

INSTITUT SUPÉRIEUR D'INFORMATIQUE, DE MODÉLISATION  
ET DE LEURS APPLICATIONS -ISIMA-

## Compte rendu

Génie logicielle et systèmes informatiques (F2)

---

# TP5 : Flux stochastiques – modèles du hasard - Nombres quasi-aléatoires - Parallélisme

---

*Réalisé par :*

MLLE. SOUMYA KADDOURI

M. ABOUBAKR EL HARRAK

*Encadré par :*

PR. DAVID HILL





## Table des matières

<b>1</b>	<b>Introduction générale</b>	<b>4</b>
<b>2</b>	<b>Installation de la bibliothèque CLHEP</b>	<b>4</b>
<b>3</b>	<b>L'archivage et la restauration des statuts pour le générateur Mersenne Twister</b>	<b>6</b>
<b>4</b>	<b>Sauvegarde de 10 États du Générateur Mersenne Twister</b>	<b>6</b>
<b>5</b>	<b>Calcul de PI par la méthode de Monte Carlo avec Restauration des Statuts</b>	<b>7</b>
5.1	Comparaison entre deux processeurs différents . . . . .	8
5.2	Analyse comparative des Résultats de Simulation . . . . .	8
5.2.1	Caractéristiques des Processeurs . . . . .	8
5.2.2	Comparaison des Résultats . . . . .	9
5.2.3	Résultats d'Exécution . . . . .	9
5.2.4	Analyse des Résultats . . . . .	10
5.2.5	Conclusion . . . . .	10
<b>6</b>	<b>Parallélisation des Répliques de Monte Carlo pour le Calcul de PI</b>	<b>11</b>
6.1	Conclusion . . . . .	11
<b>7</b>	<b>Conclusion générale</b>	<b>12</b>
	<b>Annexe</b>	<b>13</b>

## Table des figures

1	Verification de mis a jour de la date . . . . .	4
2	Résultat d'exécution de testRandom.cc . . . . .	5
3	Résultat de la question 2 . . . . .	6
4	Résultat de la question 3 . . . . .	7
5	Résultat de la question 4 (Machine 1) . . . . .	8
6	Résultat de la question 4 (Machine 2) . . . . .	8
7	Résultat de la question 5 . . . . .	11

## 1 Introduction générale

Ce compte rendu décrit notre expérience lors du TP axé sur l'utilisation de la bibliothèque CLHEP pour la simulation en physique des hautes énergies. Notre objectif principal était de se familiariser avec cette bibliothèque de taille professionnelle et de réaliser un calcul stochastique distribué. Nous avons suivi attentivement les consignes, installé la bibliothèque, et effectué des tests avec des codes C++ fournis. Les étapes suivantes ont impliqué la manipulation de statuts pour le générateur Mersenne Twister, la réalisation de répliques du calcul de PI, et enfin, la parallélisation de ces calculs. Ce compte rendu détaillera notre approche et les résultats obtenus.

## 2 Installation de la bibliothèque CLHEP

Nous avons bien suivi les conseils pour installer la bibliothèque CLHEP. Tout d'abord, nous avons regardé où se trouvaient les différentes parties de la bibliothèque, comme les sources et les codes de test. Ensuite, nous avons compilé et lié la bibliothèque, en nous assurant de vérifier la date des fichiers que nous avons obtenus (.a et .so). On nous a dit que si tout s'était bien passé, ces fichiers seraient datés du jour même.

```
souma20@SoumyaKaddouri:~/TP5-IDM/Random$ ls -l lib
total 24052
-rw-r--r-- 1 souma20 souma20 18035388 Nov 21 14:36 libCLHEP-Random-2.1.0.0.a
-rwxr-xr-x 1 souma20 souma20 6587648 Nov 21 14:36 libCLHEP-Random-2.1.0.0.so
```

FIGURE 1 – Verification de mis a jour de la date

Après cela, nous avons regardé les nouveaux dossiers qui ont été créés et les endroits où les bibliothèques ont été stockées. Pour être sûr que tout fonctionnait correctement, nous avons testé la bibliothèque en utilisant le code C++ fourni à la fin du TP et le code de test (testRandom.cc).

