Sensibilisation à l'optimisation application à l'équation des ondes

Ce premier projet a pour objectif d'introduire les aspects liés à l'efficacité d'une implémentation **monoprocesseur**. Le problème physique étudié est celui de la **propagation d'ondes dans un milieu**. On rappellera les notions de **complexité algorithmique** ainsi que la **sensibilité de la solution au maillage**. On proposera trois implémentations d'un même schéma numérique, en `Python`, `Fortran` et `C`, puis on procédera à une **comparaison des performances** de ces langages pour un même algorithme. La dernière partie sera consacrée aux techniques d'optimisation, notamment la **vectorisation** et la **bonne utilisation de la mémoire cache**. Cela constituera une première sensibilisation des étudiants aux problématiques d'optimisation haute performance.

1 Equation d'ondes :exemples issus de la physique

1.1 Ondes électromagnétiques

Les équations de Maxwell dans le vide décrivent l'évolution de l'induction magnétique B et du champ électrique E :

$$\partial_t B + \nabla \times E = 0, \qquad \nabla \cdot B = 0, \tag{1}$$

$$\partial_t E - \frac{1}{\varepsilon} \nabla \times B = 0, \qquad \nabla \cdot E = 0, \tag{2}$$

où ε et μ désignent respectivement la permittivité et la perméabilité du vide. En combinant ces équations (en prenant la dérivée temporelle puis en utilisant les identités vectorielles classiques), on obtient :

$$\partial_{tt} B = \frac{1}{\varepsilon\mu} \nabla^2 B, \tag{3}$$

$$\partial_{tt} E = \frac{1}{\varepsilon\mu} \nabla^2 E. \tag{4}$$

Les champs électrique et magnétique satisfont donc l'équation des ondes, avec pour vitesse

$$c = \frac{1}{\sqrt{\varepsilon\mu}},$$

qui est précisément la vitesse de propagation de la lumière dans le vide.

1.2 Élasticité : la corde vibrante

Considérons une corde horizontale de densité linéique ρ uniforme. On note $u(x,t)$ le déplacement de la corde par rapport à sa position d'équilibre, et $T(x,t)$ la tension. La loi de comportement élastique est donnée par la loi de Hooke :

$$T = k\, \nabla u,$$

où k est la raideur du matériau. Sous l'hypothèse d'un mouvement purement transverse (les déplacements longitudinaux sont négligés), et en appliquant le principe fondamental de la dynamique, la seule force agissant étant la tension, on obtient l'équation différentielle suivante pour la composante verticale du déplacement :

$$\partial_{tt} u - \frac{T}{\rho} \partial_{xx} u = 0. \tag{5}$$

Des modèles similaires existent en dimension deux, par exemple pour les vibrations de la peau d'un tambour, ce qui conduit à une équation des ondes sur un domaine 2D.

1.3 Ondes acoustiques : équations de la dynamique des gaz

Les équations de la dynamique des gaz proviennent des lois de conservation de la masse et de la quantité de mouvement. Elles gouvernent l'évolution de la densité ρ et de la vitesse u :

$$\partial_t \rho + \nabla_x \cdot (\rho u) = 0, \tag{6}$$
$$\partial_t (\rho u) + \nabla_x \cdot (\rho\, u \otimes u + p(\rho)\, I) = 0, \tag{7}$$

où $p(\rho)$ est la pression, donnée par une loi de comportement. Par exemple, pour un gaz parfait adiabatique, on a la loi

$$pV^\gamma = \text{cste}.$$

Pour de faibles perturbations autour d'un état d'équilibre

$$\rho = \rho_0 + O(\varepsilon), \qquad u = 0 + O(\varepsilon),$$

on linéarise les équations pour obtenir le système :

$$\begin{cases} \partial_t \rho + \rho_0 \nabla_x \cdot u = 0, \\ \partial_t u + \dfrac{p'(\rho_0)}{\rho_0} \nabla_x \rho = 0. \end{cases} \tag{8}$$

En combinant ces deux équations, on obtient :

$$\partial_{tt} \rho - p'(\rho_0)\, \Delta \rho = 0. \tag{9}$$

Il s'agit de l'équation des ondes, décrivant ici les ondes acoustiques. On peut obtenir des modèles analogues pour les ondes longitudinales dans les matériaux solides élastiques (ondes sismiques).

