité asymptotique optimale en $n\ln(n)$ et c'est cette approche que nous avons adopté.

2.3 Calcul des performances

Pour comparer les performances, on trace la courbe obtenue après une série de mesures représentative pour un nombre d'éléments allant de 27 à 500000 pour le tri par comparaison et de 27 à 8200000 pour le tri par peigne. On peut constater que le résultat est sans appel. Là où le temps de calcul se compte en minutes pour le tri par comparaison, on parle de milli-secondes pour le tri par peigne. C'est d'ailleurs pour cette raison que nous avons pu aller plus loin en ce qui concerne le nombre d'éléments pour ce second algorithme.

La courbe parabolique obtenue pour le tri par comparaison (Fig. 1) laisse bien apparaitre la complexité en $O(n^2)$.

En ce qui concerne la tri par peigne (Fig. 2), on reste dans le régime linéaire et il n'est pas possible de voir apparaitre la composante logarithmique. Néanmoins, notre tableau étant déjà presque entièrement trié du fait de l'algorithme de génération choisi pour le réseau, ce dernier termine très rapidement, ce qui laisse pensé que la complexité se rapproche dans ce cas du $O(n)$.

Conclusion

Le gain de performance apporté par le tri par peigne par rapport au tri par comparaison ne fait aucun doute. Néanmoins, il pourrait être intéressant d'implémenter le tri rapide

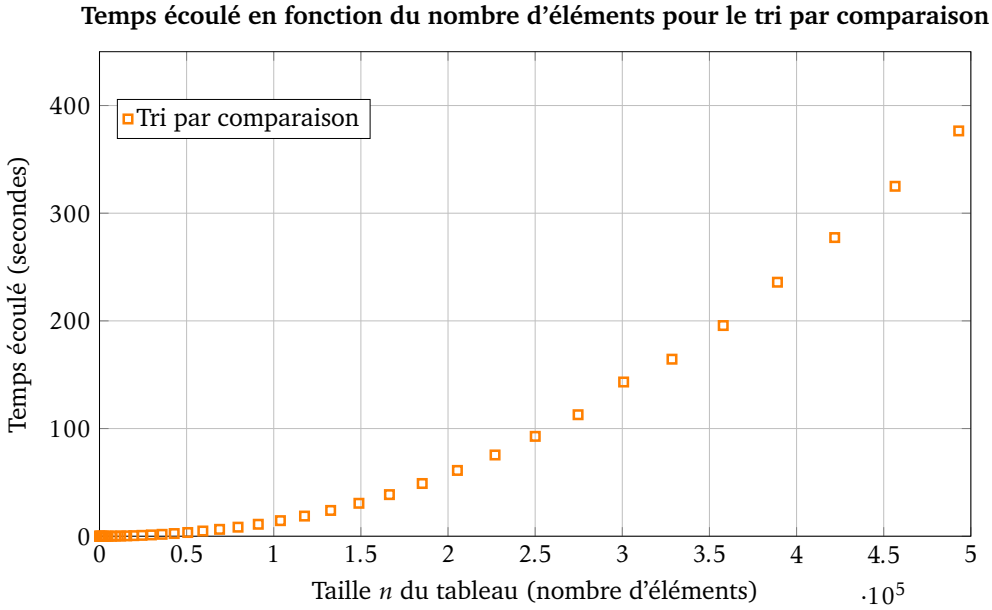

FIGURE 1: Mesures expérimentales de la complexité temporelle de l'algorithme de tri par comparaison.

