

Enoncé des travaux pratiques du cours OpenMP

1 – Description

Les travaux pratiques se dérouleront sur les nœuds **vargas** (grappes de 3584 coeurs IBM SP6) dans le répertoire **\$WORKDIR/OpenMP_tp**. Ils sont constitués de neuf exercices indépendants **tp1**, **tp2**, ..., **tp9**. Chaque exercice se trouve dans un répertoire contenant systématiquement un fichier **Makefile** pour la compilation et l'édition de liens, un fichier **batch.sh** pour la soumission en traitement par lot, un ou plusieurs **fichiers sources Fortran à compléter**.

- ➡ Dans le répertoire **sans_indications_openmp**, on trouve les sources du code séquentiel.
- ➡ Dans le répertoire **avec_indications_openmp**, on vous aide en vous indiquant explicitement les endroits où il faut insérer les directives OpenMP.
- ➡ Enfin, le répertoire **solution** contient une solution de l'exercice à ne consulter, bien entendu, qu'une fois vos forces épuisées.

Remarques générales

- ➡ Taper la commande `gmake mono`, pour une compilation séquentielle.
- ➡ Taper la commande `gmake para`, pour une compilation parallèle avec interprétation des directives `OpenMP`.
- ➡ Taper la commande `gmake clean` pour effacer les fichiers objets et core ou `gmake cleanall` pour les fichiers objets, core et exécutables.
- ➡ Tapez éventuellement la commande `lsubmit batch.sh`, pour une soumission en *batch*. Celle-ci inclut une exécution séquentielle et parallèle sur 2, 4, 6 et 8 tâches. **Attention**, les exécutables monoprocesseurs et parallèles doivent avoir été générés préalablement. La commande `Qstat` permet de suivre l'évolution du travail soumis. Une fois le travail terminé normalement, le résultat de l'exécution sera dans un fichier dont le nom possède le suffixe `.res`.

Bon courage...

2 – tp1 : produit de matrices

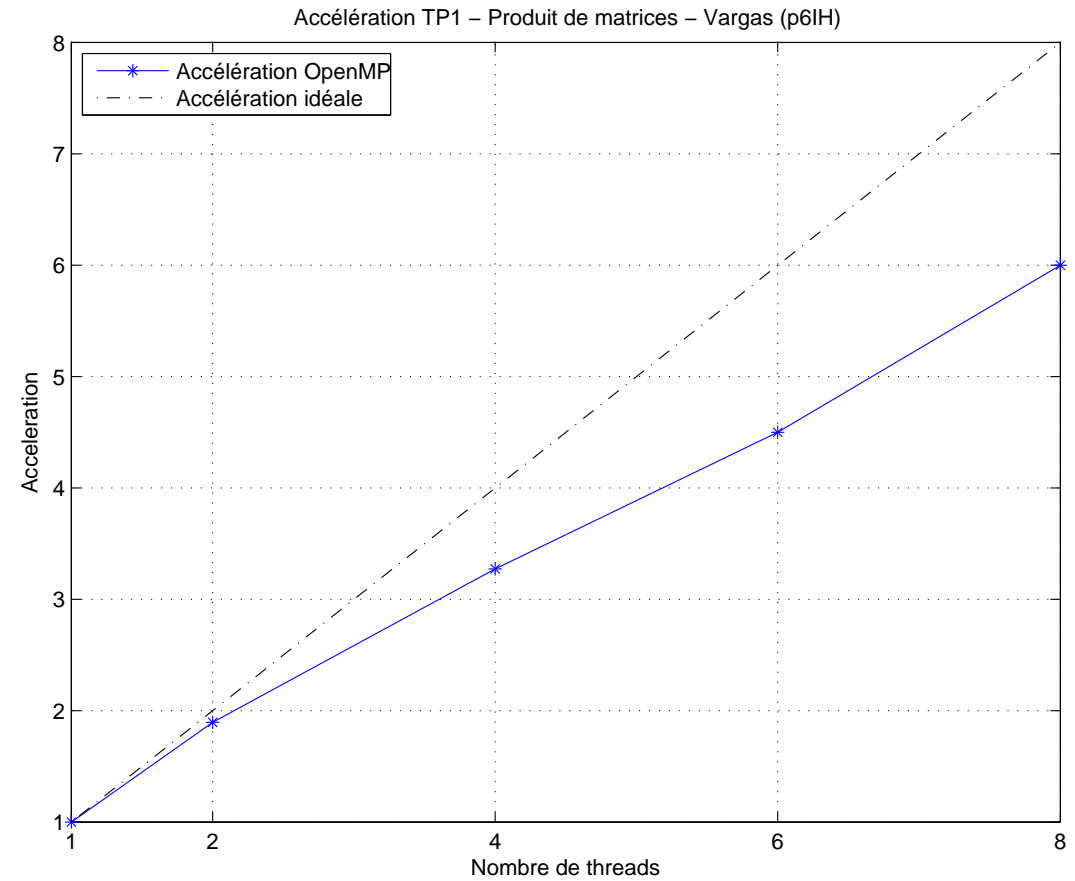
Le code, contenu dans le fichier `prod_mat.f90`, calcule le produit de matrices :

$$C = A \times B$$

Dans cet exercice, il s'agit :

