TP 1 : Utilisation d'une machine à distance – BLAS – Évaluation de la performance

L'objectif de cette séance est double :

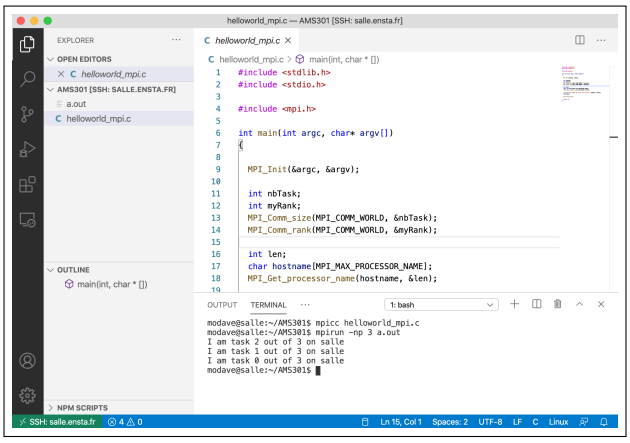
1. Installer les outils qui seront nécessaires pour le cours, et se familiariser avec un environnement pour travailler à distance sur une machine de calcul de l'école;
2. Evaluer les performances d'une routine en terme de temps de calcul (en sec) et de débit arithmétique (en GFLOP/sec), et comparer les performances d'une routine de librairie scientifique (BLAS) et d'une routine écrite soi-même.

Les codes sont disponibles sur le site Internet du cours : https://sim203.pages.math.cnrs.fr/

Partie 1 – Installation des outils et connexion à distance

Tâche 1.1. Installation de VS Code sur sa machine personnelle

Pour travailler à distance sur les machines de l'école, il est recommandé d'utiliser une version récente du programme VS Code avec l'extension "Remote SSH". Vous pourrez alors parcourir/éditer vos codes à distance, ainsi que les compiler/exécuter à distance grâce à un terminal intégrée dans l'interface.



1. Installer VS Code sur votre machine personnelle :
   https://code.visualstudio.com/
2. Installer l'extension "Remote SSH" :
   https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh

Tâche 1.2. Connexion à distance à une machine de l'ENSTA

1. Dans VS Code, cliquer sur l'icône  du menu de droite pour accéder à l'interface de connexion à distance. Dans la colonne de droite, vous verrez apparaître les différentes connexions *ssh* qui sont déjà configurées sur votre machine.
2. Dans l'interface de connexion, ajouter une nouvelle connexion *ssh* en cliquant sur le symbole +, qui apparait lorsque le curseur se trouve sur la colonne de gauche. Voici la ligne à donner :

   ```
   ssh -J IdENSTA@relais.ensta.fr IdENSTA@MACHINE.ensta.fr
   ```

   Dans cette commande, `IdENSTA` doit être remplacé par votre identifiant ENSTA, et `MachineENSTA` doit être remplacé par le nom de la machine qui vous sera attribué pour le cours.
3. Tester la connexion.

Tâche 1.3. **Accéder au outils Intel sur la machine de l'ENSTA**

Dans le cadre du cours, les compilateurs Intel seront utilisés. Ils sont déjà installés sur les machines de l'ENSTA, mais quelques manipulations sont nécessaires pour y avoir accès.

1. Ajouter les commandes suivantes à la fin du fichier `.profile` dans votre répertoire principal *(créez ce fichier si il n'existe pas déjà)* :

```
source envintel
export PATH=/auto/appy/ensta/pack/parallel2020.4/bin/:$PATH
export LD_LIBRARY_PATH=/auto/appy/ensta/pack/parallel2020.4/mkl/lib/intel64:$LD_LIBRARY_PATH
```

2. Exécuter la commande suivante : `source .profile`
3. Tester l'accès au compilateur en exécutant la commande suivante : `icpc --version`

Remarques. **Quelques commandes utiles lorsque vous êtes connectés sur machine ...**

- Pour télécharger un fichier depuis Internet : `wget https://blablabla`
  Dans cette commande, `https://blablabla` doit être remplacé par l'adresse du fichier à télécharger.
- Pour décompresser une archive : `tar -xf archive.tar`
  Les fichiers contenus dans l'archive seront déposés dans le dossier courant, ou dans un dossier déposé dans le dossier courant.
- Pour vérifier qui est connecté à la machine : `who`
- Pour vérifier les programmes qui tournent sur la machine : `top`

Partie 2 – Evaluation de la performance

La commande `icpc` correspond au compilateur propriétaire `C++` d'Intel. On considère les routines axpy/gemv/gemm de la librairie propriétaire mkl d'Intel. Des informations techniques sur ces routines se trouvent aux liens ci-dessous :

`cblas_?axpy`	(lien cliquable)	$\mathbf{y} = \alpha\mathbf{x} + \mathbf{y}$	BLAS-1
`cblas_?gemv`	(lien cliquable)	$\mathbf{y} = \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$	BLAS-2
`cblas_?gemm`	(lien cliquable)	$\mathbf{C} = \alpha\mathbf{A}\mathbf{B} + \beta\mathbf{C}$	BLAS-3

Dans le nom des routines, `?` doit être remplacer par `s` ou `d` pour des calculs en précision simple ou double. Les tableaux stockés sont alors de type `float` ou `double`.

