OpenMP tutorials

# 1 – Introduction

The tutorials will take place on the Vargas computer (IBM Power6 with 3586 cores). They are located in the $WORKDIR/OpenMP_tp directory of each student account. They consist of 9 independent tutorials ; each of them has its own directory which includes a Makefile for compilation, a batch.sh file to launch a dedicated batch execution and, of course, one or several Fortan source files that you'll have to edit and modifiy (according to the instructions).

In each tutorial directory, you can find the following sub-directories :

- sans_indications_openmp has the plain sequential source files.
- avec_indications_openmp has source files that include clues about the modifications you have to work on.
- solution has the source file that you're supposed to look at when you're done, to check if your work is correct.

## General instructions

- type gmake mono to buid the sequential executable (the compiler sees the OpenMP directives as comments).
- type gmake para to build the parallel executable, with the compiler interpreting the OpenMP directives.
- type gmake clean to wipe out object files build by a previous compilation and core files resulting from a previous execution ; type gmake cleanall to erase previous executable files too.
- type llsubmit batch.sh to submit a job in the batch system. The job will run on dedicated nodes, first a sequential execution, then parallel ones with 2, 4, 6, 8 threads. The executables must have been build before (see previous gmake commands).
- The Qstat command lists the submitted/running jobs. Once the batch job is done, the results is available in a .res file in the current directory.

Good work !

## 2 – tp1 : matrix product

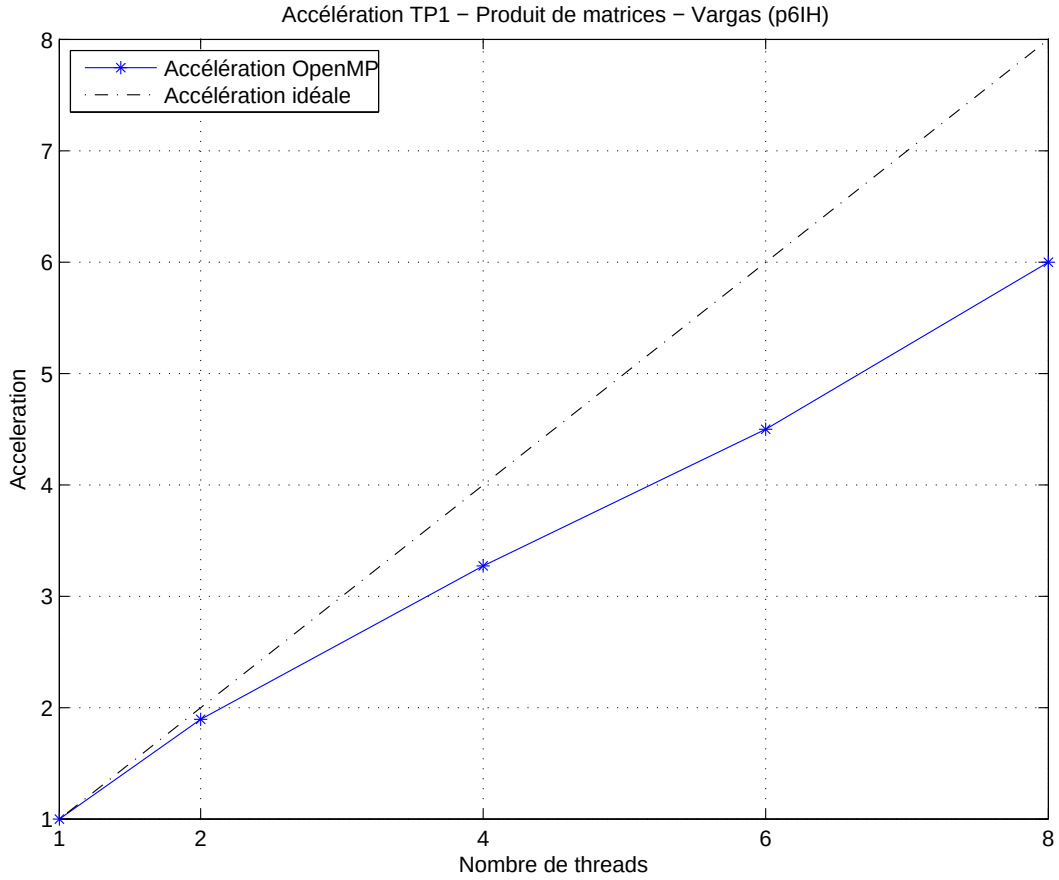
The prod_mat.f90 source file computes a matrix product :

$$C = A \times B$$

In this basic tutorial, you're asked you to :

1. Insert proper OpenMP directives.
2. Analyse the parallel performance on 2, 4, 6, 8 threads comparing to a sequential execution (please do a dedicated execution by submitting a *batch* job). You can test the various loop distribution modes (STATIC, DYNAMIC, GUIDED) and test various chunk sizes.
3. Plot the acceleration curves.