1. d'insérer les directives OpenMP appropriées dans le fichier `prod_mat.f90`.
2. d'analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`). Ne pas hésiter à tester les différents modes (**STATIC**, **DYNAMIC**, **GUIDED**) de répartition des itérations d'une boucle, ainsi qu'à faire varier la taille des paquets.
3. de tracer les courbes d'accélération obtenues.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



3 – tp2 : méthode de Jacobi

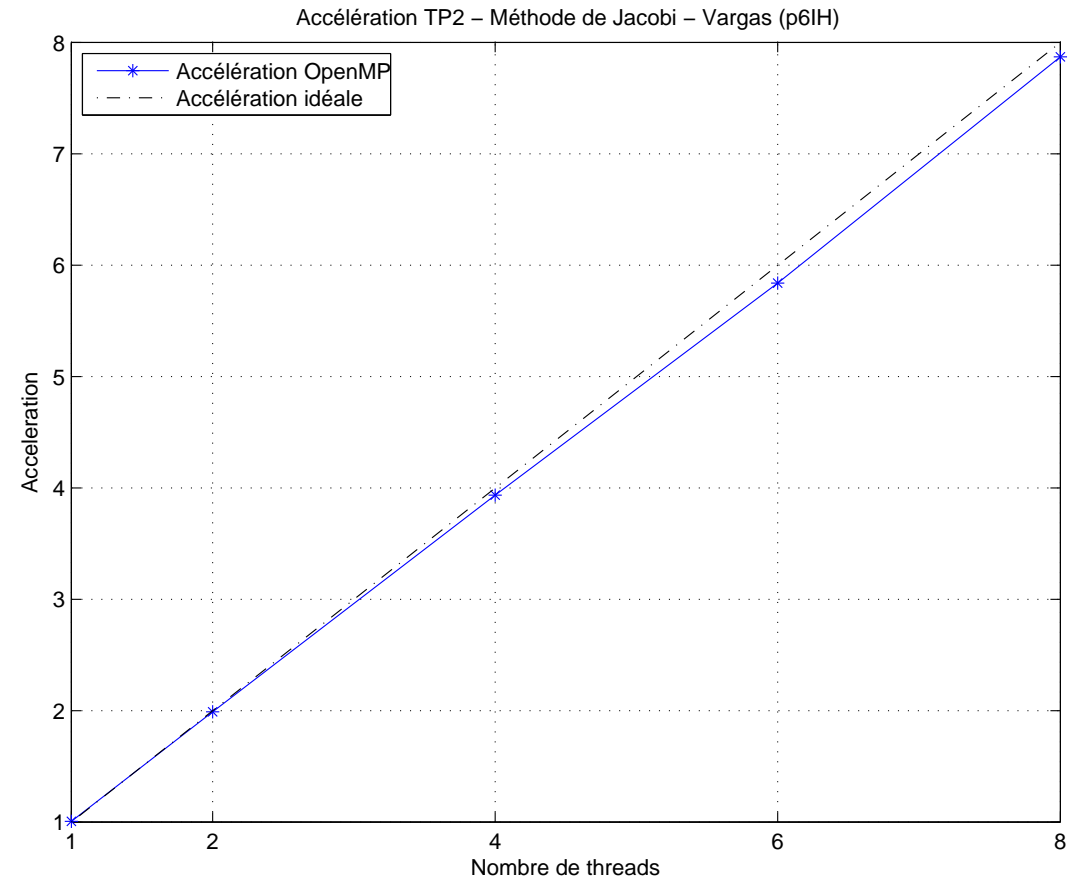
Le programme, contenu dans le fichier `jacobi.f90`, résout un système linéaire général

$$A \times x = b$$

par la méthode itérative de JACOBI.

1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans le fichier `jacobi.f90`.
2. Analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`).
3. Tracer les courbes d'accélération obtenues.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