Voici quelques figures de ce qui a été obtenu :

```

Shooting an array of 5 flat numbers ...

0.875564 0.780428 0.0335104 0.881222 0.606802

----- Press <ENTER> to continue -----

----- Shooting test skeeping the generator -----

>>> Select a Random Engine <<<

a. HepJamesRandom (default)
b. Rand
c. DRand48
d. Ranlux
e. Ranlux64
f. Ranecu
g. Hurd160
h. Hurd288
i. MTWist
j. Ranshi
k. DualRand
l. TripleRand

> |

----- Shooting test skeeping the generator -----

>>> Select a Random Engine <<<

a. HepJamesRandom (default)
b. Rand
c. DRand48
d. Ranlux
e. Ranlux64
f. Ranecu
g. Hurd160
h. Hurd288
i. MTWist
j. Ranshi
k. DualRand
l. TripleRand

> l

Flat ]0,1[      : 0.703114
Flat ]0,5[      : 4.40069
Flat ]-5,3[     : -0.431344
Exp (m=1)       : 1.52181
Exp (m=3)       : 1.48869
Gauss (m=1)     : -1.0078
Gauss (m=3,v=1) : 1.62241
Wigner(1,0.2)   : 1.28659
Wigner(1,0.2,1) : 0.581033
Wigner2(1,0.2)  : 0.947029
Wigner2(1,0.2,1): 0.941853
IntFlat [0,99[   : 47
IntFlat [-99,37[ : 34
Poisson (m=3.0)  : 2
Binomial(n=1,p=0.5) : 0
Binomial(n=-5,p=0.3): -1
ChiSqr (a=1)     : 0.00638527
ChiSqr (a=-5)    : -1
Gamma (k=1,l=1)  : 0.579851
Gamma (k=3,l=0.5) : 7.78499
StudT (a=1)      : 5.36813
StudT (a=2.5)    : 0.324093

Shooting an array of 5 flat numbers ...

0.0778183 0.667179 0.244365 0.138704 0.996181

```

FIGURE 2 – Résultat d'exécution de testRandom.cc

En résumé, nous avons suivi les instructions pour installer CLHEP et nous avons vérifié que tout fonctionnait correctement en testant la bibliothèque avec les codes fournis.

### 3 L'archivage et la restauration des statuts pour le générateur Mersenne Twister

Lors de cette étape, nous avons utilisé la méthode 'saveStatus' pour archiver quelques fichiers de statuts du générateur Mersenne Twister (MT). Pour ce faire, nous avons séparé ces statuts après un petit nombre de tirages, en choisissant, par exemple, 10 nombres. L'objectif était de tester la restauration à un statut donné en utilisant la méthode 'restoreStatus'.

La restauration des statuts nous a permis de retrouver les mêmes nombres pseudo-aléatoires, comme mentionné dans l'énoncé du TP, afin de reproduire une séquence spécifique, déboguer le code, et assurer la reproductibilité des résultats. La répétabilité bit à bit a été soulignée comme étant cruciale dans le contexte du débogage informatique, mettant en évidence que si nous ne pouvons plus déboguer, l'informatique perd sa fonctionnalité.