2 Modèle mathématique

Nous souhaitons résoudre l'équation des ondes dans un domaine $\Omega =]0, L[\times]0, H[$. La source sonore est notée $s(x,t)$. L'inconnue est la pression $p(x,t)$, définie pour $x \in \Omega$ et $t \in [0,T]$, solution de l'équation des ondes

$$p_{tt} - c^2 \Delta p = s. \tag{10}$$

Sur le bord, nous considérons des conditions aux limites dissipatives. Soit une onde plane incidente

$$p_0(x,t) = A \exp(-i\omega t + ik \cdot x),$$

où A est l'amplitude, ω la pulsation, et k le vecteur d'onde. Rappelons que la pulsation est liée à la fréquence f par

$$\omega = 2\pi f, \qquad \lambda = \frac{c}{f}.$$

Cette fonction est solution de (10) si

$$-\omega^2 + c^2k^2 = 0, \qquad \text{soit} \qquad \omega = c|k|.$$

En dimension 2, on peut écrire

$$k = \frac{\omega}{c}\begin{pmatrix}\cos\theta\\ \sin\theta\end{pmatrix},$$

où θ est l'angle d'incidence par rapport à la paroi de normale $n = (1, 0)$. Le vecteur d'onde réfléchi est

$$k' = k - 2(k \cdot n)n = \frac{\omega}{c}\begin{pmatrix}-\cos\theta\\ \sin\theta\end{pmatrix}.$$

L'onde réfléchie s'écrit alors :

$$p_1 = rA\exp(-i\omega t + ik' \cdot x),$$

où le coefficient de réflexion $r \in [0, 1]$. L'onde totale, ie solution de (10), est la somme des ondes incidente et réfléchie :

$$p = p_0 + p_1.$$

On cherche à estimer sa dérivée normale pour en déduire une condition aux limites. Cette derniére s'écrit :

$$\frac{\partial p}{\partial n} = iA\frac{\omega}{c}e^{-i\omega t}\cos\theta\left(e^{ik\cdot x} - re^{ik'\cdot x}\right),$$

tandis que

$$p = Ae^{-i\omega t}\left(e^{ik\cdot x} + re^{ik'\cdot x}\right).$$

Sur une frontiére verticale en $(0, x_2)$,

$$\frac{\partial p}{\partial n} = i\frac{\omega}{c}\frac{1-r}{1+r}p + O(\theta^2).$$

Une condition aux limites approchée est donc

$$\frac{\partial p}{\partial n} = i\frac{\omega}{c}\frac{1-r}{1+r}\,p, \tag{11}$$

ou encore

$$\frac{\partial p}{\partial n} + \frac{1}{c}\frac{1-r}{1+r}\,p_t = 0. \tag{12}$$

Il est à noter que cette condition limite entraîne une décroissance de l'énergie acoustique

$$E = \frac{1}{2}\left(p_t^2 + |\nabla p|^2\right).$$

En multipliant (10) par p_t et en intégrant on obtient :

$$\frac{dE}{dt} = -\int_{\partial\Omega}\frac{1}{c}\frac{1-r}{1+r}(p_t)^2\,d\Gamma \le 0.$$

On voit que l'énergie est conservée si $r = 1$. Nous adoptant cette valeur pour la suite du projet car elle conduit à une condition limite reflective simple à implémenter $\partial_n p = 0$ et surtout exacte (pas de d'approximation en θ petit).

3 Approximation par différences finies en 1D

On discrétise l'intervalle avec $N > 1$ points :

$$\Delta x = \frac{L}{N-1}, \qquad x_i = i\Delta x.$$

On note l'approximation :

$$p_i^n \approx p(x_i, n\Delta t).$$

Le schéma saute-mouton en 1D s'écrit

$$\frac{p_i^{n+1} - 2p_i^n + p_i^{n-1}}{\Delta t^2} - c^2 \frac{p_{i-1}^n - 2p_i^n + p_{i+1}^n}{\Delta x^2} = 0.$$

On obtient

$$p_i^{n+1} = -p_i^{n-1} + 2(1-\beta^2)p_i^n + \beta^2\left(p_{i-1}^n + p_{i+1}^n\right), \qquad \beta = \frac{c\Delta t}{\Delta x} < 1. \tag{13}$$

On considère maintenant le traitement des conditions aux limites.