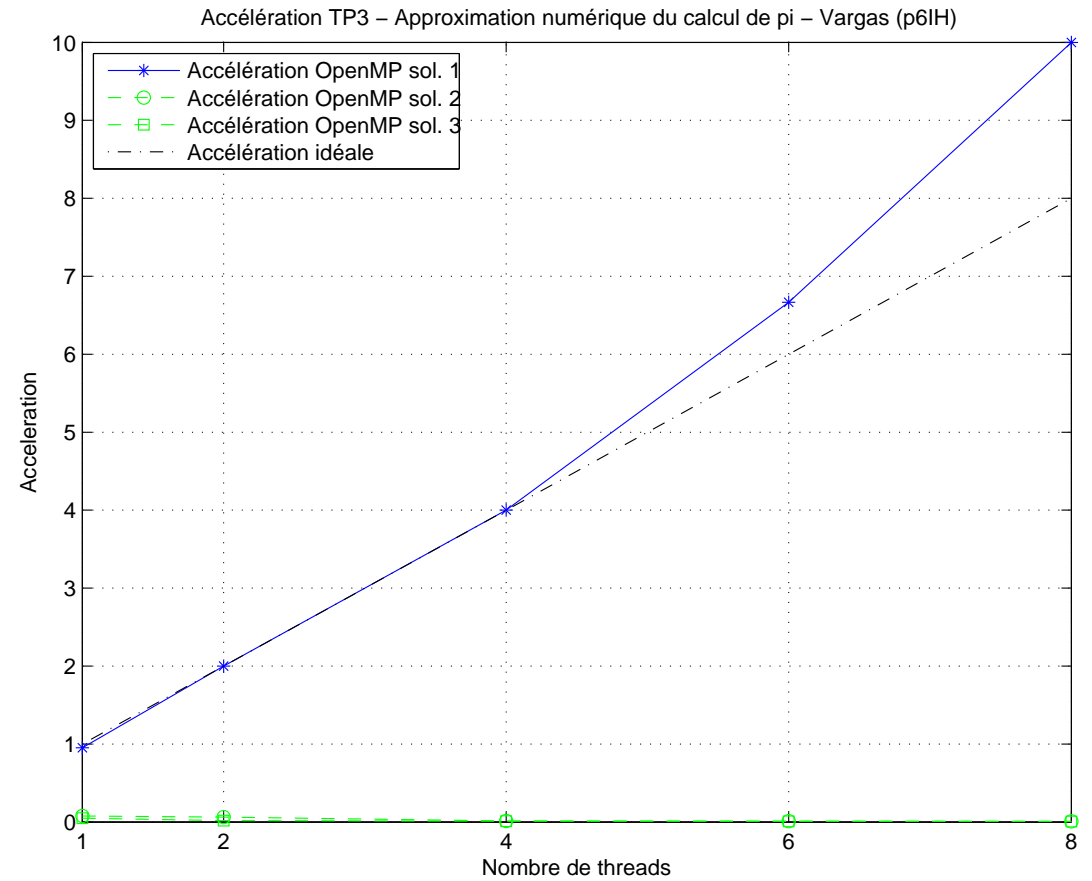
4 – tp3 : calcul de π

Il s'agit de calculer π par intégration numérique sachant que : $\int_0^1 \frac{4}{1+x^2} dx = \pi$

Le fichier `pi.f90` contient le programme permettant de calculer la valeur de π par la méthode des rectangles (point milieu). Soit $f(x) = \frac{4}{1+x^2}$ la fonction à intégrer et N et $h = \frac{1}{N}$ respectivement le nombre de points et le pas de discrétisation de l'intervalle d'intégration $[0, 1]$.

1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans le fichier `pi.f90`. Cet exercice peut être parallélisé de trois façons différentes (i.e. utilisation de directives OpenMP différentes pour chaque version).
2. Analyser les performances des trois codes sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`).
3. Optimiser les deux versions les moins performantes (sans changer le type des directives OpenMP utilisées), de façon à obtenir des performances identiques pour les trois versions parallèles. Tracer les courbes d'accélération obtenues.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



5 – tp4 : méthode du gradient conjugué

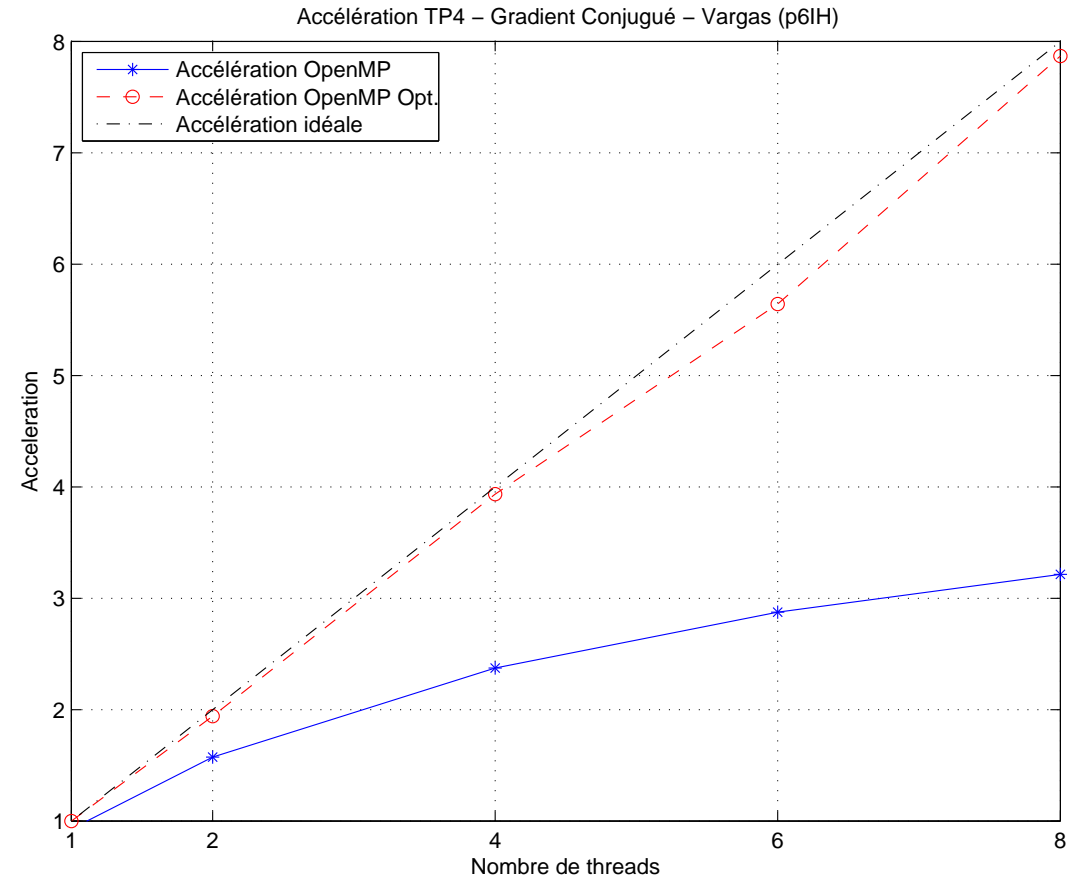
Le programme, contenu dans le fichier `gradient_conjugué.f90`, résout un système linéaire symétrique