La Figure ci-dessous montre le résultat de notre code. Nous avons initialement généré 10 nombres aléatoires en utilisant la méthode 'flat()' du générateur MT. Ensuite, nous avons sauvegardé le statut de notre générateur, tiré 10 autres nombres, puis restauré le statut sauvegardé du générateur et tiré à nouveau 10 nombres aléatoires. Nous avons remarqué que les 10 nombres tirés après la sauvegarde du statut du générateur sont les mêmes que ceux générés après la restauration du statut.

```
=> The first 10 random numbers generated:
0.176117 0.0202835 0.863258 0.118952 0.463625 0.0618998 0.823987 0.571751 0.0917234 0.777519
=> The 10 random numbers after saving the status:
0.54103 0.42616 0.0279915 0.282957 0.24201 0.227266 0.342152 0.0723833 0.0445422 0.655051
=> The 10 random numbers after restoring the status:
0.54103 0.42616 0.0279915 0.282957 0.24201 0.227266 0.342152 0.0723833 0.0445422 0.655051
```

FIGURE 3 – Résultat de la question 2

### 4 Sauvegarde de 10 États du Générateur Mersenne Twister

Dans cette partie du TP, nous avons écrit un code pour sauvegarder 10 statuts distincts du générateur Mersenne Twister. Le code utilise la bibliothèque CLHEP et la classe MTWistEngine.

Le code génère 2 milliards de nombres aléatoires à l'aide de la méthode 'flat()'. Après chaque ensemble de 2 milliards de tirages, le statut actuel du générateur est sauvegardé dans un fichier distinct. Les fichiers de statut sont nommés "status0.txt", "status1.txt", etc (voir Figure 4).

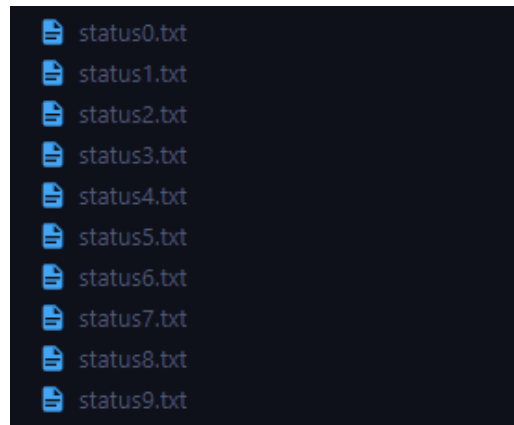


FIGURE 4 – Résultat de la question 3

Cette approche assure la conservation des différents états du générateur, ce qui peut être utile pour reproduire des séquences spécifiques ou pour assurer la reproductibilité des résultats.

## 5 Calcul de PI par la méthode de Monte Carlo avec Restauration des Statuts

Dans cette section du TP, nous avons développé un code pour enchaîner 10 réplifications d'un calcul de PI en utilisant la méthode de Monte Carlo. Chaque simulation consiste à tirer 1 milliard de points afin de calculer une estimation de PI. Les 2 milliards de nombres requis pour chaque réplification ont été préalablement calculés à la question précédente en sauvegardant les statuts du générateur Mersenne Twister.

L'intérêt de cette approche réside dans la conservation de l'indépendance des flux pseudo-aléatoires pour chaque réplification. En restaurant le générateur dans des états pré-calculés, nous assurons que chaque simulation utilise un flux pseudo-aléatoire différent, préservant ainsi l'aspect stochastique du calcul de PI.

La mesure du temps de calcul séquentiel pour les 10 réplifications est également intégrée au code. Nous utilisons la bibliothèque '<chrono>' pour capturer le moment de début et de fin de l'exécution, permettant ainsi de quantifier le temps écoulé pour l'ensemble des simulations.

Le choix de générer des fichiers de statuts après les avoir préalablement calculés et de les utiliser dans le cadre d'un calcul séquentiel vise à démontrer la reproductibilité des simulations. Cette approche offre la possibilité de retrouver exactement les mêmes conditions initiales, permettant ainsi la comparaison et la validation de résultats pour des besoins de débogage ou de vérification de la cohérence des calculs effectués.