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 3 – tp2 : Jacobi Method

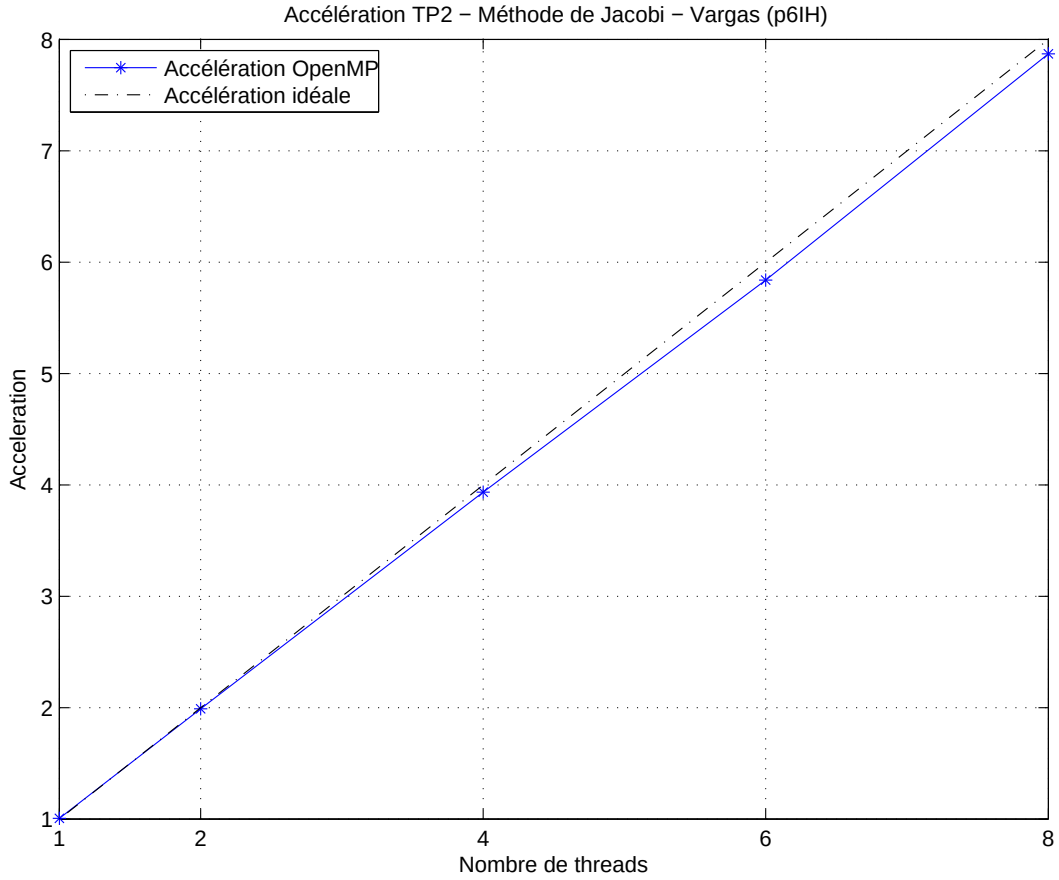
The jacobi.f90 source file contains the resolution of a linear system

$$A \times x = b$$

with the Jacobi iterative method.

1. Analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the jacobi.f90 source file.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a batch job).
3. Plot the acceleration curve.

