Mini-Projet No. 2

PRÉ-REQUIS

- Lire les chapitre 1 à 5 dans [2].
- Compléter les labs 5 et 6.
- Ajouter « cl.exe » dans les variables d'environnement si vous utiliser votre ordinateur personnel (voir Annexe A)

APPROXIMATION DE L'INVERSION MATRICIELLE PAR SÉRIES DE NEUMANN (NSA) [1]

L'inversion de matrices de grandes tailles est une opération prédominante dans de multiples domaines d'ingénierie, notamment dans le domaine des télécommunications avec le massive MIMO (Multiple Input Multiple Output). Afin de réduire la complexité de calcul de $\mathbf{A}^{-1}$ par rapport aux méthodes d'inversion directes (et exactes), il est possible d'utiliser l'approximation des séries de Neumann (NSA).

Premièrement, on décompose la matrice **A** par sa diagonale **D** et ses éléments hors diagonale **E** où **A=D+E.** ensuite on approxime l'inverse de **A** selon (1)

$$\boldsymbol{A}^{-1} \cong \sum_{\ell=0}^{L-1} (-\boldsymbol{D}^{-1}\boldsymbol{E})^{\ell}\, \boldsymbol{D}^{-1}$$

$$= \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1}\boldsymbol{E}\boldsymbol{D}^{-1} + (\boldsymbol{D}^{-1}\boldsymbol{E})^{2}\boldsymbol{D}^{-1} + \textstyle\sum_{\ell=3}^{L-1}(-\boldsymbol{D}^{-1}\boldsymbol{E})^{\ell}\, \boldsymbol{D}^{-1} \qquad (1)$$

La complexité de calcul des raisonnable lorsque *L* est inférieur à 3. En d'autres mots,

$$\boldsymbol{A}^{-1} \cong \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1}\boldsymbol{E}\boldsymbol{D}^{-1} + (\boldsymbol{D}^{-1}\boldsymbol{E})^{2}\boldsymbol{D}^{-1} \qquad (2)$$

Proposez une implémentation Cuda efficace de l'équation (2) en utilisant le tiling [Chap. 4, 2] et les techniques d'accès coalescent de données [Chap. 5, 2].

L'implémentation se limitera à des matrices carées de tailles 64x64. L'annexe B fourni un simple code MATLAB pouvant générer une matrice diagonalement dominante **A** que vous utiliserez.

HINTS

1. Vous avez appris l'addition matricielle sur Cuda au laboratoire 5 que vous pouvez réutiliser dans le projet.
2. Le laboratoire 6 a démontré l'implémentation de la multiplication matricielle avec et sans tiling.
3. Référez-vous au chapiter 5 pour le tiling et au chapitre 6 pour la coalescence des données.

4. Validez le résultat de l'implémentation en comparant avec celui effectuer sur MATLAB.
5. Profilez vos kernels à l'aide **nvprof** et **NSight** afin de connaître leurs performances.
6. Discutez de l'efficacité de vos kernels au niveau de la vitesse d'exécution ainsi que l'utilisation des ressources (mémoire partagée, registres, etc). Discutez également de l'efficacité des accès mémoire, l'efficacité de votre algorithme de tiling ainsi que la coalescence des données.

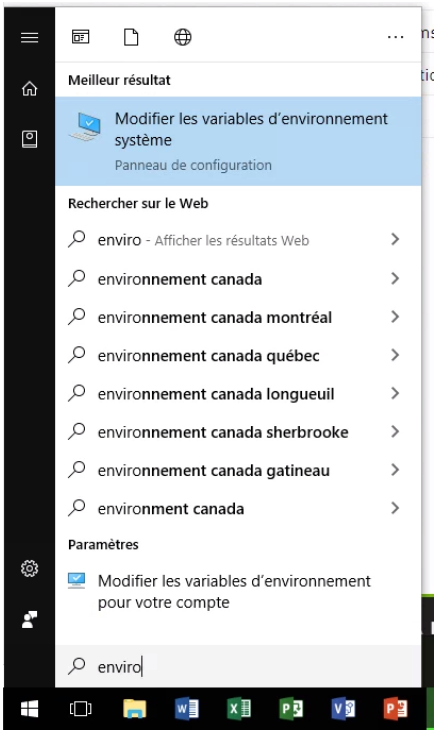
REFERENCES

[1] Wu, M., Yin, B., Wang, G., Dick, C., Cavallaro, J.R., & Studer, C. (2014). Large-scale MIMO detection for 3GPP LTE: algorithms and FPGA implementations. IEEE Journal of Selected Topics in Signal Processing, 8(5), 916–929.

