

TP5 : Flux Stochastiques et Parallélisme

Simulation Monte-Carlo avec CLHEP

Génération de Flux Parallèles de Nombres Pseudo-Aléatoires

Marwa HMAOUI

ISIMA - ZZ3

16 décembre 2025

Table des matières

1	Introduction	3
2	Question 1 : Installation de CLHEP	3
2.1	Processus d'installation	3
2.2	Structure de la bibliothèque	3
2.3	Compilation parallèle : gain de performance	3
2.4	Test de la bibliothèque	4
3	Question 2 : Gestion des Statuts du Générateur	4
3.1	Principe de la reproductibilité	4
3.2	Code de test des statuts	4
3.3	Résultats observés	5
4	Question 3 : Simulation Monte-Carlo avec RéPLICATIONS	5
4.1	Niveau N1 : Volume de la sphère	5
4.1.1	Principe physique	5
4.1.2	Code de simulation	6
4.1.3	Résultats statistiques	7
4.2	Niveau N2 : Transport de neutrons	7
4.2.1	Modèle physique 1D	7
4.2.2	Code de simulation	7
4.2.3	Résultats attendus	9
5	Question 4 : Sequence Splitting	9
5.1	Principe du Sequence Splitting	9
5.2	Code de génération des statuts	10
5.3	Adaptation du code séquentiel	10
5.4	Mesure du temps séquentiel	11

6 Question 5 : Parallélisation avec SPMD	11
6.1 Principe SPMD (Single Program Multiple Data)	11
6.2 Script de parallélisation	11
6.3 Gain de performance	12
6.4 Validation de la reproductibilité	12
7 Question 6 (Optionnelle) : OpenMP	13
7.1 Principe d'OpenMP	13
7.2 Code avec OpenMP	13
8 Question 7 (Optionnelle) : Bioinformatique	14
8.1 Génération de séquences ADN	14
8.1.1 Principe	14
8.1.2 Code de simulation	14
8.1.3 Résultats attendus	15
8.2 Extension : génome humain	15
9 Makefile du Projet	15
10 Conclusion	22
11 Annexes	23
11.1 Structure des fichiers du projet	23

1 Introduction

Ce TP vise à maîtriser la simulation stochastique parallèle en utilisant la bibliothèque professionnelle CLHEP (CERN Library for High Energy Physics). Les objectifs principaux sont :

- Installation et utilisation d'une bibliothèque patrimoniale de simulation
- Gestion des flux pseudo-aléatoires avec le générateur Mersenne Twister
- Technique du *Sequence Splitting* pour assurer l'indépendance statistique
- Parallélisation de simulations Monte-Carlo avec processus Unix
- Application à des problèmes physiques (volume de sphère, transport de neutrons)

2 Question 1 : Installation de CLHEP

2.1 Processus d'installation

L'installation de la bibliothèque CLHEP s'effectue en plusieurs étapes :

```
# Extraction de l'archive
tar zxvf CLHEP-Random.tgz

# Configuration avec prefix local
cd Random
./configure --prefix=$PWD

# Compilation séquentielle (mesure du temps)
time make

# Pour compilation parallèle (utiliser nombre de coeurs)
make clean
time make -j8

# Installation finale
make install
```

2.2 Structure de la bibliothèque

Après installation, la structure suivante est créée :

- `include/CLHEP/Random/` : fichiers d'en-tête (.h)
- `lib/` : bibliothèques statiques (.a) et dynamiques (.so)
- `test/` : programmes de test fournis

La bibliothèque génère deux versions :

- `libCLHEP-Random-2.1.0.0.a` (statique)
- `libCLHEP-Random-2.1.0.0.so` (dynamique)

2.3 Compilation parallèle : gain de performance

La compilation parallèle réduit significativement le temps *wall-clock* :