$$A \times x = b$$

par la méthode du gradient conjugué préconditionné.

1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans le fichier `gradient_conjugué.f90`.
2. Analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`). Quelles sont vos conclusions quant à l'efficacité de la directive **WORKSHARE** ?
3. Optimiser la version parallèle du code en modifiant légèrement le code source pour ne plus avoir recours à la directive **WORKSHARE**. Tracer les courbes d'accélération correspondantes.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		

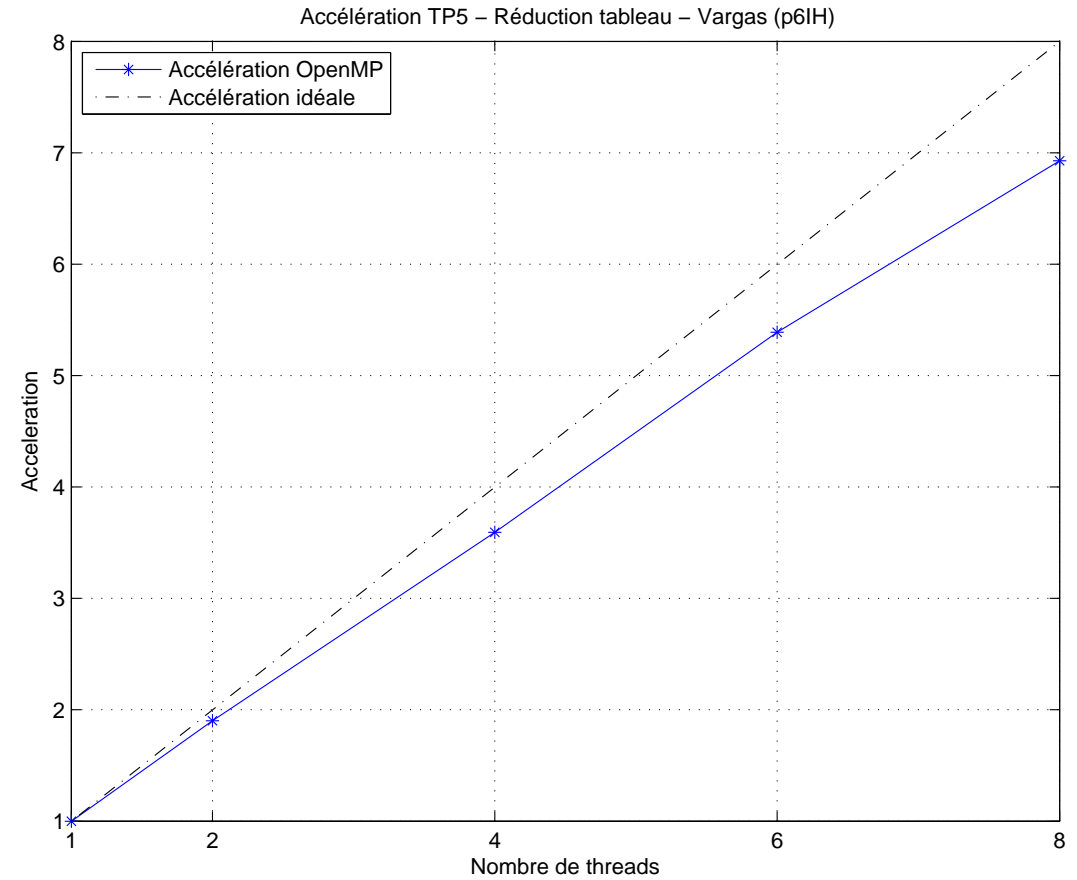


6 – tp5 : réduction d'un tableau

Le programme contenu dans le fichier `reduction_tab.f90` est extrait d'un code de chimie. Il s'agit de réduire un tableau tridimensionnel en un vecteur. Le but de ce TP est de paralléliser ce noyau de code de trois façons différentes.

