Enoncé des travaux pratiques du cours OpenMP

1 – Description

Les travaux pratiques se dérouleront sur la machine Jean Zay (HPE SGI 8600, 71560 cœurs, 40 cœurs par nœud SMP) dans le répertoire $WORK/OpenMP_tp. Ils sont constitués de onze exercices indépendants tp0, tp1, tp2, ..., tp10. Chaque exercice se trouve dans un répertoire contenant systématiquement un fichier Makefile pour la compilation, un fichier batch.sh pour la soumission en traitement par lots et un ou plusieurs fichiers sources à compléter. Les fichiers sources sont disponibles en Fortran et en C.

- ☞ 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 make mono, pour compiler une version séquentielle.
- Taper la commande make para, pour interpréter les directives `OpenMP` et générer une version parallèle.
- Taper la commande make clean pour effacer les fichiers objets et core ou make cleanall pour les fichiers objets, core et exécutables.
- Tapez ensuite la commande sbatch batch.sh, pour une soumission en *batch*. Celle-ci inclut une exécution séquentielle, puis parallèle sur 2, 4, 6 et 8 threads. Attention, les exécutables monoprocesseurs et parallèles doivent avoir été générés au préalable. La commande squeue -u $USER permet de suivre l'évolution du travail soumis. Une fois le travail terminé normalement, le résultat de l'exécution sera écrit dans un fichier dont le nom possède le suffixe .res.
- Taper la commande make visu pour générer et afficher la courbe d'accélération correspondant au résultat de l'exécution stocké dans le fichier .res.

Enoncé général des TPs

Pour chaque TP, il faut :

1. analyser le statut des variables et paralléliser le code en utilisant des directives OpenMP.
2. analyser les performances du code sur 2, 4, 6 et 8 threads par rapport à une exécution séquentielle (utiliser la soumission en *batch* avec le fichier batch.sh).
3. tracer les courbes d'accélération obtenues.

Bon courage...

2 – tp0 : Hello World

Dans cet exercice très simple, il s'agit :

1. d'écrire un programme OpenMP affichant le nombre de threads utilisés pour l'exécution et le rang de chacun des threads.
2. de compiler le code manuellement afin d'obtenir un exécutable monoprocesseur et un exécutable parallèle.
3. de tester les programmes obtenus sans passer par la soumission en *batch*, en faisant varier le nombre de threads pour le programme parallèle.

Exemple de sortie pour le programme parallèle avec 4 threads :

```
Bonjour depuis le thread de rang 2
Bonjour depuis le thread de rang 1
Bonjour depuis le thread de rang 3
Bonjour depuis le thread de rang 0
Execution de hello_world en parallele avec 4 threads
```

3 – tp1 : produit de matrices

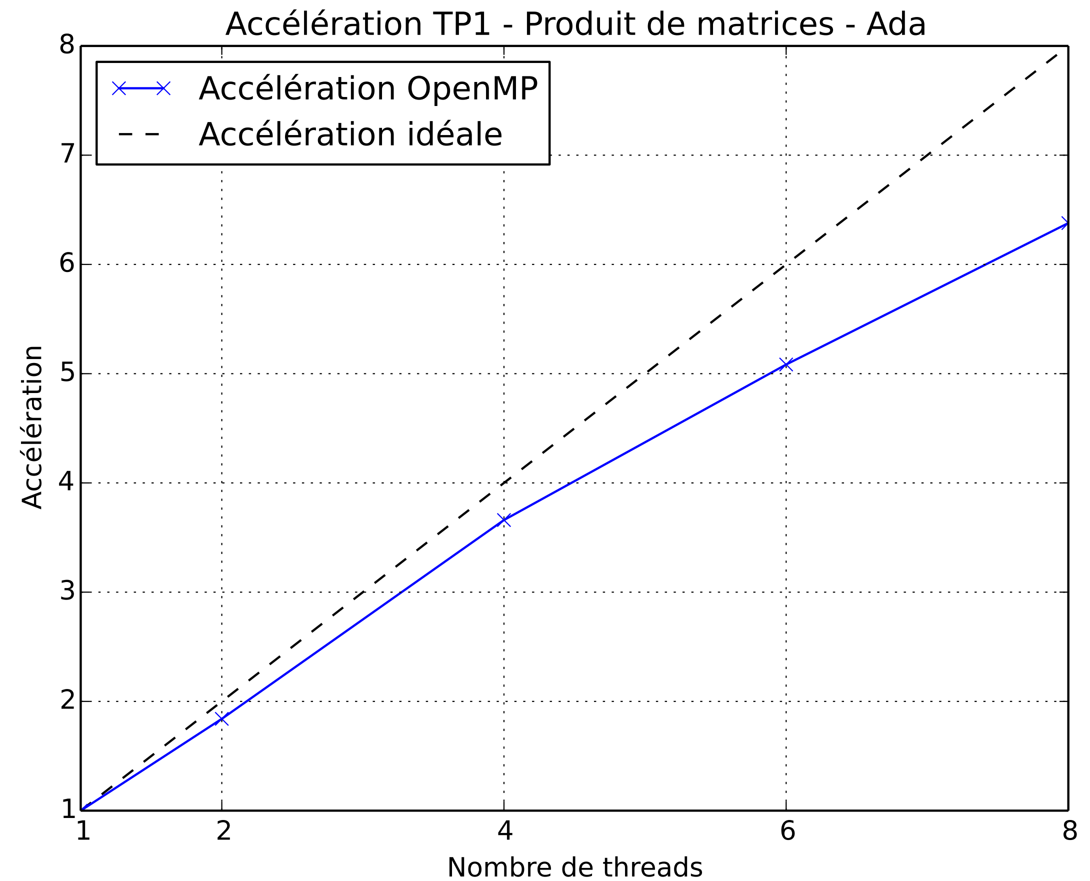
Le code, contenu dans le fichier prod_mat.f90, calcule le produit de matrices :