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



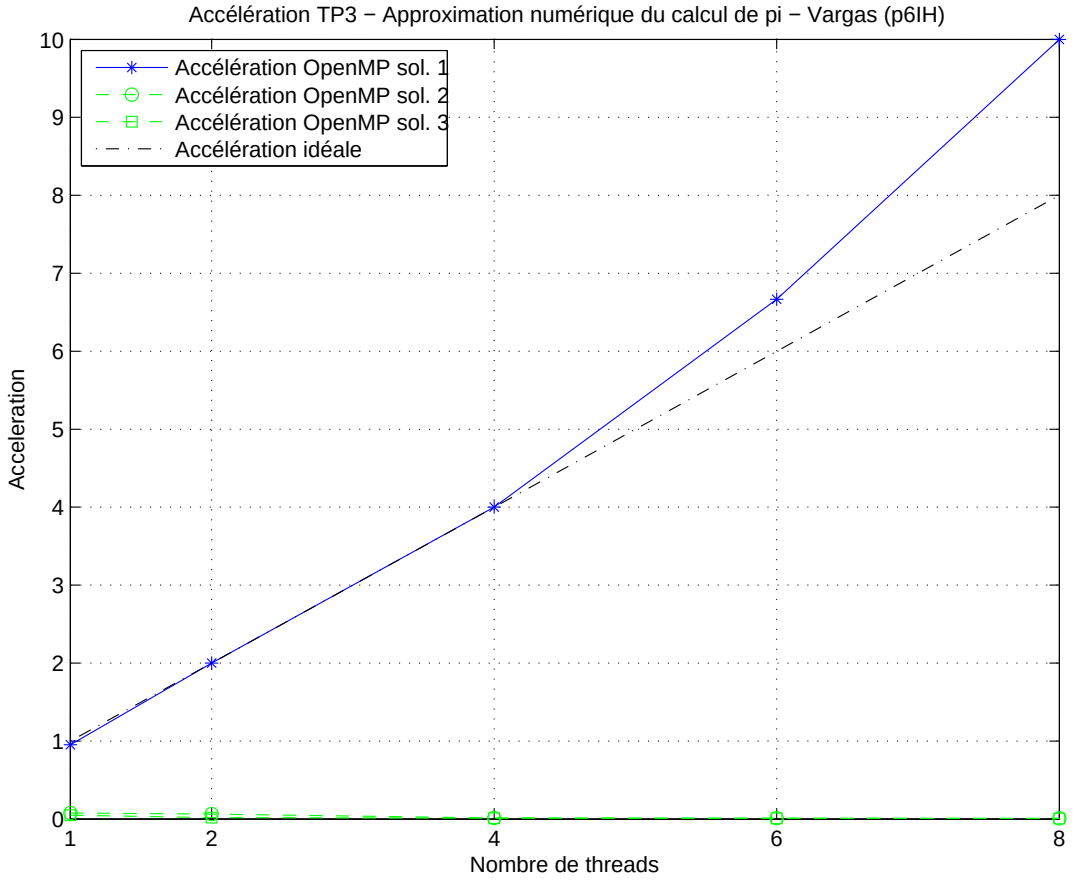
## 4 – tp3 : computation of $\pi$

The aim of this exercice is to compute *pi* by numerical integration. In the pi.f90 source file you can find the program that computes *pi* using the rectangle method (mean point). Let $f(x) = \frac{4}{1+x^2}$ the function to integrate and $N$, $h = \frac{1}{N}$ the number of points and the discretization interval $[0, 1]$, respectively.

1. Analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the pi.f90 source file. This tutorial's point is to use three different methods (three different OpenMP directives, one for each version) to parallelize the code.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a *batch* job).
3. Optimize the 2 less performing versions, without changing the OpenMP directives that were used, to reach the same performance for the three versions.
4. Plot the acceleration curves.

# Description of tp3

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 5 – tp4 : Conjugate Gradient ($CG$) Method