La condition (12) s'écrit en 1D :

$$p_x \approx \frac{p_1^n - p_{-1}^n}{2\Delta x} = -\frac{1}{c}\frac{1-r}{1+r}\frac{p_0^{n+1} - p_0^{n-1}}{2\Delta t}.$$

On obtient

$$p_{-1}^n = p_1^n - \frac{1}{\beta}\frac{1-r}{1+r}\left(p_0^{n+1} - p_0^{n-1}\right). \tag{14}$$

De même au bord droit :

$$p_N^n = p_{N-2}^n - \frac{1}{\beta}\frac{1-r}{1+r}\left(p_{N-1}^{n+1} - p_{N-1}^{n-1}\right). \tag{15}$$

En les réinjectant dans (13), on obtient les mises à jour de bord :

$$p_0^{n+1} = \frac{1}{1+r^*}\Big[(r^*-1)p_0^{n-1} + 2\beta^2(p_1^n - p_0^n) + 2p_0^n\Big], \tag{16}$$

$$p_{N-1}^{n+1} = \frac{1}{1+r^*}\Big[(r^*-1)p_{N-1}^{n-1} + 2\beta^2(p_{N-2}^n - p_{N-1}^n) + 2p_{N-1}^n\Big], \tag{17}$$

avec

$$r^* = \beta\frac{1-r}{1+r}.$$

3.1 Travail demandé

Implémenter ce schéma en python, fortran et C. Étudier la sensibilité de la solution au maillage. Comparer les performances des trois implémentations pour un même maillage et une même durée de simulation.

4 Approximation 2D

On ajoute la direction y :

$$\Delta y = \frac{H}{M-1}, \qquad y_j = j\Delta y.$$

On note :

$$p_{i,j}^n \approx p(x_i, y_j, n\Delta t), \qquad \beta_x = \frac{c\Delta t}{\Delta x}, \quad \beta_y = \frac{c\Delta t}{\Delta y}.$$

Le schéma saute-mouton devient

$$p_{i,j}^{n+1} = -p_{i,j}^{n-1} + 2(1-\beta_x^2-\beta_y^2)p_{i,j}^n + \beta_x^2(p_{i-1,j}^n + p_{i+1,j}^n) + \beta_y^2(p_{i,j-1}^n + p_{i,j+1}^n) \tag{18}$$

$$+ \Delta t^2\, s_{i,j}^n. \tag{19}$$

Les conditions limites suivent le même traitement que précédemment, en appliquant la formule ghost-point

$$p_{-1,j}^n = p_{1,j}^n - \frac{1}{\beta_x}\frac{1-r}{1+r}\left(p_{0,j}^{n+1} - p_{0,j}^{n-1}\right),$$

etc.

On introduit

$$\gamma = \frac{1-r}{1+r}, \qquad \alpha = \frac{1}{1+\beta_x\gamma},$$

et on obtient finalement, au bord gauche :

$$p_{0,j}^{n+1} = (1-2\alpha)p_{0,j}^{n-1} + 2\alpha(1-\beta_x^2-\beta_y^2)p_{0,j}^n + 2\alpha\beta_x^2 p_{1,j}^n \tag{20}$$

$$+ \alpha\beta_y^2(p_{0,j-1}^n + p_{0,j+1}^n) + \alpha\Delta t^2 s_{0,j}^n. \tag{21}$$

Les autres bords se traitent de manière analogue.

Cas test : propagation dans un tube 2D avec onde non plane incidente

On considère un tube bidimensionnel

$$\Omega = (0,L)\times(0,H),$$

dans lequel la pression acoustique $p(x,y,t)$ satisfait l'équation des ondes

$$p_{tt} - c^2\Delta p = 0 \qquad \text{dans } \Omega\times(0,T). \tag{22}$$

Parois horizontales (tube rigide). Les parois supérieure et inférieure du tube sont supposées parfaitement réfléchissantes (murs rigides). On impose donc des conditions de Neumann homogènes :

$$\partial_y p(x,0,t) = 0, \qquad \partial_y p(x,H,t) = 0, \qquad \forall x\in(0,L),\ \forall t>0. \tag{23}$$

Entrée du tube (onde incidente non plane). À l'entrée $x=0$, on impose une onde incidente dont le profil dépend de y :

$$p(0,y,t) = g(y)\,\sin(\omega t), \qquad y\in(0,H),\ t>0, \tag{24}$$

où $g(y)$ est une fonction non constante, par exemple une gaussienne centrée dans le tube :

$$g(y) = A\exp\left(-\frac{(y-y_0)^2}{\sigma^2}\right), \qquad y_0 = \frac{H}{2}, \tag{25}$$

avec amplitude $A>0$ et largeur $\sigma>0$. Cette excitation n'est pas plane car la valeur imposée en $x=0$ dépend de y.

Sortie du tube (extrémité ouverte). À la sortie $x=L$, on peut modéliser une ouverture vers l'extérieur par une condition limite absorbante de premier ordre (type Sommerfeld) :

$$\partial_x p(L,y,t) + \frac{1}{c}\,p_t(L,y,t) = 0, \qquad y\in(0,H),\ t>0. \tag{26}$$

Cette condition laisse sortir une grande partie de l'énergie acoustique et limite les réflexions non physiques au bord droit.

