TP 1 : Préparation

A chaque étape, regarder les messages affichés pour voir si tout s'est bien passé !

1. Récupérer l'archive `TP1.tar.gz` et extraire les fichiers.
2. Ouvrir un terminal et se placer dans le répertoire `I03_TP1` qui vient d'être créé
3. préparer la compilation du code du TP avec les commandes :

   mkdir -p build
   cd build
   cmake ../src
   cd ..

4. Se remettre dans le répertoire `I03_TP1` et compiler:

   make -C build

5. Executer le code avec la commande:
   ./build/PoissonSeq

6. A la fin de l'exécution, les résultats sont sauvegardés au format VTK dans un répertoire "results_..." (le nom précis est affiché à l'écran)

Si on modifie un ou plusieurs fichiers sources (dans le sous-répertoire src), il faut recompiler (point 4).

Si on ajoute un nouveau fichier ou on enlève un fichier existant (dans le sous-répertoire src), il faut adapter les fichiers CMakeLists.txt et refaire les points 3 et 4.

Mesure du temps de calcul global

Afficher le temps de calcul global avec `time` :

time ./build/PoissonSeq

A l'écran:

```
real    0m30,283s
user    0m30,186s
sys     0m0,096s
```

- Avantage : n'est pas intrusif

  pas besoin de modifier le code, ni de le compiler avec des options spécifiques.

- Désavantage : donne une information globale

  on ne sait pas dans quelle partie du code, on passe peu/beaucoup de temps, ni pourquoi.

Mesure plus précise : utilisation d'outils de "profiling"

Exemple : `gprof`

Fait partie de la famille gcc/g++/gfortran.

- Avantage : calcule le nombre d'appels de chaque fonction et le (pourcentage du) temps qui y est passé
- Avantage : n'est pas très intrusif (pas besoin de modifier le code, mais il faut le recompiler avec une option spécifique: -pg).
- Désavantage : le temps passé dans une fonction est peu précis dans une fonction "courte"
- Désavantage : ne mesure pas le temps dans les différentes parties d'une fonction.
- Désavantage : ne rentre pas dans les librairies dynamiques.

Mode de fonctionnement:

Ajoute dans chaque fonction, un comptage du nombre d'appels de cette fonction et évalue statistiquement le temps passé dans cette fonction (tous les 0.01 secondes on enregistre dans quelle fonction on se trouve).

Utilisation de `gprof`

Recompiler en utilisant l'option -pg:

```
mkdir -p build_gprof
cd build_gprof
cmake -DCMAKE_CXX_FLAGS="-pg" ../src
cd ..
make -C build_gprof
```

Exécuter le code:

```
./build_gprof/PoissonSeq
```

Collecter les mesures

```
gprof ./build_gprof/PoissonSeq
```

Mesure plus précise : utilisation d'outils de "profiling"

Exemple : `perf`

Outil spécifique linux

- Avantage : très puissant (mesure le temps passé dans une fonction, une instruction C/C++/fortran, une instruction binaire, multiples indicateurs de performance)
- Avantage : non intrusif
- Désavantage : plus compliqué à utiliser
- Désavantage : ne rentre pas toujours dans les librairies dynamiques.
- Désavantage : nécessite que la machine soit configurée correctement.

Outil á privilégier quand c'est possible

Mode de fonctionnement:

Enregistre les événements dans le noyau Linux, évalue statistiquement le temps passé dans les fonctions et les instructions.

Utilisation simple de `perf`

Recompiler en utilisant l'option -pg:

```
mkdir -p build_gprof
cd build_gprof
cmake -DCMAKE_CXX_FLAGS="-pg" ../src
cd ..
make -C build_gprof
```

Exécuter le code:

```
./build_gprof/PoissonSeq
```

Collecter les mesures

```
gprof ./build_gprof/PoissonSeq
```