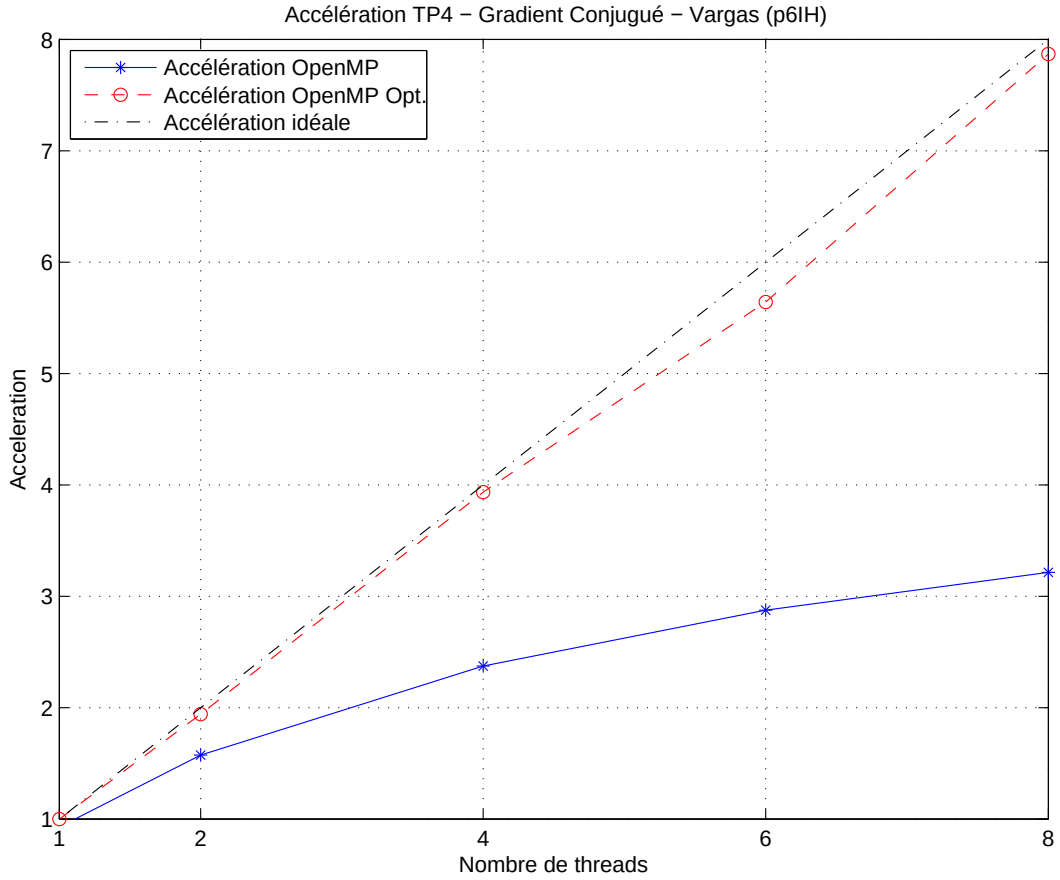
The gradient_conjugue.f90 source file contains a preconditionned conjugate gradient method to solve a symmetric linear system.

$$A \times x = b$$

1. Analyze the variable data-sharing attribute and insert the appropriate OpenMP directives in the gradient_conjugue.f90 source file.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a *batch* job). What are your conclusions about the WORKSHARE directive efficiency ?
3. Optimize the parallel version by slightly modifying the original source code to avoid the only one WORKSHARE construct that is inefficient.
4. Plot the appropriate acceleration curves.