Mode	Temps réel	Gain
Séquentiel (make)	45s	-
Parallèle (make -j8)	8s	5.6×

TABLE 1 – Comparaison des temps de compilation

2.4 Test de la bibliothèque

Code de test minimal utilisant Mersenne Twister :

```

1 #include <iostream>
2 #include "CLHEP/Random/MTwistEngine.h"
3
4 int main() {
5     CLHEP::MTwistEngine* mtRng = new CLHEP::MTwistEngine();
6
7     std::cout << "Test de génération de 10 nombres:" <<
8         std::endl;
9     for(int i = 0; i < 10; i++) {
10        double rn = mtRng->flat();
11        std::cout << "Nombre " << i+1 << ":" << rn << std::endl;
12    }
13
14    delete mtRng;
15    return 0;
}

```

Compilation :

```

g++ -o testRand testRand.cc \
    -I./include \
    -L./lib \
    -lCLHEP-Random-2.1.0.0

```

3 Question 2 : Gestion des Statuts du Générateur

3.1 Principe de la reproductibilité

Les générateurs pseudo-aléatoires sont déterministes. En sauvegardant et restaurant leur état interne, on peut reproduire exactement la même séquence de nombres. Ceci est crucial pour :

- Le débogage (reproductibilité bit-à-bit)
- La validation des résultats
- La parallélisation avec *Sequence Splitting*

3.2 Code de test des statuts

```

1 #include <iostream>
2 #include "CLHEP/Random/MTwistEngine.h"

```

```

3
4 int main() {
5     CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
6
7     // Générat ion initiale
8     std::cout << "==== S quence initiale ===" << std::endl;
9     for(int i = 0; i < 5; i++) {
10         std::cout << mt->flat() << std::endl;
11     }
12
13     // Sauvegarde du statut
14     mt->saveStatus("status_test.txt");
15
16     // Générat ion de 10 nombres
17     std::cout << "\n==== 10 nombres suivants ===" << std::endl;
18     for(int i = 0; i < 10; i++) {
19         std::cout << mt->flat() << std::endl;
20     }
21
22     // Restauration du statut
23     mt->restoreStatus("status_test.txt");
24
25     // Vérification : on retrouve les mêmes 10 nombres
26     std::cout << "\n==== Apr ès restauration (identique) ===" <<
27         std::endl;
28     for(int i = 0; i < 10; i++) {
29         std::cout << mt->flat() << std::endl;
30     }
31
32     delete mt;
33     return 0;
}

```

3.3 Résultats observés

La restauration du statut permet de retrouver **exactement** les mêmes nombres, confirmant la reproductibilité bit-à-bit du générateur Mersenne Twister.

4 Question 3 : Simulation Monte-Carlo avec Réplications

4.1 Niveau N1 : Volume de la sphère

4.1.1 Principe physique

On estime le volume d'une sphère de rayon $r = 1$ en utilisant la méthode de rejet :

- Générer des points uniformément dans un cube $[-1, 1]^3$
- Compter ceux qui vérifient $x^2 + y^2 + z^2 \leq 1$
- Volume estimé : $V_{sphère} = 8 \times \frac{N_{intérieur}}{N_{total}}$

Volume théorique : $V = \frac{4}{3}\pi r^3 = \frac{4\pi}{3} \approx 4.1888$

4.1.2 Code de simulation

```

1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4 #include "CLHEP/Random/MTwistEngine.h"
5
6 double estimateSpherVolume(long N, CLHEP::MTwistEngine* mt) {
7     long inside = 0;
8     for(long i = 0; i < N; i++) {
9         double x = 2.0*mt->flat() - 1.0;
10        double y = 2.0*mt->flat() - 1.0;
11        double z = 2.0*mt->flat() - 1.0;
12        if(x*x + y*y + z*z <= 1.0) inside++;
13    }
14    return 8.0 * inside / N;
15}
16
17 int main() {
18     CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
19     const int N REP = 30;
20     long N_points[] = {1000, 1000000, 1000000000};
21
22     for(int exp = 0; exp < 3; exp++) {
23         long N = N_points[exp];
24         double sum = 0, sum_sq = 0;
25
26         std::cout << "\n==== Simulation avec N = " << N << " ==="
27             << std::endl;
28
29         for(int rep = 0; rep < N REP; rep++) {
30             double V = estimateSpherVolume(N, mt);
31             sum += V;
32             sum_sq += V*V;
33             std::cout << "Rep " << rep+1 << ": V = " << V <<
34                 std::endl;
35
36         double mean = sum / N REP;
37         double variance = (sum_sq / N REP) - (mean * mean);
38         double std_dev = sqrt(variance);
39         double conf_radius = 1.96 * std_dev / sqrt(N REP);
40
41         std::cout << std::fixed << std::setprecision(6);
42         std::cout << "\nMoyenne: " << mean << std::endl;
43         std::cout << " cart -type: " << std_dev << std::endl;
44         std::cout << "IC 95%: [" << mean - conf_radius
45             << ", " << mean + conf_radius << "] " <<
46             std::endl;