## 5.1 Comparaison entre deux processeurs différents

La figure ci-dessous montre le résultat obtenu pour l'exécution du code sur les deux machines :

```
souma20@SoumyaKaddouri:~/TP5-IDM/Random$ g++ -o question4 question4.cc -I ./include ./lib/libCLHEP-Random-2.1.0.0.a
souma20@SoumyaKaddouri:~/TP5-IDM/Random$ ./question4
Estimation of PI: 3.14173
Estimation of PI: 3.14163
Estimation of PI: 3.14158
Estimation of PI: 3.14151
Estimation of PI: 3.14154
Estimation of PI: 3.14166
Estimation of PI: 3.14159
Estimation of PI: 3.14161
Estimation of PI: 3.14154
Estimation of PI: 3.1415
Elapsed time: 166.844 seconds
```

FIGURE 5 – Résultat de la question 4 (Machine 1)

```
aboubakr@ubuntu2004:/home/ubuntu/TP_IDM/tp_5/CLHEP/Random$ g++ -o question4 question4.cc -I ./include/ ./lib/libCLHEP-Random-2.1.0.0.a
aboubakr@ubuntu2004:/home/ubuntu/TP_IDM/tp_5/CLHEP/Random$ ./question4
Estimation of PI: 3.14173
Estimation of PI: 3.14163
Estimation of PI: 3.14158
Estimation of PI: 3.14151
Estimation of PI: 3.14154
Estimation of PI: 3.14166
Estimation of PI: 3.14159
Estimation of PI: 3.14161
Estimation of PI: 3.14154
Estimation of PI: 3.1415
Elapsed time: 133.238 seconds
aboubakr@ubuntu2004:/home/ubuntu/TP_IDM/tp_5/CLHEP/Random$
```

FIGURE 6 – Résultat de la question 4 (Machine 2)

## 5.2 Analyse comparative des Résultats de Simulation

Dans le cadre de l'expérience de simulation, nous avons exécuté un calcul de Monte Carlo pour estimer la valeur de  $\pi$  sur deux machines distinctes : Machine 1 équipée d'un processeur Intel i7-1065G7, et Machine 2 dotée d'un processeur AMD Ryzen 5 5600H. Les résultats ont été comparés en termes de temps d'exécution séquentiel.

### 5.2.1 Caractéristiques des Processeurs

#### Machine 1 - Intel i7-1065G7 :

- Cœurs Physiques : 4 (Quad-Core)
- Threads : 8
- Fréquence de Base : 1.3 GHz
- Fréquence Boost : 3.9 GHz
- Cache L3 partagé : 8 MB
- Lithographie : 10 nm

**Machine 2 - AMD Ryzen 5 5600H :**

- Cœurs Physiques : 6 (Hexa-Core)
- Threads : 12
- Fréquence de Base : 3.3 GHz
- Fréquence Boost : 4.2 GHz
- Cache L3 partagé : 16 MB
- Lithographie : 7 nm

**5.2.2 Comparaison des Résultats****1. Cœurs et Threads :**

- **Machine 1** : 4 cœurs physiques, 8 threads
- **Machine 2** : 6 cœurs physiques, 12 threads
- **Observation** : La Machine 2 dispose d'une plus grande capacité de traitement parallèle grâce à ses cœurs et threads supplémentaires.

**2. Fréquence de l'Horloge :**

- **Machine 1** : Base 1.3 GHz, Boost 3.9 GHz
- **Machine 2** : Base 3.3 GHz, Boost 4.2 GHz
- **Observation** : La Machine 2 a une fréquence de base significativement plus élevée, ce qui peut contribuer à des calculs plus rapides.

**3. Cache L3 :**

- **Machine 1** : 8 MB partagés
- **Machine 2** : 16 MB partagés
- **Observation** : La Machine 2 dispose d'un cache L3 plus grand, permettant un accès plus rapide aux données partagées entre les cœurs.

**5.2.3 Résultats d'Exécution**

Les temps d'exécution séquentiels sont les suivants :

- **Machine 1** : 166.844 secondes
- **Machine 2** : 133.238 secondes

#### 5.2.4 Analyse des Résultats

1. **Performances Globales** : La Machine 2 (AMD Ryzen) a démontré des performances meilleures avec un temps d'exécution inférieur.