# Description of tp4

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |

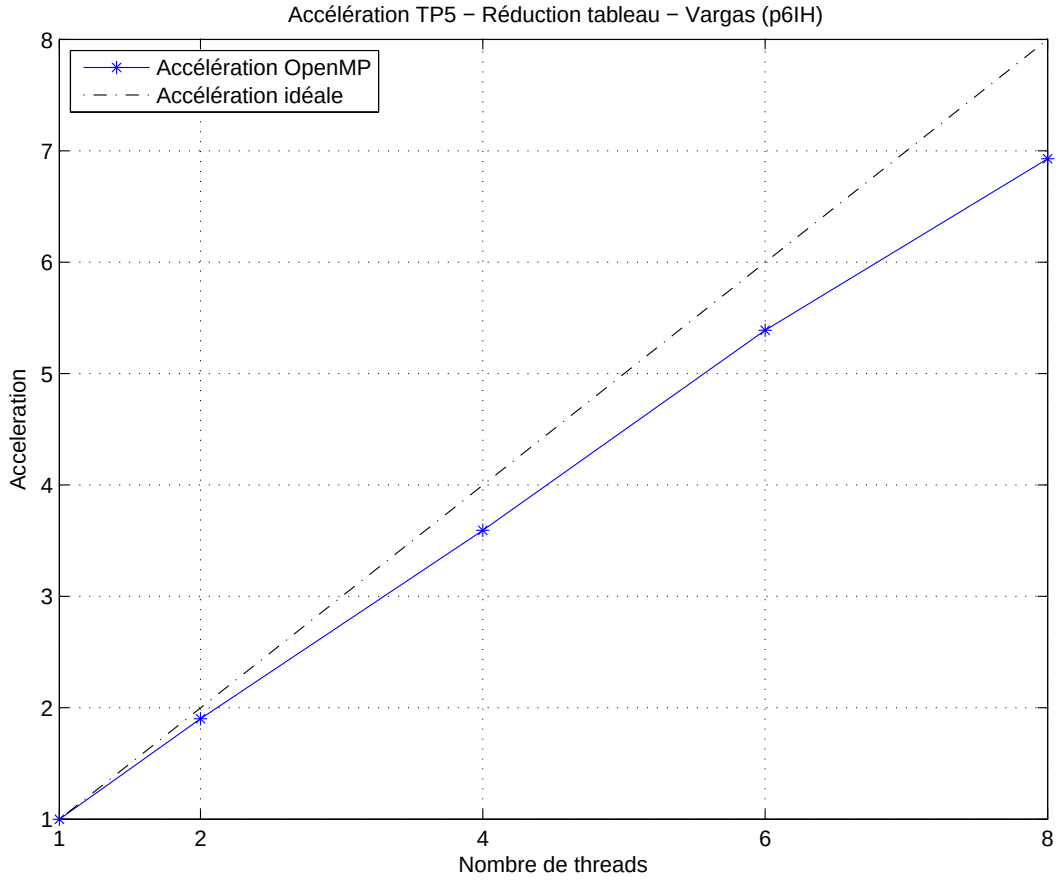


## 6 – tp5 : array reduction

The reduction_tab.f90 source code comes from a chemistry code ; it computes the reduction of a 3D array into a vector. The aim of this exercice is to use three different ways to parallelize this code.

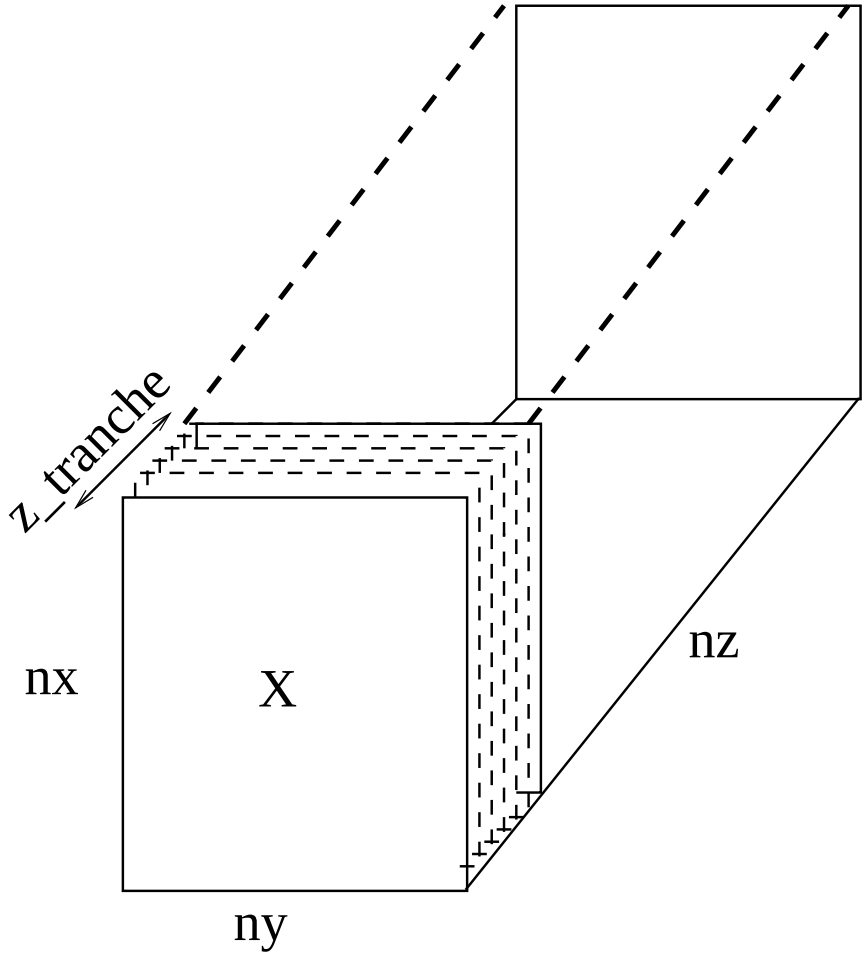
1. Version 1 : analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the reduction_tab.f90 source file. Do not make any modification to the original sequential code.
2. Version 2 : analyze the variable data-sharing attributes and modifiy the original loop order, so that one PARALLEL DO directive applies the the outermost loop.
3. Version 3 : without changing the loop order (k, j, i), modify the source code to parallelize the k outermost loop.
4. For each version, analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a emphbatch job). Plot the acceleration curves. How can version 2's bad performance be explained ?