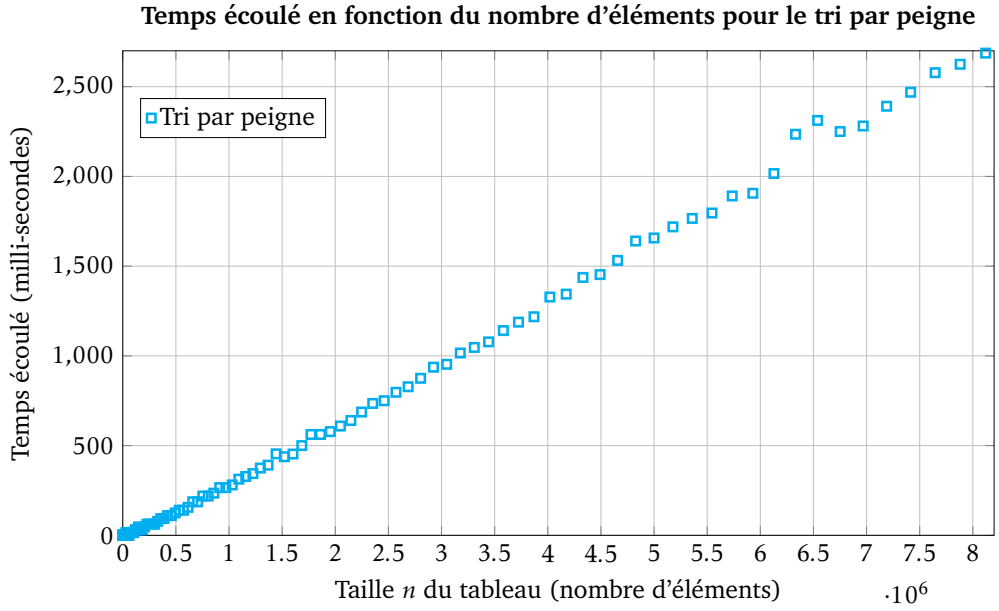

FIGURE 2: Mesures expérimentales de la complexité temporelle de l'algorithme de tri par peigne.

ou l'une de ses variantes, ou encore le tri par tas, étant donné que ce dernier algorithme est asymptotiquement optimal.

On pourrait aussi discuter de l'algorithme choisi pour la génération du réseau et discuter de l'influence de ce dernier sur la performance des algorithmes de tri. On pourrait également tester notre algorithme sur d'autre réseaux, avec éventuellement plusieurs atomes par maille primitive et encore une fois discuter de l'influence de ces paramètres sur les performances des algorithmes de tri.

3 Annexes

A Programme principal

On trouvera ci-dessous le code du programme principal appelant la sous-routine lançant le tri d'un réseau généré automatiquement.

Listing 1: Programme principal

```
Program main
    integer nmax, nat_cell, max_neighbour, n_prop, tri
    parameter (nat_cell=1, max_neighbour = 2, tri = 1, nat = 19)
    double precision, dimension(4,3) :: vu
    double precision, dimension(4,nat_cell) :: unit_cell
    integer :: n_net, n
    real :: time

    n_net = 0
    n_prop = 40

    !Definition des vecteurs unite
    vu(1,1) = 0.5d0
    vu(2,1) = 0.5d0
    vu(3,1) = 0.0d0

    vu(1,2) = 0.0d0
    vu(2,2) = 0.5d0
    vu(3,2) = 0.5d0

    vu(1,3) = 0.5d0
    vu(2,3) = 0.0d0
    vu(3,3) = 0.5d0

    !Definition des atomes primitifs
    unit_cell(1,1) = 0.d0
    unit_cell(2,1) = 0.d0
    unit_cell(3,1) = 0.d0

    !Boucle sur les algorithmes de tri
    !et recuperation du temps de calcul
    open(10, file = 'tri_1.dat', status = 'replace')
    do n = 1,n_prop
```

```
        call sort(vu, unit_cell, n, nat_cell,1,time,n_net)
        write(10,666) n, n_net, time
    end do
    close(unit=10)

    open(20, file = 'tri_3.dat', status = 'replace')
    do n = 1,n_prop
        call sort(vu, unit_cell, n, nat_cell,3,time,n_net)
        write(20,666) n, n_net, time
    end do
    close(unit=20)
    666  format(I4,' ',I4,' ',G15.6)

end program
```