$$C = A \times B$$

Dans cet exercice, il s'agit :

1. d'insérer les directives OpenMP appropriées et analyser les performances du code.
2. de tester les différents modes (STATIC, DYNAMIC, GUIDED) de répartition des itérations d'une boucle et faire varier la taille des paquets.

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



4 – tp2 : méthode de Jacobi

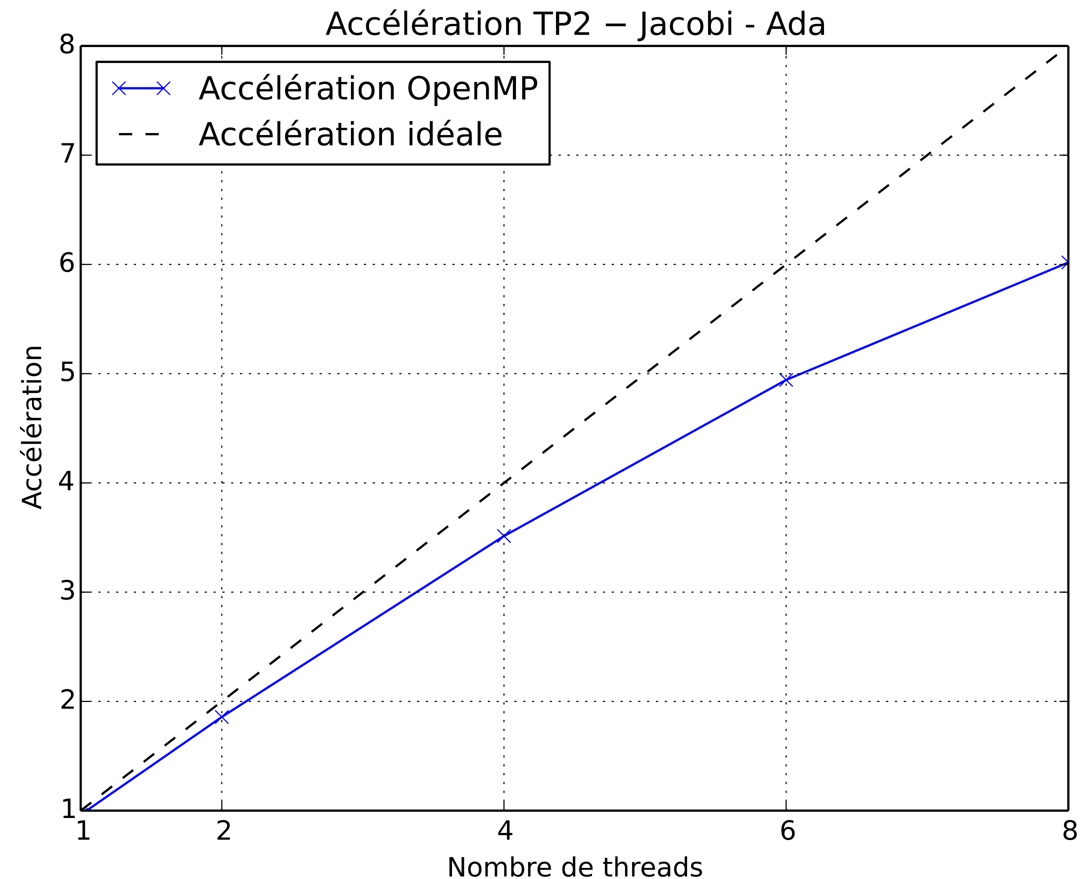
Le programme, contenu dans le fichier jacobi.f90, résout un système linéaire général

$$A \times x = b$$

par la méthode itérative de Jacobi.

Dans cet exercice, il s'agit de paralléliser la résolution du système.