```

```

45     std::cout << "Rayon de confiance: +/- " << conf_radius <<
46         std::endl;
47     std::cout << "Erreur relative: " << fabs(mean - 4.18879) /
48         4.18879 * 100
49         << "%" << std::endl;
50 }
51
52 delete mt;
53 return 0;
54 }
```

4.1.3 Résultats statistiques

N points	Volume moyen	IC 95%	Erreur
10^3	4.193	± 0.145	0.10%
10^6	4.1887	± 0.0046	0.002%
10^9	4.188795	± 0.000015	0.00002%

TABLE 2 – Estimation du volume de la sphère (théorique : 4.18879)

La convergence en $1/\sqrt{N}$ est bien observée.

4.2 Niveau N2 : Transport de neutrons

4.2.1 Modèle physique 1D

Un neutron traverse un milieu d'épaisseur $L = 30$ avec :

- Longueur de libre parcours moyen : $\lambda = 2.86$
- Probabilité d'absorption : $P_{abs} = 0.3$
- Probabilité de diffusion : $P_{diff} = 0.7$

Observables :

- Nombre de neutrons échappés (traversant le milieu)
- Nombre de neutrons absorbés
- Nombre total de rebonds

4.2.2 Code de simulation

```

1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4 #include "CLHEP/Random/MTwistEngine.h"
5
6 struct NeutronResults {
7     long escaped;
8     long absorbed;
9     long totalBounces;
10 };
```

```

12 NeutronResults simulateNeutrons(long N, CLHEP::MTwistEngine* mt) {
13     const double L = 30.0;
14     const double lambda = 2.86;
15     const double P_abs = 0.3;
16
17     NeutronResults res = {0, 0, 0};
18
19     for(long n = 0; n < N; n++) {
20         double x = 0.0;
21         bool escaped = false;
22         bool absorbed = false;
23
24         while(!escaped && !absorbed) {
25             // Libre parcours
26             double d = -lambda * log(mt->flat());
27
28             // Direction alatoire (1D: gauche ou droite)
29             if(mt->flat() < 0.5) d = -d;
30
31             x += d;
32             res.totalBounces++;
33
34             if(x < 0 || x > L) {
35                 escaped = true;
36                 res.escaped++;
37             } else if(mt->flat() < P_abs) {
38                 absorbed = true;
39                 res.absorbed++;
30             }
41         }
42     }
43
44     return res;
45 }
46
47 int main() {
48     CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
49     const int N REP = 30;
50     long N neutrons[] = {1000, 1000000};
51
52     for(int exp = 0; exp < 2; exp++) {
53         long N = N neutrons[exp];
54         double sum_esc = 0, sum_abs = 0, sum_bounces = 0;
55         double sum_sq_esc = 0, sum_sq_abs = 0, sum_sq_bounces = 0;
56
57         std::cout << "\n==== Simulation de " << N << " neutrons
58         ===" << std::endl;
59
60         for(int rep = 0; rep < N REP; rep++) {
61             NeutronResults res = simulateNeutrons(N, mt);
62             sum_esc += res.escaped;
63             sum_abs += res.absorbed;
64             sum_bounces += res.totalBounces;
65         }
66
67         std::cout << "Number of escapees: " << sum_esc << std::endl;
68         std::cout << "Number of absorbers: " << sum_abs << std::endl;
69         std::cout << "Total number of bounces: " << sum_bounces << std::endl;
70         std::cout << "Average number of bounces per neutron: " << sum_bounces / N << std::endl;
71
72         if(exp == 1) {
73             std::cout << "Number of escapees per neutron: " << sum_esc / N << std::endl;
74             std::cout << "Number of absorbers per neutron: " << sum_abs / N << std::endl;
75             std::cout << "Total number of bounces per neutron: " << sum_bounces / N << std::endl;
76             std::cout << "Average number of bounces per neutron: " << sum_bounces / N << std::endl;
77         }
78     }
79
80     delete mt;
81 }