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 7 – tp6 : multiple Fast Fourier Transform

The fft.f90 source file contains a 3D matrix direct real-complex forward and reverse FFT. Parallelization consists in explicitly distributing the work by cutting the 3D matrix following the 3rd dimension, with as many slices as threads : each thread computes the FFT on its 2D array.



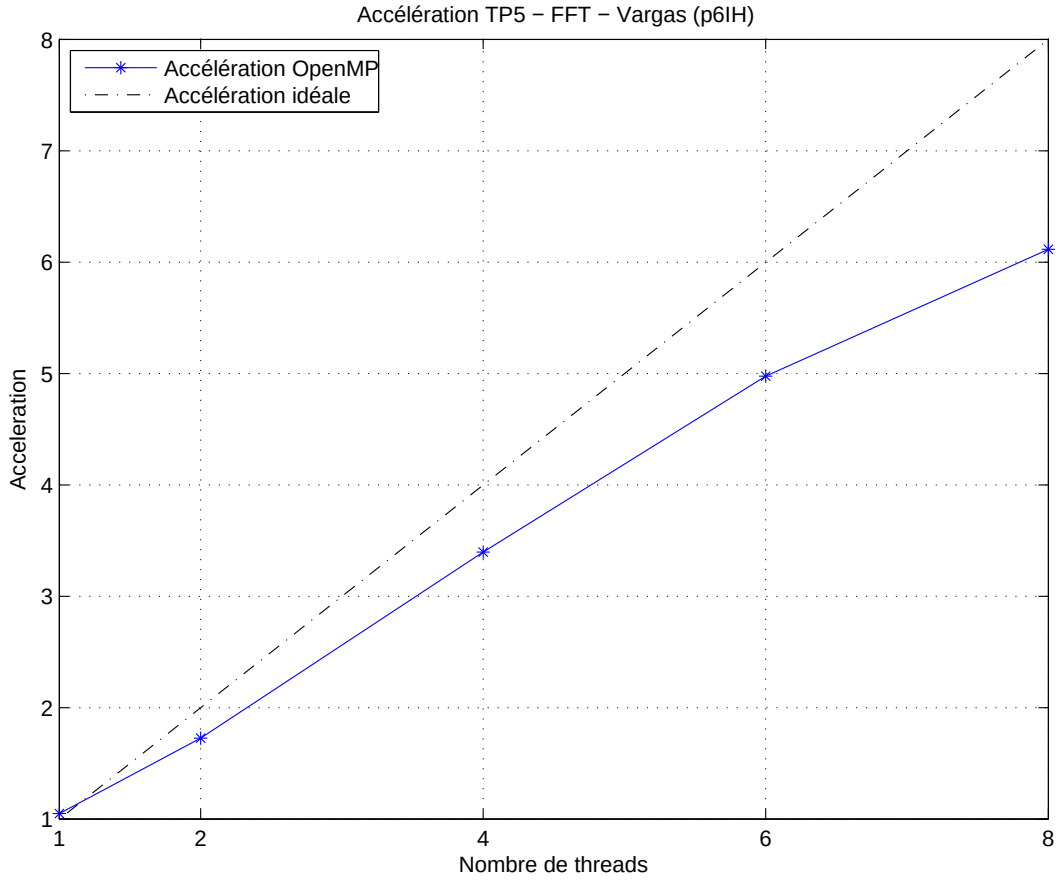
# Description of tp6

1. Analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the fft.f90 source file. In this tutorial, use conditional execution to take care of sequential execution.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a *batch* job).
3. Plot the acceleration curve.

Please note : the jmfft.a FFT library file should not be modified.
It includes the references to the scfftm and csfftm subroutines
that are called within fft.f90.

# Description of tp6

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 8 – tp7 : Biconjugate gradient stabilized method

The principal.f90 source file contains the main program, which calls the bi-cgstab subprogram (bi-cgstab.f90 source file) to solve a linear system with multiple right-hand vectors :