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



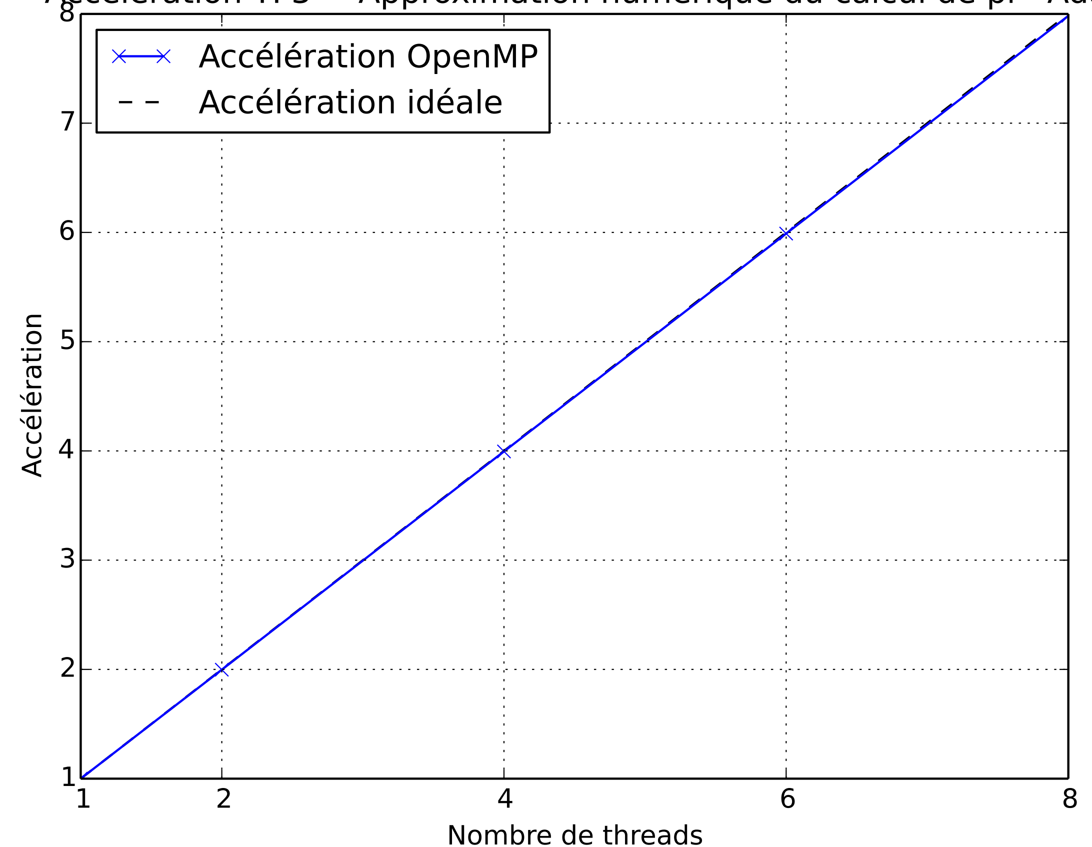
5 – tp3 : calcul de π

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 $[0, 1]$.

Cet exercice peut être parallélisé de trois façons différentes (i.e. utilisation de directives OpenMP différentes pour chaque version). Analyser les performances des trois codes puis optimiser les versions les moins performantes (sans changer le type des directives OpenMP utilisées), de façon à obtenir des performances identiques pour les trois versions parallèles.

