

Rapport d'élève ingénieur

TP 5 IDM

Filière : Génie logiciel et Systèmes Informatiques

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

*Présenté par : Antoine VITON*

Encadré par David Hill

Campus des Cézeaux. 1 rue de la Chebarde. TSA 60125. 63178 Aubière CEDEX

# Table des matières

I	Introduction . . . . .	4
II	Installation de la librairie CHELP (Question 1) . . . . .	5
	II.1 Manière Séquentielle . . . . .	5
	II.2 Manière parallèle . . . . .	5
III	Vérification de la répétabilité (Question 2) . . . . .	6
IV	Sauvegarde de statuts (Question 3) . . . . .	7
	IV.1 Explication du choix de réaliser trois milliard de tirage . . . . .	7
	IV.2 Explication de la sauvegarde . . . . .	7
V	Estimation du volume d'une sphère en séquentiel (Question 4) . . . . .	8
	V.1 Explication du principe de la simulation . . . . .	8
	V.2 Résultats . . . . .	9
VI	Estimation du volume d'une sphère en parallèle (Question 5) . . . . .	10
	VI.1 Avec la bibliothèque standard . . . . .	10
	VI.2 En utilisant un script bash . . . . .	11
VII	OpenMP (Question 6) . . . . .	12
	VII.1 Explication du principe d'OpenMP . . . . .	12
VIII	Conclusion . . . . .	14

# Table des figures

1	Configuration Huawei matebook D14 . . . . .	4
2	Temps d'exécution en séquentiel . . . . .	5
3	Temps d'exécution en parallèle . . . . .	5
4	Vérification répétabilité bit à bit . . . . .	6
5	Sauvegarde de 10 statuts espacés de 3 milliard de tirages . . . . .	7
6	Simulation de Monte Carlo pour estimer le volume d'une sphère . . . . .	8
7	10 réplifications séquentielles de la simulation de Monte Carlo . . . . .	9
8	Résultats obtenus et qualité de ces derniers lors de l'approximation du volume d'une sphère de rayon 1 . . . . .	9
9	Résultats obtenus avec la parallélisation en utilisant les threads de la bibliothèque standard . . . . .	10
10	Résultats obtenus avec la parallélisation en utilisant un script bash . . . . .	11
11	Utilisation d'OpenMP . . . . .	12
12	Résultats obtenus avec la parallélisation en utilisant OpenMP . . . . .	13

## I Introduction

Le but de ce TP est de découvrir la librairie CHELP à travers la mise en place d'une simulation de Monte Carlo, en générant des nombres pseudo-aléatoires à l'aide du générateur Mersenne Twister. En plus de cela nous paralléliserons les différentes répliques de notre simulation tout en veillant à utiliser des flux stochastiques indépendants.

Les différents extraits des exécutions ont été réalisés sur un Huawei matebook D14 avec la configuration que l'on peut retrouver sur la figure 1.

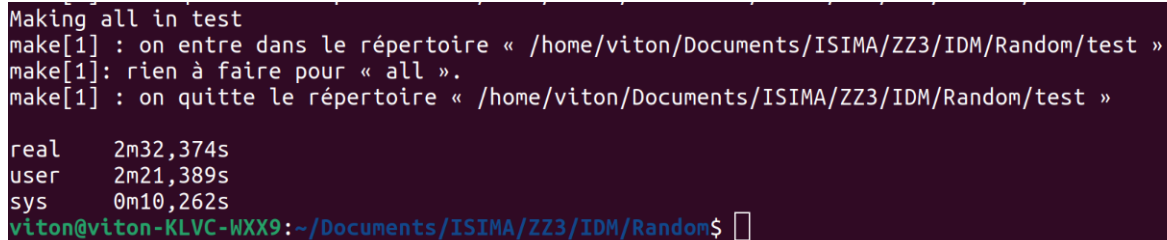
```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ lscpu
Architecture : x86_64
Mode(s) opératoire(s) des processeurs : 32-bit, 64-bit
Address sizes: 39 bits physical, 48 bits virtual
Boutisme : Little Endian
Processeur(s) : 8
Liste de processeur(s) en ligne : 0-7
Identifiant constructeur : GenuineIntel
Nom de modèle : Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz
Famille de processeur : 6
Modèle : 142
Thread(s) par cœur : 2
Cœur(s) par socket : 4
Socket(s) : 1
Révision : 12
Vitesse maximale du processeur en MHz : 4900,0000
Vitesse minimale du processeur en MHz : 400,0000
```

FIGURE 1 – Configuration Huawei matebook D14

## II Installation de la librairie CHELP (Question 1)

Pour réaliser ce TP il a fallu compiler la librairie CHELP en suivant les consignes.

