# Calcul de Pi

Avant de commencer le TP, faites :

```
source environnement.sh
```

Cela permet aux commandes `mpicc` et `mpirun` d'être exécutées dans votre terminal. Pour compiler les programmes utilisez le compilateur `mpicc` :

```
mpicc pi.c -o pi
```

Pour lancer le programme `pi` sur 4 processeurs utilisez la commande `mpirun` :

```
mpirun -np 4 ./pi
```

## I. MÉTHODE DÉTERMINISTE

$$\int_0^1 \frac{1}{1+x^2} = \frac{\pi}{4} \tag{1}$$

Nous calculerons $\pi$ en calculant l'intégrale ci-dessus avec la méthode des trapèzes.

### A. Mesure de scalabilité

On utilisera le programme `pi_scalabilite.c`.

Comparez les temps de calcul et la précision pour $n = 5000$ et $n = 1$ milliard de points d'intégration sur un seul processeur.

Étudiez la scalabilité avec $n = 1$ milliard pour 1, 2, et 4 processus.

### B. Équilibrage de charge

Le découpage du domaine en zones continues de tailles identiques est parfaitement adapté lorsque le calcul de chacun des points prend un temps de calcul constant. Cependant, lorsque le temps calcul n'est pas homogène, le temps de restitution du calcul sera égal au temps passé à calculer la partie la plus longue. Il en résultera une mauvaise efficacité parallèle.

Lorsque le temps de calcul de chacun des points est variable, il se pose le problème d'*équilibrage de charge*.

Nous allons artificiellement déséquilibrer le calcul entre les différents processeurs. Pour cela, nous ajoutons une pause (`sleep`) proportionnelle à la valeur de l'intégrale partielle calculée : plus la valeur est grande plus la pause est longue. Ces modifications ont été faites dans `pi_charge.c`.

Modifiez le programme afin de minimiser le déséquilibre dans le temps de calcul sur chacun des processus MPI.

## II. MÉTHODE STOCHASTIQUE

On cherche à calculer $\pi$ avec un algorithme de type Monte Carlo.

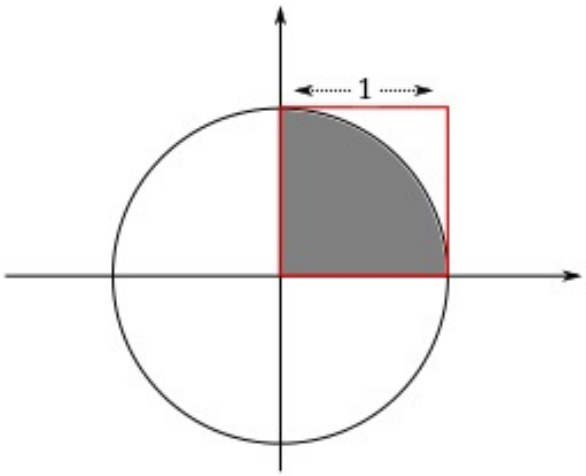


FIGURE 1: La surface grise vaut $\pi/4$

L'aire du cercle de rayon unité est $\pi$, donc son aire sur $(x, y) \in [0, 1] \times [0, 1]$ est $\pi/4$. La fonction tracée dans le carré rouge de la figure 1 est $y = \sqrt{1 - x^2}$, donc

$$\int_0^1 \sqrt{1 - x^2} dx = \frac{\pi}{4} \tag{2}$$

*Algorithme*

1. $N_{\text{in}} = 0$ ; $N_{\text{total}} = 0$
2. Tirer $x$ uniformément dans $[0, 1]$

3. Tirer $y$ uniformément dans $[0, 1]$
4. Si $x^2 + y^2 < 1$, $(x, y)$ est dans le cercle et on incrémente $N_{\text{in}}$
5. On incrémente $N_{\text{total}}$
6. Go to 2
7. $\frac{N_{\text{in}}}{N_{\text{total}}} = \frac{\pi}{4}$