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



6 – tp4 : méthode du gradient conjugué

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

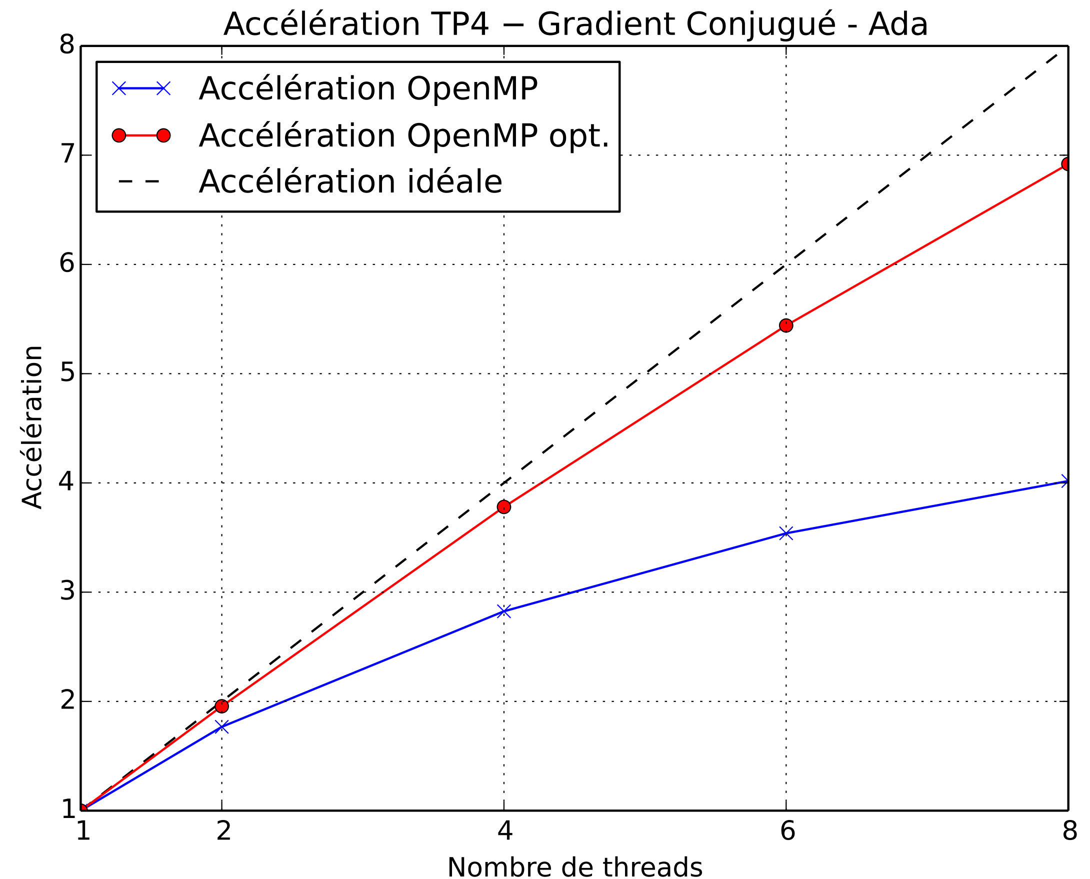
$$A \times x = b$$

par la méthode du gradient conjugué préconditionné.

En Fortran, ce programme peut être parallélisé en utilisant notamment des constructions WORKSHARE.

1. Après avoir introduit les directives OpenMP appropriées, analyser les performances du code.
2. 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 aux endroits où elle pose problème.

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

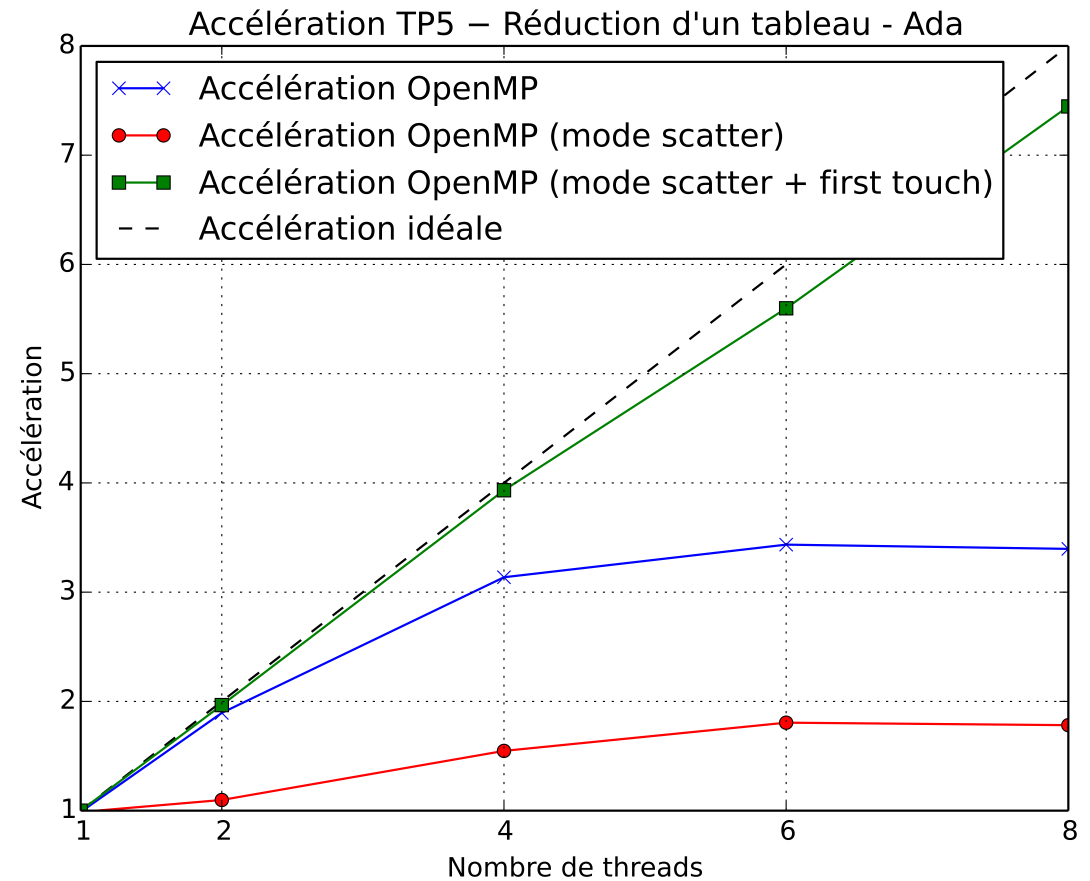


7 – tp5 : réduction d'un tableau

Le programme contenu dans le fichier reduction_tab.f90 est extrait d'un code de chimie. Il s'agit de réduire un tableau tridimentionnel en un vecteur. Le but de ce TP est de paralléliser ce noyau de calcul sans toucher à l'ordre initial des boucles (i.e. k,j,i).