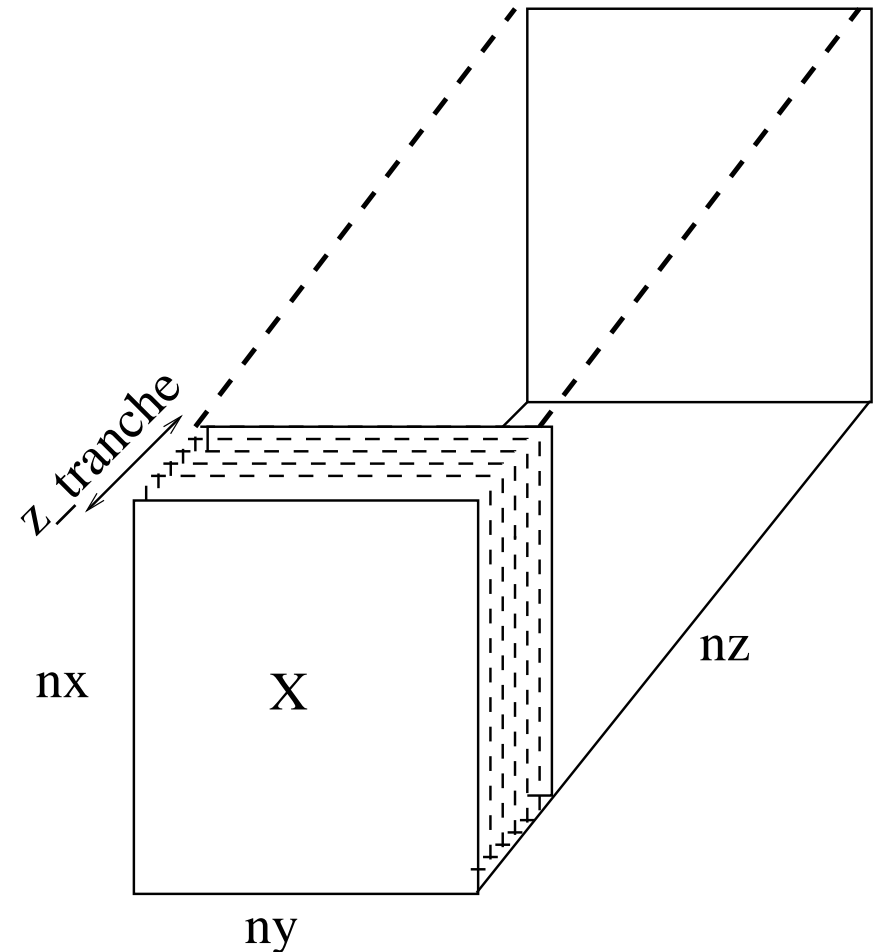
1. Version 1 - Analyser le statut des variables et insérer les directives OpenMP appropriées dans le fichier `reduction_tab.f90` sans aucune modification du code source.
2. Version 2 - Analyser le statut des variables et changer l'ordre des boucles de façon à paralléliser la boucle la plus externe avec une seule directive **PARALLEL DO**.
3. Version 3 - Sans toucher à l'ordre initial des boucles (i.e. k,j,i), adapter le code source de façon à paralléliser la boucle la plus externe en k.
4. Pour chacune des trois versions, analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`). Tracer les courbes d'accélération correspondantes. Comment expliquer les mauvaises performances de la version 2 ?

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



7 – tp6 : transformée de Fourier multiple

Le programme contenu dans le fichier `fft.f90` calcule la FFT réelle-complexe directe et inverse d'une matrice 3D x . La parallélisation est réalisée par répartition explicite du travail en découpant le tableaux x suivant la 3ème dimension par autant de tranches qu'il y a de tâches. Chaque tâche applique ensuite la FFT sur la tranche qui lui a été affectée indépendamment des autres.