### II.1 Manière Séquentielle



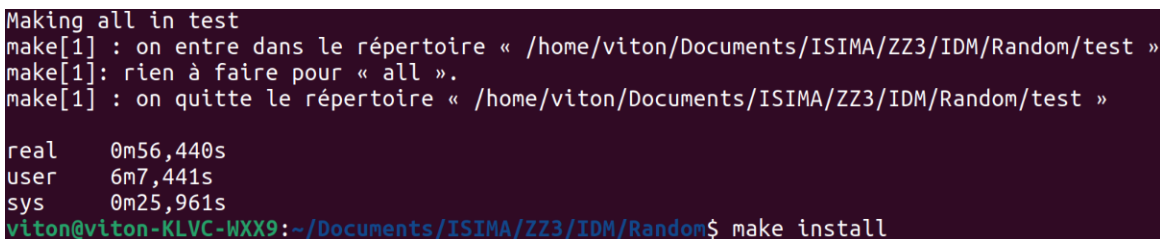
```
Making all in test
make[1] : on entre dans le répertoire « /home/viton/Documents/ISIMA/ZZ3/IDM/Random/test »
make[1]: rien à faire pour « all ».
make[1] : on quitte le répertoire « /home/viton/Documents/ISIMA/ZZ3/IDM/Random/test »

real    2m32,374s
user    2m21,389s
sys     0m10,262s
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/Random$
```

FIGURE 2 – Temps d'exécution en séquentiel

Le processus de compilation (commande make) est exécuté de façon classique, sans parallélisme. Le temps "real" (temps réel) pris est de 2m32,374s. Le temps utilisateur "user" est de 2m21,389s et le temps système "sys" est de 0m10,262s.

### II.2 Manière parallèle



```
Making all in test
make[1] : on entre dans le répertoire « /home/viton/Documents/ISIMA/ZZ3/IDM/Random/test »
make[1]: rien à faire pour « all ».
make[1] : on quitte le répertoire « /home/viton/Documents/ISIMA/ZZ3/IDM/Random/test »

real    0m56,440s
user    6m7,441s
sys     0m25,961s
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/Random$ make install
```

FIGURE 3 – Temps d'exécution en parallèle

Dans ce second cas, la commande make est utilisée avec le parallélisme, avec l'option -j pour activer plusieurs threads. Le temps "real" est ici 0m56,440s, ce qui montre une réduction significative du temps total d'exécution grâce au parallélisme. Le temps utilisateur "user" est cependant beaucoup plus élevé à 6m7,441s, ce qui reflète l'effort collectif des différents threads pour exécuter les tâches en parallèle. Le temps système "sys" est aussi un peu plus élevé à 0m25,961s, ce qui est attendu en raison de la gestion accrue des processus parallèles.

### III Vérification de la répétabilité (Question 2)

Nous avons ensuite testé la répétabilité bit à bit de la génération de nombres pseudo-aléatoires en utilisant Mersenne Twister contenu dans la librairie CHELP.

On teste aussi le fonctionnement des méthodes de sauvegarde et de restauration de statut du générateur Mersenne Twister.

```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./TP5 2
Réponse question : 2
Première séquence de 10 nombres :
0.176117 0.0202835 0.863258 0.118952 0.463625 0.0618998 0.823987 0.571751 0.0917234 0.777519
Deuxième séquence après restauration (doit être identique) :
0.176117 0.0202835 0.863258 0.118952 0.463625 0.0618998 0.823987 0.571751 0.0917234 0.777519
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$
```

FIGURE 4 – Vérification répétabilité bit à bit

Pour tester la répétabilité du générateur, on sauvegarde le statut du générateur dans un fichier, puis on tire dix nombres pseudo-aléatoires que l'on affiche, ensuite on restaure le statut depuis le fichier et on finit par tirer à nouveau 10 nombres et on les affiche également. On peut ainsi voir la trace de l'exécution sur la figure 4 où l'on obtient bien les deux mêmes séquences de nombres.

