

# TP PRECIS

# TP RARE-BLAS

*Objectif : l'objectif du TP est de prendre en main une librairie de calcul d'algèbre linéaire plus précise qu'une librairie "ordinaire".*

## Exercice 1. Découverte de l'environnement de travail

*Objectif : l'objectif de l'exercice est de mettre en place l'environnement de travail.*

- Connexion sur le serveur du meso-centre :

```
\$> ssh tpfrejus@194.57.87.10
```

- Connexion à un noeud de calcul :

```
\$>qlogin -q all.q -l proctype=sandybridge
```

- Création du répertoire de travail :

```
\$>mkdir /Work/Users/tpfrejus/<mon_nom>
\$>cd /Work/Users/tpfrejus/<mon_nom>
```

- Copie des fichiers :

```
\$>cp -r /Work/Users/tpfrejus/dparello/precis2017/work/ ./
```

- Arborescence :

```
.
|-blas          /* Test des libraires */
|-.dotgen      /* Generateur de données pour DOT */
|-.exactsum
|-.gemvgen     /* Generateur de données pour GEMV */
|-.linearalgebra
|-.precis      /* Code de test (Répertoire de travail) */
|---build
|---datadot
|---datagemv
|---datasum
|---run
|---scripts
|---src
|-reprodblas  /* Librairie de BLAS reproductible */
|-rtnblas     /* Librairie RARE-BLAS */
|-.sumgen     /* Generateur de données pour la somme */
|-.summation  /* Librairie de fonctions pour la somme */
```

- Fixer les bonnes variables d'environnement :

```
\$>source ~/dparello/env.bashrc
```

- Compilation :

```
\$>make
```

## Exercice 2. Quelques algorithmes de sommation

*Objectif : tester quelques algorithmes de sommation*

- Se placer dans le répertoire `precis` et observer le fichier de configuration des paramètres de l'expérience :

```
\$> less scripts/parameters.py
```

- Compiler et lancer l'expérience :

```
\$>make
\$>make generateresults
```

- Observer le fichier généré `testresultscpu.m` :

Les champs qui nous intéressent sont les champs 8 et 9 fournissant respectivement le résultat de l'algorithme et résultat exact arrondi au plus près.

- Identifier les algorithmes fournissant une somme arrondie au plus près :
- Faire varier le conditionnement et la taille des vecteurs et observer les résultats numériques ainsi que la performance (la colonne 10 indique le temps d'exécution en nombre de cycles processeur).

## Exercice 3. Quelques algorithmes de produits scalaires

*Objectif : réaliser quelques mesures de performances*

- Se déconnecter puis se reconnecter en réservant 16 coeurs sur le noeud de calcul :

```
\$>qlogin -q all.q -l proctype=sandybridge -pe openmp 16
\$>source ~/dparello/env.bashrc
```

Ne pas oublier de re-configurer l'environnement :

```
\$>source ~/dparello/env.bashrc
```

- Se placer dans le répertoire `precis` et modifier le fichier `scripts/parameters.py` afin de tester les algorithmes de produits scalaires
- Lancer la génération de résultats :

```
\$>make generateresults
```

- Observer le fichier généré `testresultscpu.m` (la colonne 10 indique de temps d'exécution en nombre de cycles processeur).
- Lancer la génération de graphiques et observer les performances :

```
\$>make generateplots
\$>make showplots
```

L'axe des ordonnées représente le temps d'exécution en cycles divisé par la taille des vecteurs.

- Modifier le fichier `scripts/parameters.py` afin de tester les algorithmes de produits scalaires en version parallèles (OpenMP : `OMP_THREADS = [4, 8, 16]`)
- Lancer la génération de résultats et de graphiques et observer les performances :

```
\$>make generateresults
\$>make generateplots
\$>make showplots
```

*Il est préférable de copier sur sa machine les fichiers `testresultscpu.m.eps` pour visualiser les graphiques*

#### **Exercice 4. Quelques algorithmes de produits matrice-vecteur**

*Objectif : réaliser quelques mesures de performances*

- Se placer dans le répertoire `precis` et modifier le fichier `scripts/gemmv_parameters.py` afin de tester les algorithmes de produits matrice-vecteur
- Lancer la génération de résultats :

```
\$>make generateresultsgemv
```

- Observer le fichier généré `testresultscpu.m` :  
La colonne 10 indique de temps d'exécution en nombre de cycles processeur.
- Lancer la génération de graphiques et observer les performances :

```
\$>make generateplotsgemv
\$>make showplots
```

L'axe des ordonnées représente le temps d'exécution en cycles divisé par la taille des vecteurs.

- Modifier le fichier `scripts/gemv_parameters.py` afin de tester les algorithmes de produits scalaires en version parallèles (OpenMP : `OMP_THREADS = [4, 8, 16]`).
- Lancer la génération de résultats et de graphiques et observer les performances :

```
\$>make generateresults
\$>make generateplots
\$>make showplots
```

#### **Exercice 5. Ultimate non-reproducible experience**

- Exercice : utiliser une fonction de la librairie pour un de vos codes de calcul possédant une sensibilité particulière à la précision.