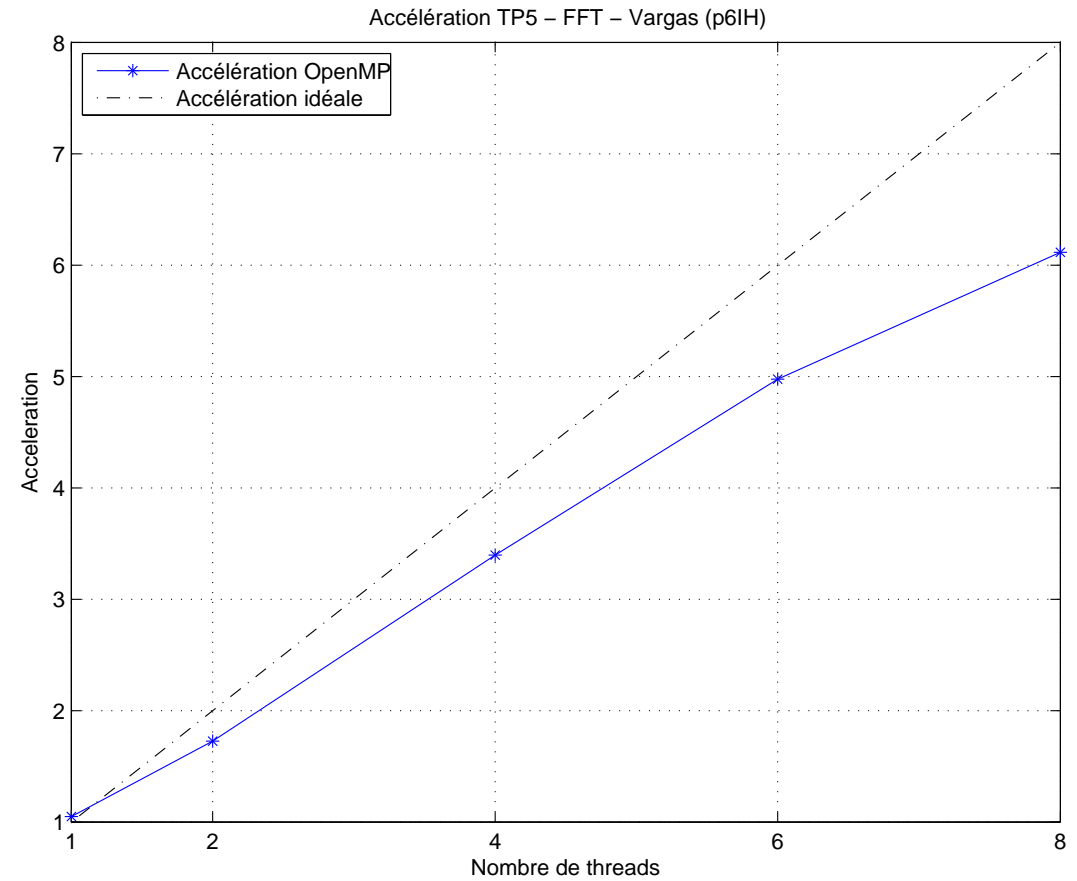


1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans le fichier `fft.f90` (utiliser la compilation conditionnelle pour prévoir le cas d'une exécution séquentielle).
2. Analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`).
3. Tracer les courbes d'accélération obtenues.

Remarque : la bibliothèque de FFT `libjfft.a` ne doit pas être modifiée.

Elle contient les références aux sous-programmes `scfft` et `csfft` utilisés dans le fichier `fft.f90`.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



8 – tp7 : méthode du Bi-Gradient Conjugué STABilisé

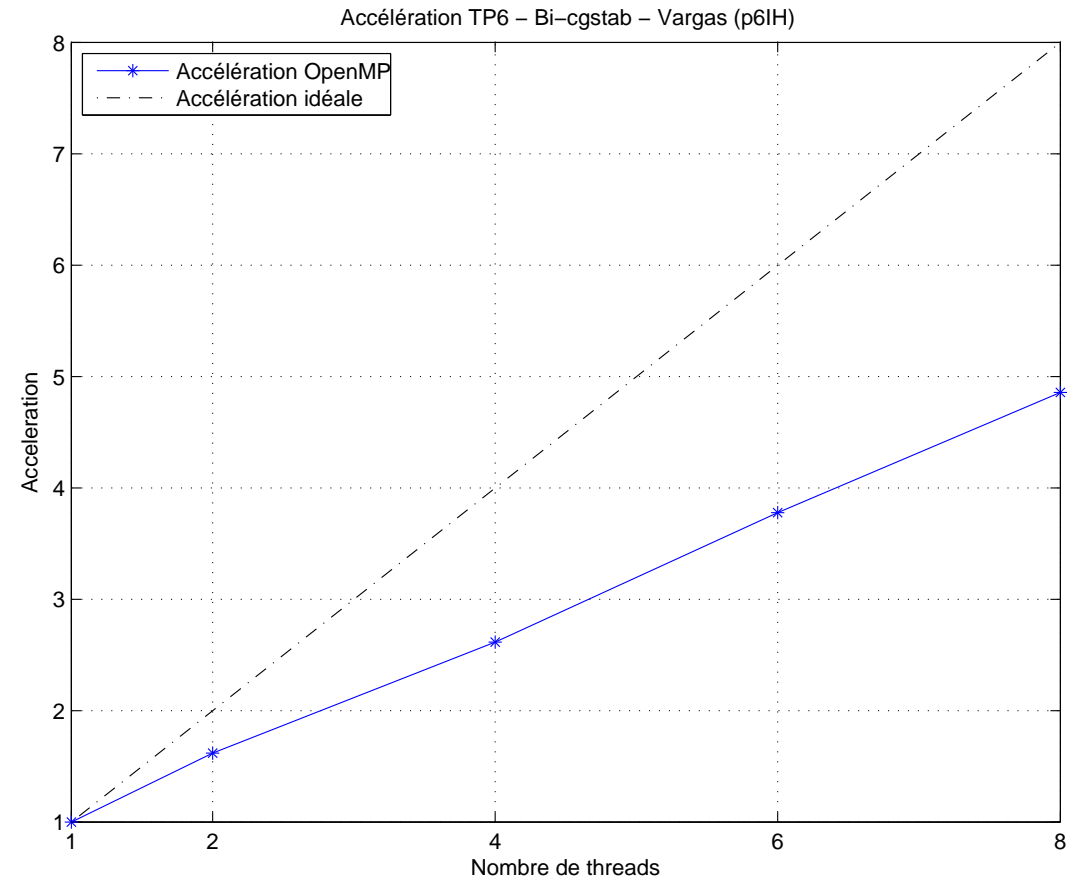
Le programme principal, contenu dans le fichier `principal.f90`, appelle le sous-programme `bi-cgstab`, défini dans le fichier `bi-cgstab.f90`, pour résoudre un système linéaire général à plusieurs seconds membres :