## IV Sauvegarde de statuts (Question 3)

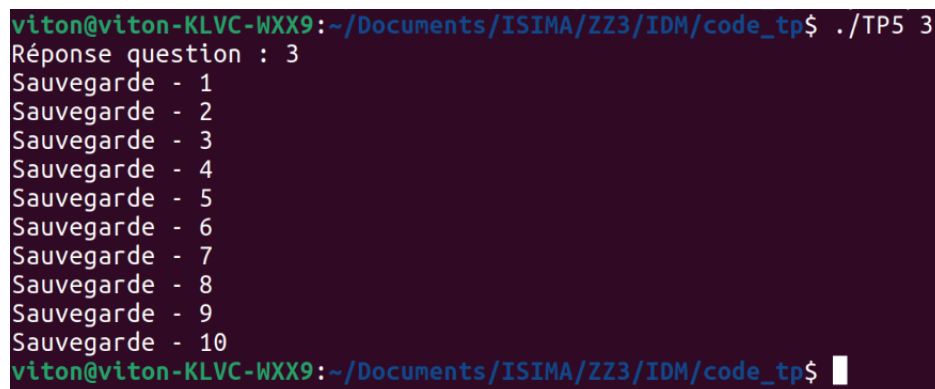
Dans cette partie nous allons produire un code pour sauvegarder 10 statuts espacés de 3 milliard de tirages.

### IV.1 Explication du choix de réaliser trois milliard de tirage

J'ai pu lire la suite de l'énoncé du TP qui consiste à réaliser une simulation de Monte Carlo dans le but d'estimer le volume d'une sphère de rayon 1. Or pour réaliser cette simulation dans un espace tri-dimensionnelle (le volume est une mesure qui a lieu dans un espace 3D), il est nécessaire de tirer de manière aléatoire des points dont les coordonnées sont composés de 3 nombres (x, y et z), donc si l'on souhaite que chaque réplication de notre simulation tire 1 milliard de points nous avons besoin de 3 milliard de nombres aléatoires à chaque réplication.

### IV.2 Explication de la sauvegarde

Pour réaliser la sauvegarde de statu j'ai créé un fonction qui se nomme *saveStatusWithParameter* qui permet de spécifier le nombre de statuts que l'on souhaite sauvegarder et le nombre de tirages qui séparent la sauvegarde de chaque statut.



```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./TP5 3
Réponse question : 3
Sauvegarde - 1
Sauvegarde - 2
Sauvegarde - 3
Sauvegarde - 4
Sauvegarde - 5
Sauvegarde - 6
Sauvegarde - 7
Sauvegarde - 8
Sauvegarde - 9
Sauvegarde - 10
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$
```

FIGURE 5 – Sauvegarde de 10 statuts espacés de 3 milliard de tirages

## V Estimation du volume d'une sphère en séquentiel (Question 4)

Dans cette partie nous allons approximer le volume d'une sphère de rayon 1 en utilisant une simulation de Monte Carlo que nous allons répliquer 10 fois.

### V.1 Explication du principe de la simulation

Le principe de notre simulation est de tirer un milliard de points aléatoires et de vérifier à chaque fois si ce point est dans la sphère.

```
double calculateSphereVolume(CLHEP::MTwistEngine* mt, long points) {  
    double inside = 0;  
  
    for (long i = 0; i < points; ++i) {  
        double x = mt->flat();  
        double y = mt->flat();  
        double z = mt->flat();  
  
        if (x * x + y * y + z * z <= 1.0) {  
            ++inside;  
        }  
    }  
  
    return 8.0 * inside / points;  
}
```