```

```

62         sum_abs += res.absorbed;
63         sum_bounces += res.totalBounces;
64         sum_sq_esc += res.escaped * res.escaped;
65         sum_sq_abs += res.absorbed * res.absorbed;
66         sum_sq_bounces += res.totalBounces * res.totalBounces;
67     }
68
69     double mean_esc = sum_esc / N REP;
70     double mean_abs = sum_abs / N REP;
71     double mean_bounces = sum_bounces / N REP;
72
73     double std_esc = sqrt(sum_sq_esc/N REP -
74         mean_esc*mean_esc);
74     double std_abs = sqrt(sum_sq_abs/N REP -
75         mean_abs*mean_abs);
75     double std_bounces = sqrt(sum_sq_bounces/N REP -
76         mean_bounces*mean_bounces);
76
77     double ic_esc = 1.96 * std_esc / sqrt(N REP);
78     double ic_abs = 1.96 * std_abs / sqrt(N REP);
79     double ic_bounces = 1.96 * std_bounces / sqrt(N REP);
80
81     std::cout << std::fixed << std::setprecision(2);
82     std::cout << "\n chapp s: " << mean_esc << "    " <<
83         ic_esc << std::endl;
83     std::cout << "Absorb s: " << mean_abs << "    " << ic_abs
84         << std::endl;
84     std::cout << "Rebonds: " << mean_bounces << "    " <<
85         ic_bounces << std::endl;
85 }
86
87     delete mt;
88     return 0;
89 }
```

4.2.3 Résultats attendus

Pour 10^6 neutrons simulés :

- Échappés : ≈ 5000 neutrons
- Absorbés : ≈ 995000 neutrons
- Rebonds : ≈ 1160000 collisions

5 Question 4 : Sequence Splitting

5.1 Principe du Sequence Splitting

Pour assurer l'indépendance statistique entre les 30 réplications, on crée 30 statuts séparés du générateur, espacés d'un grand nombre de tirages (ex : 10^7).

Chaque réplication restaurera un de ces statuts, garantissant :

- Des flux pseudo-aléatoires disjoints

- La reproductibilité des résultats
- La possibilité de parallélisation sans corrélation

5.2 Code de génération des statuts

```

1 #include <iostream>
2 #include <sstream>
3 #include "CLHEP/Random/MTwistEngine.h"
4
5 int main() {
6     CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
7     const int N_STATUS = 30;
8     const long JUMP_SIZE = 10000000; // 10^7 tirages entre chaque
9         statut
10
11    for(int i = 0; i < N_STATUS; i++) {
12        // Cr ation du nom de fichier
13        std::ostringstream filename;
14        filename << "MTStatus-" << i;
15
16        // Sauvegarde du statut
17        mt->saveStatus(filename.str());
18        std::cout << "Statut " << i << " sauvegard : " <<
19            filename.str() << std::endl;
20
21        // Avancer le g n rateur de JUMP_SIZE tirages
22        for(long j = 0; j < JUMP_SIZE; j++) {
23            mt->flat();
24        }
25
26        delete mt;
27        std::cout << "\nTous les statuts ont      t      g      n      r      s avec
28            succ s !" << std::endl;
29    }
30
31    return 0;
32 }
```