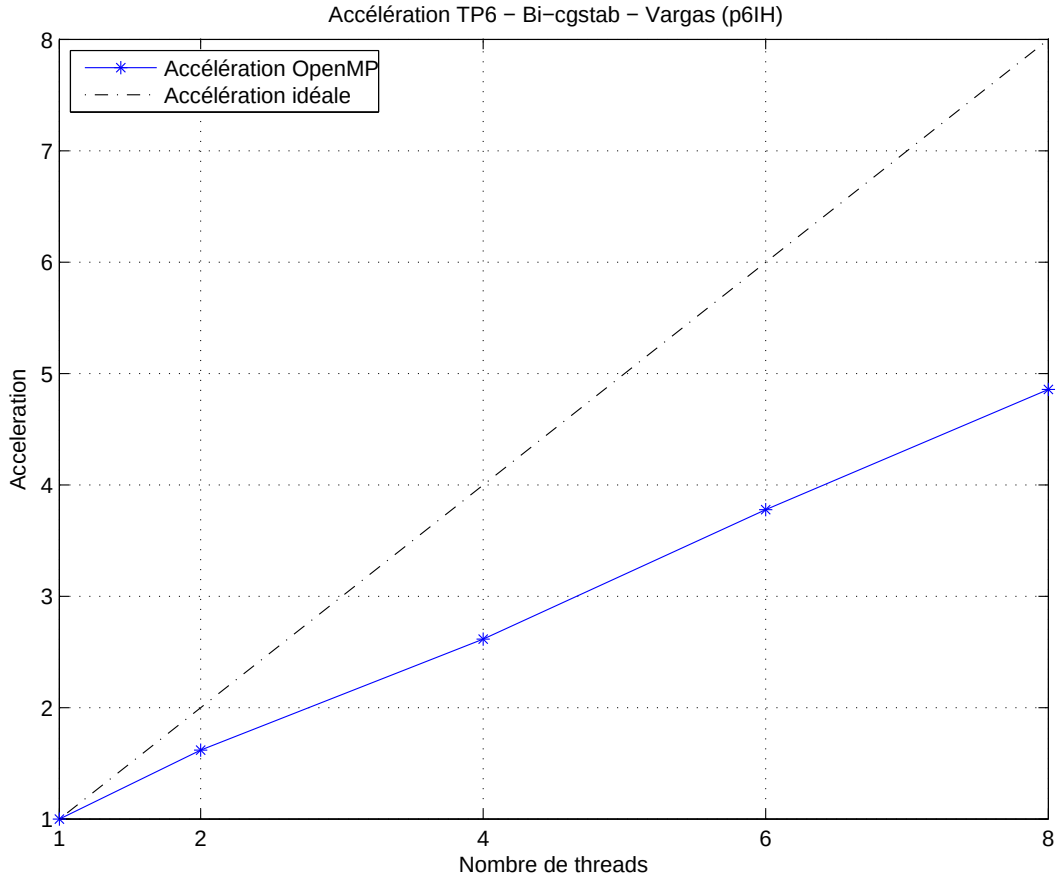
$$A \times x = B$$

with the biconjugate gradient stabilized method (BiCGSTAB)

1. Analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the principal.f90 and bi-cgstab.f90 source file. The bi-cgstab subprogram should be modified to behave as an orphaned procedure.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a *batch* job).
3. Plot the acceleration curve.

# Description of tp7

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 9 – tp8 : resolution equation of Poisson

The poisson.f90 and gradient_conjugue.f90 source files are used to solve the POISSON problem (1). The analytic solution $u_a(x, y)$ is given by :

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

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

We will use a center finite difference method in the $x$ direction followed by an FFT (sinus) in the $y$ direction. Let be $\tilde{u}$ and $\tilde{b}$ the result of the FFT of $u$ and $b$ respectively and apply the FFT to POISSON (1) equation :

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

The sinus transformation $\widetilde{\frac{\partial^2 u}{\partial y^2}}$ of $\frac{\partial^2 u}{\partial y^2}$ operator is diagonal, each element representing an eigenvalue of the operator. These eigenvalues can be calculated analytically. Let $j = 1, \ldots, N_j$ represents the indice of discretization and $h_y$ the spatial discrezation in the $y$ direction, then the eigenvalues are :

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

Hence, in the basis composed of the eigenvectors, the POISSON problem turns out to be a resolution of $N_j - 2$ symetric tridiagonal linear systems (use Conjugate Gradient method to solve them) of size $(N_i - 2) \times (N_i - 2)$ each, where $N_i$ represents the number

of discretization points in the $x$ direction :