FIGURE 6 – Simulation de Monte Carlo pour estimer le volume d'une sphère



## V.2 Résultats

```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./TP5 4
Réponse question 4 :
Réplication 1: Calcul du volume de la sphère...
Volume estimé : 4.1888326000
Réplication 2: Calcul du volume de la sphère...
Volume estimé : 4.1889602720
Réplication 3: Calcul du volume de la sphère...
Volume estimé : 4.1886184000
Réplication 4: Calcul du volume de la sphère...
Volume estimé : 4.1885927680
Réplication 5: Calcul du volume de la sphère...
Volume estimé : 4.1889140320
Réplication 6: Calcul du volume de la sphère...
Volume estimé : 4.1886768640
Réplication 7: Calcul du volume de la sphère...
Volume estimé : 4.1886023760
Réplication 8: Calcul du volume de la sphère...
Volume estimé : 4.1886803920
Réplication 9: Calcul du volume de la sphère...
Volume estimé : 4.1888911680
Réplication 10: Calcul du volume de la sphère...
Volume estimé : 4.1890580560
```

FIGURE 7 – 10 réplifications séquentielles de la simulation de Monte Carlo

Sur la figure ci-dessus on peut voir l'approximation du volume d'une sphère de rayon 1 lors de nos 10 réplifications de la simulation de Monte Carlo.

```
Volume moyen sur les 10 réplifications : 4.1887826928
Variance : 0.0000000256
Ecart type : 0.0001600343
Intervalle de confiance à 95% : [4.1886835025 - 4.1888818831]
Temps total pour 10 réplifications : 265.6167711460 secondes
```

FIGURE 8 – Résultats obtenus et qualité de ces derniers lors de l'approximation du volume d'une sphère de rayon 1

Sur la figure 8 on peut voir la moyenne calculée avec les 10 réplifications, elle est de 4.18878. L'écart type est de  $2.56 \times 10^{-8}$  et on a alors un intervalle de confiance à 95% qui vaut [4.1886835025 - 4.1888818831].

Enfin on note que le temps moyen d'exécution de 10 réplifications est de 264 secondes en séquentiel (266 secondes pour l'exécution présentée).

## VI Estimation du volume d'une sphère en parallèle (Question 5)

Le but dans cette partie est de réaliser 10 réplifications de Monte Carlo comme à la question précédente mais cette fois-ci en parallélisant l'exécution en utilisant les threads de la bibliothèque standard dans un premier temps et dans un second en utilisant un script bash, qui lance 10 réplifications en précisant à chaque fois le chemin correspondant au statut du générateur qu'il faut utiliser.

### VI.1 Avec la bibliothèque standard

Dans cette partie nous avons utilisé les threads de la bibliothèque standard pour paralléliser l'exécution des 10 réplifications de la simulation de Monte Carlo, qui sert à approximer le volume d'une sphère de rayon 1. Nous avons utilisé un thread par réplification soit 10 threads au total.

```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./TP5 5
Réponse question 5 :
Réplication 2: calcul...
Réplication 1: calcul...
Réplication 3: calcul...
Réplication 4: calcul...
Réplication 6: calcul...
Réplication 7: calcul...
Réplication 10: calcul...
Réplication 9: calcul...
Réplication 5: calcul...
Réplication 8: calcul...
Réplication 10: Volume estimé : 4.1890580560
Réplication 5: Volume estimé : 4.1889140320
Réplication 4: Volume estimé : 4.1885927680
Réplication 2: Volume estimé : 4.1889602720
Réplication 3: Volume estimé : 4.1886184000
Réplication 8: Volume estimé : 4.1886803920
Réplication 7: Volume estimé : 4.1886023760
Réplication 6: Volume estimé : 4.1886768640
Réplication 9: Volume estimé : 4.1888911680
Réplication 1: Volume estimé : 4.1888326000
Toutes les réplifications sont terminées.

Volume moyen sur les 10 réplifications : 4.1887826928
Variance: 0.0000000256
Ecart type : 0.0001600343
Intervalle de confiance à 95% : [4.1886835025 - 4.188818831]
Temps total pour 10 réplifications : 81.0445003290 secondes
```