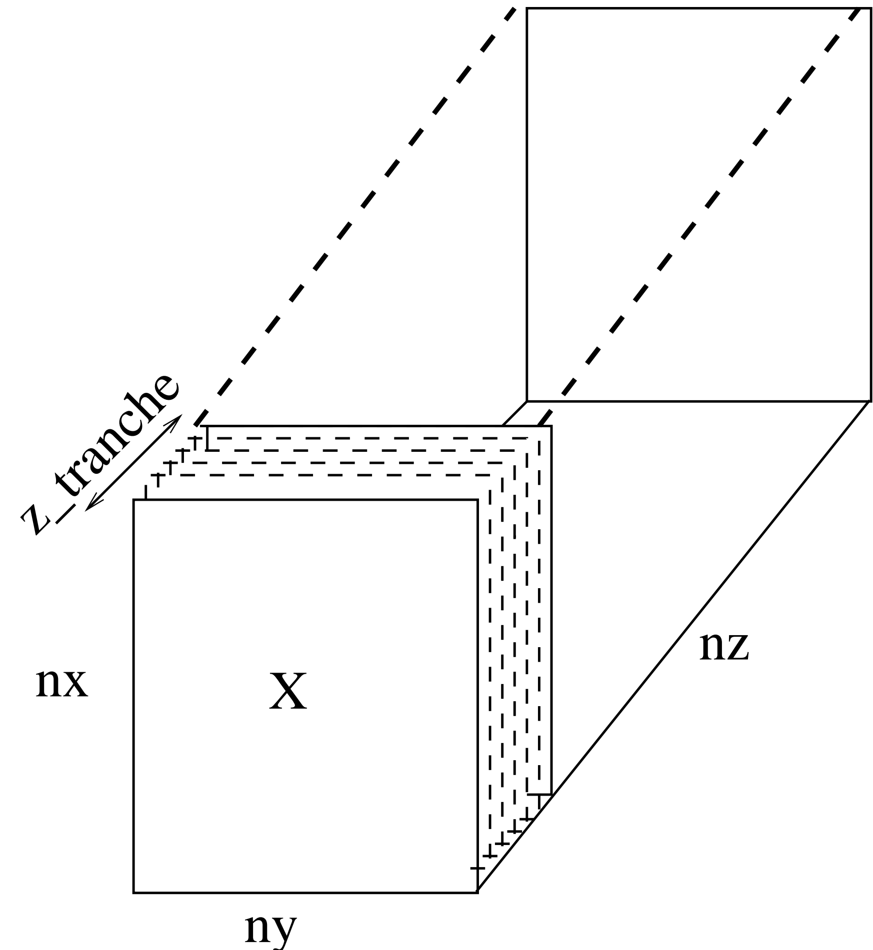
1. Analyser le statut des variables et adapter le code source de façon à paralléliser la boucle la plus externe en k.
2. Comparer les performances obtenues en utilisant l'association thread/cœur d'exécution par défaut sur Ada et en utilisant le mode d'association *scatter*. Proposer une explication quant aux mauvaises performances de ce dernier.
3. Optimiser le code source pour le mode *scatter* en prenant en compte l'affinité mémoire. Pourquoi cette troisième série d'exécutions permet-elle d'obtenir les meilleures performances ?

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



8 – 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 threads. Chaque thread applique ensuite la FFT sur la tranche qui lui a été affectée indépendemment des autres.

