LINMA1170 – TP 1: Langage C et structures matricielles

Mattéo Couplet, Thomas Leyssens, Prof. Jean-François Remacle

2022–2023

Cette première séance de tutorat est une mise en jambes. Vous avez tous utilisé le langage C durant le projet P3. Cette séance a pour but de rappeler certains concepts importants pour la suite.

Les bases de la compilation

Avant de (re)commencer à coder en C, il est important que vous compreniez bien comment fonctionne la compilation d'un petit projet en C. Pour cela, un **guide de compilation en C** est disponible sur Moodle. Avant de commencer le TP, prenez connaissance de ce guide et codez et compilez les exemples donnés : ce sera très utile pour la suite.

Le stockage de matrices

Le cours d'analyse numérique commence par l'étude de certains algorithmes liés aux matrices. Une matrice *dense* est une matrice ne contenant pas beaucoup de zéros. Conceptuellement, les matrices denses correspondent aux *systèmes fortement couplés*. Une matrice dense $A \in \mathbb{R}^{m \times n}$ sera donc représentée par une structure de données contenant $m \times n$ nombres réels.

Il existe différentes façons de stocker des tableaux multidimensionnels dans une mémoire linéaire telle que la mémoire vive d'un ordinateur :

- Dans le stockage ***row-major***, les éléments consécutifs d'une *ligne* de la matrice se trouvent les uns à côté des autres.
- Dans le stockage ***column-major***, les éléments consécutifs d'une *colonne* de la matrice se trouvent les uns à côté des autres.

Pour être plus concrets, considérons la matrice

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Dans le cas *row-major*, les éléments de cette matrice seront stockés en mémoire de la façon suivante

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} \end{bmatrix}.$$

Dans le cas *column-major*, les éléments de cette matrice seront stockés en mémoire de la façon suivante

$$A = \begin{bmatrix} a_{11} & a_{21} & a_{12} & a_{22} & a_{13} & a_{23} \end{bmatrix}.$$

En langage C, les tableaux multidimensionnels sont stockés en *row-major*. Les matrices peuvent être représentées par un tableau de pointeurs vers chaque ligne ou vers chaque colonne de la matrice, en fonction de la convention choisie. Il est préférable néanmoins de *stocker les éléments de la matrice dense dans un espace contigu.* Utiliser un espace contigu est nécessaire pour une interface avec BLAS (*Basic Linear Algebra Subroutines*) que nous utiliserons par la suite. Il ne faut donc pas allouer la matrice ligne par ligne (ou colonne par colonne) mais allouer un grand tableau contigu de taille $m \times n$.

Exercices

1. Le script `rowmajor.c` montre que les tableaux multidimensionnels sont bien stockés en *row-major* en C. Compilez et exécutez le programme `rowmajor` ; on vous rappelle les commandes :

```
$ gcc -c rowmajor.c                # On compile
$ gcc -o rowmajor rowmajor.o    # On link
$ ./rowmajor                    # On exécute
A = { 1.00 2.00 3.00 4.00 5.00 6.00 }
```

2. On vous donne le squelette d'un module `matrix` permettant de manipuler des matrices denses stockées contigûment en mémoire. On vous donne également un script `test_matrix.c` qui permet de tester le module. Complétez les fonctions `allocate_matrix` et `free_matrix`. Compilez le programme en linkant bien le module `matrix`.

3. Écrire un algorithme qui permet de multiplier deux matrices : soit $A \in \mathbb{R}^{m \times p}$ et $B \in \mathbb{R}^{p \times n}$, on vous demande de calculer $C = AB \in \mathbb{R}^{m \times n}$ avec

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}.$$

   Voir la signature de la fonction dans `matrix.h`. On vous donne un script `test_mult_matrix.c` pour tester votre code. Compilez un nouveau programme `test_mult_matrix`.

4. Calculez le nombre théorique d'opérations en virgules flottantes (FLOP) nécessaires pour effectuer ce produit matriciel. Une multiplication ou une addition comptent comme une opération.

5. Calculez (ou renseignez-vous sur internet) le nombre maximum d'opérations en virgule flottantes (FLOP) que votre ordinateur est capable de réaliser par seconde (FLOPS). L'équation permettant de calculer un FLOPS est :

$$\text{FLOPS} = \text{cœurs} \times \text{fréquence} \times \frac{\text{FLOP}}{\text{cycle}}.$$

   En 2022, la plupart des microprocesseurs sont capables de réaliser plusieurs opérations en virgule flottante par cycle (extensions SIMD, Single instruction, multiple data) qui permettent de chaîner (vectoriser) des différentes étapes d'un FLOP. Par exemple, la norme AVX2 (jeu d'instructions de l'architecture x86 d'Intel et AMD, proposé par Intel en mars 2008) permet théoriquement de faire 4 FLOP par cycle en double précision et 8 FLOP par cycle en simple précision.

6. Calculer le temps d'horloge (*wall clock time*) pour effectuer le produit de deux matrices remplies avec des nombres aléatoires (comme dans `test_matrix` avec $m = 1000$, $p = 2000$ et $n = 3000$ et comparer ce temps avec le temps théorique que votre ordinateur aurait du prendre pour effecture ce produit.

7. Le temps d'horloge que vous obtenez est largement supérieur au temps théorique. Vous n'utilisez donc qu'une toute petite partie des FLOPS disponibles. Comment expliquez vous cette différence ?

8. **Pour les motivés**. Il existe un standard international appelé BLAS[1]. La fonction `dgemm`[2] permet d'effectuer un produit de matrices en double précision. Vérifier que votre ordinateur possède bien les librairies BLAS et effectuez le produit de matrices avec BLAS. Les performances sont-elles meilleures ?

[1] https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

[2] http://www.netlib.org/lapack/explore-html/d1/d54/group___double___blas___level3_gaeda3cbd99c8fb834a60a6412878226e1.html