2. **Impact des Caractéristiques du Processeur** : Les caractéristiques supérieures de la Machine 2, notamment le nombre de cœurs/threads, la fréquence de base plus élevée, et un cache L3 plus important, ont probablement contribué à l'amélioration des performances.

3. **Lien avec l'expérience de simulation** : L'expérience impliquait des calculs intensifs, où la capacité de traitement parallèle et la vitesse d'exécution sont cruciales. Les caractéristiques avancées de la Machine 2 ont probablement eu un impact positif sur la réduction du temps d'exécution.

#### 5.2.5 Conclusion

La comparaison des résultats entre les deux machines met en lumière l'influence significative des caractéristiques du processeur sur les performances d'une simulation. La Machine 2, avec son architecture plus avancée, a surpassé la Machine 1 dans ce contexte spécifique de calcul intensif, démontrant ainsi l'importance de choisir un matériel adapté aux charges de travail spécifiques.

## 6 Parallélisation des Réplifications de Monte Carlo pour le Calcul de PI

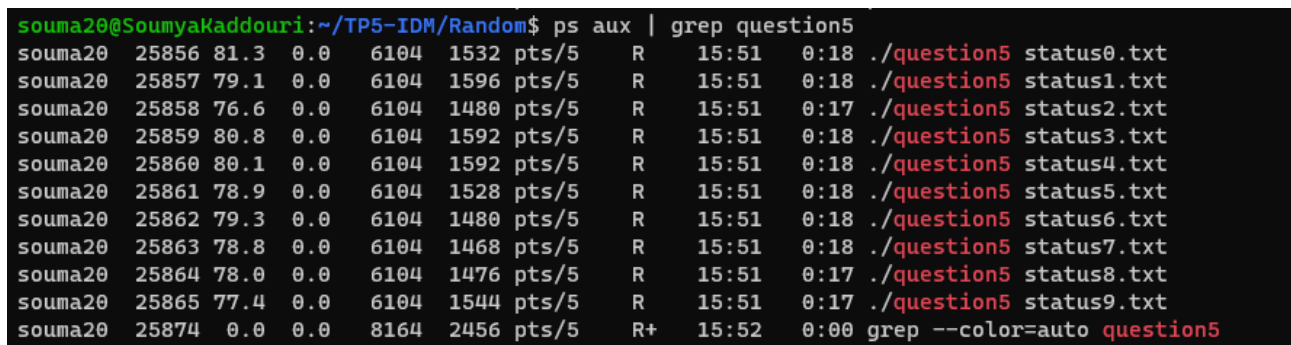
Dans cette phase du TP, nous avons cherché à paralléliser l'exécution en utilisant la technique du "sequence splitting" pour calculer simultanément les 10 réplifications d'une simulation de Monte Carlo visant à estimer PI. L'approche adoptée était de maintenir notre programme séquentiel tout en utilisant le système multitâche pour réaliser du "SPMD" (Simple Program Multiple Data).

Pour ce faire, nous avons créé un programme C++ prenant en paramètre le fichier de statut et calculant une estimation de PI pour cette séquence aléatoire. Un script Bash a été développé pour lancer 10 instances de ce programme en arrière-plan, chacune avec un fichier de statut différent. Les estimations de PI résultantes ont été sauvegardées dans les fichiers "EstimationPI-0", "EstimationPI-1", etc., jusqu'à "EstimationPI-9".