B Sous routine de tri

Listing 2: Sous-routine de tri

```
subroutine sort(vu, unit_cell, n_prop, nat_cell,tri,time,n_net)
    integer :: n_net, counter, tri, n_prop, n_intervalle, nat_cell
    double precision :: buffer
    double precision, dimension(4,3) :: vu
    double precision, dimension(4,nat_cell) :: unit_cell
    double precision, dimension(6,(2*n_prop+1)**3) :: net
    logical :: echange
    real :: time

    INTEGER :: nb_ticks_initial ,nb_ticks_final ,nb_ticks_max,nb_ticks_s
! number of clock ticks of the code
    REAL :: elapsed_time  ! real time in seconds

    CALL SYSTEM_CLOCK(COUNT_RATE=nb_ticks_sec, COUNT_MAX=nb_ticks_max)

    !Generation du reseau, initialisation de net
    call net_gen(n_net,net,vu,unit_cell,n_prop,nat_cell)


    echange = .TRUE.
```

```
CALL SYSTEM_CLOCK(COUNT=nb_ticks_initial)

if(tri.eq.1) then

    !tri par comparaison
    do k=1,n_net-1
        do j=k+1,n_net
            if(net(4,k).GT.net(4,j)) then
                do i=1,4
                    buffer = net(i,k)
                    net(i,k) = net(i,j)
                    net(i,j) = buffer
                end do
            end if
        end do
    end do

elseif(tri.eq.2) then

    !tri par bulle
    do k = 1,n_net-1
        do j = 1, n_net-k
            if(net(4,j).GT.net(4,j+1)) then
                do i=1,4
                    buffer = net(i,j+1)
                    net(i,j+1) = net(i,j)
                    net(i,j) = buffer
                end do
            end if
        end do
    end do

else

    !tri par peigne
    n_intervalle = n_net
    do while(echange.OR.(n_intervalle.GT.1))
        !Initialisation de la taille de lintervalle
        n_intervalle = int(7*n_intervalle/9)
        if(n_intervalle.eq.1) then
            n_intervalle = 1
        end if
```

```
            counter = 1
            echange = .FALSE.

            do while ((n_net-n_intervalle).GT.(counter-1))
                if(net(4,counter).GT.net(4,counter+n_intervalle)) then
                    do i=1,4
                        buffer = net(i,counter)
                        net(i,counter) = net(i,counter+n_intervalle)
                        net(i,counter+n_intervalle) = buffer
                    end do
                    echange = .TRUE.
                end if
                counter = counter + 1
            end do
        end do

    end if

    CALL SYSTEM_CLOCK(COUNT=nb_ticks_final)

    nb_ticks = nb_ticks_final - nb_ticks_initial
    IF (nb_ticks_final < nb_ticks_initial) &
    nb_ticks = nb_ticks + nb_ticks_max
    elapsed_time    = REAL(nb_ticks) / nb_ticks_sec
    time = elapsed_time
    write(*,*) time


    return
end
```

C Sous routine de génération du réseau

Listing 3: Sous routine de génération du réseau

```
subroutine net_gen(n_net,net,vu, unit_cell, n_prop, nat_cell)
    integer :: n_net, nat_cell, n_prop
    double precision, dimension(4,3) :: vu
    double precision, dimension(4,nat_cell) :: unit_cell
```

```
    double precision, dimension(6,(2*n_prop+1)**3) :: net

    n_net = 0
    do i = -n_prop,n_prop
        do j = -n_prop,n_prop
            do k = -n_prop,n_prop
                !incrementation du compteur
                n_net = n_net + 1
                !Ajout de l'atome au reseau
                do m = 1,3
                    net(m,n_net) = unit_cell(m,1) &
                     + i*vu(m,1) + j*vu(m,2) + k*vu(m,3)
                    net(m,n_net) = net(m,n_net)
                end do
                !Calcul de la norme
                net(4,n_net) = sqrt(net(1,n_net)**2 &
                 + net(2,n_net)**2 + net(3,n_net)**2)
            end do
        end do
    end do

end
```