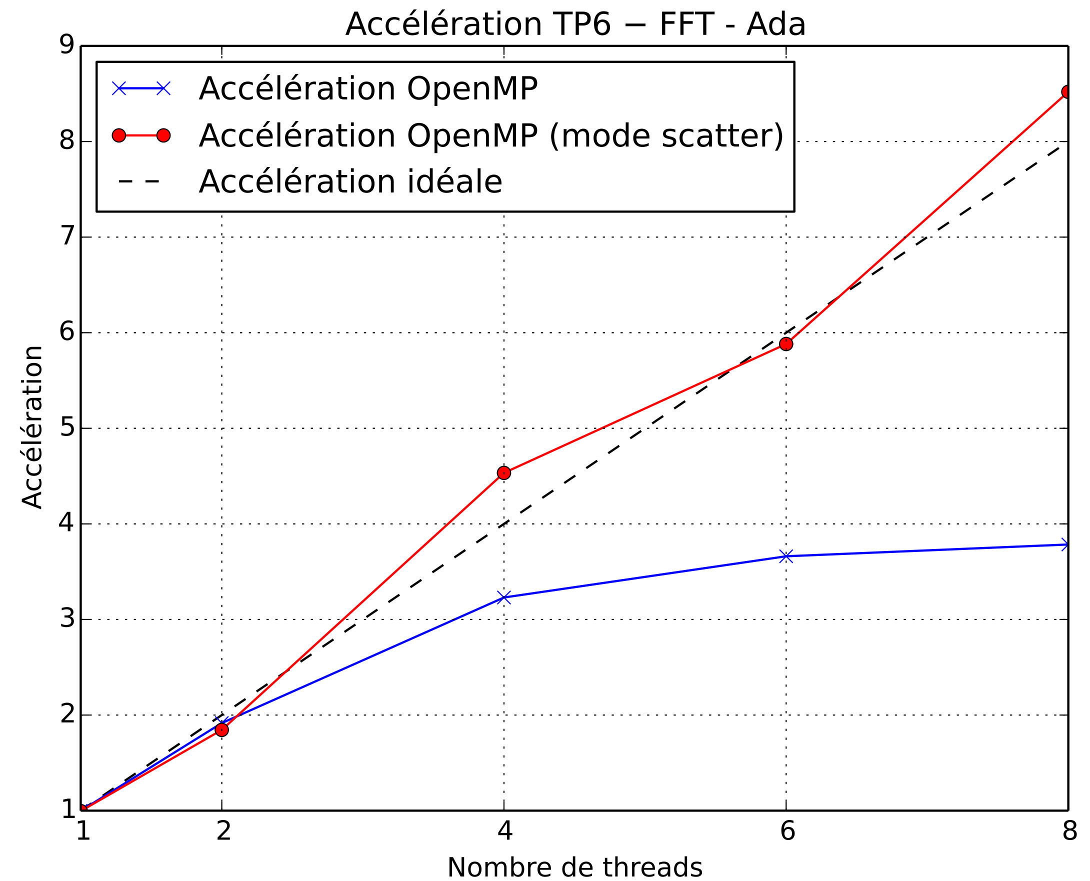


1. Insérer les directives OpenMP appropriées dans le fichier fft.f90 (utiliser la compilation conditionnelle pour prévoir le cas d'une exécution séquentielle).
2. Analyser les performances du code et tracer les courbes d'accélération obtenues.
3. Faire de même pour le mode d'association thread/cœur *scatter*. Pourquoi observe-t-on de meilleures performances sans même avoir à modifier le code source ?

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

Elle contient les références aux sous-programmes scfftm et csfftm utilisés par le programme principal.

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



9 – tp7 : méthode du Bi-Gradient Conjugué STABilisé

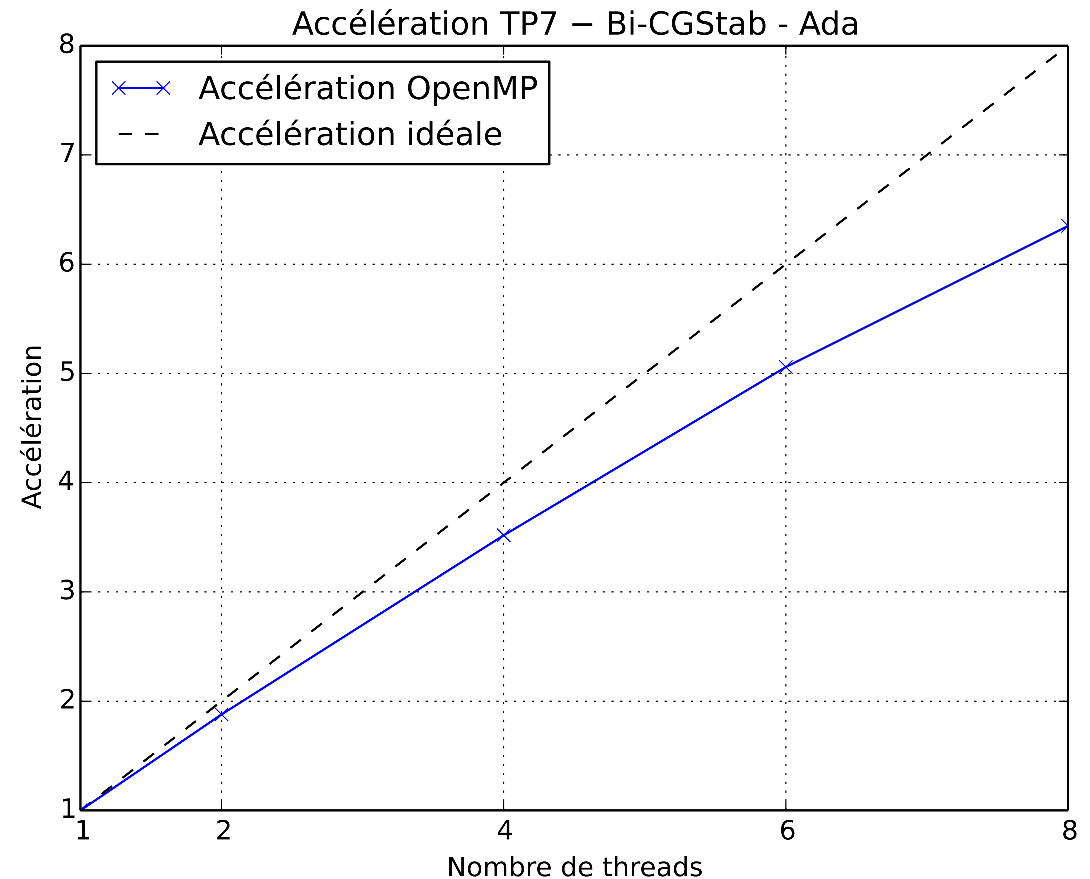
Le programme principal, contenu dans le fichier principal.f90, appelle le sous-programme bi-cgstab, défini dans le fichier bi-cgstab.f90, pour résoudre un système linéaire général à plusieurs seconds membres :