Nous avons confirmé le lancement réussi des processus en utilisant la commande

```
ps aux | grep question5
```

La Figure ci-dessous illustre cette liste, mettant en évidence les 10 processus lancés en parallèle.



```
souma20@SoumyaKaddouri:~/TP5-IDM/Random$ ps aux | grep question5
souma20 25856 81.3 0.0 6104 1532 pts/5 R 15:51 0:18 ./question5 status0.txt
souma20 25857 79.1 0.0 6104 1596 pts/5 R 15:51 0:18 ./question5 status1.txt
souma20 25858 76.6 0.0 6104 1480 pts/5 R 15:51 0:17 ./question5 status2.txt
souma20 25859 80.8 0.0 6104 1592 pts/5 R 15:51 0:18 ./question5 status3.txt
souma20 25860 80.1 0.0 6104 1592 pts/5 R 15:51 0:18 ./question5 status4.txt
souma20 25861 78.9 0.0 6104 1528 pts/5 R 15:51 0:18 ./question5 status5.txt
souma20 25862 79.3 0.0 6104 1480 pts/5 R 15:51 0:18 ./question5 status6.txt
souma20 25863 78.8 0.0 6104 1468 pts/5 R 15:51 0:18 ./question5 status7.txt
souma20 25864 78.0 0.0 6104 1476 pts/5 R 15:51 0:17 ./question5 status8.txt
souma20 25865 77.4 0.0 6104 1544 pts/5 R 15:51 0:17 ./question5 status9.txt
souma20 25874 0.0 0.0 8164 2456 pts/5 R+ 15:52 0:00 grep --color=auto question5
```

FIGURE 7 – Résultat de la question 5

De plus, nous avons vérifié que les fichiers "EstimationPI-i" contiennent effectivement les mêmes valeurs calculées dans la question précédente, démontrant ainsi la reproductibilité des résultats lors de l'exécution en parallèle.

### 6.1 Conclusion

L'importance de la parallélisation réside dans la réduction significative du temps de calcul, permettant une exploitation plus efficace des ressources disponibles sur un serveur multi-cœur. Cependant, cette approche n'est pas sans inconvénients. L'utilisation d'un script shell pour paralléliser le processus peut introduire des problèmes de gestion des ressources et de coordination entre les différents processus. De plus, cela peut entraîner une consommation élevée de ressources système et nécessiter une gestion attentive des dépendances entre les processus afin d'éviter des résultats incohérents.

## 7 Conclusion générale

En conclusion de ce TP, nous avons exploré divers aspects de la simulation stochastique en utilisant la bibliothèque CLHEP. Nous avons réussi l'installation, exploré l'archivage et la restauration des statuts pour le générateur Mersenne Twister, puis sauvegardé 10 états distincts pour des répliques ultérieures. Nous avons également implémenté un calcul de PI par la méthode de Monte Carlo, en étudiant la comparaison entre deux processeurs.

Dans cette expérience, nous avons abordé la parallélisation des répliques de Monte Carlo pour le calcul de PI, en adoptant la technique du "sequence splitting" avec l'utilisation d'un script shell. Cependant, il est essentiel de mentionner qu'il existe d'autres méthodes de parallélisation, telles qu'OpenMP, que nous avons récemment explorées dans le cours d'infrastructure distribuée pour le passage à l'échelle.

L'avantage d'approches comme OpenMP réside dans leur facilité d'utilisation et leur capacité à tirer parti de manière transparente des architectures multi-cœurs. Ces méthodes offrent une parallélisation plus intégrée et simplifiée des tâches, ce qui peut considérablement améliorer les performances, notamment pour des calculs intensifs.

Nous prévoyons d'approfondir notre compréhension de ces techniques de parallélisation, en particulier OpenMP, dans les futurs TP après les vacances. Cela nous permettra d'explorer davantage les avantages de la parallélisation pour des simulations plus complexes et de maximiser l'efficacité de nos calculs sur des ressources informatiques modernes.

## Annexe

Annexe 1    Lien vers Github

## Références

- [1] <<https://korben.info/tuto-pour-installer-linux-sous-windows-10-avec-wsl.html>>. Tuto pour installer Linux sous Windows 10 avec WSL.
- [2] Hill D. *Installation TIPS fo CLHEP*. ISIMA, 2023.