Pour utiliser ces routines, il faut :
- inclure le fichier d'entête avec `#include <mkl.h>` *(à ajouter au dessus du programme)*,
- relier la librairie avec l'option de compilation `-mkl` *(à ajouter à la ligne de compilation)*,
- utiliser les fonctions dont les entêtes sont décrites sur les pages référencées ci-dessus.

Un exemple d'utilisation pour la routine `cblas_dgemm` est fourni dans le fichier `dgemmMKL.cpp`. La commande pour compiler ce programme se trouve au début du fichier.

Remarques : Lorsque vous vous connectez à une machine pour évaluer des temps de calcul, vérifiez que vous êtes seuls/seules dessus avec la commande `who`.

Tâche 2.1. **Compréhension et vérification du code**

La première étape, en programmation, est toujours de vérifier que le code de calcul donne bien le résultat attendu. Il est inutile d'optimiser un code qui ne donne pas le bon résultat, ou qui est mal compris.

Sur un cas de base dont vous connaissez la solution, vérifiez que le code de calcul `dgemmMKL.cpp` donne bien le résultat attendu. Cela peut se faire sur un petit cas *(ex. matrices 3×3)* avec une solution calculée à la main ou avec un logiciel de référence *(ex. matlab)*.

Les entrées de la fonction dgemm servent à indiquer si les matrices sont stockées par ligne ou par colonne, à indiquer leurs dimensions (m, n, k) ou la dimension de l'espace de stokage (lda, ldb, ldc).

Tâche 2.2. Évaluation du temps de calcul

Pour pouvoir évaluer la performance de calcul, il faut évaluer correctement le temps de calcul. Dans le cas présent, on s'intéresse au temps associé à la routine dgemm.

Plusieurs commandes sont possibles pour évaluer un temps de calcul. On considère les fonctions `clock()`, `dsecnd()` et `chrono::high_resolution_clock::now()`. La première fonction provient de la bibliothèque standard `C` (fichier d'entête `cmath` – documentation), tandis que les autres proviennent de la librairie mkl (fichier d'entête `mkl.h` – documentation) et la libraire standard std::chrono (fichier d'entête `chrono` – documentation).

Ces fonctions sont implémentées dans le fichier `dgemmMKL.cpp`. Décommentez-les et testez-les sur un cas demandant un temps de calcul significatif *(c'est-à-dire plusieurs secondes)*.

Remarques :

- *Il se peut que la première commande (`clock()`) donne un temps de calcul environ 2, 4 ou 8 fois supérieur aux autres (et à la réalité). Sur certaines implémentations, cette commande donne le temps cumulé correspondant aux différents cœurs utilisés. Le facteur que vous observez correspond donc au nombre de cœurs utilisés.*
- *Si le temps de calcul d'une opération est trop petit, les commandes ne permettront pas de donner un temps fiable. Dans ce cas de figure, il vaut mieux effectuer l'opération un grand nombre de fois (grâce à une boucle `for`), calculer le temps de calcul sur l'ensemble de la boucle, et diviser le temps obtenu par le nombre de fois que l'opération a été effectuée.*

Tâche 2.3. Évaluation de la performance arithmétique

La performance arithmétique est une mesure du taux de calcul du processeur pour une implémentation d'un algorithme donné. On le calcule en divisant le nombre d'opérations à virgule flottante effectuée par le temps de calcul.

Modifiez le code pour évaluer de manière automatique le débit arithmétique (GFLOP/s) de la routine dgemm pour des matrices carrées $n \times n$ de différentes tailles.

- Calculez le quantité théorique d'opérations à virgule flottante N_{FLOP}. En pratique, on ignore les opérations sur les entiers, seules les opérations sur les nombres à virgule flottantes sont considérées.
- Modifiez le code pour effectuer l'opération un nombre suffisant de fois pour avoir un temps de calcul moyen t_{moyen} fiable, et estimer le débit arithmétique avec la formule $N_{\text{FLOP}}/t_{\text{moyen}}$.
- Ajoutez une boucle externe pour calculer et afficher le débit arithmétique pour des matrices de différentes tailles.

Générez, comparez et analysez les figures du nombre d'opérations à virgule flottante N_{FLOP} et du débit arithmétique en fonction de la taille n.

Tâche 2.4. Performance de votre propre routine

Vient le moment de l'humilité!

Écrivez, validez et optimisez un code de calcul qui effectue une opération dgemm pour des matrices carrées $n \times n$ (sans utiliser de librairie). Sur quelques tailles de matrices, comparez les temps de calcul que vous obtenez avec ceux obtenus avec la routine de la librairie mkl.

Pour aller plus loin ...

Réalisez des codes similaires pour les opérations daxpy et dgemv. Comparez les performances des différentes opérations.

Références

- Developer Reference for Intel oneAPI MKL (lien cliquable)
- Developer Reference for Intel oneAPI MKL – BLAS and Sparse BLAS Routines (lien cliquable)
- Developer Reference for Intel oneAPI MKL – Support functions (lien cliquable)