$$A \times x = b$$

par la méthode du Bi-CGSTAB (Bi-Gradient Conjugué STABilisé).

1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans les fichiers `principal.f90` et `bi-cgstab.f90` en considérant le sous-programme `bi-cgstab` comme **orphelin**.
2. Analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`).
3. Tracer les courbes d'accélération obtenues.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



9 – tp8 : problème de Poisson

Les fichiers `poisson.f90` et `gradient_conjugue.f90` (ici étendu à la résolution de plusieurs systèmes linéaires indépendants) permettent de résoudre le problème de POISSON (1) dont la solution analytique $u_a(x, y)$ est donnée par :

$$u_a(x, y) = \cos \pi x \times \sin \pi y \quad ; \quad (x, y) \in [0, 1] \times [0, 1]$$

$$\left\{ \begin{array}{l} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = b(x, y) \\ u(0, y) = u_a(0, y) \\ u(1, y) = u_a(1, y) \\ u(x, 0) = u_a(x, 0) \\ u(x, 1) = u_a(x, 1) \end{array} \right. \quad (1)$$

La méthode numérique adoptée est mixte. Nous appliquerons une méthode de différences finies centrées dans la direction x suivie d'une FFT en sinus dans la direction y . Pour cela, notons \tilde{u} et \tilde{b} la FFT en sinus respectivement de u et de b par rapport à y et appliquons cette FFT à l'équation de POISSON (1) qui devient :

$$-\frac{\partial^2 \tilde{u}}{\partial x^2} - \widetilde{\frac{\partial^2 u}{\partial y^2}} = \tilde{b}(x, y)$$

Il se trouve que la transformée en sinus $\widetilde{\frac{\partial^2 u}{\partial y^2}}$ de l'opérateur $\frac{\partial^2 u}{\partial y^2}$ est un opérateur diagonal dont les éléments représentent les valeurs propres de la matrice associée obtenue par différences finies de l'opérateur en question. Ses valeurs propres sont analytiquement connues (c'est ce qui fait le charme de cette méthode). Si $j = 1, \dots, N_j$ désigne l'indice du point de discrétisation et h_y désigne le pas de discrétisation suivant y , ces valeurs propres s'expriment selon la formule suivante :

$$\text{vp}_j = \frac{4}{h_y^2} \sin^2 \frac{\pi(j-1)}{2(N_j-1)} \quad ; \quad j = 2, \dots, N_j - 1$$

Si bien que, dans la base des vecteurs propres, le problème de POISSON se résume à la résolution de $N_j - 2$ systèmes tridiagonaux symétriques indépendants (employer l'algorithme du gradient conjugué) de taille $(N_i - 2) \times (N_i - 2)$ chacun, où N_i représente le nombre de points de discrétisation dans la direction x :

$$\begin{pmatrix} d_j & -c_x & 0 & \dots & \dots & 0 \\ -c_x & d_j & -c_x & 0 & \dots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & 0 & -c_x & d_j & -c_x \\ 0 & \dots & \dots & 0 & -c_x & d_j \end{pmatrix} \begin{pmatrix} \tilde{u}_{2,j} \\ \tilde{u}_{3,j} \\ \vdots \\ \vdots \\ \tilde{u}_{N_i-2,j} \\ \tilde{u}_{N_i-1,j} \end{pmatrix} = \begin{pmatrix} \tilde{b}_{2,j} + \text{CL} \\ \tilde{b}_{3,j} \\ \vdots \\ \vdots \\ \tilde{b}_{N_i-2,j} \\ \tilde{b}_{N_i-1,j} + \text{CL} \end{pmatrix}$$