$$A \times x = b$$

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

1. 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 et tracer les courbes d'accélération obtenues.

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



10 – 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 :

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

$$\begin{cases} -\dfrac{\partial^2 u}{\partial x^2} - \dfrac{\partial^2 u}{\partial y^2} = b(x, y) \\ u(0, y) = u_a(0, y) \\ u(1, y) = u_a(1, y) \\ u(x, 0) = u_a(x, 0) \\ u(x, 1) = u_a(x, 1) \end{cases} \tag{1}$$

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 $j = 1, \ldots, N_j$ 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 :

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

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 :

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

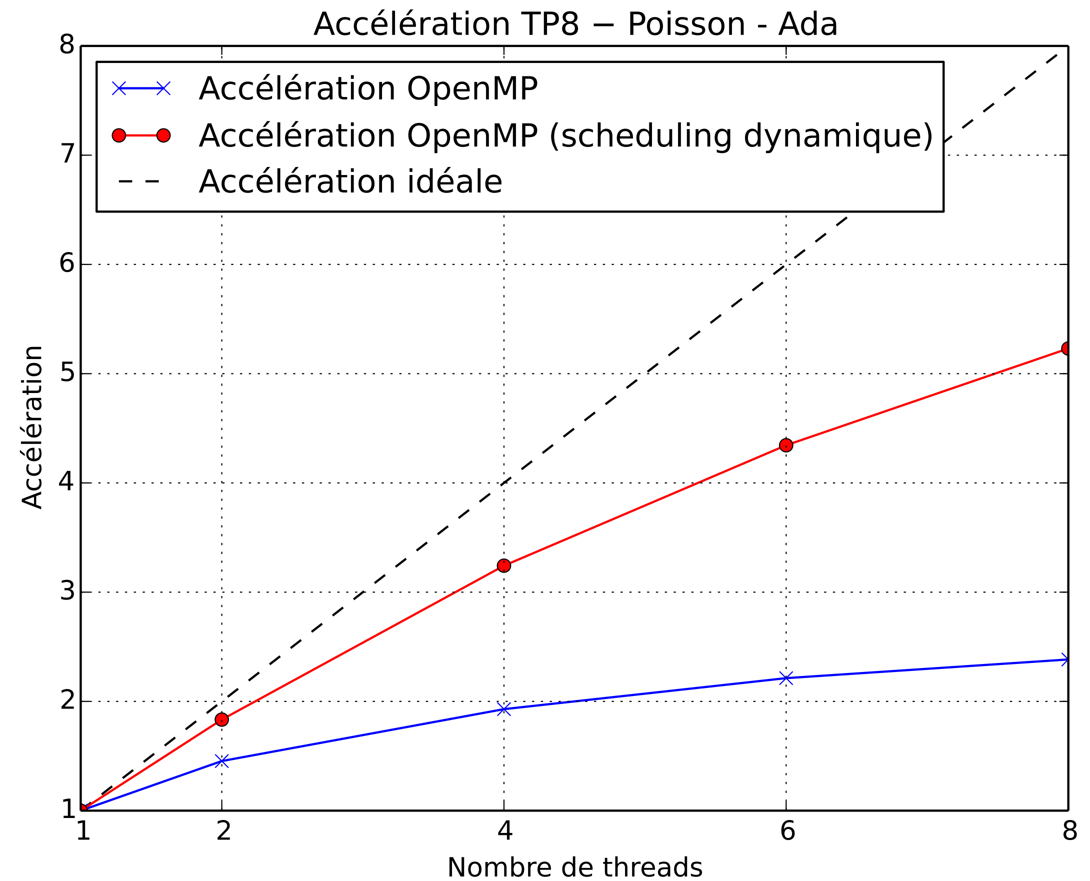
où, $c_x = \frac{1}{h_x^2}$, $c_y = \frac{1}{h_y^2}$, $d_j = 2c_x + \mathrm{vp}_j$ et h_x est le pas de discrétisation suivant la direction x. Le terme CL contient la contribution des conditions sur les frontières.

