LINMA1170 – TP1: Python, Numba et complexité

Mattéo Couplet, Gatien Dony

2023–2024

Cette première séance de tutorat est une mise en jambes. Vous êtes maintenant familiers avec le langage Python, même s'il s'avère qu'il n'est pas toujours le plus efficace. Nous verrons dans cette séance comment utiliser Numba pour accélerer des codes numériques en Python. Nous verrons aussi comment analyser la complexité d'un code numérique.

Les outils

Nous vous recommandons d'utiliser Anaconda, qui est une **distribution Python** particulièrement adaptée au calcul scientifique, contenant déjà les paquets dont vous aurez besoin. Suivez les instructions d'installation en fonction de votre système d'exploitation.

Vous aurez besoin d'un **éditeur de code**. Nous recommandons Visual Studio Code (à ne pas confondre avec Visual Studio, qui est un environnement de développement !), mais vous êtes libres d'utiliser un autre éditeur si vous le souhaitez.

Dans VSCode, écrivez un premier script Python `tp1.py` où vous importez les paquets utiles :

```python
import time         # pour mesurer la performance de nos algorithmes
import numpy as np  # pour gérer des tableaux n-d
import numba        # pour accélérer des bouts de code critiques

print('Hello, world!')
```

Ouvrez un terminal, activez l'environnement virtuel `base` d'Anaconda, puis exécutez votre script :

```
$ conda activate base
$ python tp1.py
Hello, world!
```

Premiers pas avec ~~Pumba~~ Numba

Numba est un compilateur "*just-in-time*" qui peut, au moment de l'exécution, compiler des morceaux de votre code Python en langage machine extrêmement rapide. Il vous permettra d'écrire des codes élégants (proches du pseudo-code) et très efficaces sans recourir à une minutieuse vectorisation.

Dans cette séance, on vous propose de découvrir Numba au travers d'un algorithme numérique très simple.

1. Étant donné deux matrices $A \in \mathbb{R}^{m\times n}$ et $B \in \mathbb{R}^{n\times p}$, écrivez une function `mult(A,B)` pour calculer le produit matriciel $C = AB \in \mathbb{R}^{m\times p}$. Et ce, sans utiliser de fonction NumPy ou l'opérateur `@`.

2. Générez deux matrices A et B de tailles $m = n = p = 300$ et mesurez le temps que prend votre algorithme pour calculer C, en utilisant la fonction `time.time()`. C'est lent...

3. Ajoutez un décorateur Numba à votre fonction :

   ```
   @numba.jit(nopython=True)
   def mult(A, B):
       # votre code
   ```

   Ici, `jit` signifie *just-in-time* et l'option `nopython` indique que le code peut être entièrement compilé sans utiliser l'interpréteur Python, ce qui permet d'obtenir les meilleures performances. Testez à nouveau votre code : pas mal, non ?

   Attention : la première fois que vous appelez une fonction avec un décorateur Numba, la fonction est compilée à la volée, ce qui peut prendre un certain temps. Les exécutions suivantes seront donc plus rapides !

4. Un atout de Numba est qu'il permet de paralléliser votre code très facilement. Mettez l'option `parallel=True` dans le décorateur, et, dans la boucle que vous souhaitez paralléliser, remplacez `range` par `numba.prange`. Vous devriez observer un speedup correspondant au nombre de cœurs dans votre CPU.

Analyse de complexité et de performance

Pour tout algorithme que vous implémenterez dans vos devoirs, vous devrez analyser sa complexité temporelle. On vous propose de faire cette analyse pour l'algorithme que vous venez d'implémenter.

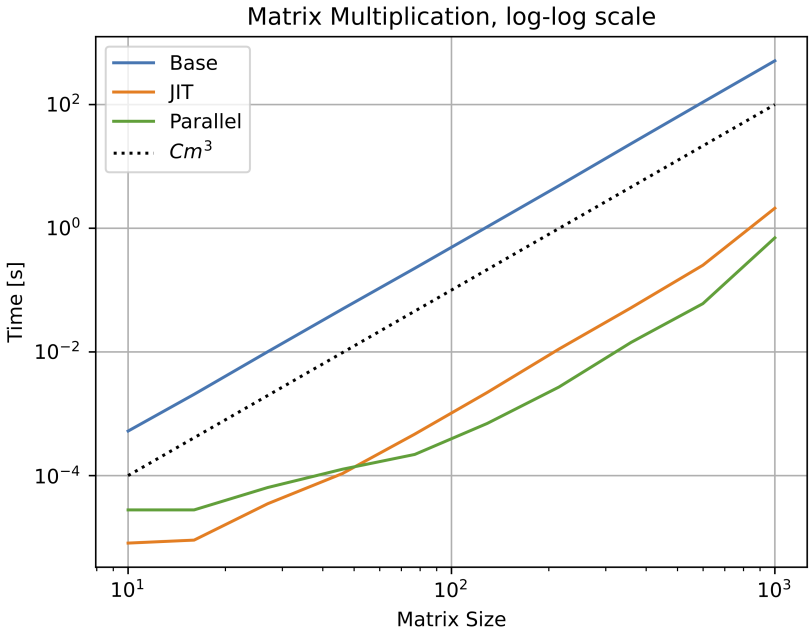
1. Quelle est la complexité temporelle de votre algorithme ?
2. Mesurez le temps d'exécution pour une série de matrices de taille croissante, et plottez un graphe taille-temps avec `matplotlib.pyplot.plot`.
3. Il est très difficile de deviner la complexité uniquement sur base de ce graphe. Quel meilleur choix pouvez-vous faire pour que la complexité

puisse se lire sur le graphique ? Montrez mathématiquement pourquoi. Voyez ci-dessous le genre de figure qui est attendue.

4. La réponse à la question précédent est valable pour toute complexité *polynomiale*. Quel choix auriez-vous fait pour une complexité exponentielle, par exemple $O(2^n)$, et pourquoi ?
5. Calculez le nombre théorique d'opérations en virgule flottante (*FLoating point OPerations, FLOP*) nécessaires pour effectuer ce produit matriciel.
6. Calculez (ou renseignez-vous sur internet) le nombre maximum d'opérations en virgule flottante que votre ordinateur est capable de réaliser par seconde (FLOPS). L'équation permettant de calculer ce FLOPS est

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

7. Quel est le temps théorique nécessaire pour une multiplication matricielles avec $n = m = p = 1000$? Comparez au temps que obtenez. Comment expliquer la différence ?



Ressources

A ~5 minute guide to Numba