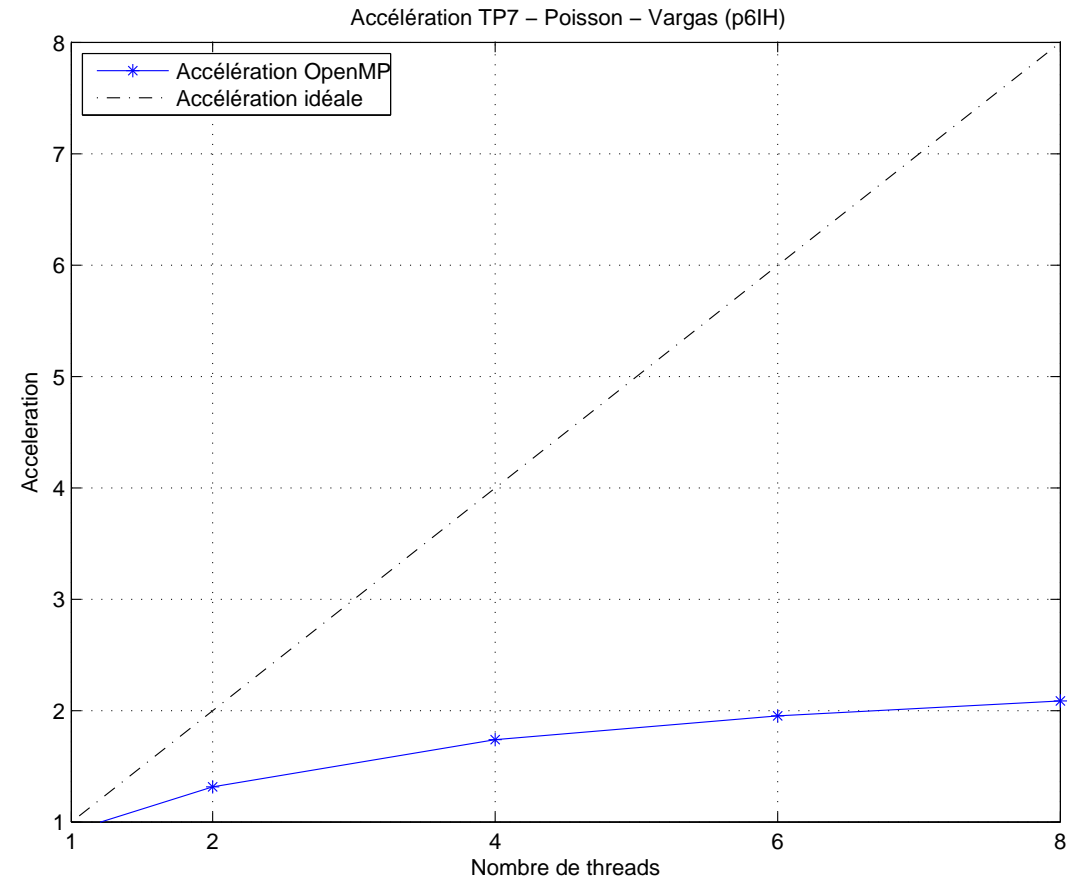
où, $c_x = \frac{1}{h_x^2}$, $c_y = \frac{1}{h_y^2}$, $d_j = 2c_x + vp_j$ et h_x est le pas de discrétisation suivant la direction x . Le terme CL contient la contribution des conditions sur les frontières.

Enfin, $(N_i - 2)$ FFT inverses indépendantes de \tilde{u} par rapport à y calcul la solution finale u dans la base canonique.

1. Analyser le statut des variables et insérer les directives OpenMP appropriées dans les fichiers `poisson.f90` et `gradient_conjugue.f90` en considérant le sous-programme `gradient_conjugue` comme `orphelin` et en n'utilisant qu'une unique région parallèle.
2. Analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`).
3. Tracer les courbes d'accélération obtenues

Remarque : le fichier `c06haf.o` ne doit pas être modifié. Il contient la référence au sous-programme `c06haf` (il réalise la FFT en sinus et son inverse) appelé dans le fichier `poisson.f90`.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		



10 – tp9 : nid de boucles avec dépendance

Le code, contenu dans le fichier `dependance.f90` est constitué de deux boucles imbriquées.

Dans cet exercice, il s'agit :

1. déterminer si les boucles sont des boucles parallèles (i.e. pas de dépendance entre les itérations). Si on force la parallélisation des boucles, que se passe-t-il ?
2. de paralléliser le code en insérant les directives OpenMP appropriées dans le fichier `dependance.f90`. Toute la difficulté de cet exercice consiste à synchroniser correctement les différents threads entre eux de façon à ne pas casser de dépendance.
3. d'analyser les performances du code sur 2, 4, 6 et 8 tâches par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier `batch.sh`). Attention, la version parallèle du code n'est valide que si la valeur affichée à l'écran pour la variable `norme` est égale à 0...
4. de tracer les courbes d'accélération obtenues.

Nb. de threads	Tps elapsed	Accélération
mono		
1		
2		
4		
6		
8		