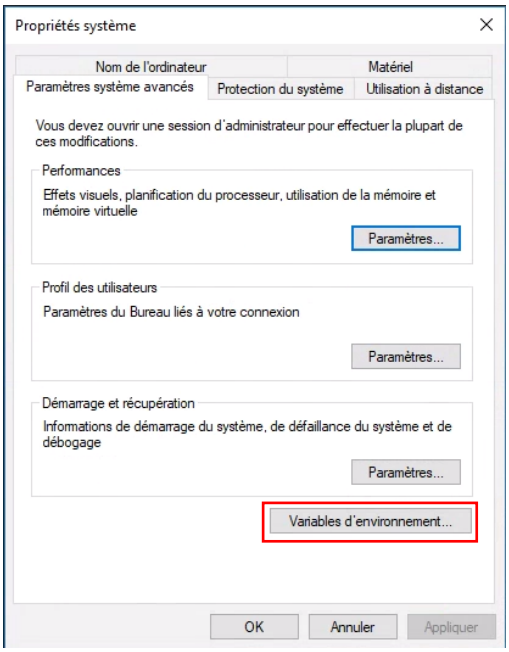
[2] D. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors- A Hands-on Approach*, 3rd Edition, Morgan Kaufmann, 2017. (Elsevier Science & Technology, ISBN: 978-0-12-811986-0).

ANNEXE A

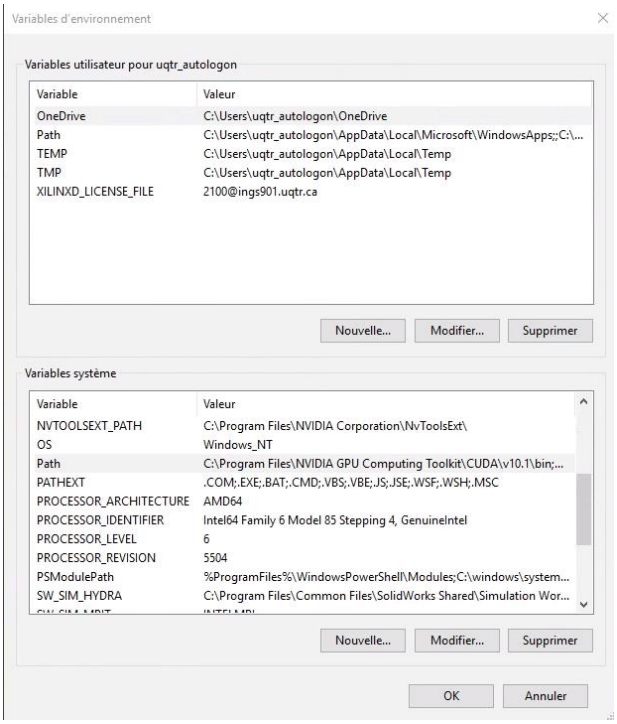
1. Add “cl.exe” to System Environment Variables
   1.1.



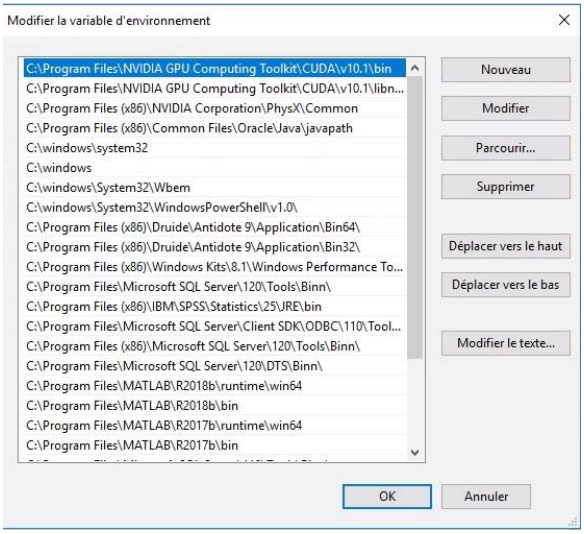
   1.2. Click on Variables d’environnement



   1.3. Select Path and click Modifier



1.4. Click on Nouveau and add this path C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin



1.5. Click ok to close the 3 windows

ANNEXE B

```
% This is a simple code to generate a diagonally dominant matrix A
% It implement Neumann series approximations of orders 2, 3 and 4
clc
clear all
close all

% Matrix dimention 64x64
A_dim=64;
H_dim=16;

% Generating a tall matrix H such that A=H'*H is diagonally dominant matrix
H=randn(H_dim*A_dim,A_dim);
% A=ceil(H'*H/(H_dim*A_dim)*2^(0));
A=(H'*H/(H_dim*A_dim)*2^(0));

% Direct (exact) matrix inversion
Ainv=inv(A);

% Isolate diagonal and off-diagonal parts of A
D=diag(diag(A));
E=A-D;
Dinv=inv(D);

% Computing Neumann series approximation of order 2, 3 and 4
Ainv2=Dinv-Dinv*E*Dinv;
Ainv3=Dinv-Dinv*E*Dinv+Dinv*E*Dinv*E*Dinv;
Ainv4=Dinv-Dinv*E*Dinv+Dinv*E*Dinv*E*Dinv-Dinv*E*Dinv*E*Dinv*E*Dinv;

% Approximate inversion errors for comparaison
Error2=norm(Ainv/norm(Ainv)-Ainv2/norm(Ainv2),'fro');
Error3=norm(Ainv/norm(Ainv)-Ainv3/norm(Ainv3),'fro');
Error4=norm(Ainv/norm(Ainv)-Ainv4/norm(Ainv4),'fro');
```