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

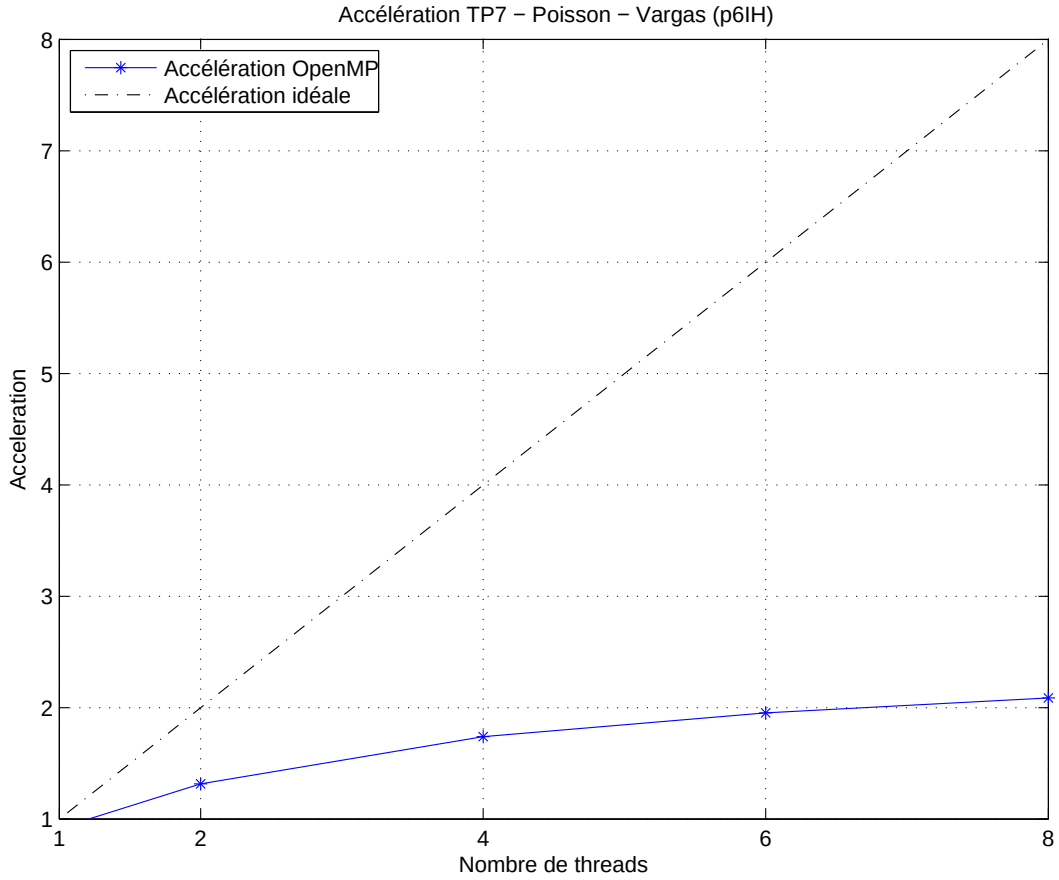
where $c_x = \frac{1}{h_x^2}$, $c_y = \frac{1}{h_y^2}$, $d_j = 2c_x + \mathrm{vp}_j$ and $h_x$ is the spatial discretization in the $x$ direction. The term CL deals with boundary conditions.

Finally, $(N_i - 2)$ independent reverse FFT of $\tilde{u}$ gives the final solution in the canonical basis.

1. Analyze the variable data-sharing attributes and insert the appropriate OpenMP directives in the poisson.f90 and gradient_conjugue.f90 source file. The gradient_conjugue subprogram should be modified to behave as an orphaned procedure, with only one parallel region.
2. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a *batch* job).
3. Plot the acceleration curve.

Please note : the file c06haf.o should not be modified. It includes the reference to the subroutine c06haf (sinus forward and reverse FFT) called within poisson.f90.

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |



## 10 – tp9 : nested loops with dependancies

The source file dependance.f90 contains two nested loops.

1. Analyse the data dependancy of the nested loops (e.g. are iterations independent one from each other ?). If loop parallelization is done nevertheless, what happens ?
2. Insert the appropriate OpenMP directives in the dependance.f90 source file. Take care about thread synchronization, to avoid breaking dependancies.
3. Analyze the parallel performance with 2, 4, 6, 8 threads referring to a sequential execution (please do a dedicated execution by submitting a batch job). The execution is validated if the norm printed during execution is 0...
4. Plot the acceleration curve.

# Description of tp9

| Nb. of threads | Elapsed time | Speedup |
|---|---|---|
| seq. | | |
| 1 | | |
| 2 | | |
| 4 | | |
| 6 | | |
| 8 | | |