Données initiales. On peut prendre des conditions initiales nulles :

$$p(x, y, 0) = 0, \qquad p_t(x, y, 0) = 0, \qquad (x, y) \in \Omega. \tag{27}$$

Dans cette configuration, l'onde générée à l'entrée $x = 0$ se propage le long du tube, se réfléchit sur les parois horizontales (conditions de Neumann) et sort en grande partie par le bord $x = L$ grâce à la condition limite convective.

4.1 Travail demandé

Vous avez à disposition le code en fortran pour ce problème. Il vous ai demandé d'analyser la complexité algorithmique du schéma 2d et de comparer avec vos expérimenations numériques.

5 Quelques aspects de l'optmisation moni-processeur

5.1 introduction : hiérarchie mémoire et localité

Sur les architectures modernes, l'écart de performance entre le processeur et la mémoire principale est très important. Pour atténuer cet écart, plusieurs niveaux de mémoire cache (L1, L2, L3) sont interposés entre le processeur et la RAM. Ces caches sont :

— beaucoup plus rapides que la mémoire principale,

— mais de capacité limitée (quelques dizaines de Ko pour L1, puis quelques Mo pour L2/L3).

Lorsqu'une donnée n'est pas présente dans le cache au moment où le processeur en a besoin, on parle de *cache miss*. Chaque cache miss entraîne un chargement de toute une ligne de cache (*cache line*), ce qui augmente la latence et consomme de la bande passante. Pour atteindre de bonnes performances, il est donc essentiel d'écrire des algorithmes qui :

— maximisent la *localité spatiale* (données contiguës en mémoire),

— maximisent la *localité temporelle* (réutiliser rapidement les mêmes données),

— minimisent le nombre de cache misses.

Afin d'améliorer la localité mémoire, le *loop tiling* (ou *cache blocking*) est une transformation de boucle qui consiste à découper l'espace d'itération en blocs (*tiles*) de taille contrôlée. L'idée est de travailler bloc par bloc, de telle sorte que les données d'un bloc tiennent dans le cache. Considérons une boucle 1D naïve :

```
for (i = 0; i < N; i++) {
  a[i] = f(a[i]);
}
```

Si le tableau `a` est très grand, il ne tient pas en cache : à chaque passage, on va streamer les données depuis la mémoire principale, et les lignes de cache seront rapidement évincées. Avec du tiling, on réécrit :

```
for (b = 0; b < N; b += bs) {
     int i_end = min(b + bs, N);
         for (i = b; i < i_end; i++) {
            a[i] = f(a[i]);
        }
}
```

où `bs` est la taille du *bloc* (blocksize). Si `bs` est choisi de sorte que les données du bloc `a[b..b+bs-1]` tiennent dans le cache L1 (ou L2), alors :

— le coût de chargement du bloc en cache est amorti sur plusieurs opérations,

— les accès se font majoritairement à des données déjà en cache,

— on réduit fortement le nombre de cache misses.

5.2 Expérimentations : le produit matrice-matrice

Le produit matriciel est une aopération de base en HPC. et reste un très bon exemple pour illsutrer les notions de localité. Dans ce qui suit vous allez implémenter et analyser différentes versions du produit matrice-matrice carré :

$$C \leftarrow AB, \qquad A, B, C \in \mathbb{R}^{N \times N}.$$

On rappelle que le nombre total d'opérations flottantes pour cette opération est

$$\text{FLOPs}_{\text{tot}} = 2N^3.$$

5.2.1 Approche naïve et optimisation par *loop tiling*

On suppose dans toute cette section un stockage *row-major* des matrices. On considère d'abord une implémentation naïve à trois boucles, dans l'ordre `i-j-k` :

```
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        for (k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Cette version est simple, mais effectue des accès mémoire peu efficaces :
- la matrice `A` est relue N fois pour chaque ligne,
- la matrice `B` est parcourue en colonnes,
- la réutilisation locale est très faible.

Pour améliorer la localité mémoire, on introduit une version par *loop tiling* (ou *cache blocking*) :