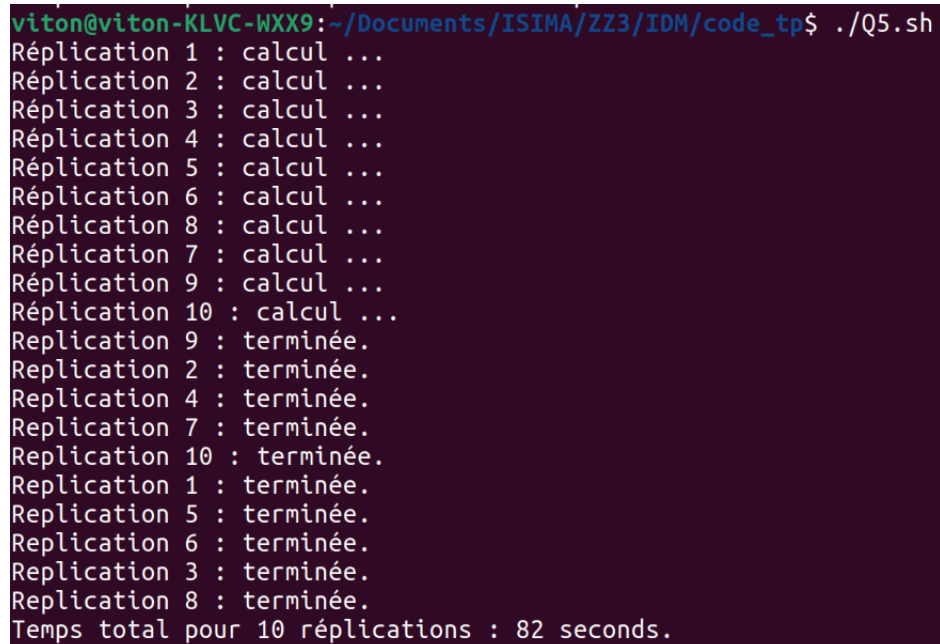
FIGURE 9 – Résultats obtenus avec la parallélisation en utilisant les threads de la bibliothèque standard

Dans un premier temps on peut remarquer que l'on obtient les mêmes résultats bit à bit que lors des réplifications réalisées séquentiellement. Ce qui garanti la reproductibilité de notre "expérience" ainsi que le bon fonctionnement de la sauvegarde et la restauration des statuts du générateur.

Ensuite au niveau du temps d'exécution on note qu'il a fallu 81 secondes environ pour réaliser les 10 réplifications soient environ 3 fois moins de temps que la manière séquentielle. Le temps moyen d'exécution est de 82 secondes pour cette version.

## VI.2 En utilisant un script bash

Dans le but de réaliser une seconde version de la parallélisation, nous avons réalisé un script bash que l'on peut retrouver dans le fichier *Q5.sh* qui lance 10 réplifications en parallèle.



```
viton@viton-KLVC-WXX9:~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./Q5.sh
Réplication 1 : calcul ...
Réplication 2 : calcul ...
Réplication 3 : calcul ...
Réplication 4 : calcul ...
Réplication 5 : calcul ...
Réplication 6 : calcul ...
Réplication 8 : calcul ...
Réplication 7 : calcul ...
Réplication 9 : calcul ...
Réplication 10 : calcul ...
Replication 9 : terminée.
Replication 2 : terminée.
Replication 4 : terminée.
Replication 7 : terminée.
Replication 10 : terminée.
Replication 1 : terminée.
Replication 5 : terminée.
Replication 6 : terminée.
Replication 3 : terminée.
Replication 8 : terminée.
Temps total pour 10 réplifications : 82 seconds.
```

FIGURE 10 – Résultats obtenus avec la parallélisation en utilisant un script bash

On remarque que l'on obtient un temps d'exécution assez similaire que celui obtenu avec la parallélisation réalisée avec la bibliothèque standard, là aussi environ 3 fois plus rapide que la manière séquentielle. On obtient pour cette version un temps d'exécution moyen de 83 secondes (sur la capture nous avons un temps de 82 secondes).

## VII OpenMP (Question 6)

Dans cette partie nous allons utiliser la librairie OpenMP dans le but de comparer son fonctionnement avec les threads de la librairie standard.