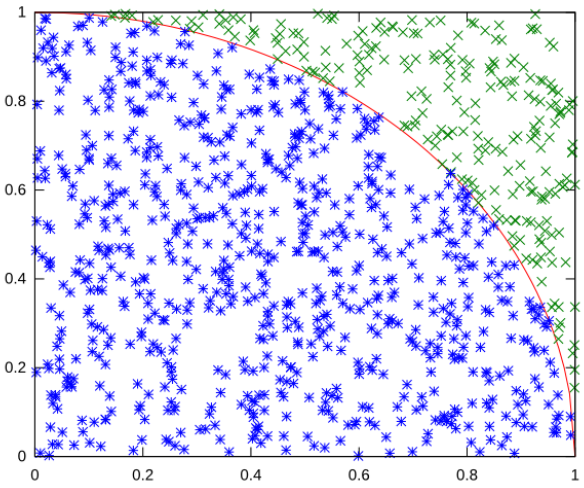


FIGURE 2: La valeur de $\pi/4$ est égale au rapport du nombre de points bleus sur le nombre total de points.

*Parallélisme*

On utilisera une implémentation de type client/serveur. Le *serveur* est un processus qui collecte les résultats partiels calculés par les *clients*. Chaque client calcule un paquet de 10 millions de tirages et estime la valeur de $\pi$. Cette valeur (échantillon) est envoyée au serveur, et à la réception du message le serveur retourne une instruction au client lui donnant l'ordre de recalculer un nouvel échantillon ou de s'arrêter.

Puisque tous les échantillons sont indépendants, ils ont une distribution gaussienne centrée sur la valeur estimée de $\pi$. La barre d'erreur est donc proportionnelle à $1/\sqrt{N}$ ou $N$ est le nombre de tirages.

Avec cette méthode, plus le nombre de tirages est grand, plus le résultat est précis. À temps de restitution constant, on utilisera la parallélisme pour améliorer la précision du

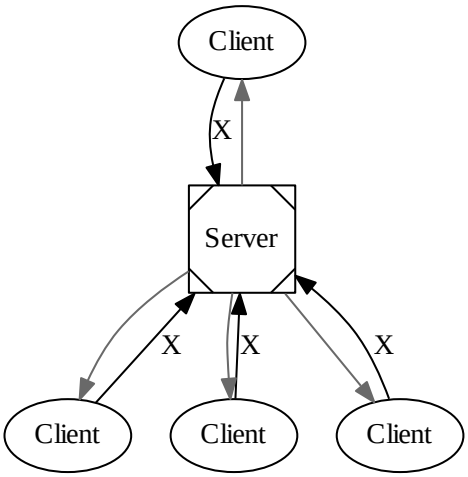


FIGURE 3: Modèle client/serveur

calcul. Notons que cet algorithme est parfaitement scalable (*embarrassingly parallel*), et peut utiliser sans problème plusieurs dizaines de milliers de cœurs.

Dans le programme `pi_stochastique.c`, on vérifiera que pour une durée de 10 secondes, le nombre de tirages augmente linéairement avec le nombre de clients.

## III. MODÈLE CLIENT/SERVEUR POUR LA MÉTHODE DÉTERMINISTE

Lorsque le temps de calcul est imprévisible, le modèle client-serveur permet d'ajuster dynamiquement la charge de calcul sur chacun des processus.

Le processus de rang `0` établit une liste de tâches à effectuer. Chaque tâche va être représentée ici par les bornes d'un petit intervalle sur lequel on va calculer l'intégrale par la méthode des trapèzes. Lorsqu'un client se connecte au serveur, il envoie le résultat qu'il a calculé puis reçoit une nouvelle tâche à effectuer.

Pour simuler un programme dont le temps de calcul est imprévisible, on a ajouté une pause aléatoire après le calcul de la tâche.

Modifiez le fichier `pi_client_server.c` pour faire fonctionner cet exemple.

Remarque : pour que l'équilibrage de charge fonctionne, il faut que le nombre de tâches sont bien supérieur au nombre de processus.