```
for (ii = 0; ii < N; ii+=BS)
    for (jj = 0; jj < N; jj+=BS)
        for (kk = 0; kk < N; kk+=BS)
            for (i = ii; i < ii+BS; i++)
                for (j = jj; j < jj+BS; j++)
                    for (k = kk; k < kk+BS; k++)
                        C[i][j] += A[i][k] * B[k][j];
```

Cette transformation consiste à découper les trois boucles externes en blocs de taille BS. L'idée est de travailler successivement sur de petits sous-blocs de matrices qui tiennent mieux dans les mémoires rapides du processeur (cache L1/L2). Ainsi, pour chaque triplet (ii, jj, kk), on multiplie uniquement un bloc de lignes de A par un bloc de colonnes de B, et on accumule le résultat dans le bloc correspondant de C. En limitant la zone mémoire active à un sous-ensemble réduit des données, on améliore significativement la *localité spatiale* et *temporelle*, ce qui réduit les accès lents à la mémoire principale et augmente l'efficacité du calcul.

Travail demandé.

1. Implémenter les deux codes ci-dessus en C++.
2. Comparer les temps d'exécution en fonction de N.
3. Étudier l'influence de la taille de bloc `BS`.
4. Expliquer qualitativement l'effet du tiling sur la localité mémoire.

5.2.2 Analyse Roofline : rôle crucial de l'ordre des boucles

Nous souhaitons maintenant comprendre le lien entre performance, organisation des boucles, et accès mémoire. Dans cette section, on compare *deux* versions sans tiling :

— l'ordre `i-j-k` : faible localité mémoire,

— l'ordre `i-k-j` : bonne réutilisation locale.

L'objectif de ce modèle analytique est d'estimer l'*intensité arithmétique* de chaque version, c'est-à-dire le rapport entre le nombre total d'opérations flottantes effectuées et le volume de données transférées depuis la mémoire principale. Comme le noyau de calcul est identique dans tous les cas, le nombre total d'opérations effectuées reste le même : une multiplication et une addition par élément de la somme, ce qui conduit à

$$\text{FLOPs}_{\text{tot}} = 2N^3.$$

Dans la version `ijk`, l'ordre d'itération est défavorable du point de vue de la hiérarchie mémoire. À chaque itération de la boucle interne, la matrice A est parcourue par lignes tandis que la matrice B est parcourue par colonnes, ce qui induit de nombreux défauts de cache. Autrement dit, la localité spatiale et temporelle est médiocre, et l'on peut considérer que les lignes de A et les colonnes de B sont relues environ N fois depuis la mémoire principale. Dans cette approximation, le volume de données transférées se comporte comme

$$\text{Bytes}_{ijk} \approx 2N^3 \times 8,$$

où le facteur 8 correspond à la taille (en octets) d'un flottant double précision. On en déduit une intensité opérationnelle constante :

$$I_{ijk} \approx \frac{2N^3}{16N^3} = \frac{1}{8} \quad \text{FLOP/Byte}.$$

À l'inverse, la version `ikj` exploite une meilleure réutilisation des données en cache. L'ordre des boucles permet de conserver plus longtemps en cache les lignes de A ainsi que les valeurs de C, ce qui diminue fortement les transferts depuis la mémoire principale. Dans un modèle simplifié, on peut alors considérer que seulement $\mathcal{O}(N^2)$ éléments de A, B et C doivent être relus, ce qui conduit à

$$\text{Bytes}_{ikj} \approx 3N^2 \times 8.$$

L'intensité arithmétique correspondante devient alors

$$I_{ikj} \approx \frac{2N^3}{24N^2} = \frac{N}{12},$$

c'est-à-dire qu'elle croît linéairement avec N.

On voit donc que l'intensité I_{ijk} reste **faible et constante**, même lorsque la taille du problème augmente, ce qui limite le débit de calcul atteignable. En revanche, I_{ikj} **augmente avec** N, traduisant une meilleure exploitation du cache et une probabilité accrue d'être limité par la puissance de calcul plutôt que par la bande passante mémoire.

Travail demandé.

1. Implémenter les deux versions `ijk` et `ikj`.
2. Mesurer les temps et calculer la performance en GFLOP/s.
3. Estimer le nombre de FLOPs effectués.
4. Estimer le volume de données déplacées (en bytes).

5. En déduire l'intensité arithmétique

$$I = \frac{\text{FLOPs}}{\text{Bytes}}$$

   pour chaque version.
6. Placer les deux points (I, P) sur un diagramme Roofline.
7. Commenter la différence de performance observée.

6 Retour sur le code equation des ondes 2D

Dans cette derniére partie , on vous propose d'analyser le code C++ de l'équation des ondes 2D que vous avez utilisé précédemment. L'idée est de voir si pouvez optimiser ce code en utilisant les notions vues précédemment (localité mémoire, tiling, etc.).