### VII.1 Explication du principe d'OpenMP

```
void question6() {  
    const int numReplications = 10;  
    const long pointsPerReplication = 1'000'000'000;  
    std::vector<double> data;  
    std::ifstream file;  
    double volume = 0.0;  
    double mean;  
    double variance;  
    double standardDeviation;  
    std::pair<double, double> confidenceInterval;  
  
    auto start = std::chrono::high_resolution_clock::now();  
  
    #pragma omp parallel for  
    for (int i = 0; i < numReplications; ++i) {  
        runSimulation(i, pointsPerReplication);  
    }  
    std::cout << "Toutes les réplifications sont terminées.\n";  
    ...  
}
```

FIGURE 11 – Utilisation d'OpenMP

Pour mettre en place la parallélisation des 10 réplifications nous utilisons des annotations qui sont des directives processeurs comme celle que nous pouvons voir dans l'extrait de code ci-dessus *pragma omp parallel for*, cela signifie que le code contenu dans la boucle va être parallélisé.

```
viton@viton-KLVC-WXX9: ~/Documents/ISIMA/ZZ3/IDM/code_tp$ ./TP5 6
Réponse question 6 :
Réplication 8: calcul...
Réplication 7: calcul...
Réplication 1: calcul...
Réplication 9: calcul...
Réplication 5: calcul...
Réplication 6: calcul...
Réplication 3: calcul...
Réplication 10: calcul...
Réplication 6: Volume estimé : 4.1886768640
Réplication 9: Volume estimé : 4.1888911680
Réplication 8: Volume estimé : 4.1886803920
Réplication 3: Volume estimé : 4.1886184000
Réplication 4: calcul...
Réplication 5: Volume estimé : 4.1889140320
Réplication 1: Volume estimé : 4.1888326000
Réplication 2: calcul...
Réplication 10: Volume estimé : 4.1890580560
Réplication 7: Volume estimé : 4.1886023760
Réplication 4: Volume estimé : 4.1885927680
Réplication 2: Volume estimé : 4.1889602720
Toutes les réplifications sont terminées.

Volume moyen sur les 10 réplifications : 4.1887826928
Variance : 0.0000000256
Ecart type : 0.0001600343
Intervalle de confiance à 95% : [4.1886835025 - 4.188818831]
Temps total pour 10 réplifications avec OpenMP : 94.6776092760 secondes
```

FIGURE 12 – Résultats obtenus avec la parallélisation en utilisant OpenMP

On remarque que les résultats sont à nouveau identiques à ceux des questions 4 et 5. Le temps d'exécution est d'environ 95 secondes pour l'exécution présentée, en moyenne il est d'environ 92 secondes.

OpenMP semble légèrement moins rapide que les versions présentées précédemment cependant j'ai utilisé le mécanisme des annotations lors de l'accès à la sortie standard (pour éviter les accès concurrents), ce qui peut ralentir l'exécution.

On remarque également qu'OpenMP lance 8 threads en parallèle (contrairement aux deux autres versions présentées, on ne précise ici pas le nombre de threads à lancer), certainement car mon ordinateur possède 8 cœurs logiques.

## VIII Conclusion

Ce TP nous a permis d'explorer de manière approfondie les mécanismes de génération de nombres pseudo-aléatoires déjà étudiés l'an dernier en cours de simulation ainsi que leur utilisation dans des simulations stochastiques complexes. En particulier, l'utilisation de la bibliothèque professionnelle CLHEP a fourni une opportunité précieuse pour comprendre les principes sous-jacents au Mersenne Twister et ses fonctionnalités de sauvegarde/restauration de statuts. Ces outils garantissent une reproductibilité essentielle dans le cadre de calculs stochastiques, notamment dans le contexte du Monte Carlo et sont également très utiles lors de la parallélisation des répliques.

La mise en œuvre d'une estimation du volume d'une sphère via la méthode de Monte Carlo a démontré la pertinence des flux indépendants et a mis en lumière les performances des approches séquentielles et parallèles. Les résultats montrent clairement l'impact du parallélisme sur la réduction du temps d'exécution, avec des gains significatifs obtenus grâce aux threads de la bibliothèque standard, aux scripts bash et à la bibliothèque OpenMP. Chaque méthode a ses avantages et ses spécificités.

En conclusion, ce travail a renforcé ma compréhension des outils et des concepts nécessaires à la simulation stochastique distribuée. Les expérimentations ont validé l'importance de techniques rigoureuses pour la gestion des flux aléatoires, mais ont également souligné les bénéfices du parallélisme dans les environnements multi-cœurs.