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

1. 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 seule région parallèle.
2. Analyser les performances du code et tracer les courbes d'accélération obtenues.
3. Faire de même en utilisant le mode de répartition des itérations DYNAMIC. Proposer une explication quant à la différence de performance observée.

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		



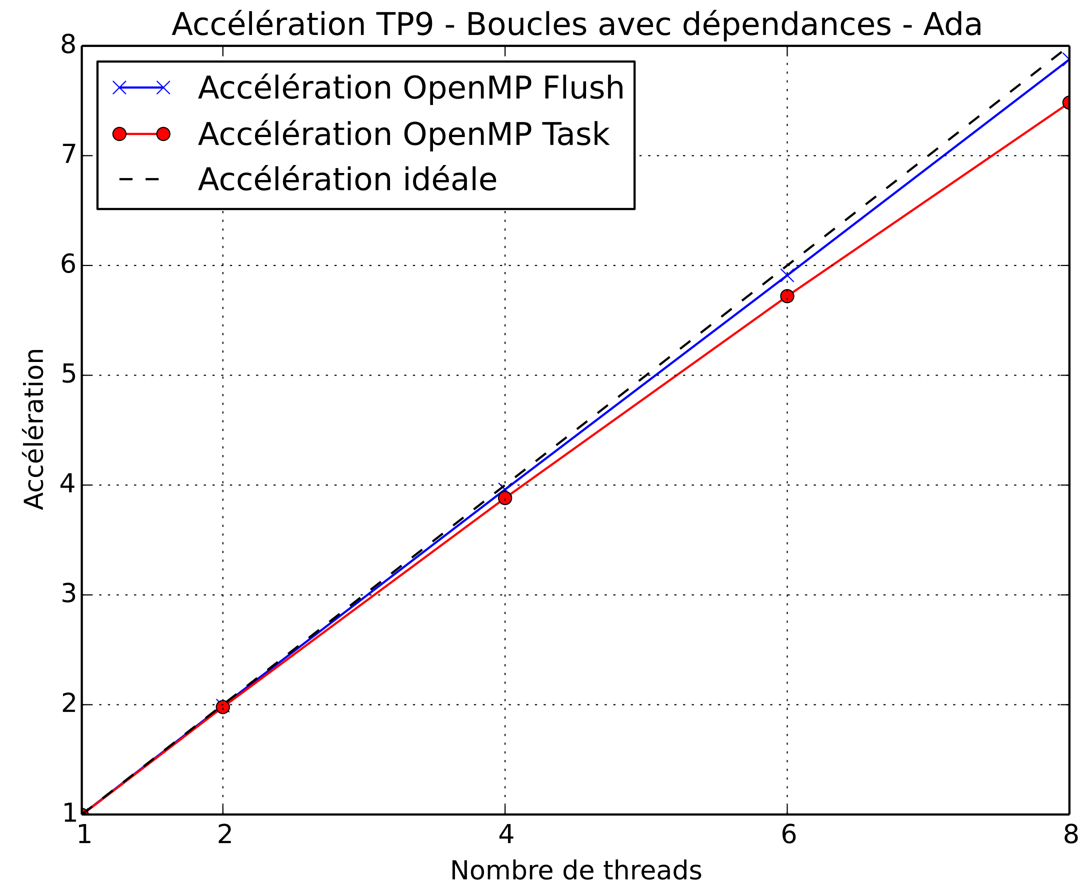
11 – tp9 : nid de boucles avec dépendances

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

Dans cet exercice, il s'agit :

1. de déterminer si les boucles sont des boucles parallèles (i.e. pas de dépendance entre les itérations). Si on force la parallèlisation des boucles, que se passe-t-il ?
2. de paralléliser le code en insérant les directives OpenMP appropriées dans le fichier dependance.f90. Deux approches sont possibles, soit avec la directive *flush* ou soit en utilisant les tâches OpenMP. Toute la difficulté de cet exercice consiste à synchroniser correctement les différents threads entre eux de façon à respecter les dépendances entre les itérations.
3. d'analyser les performances du code et de tracer les courbes d'accélération obtenues. 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...

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



12 – tp10 : produit de matrices par l'algorithme de Strassen

Le code, contenu dans le fichier strassen.F90, calcule le produit de matrices :

$$C = A \times B$$

en utilisant l'algorithme récursif de Strassen.

Dans cet exercice, il s'agit :

1. d'analyser le code et de le paralléliser en utilisant des tâches OpenMP.
2. de mesurer les performances du code et de tracer les courbes d'accélération obtenues.

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