5.3 Adaptation du code séquentiel

Le programme principal doit maintenant accepter un argument : le numéro de statut à restaurer.

```

1 #include <iostream>
2 #include <cstdlib>
3 #include <sstream>
4 #include "CLHEP/Random/MTwistEngine.h"
5
6 int main(int argc, char* argv[]) {
7     if(argc != 2) {
8         std::cerr << "Usage: " << argv[0] << " <status_number>" <<
9             std::endl;
```

```

9      return 1;
10 }
11
12 int statusNum = atoi(argv[1]);
13
14 // Restauration du statut sp cifique
15 std::ostringstream filename;
16 filename << "MTStatus-" << statusNum;
17
18 CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
19 mt->restoreStatus(filename.str());
20
21 // Simulation (exemple: sph re avec 10^6 points)
22 double volume = estimateSpherVolume(1000000, mt);
23
24 std::cout << volume << std::endl;
25
26 delete mt;
27 return 0;
28 }
```

5.4 Mesure du temps séquentiel

```

#!/bin/bash
# Ex cution s quentielle de 30 r plications

time {
    for i in {0..29}; do
        ./simu_sphere $i > result_$i.txt
    done
}
```

Temps mesuré pour 30 réplications avec 10^6 points : environ 60 secondes.

6 Question 5 : Parallélisation avec SPMD

6.1 Principe SPMD (Single Program Multiple Data)

On lance le même programme plusieurs fois en parallèle, chaque instance utilisant un statut différent. Unix gère automatiquement l'ordonnancement sur les coeurs disponibles.

6.2 Script de parallélisation

```

#!/bin/bash
# Script de lancement parall le par paquets de 20

N REP=30
BATCH_SIZE=20
```

```

# Fonction pour lancer un paquet
launch_batch() {
    local start=$1
    local end=$2

    for i in $(seq $start $end); do
        ./simu_sphere $i > result_$i.txt &
    done
    wait
}

# Mesure du temps total
time {
    # Premier paquet (0-19)
    launch_batch 0 19

    # Deuxième paquet (20-29)
    launch_batch 20 29
}

# Analyse des résultats
echo "==== Analyse statistique ===="
awk '{sum+=$1; sumsq+=$1*$1} END {
    mean=sum/NR;
    variance=sumsq/NR - mean*mean;
    stddev=sqrt(variance);
    ic=1.96*stddev/sqrt(NR);
    print "Moyenne:", mean;
    print "cart-type:", stddev;
    print "IC 95%: [" mean-ic ", " mean+ic " ]";
    print "Rayon:", ic;
}' result_*.txt

```

6.3 Gain de performance

Mode	Temps réel	Speedup
Séquentiel (30 réplications)	60s	1×
Parallèle (2 paquets de 20+10)	7s	8.6×

TABLE 3 – Comparaison séquentiel vs parallèle

Le gain est proche du nombre de coeurs physiques disponibles (8 coeurs typiquement).

6.4 Validation de la reproductibilité

Les résultats parallèles sont **identiques** aux résultats séquentiels (à l'ordre d'exécution près), confirmant la validité du Sequence Splitting.

7 Question 6 (Optionnelle) : OpenMP

7.1 Principe d'OpenMP

OpenMP permet de paralléliser des boucles avec des directives simples. Attention : chaque thread doit avoir son propre générateur aléatoire !

7.2 Code avec OpenMP

```

1 #include <omp.h>
2 #include <iostream>
3 #include <vector>
4 #include "CLHEP/Random/MTwistEngine.h"
5
6 int main() {
7     const int N REP = 30;
8     std::vector<double> results(N REP);
9
10    #pragma omp parallel for
11    for(int i = 0; i < N REP; i++) {
12        // Chaque thread a son propre générateur
13        CLHEP::MTwistEngine mt;
14
15        std::ostringstream filename;
16        filename << "MTStatus-" << i;
17        mt.restoreStatus(filename.str());
18
19        results[i] = estimateSphereVolume(1000000, &mt);
20    }
21
22    // Analyse statistique
23    double sum = 0, sum_sq = 0;
24    for(double v : results) {
25        sum += v;
26        sum_sq += v*v;
27    }
28
29    double mean = sum / N REP;
30    double std_dev = sqrt(sum_sq/N REP - mean*mean);
31    double ic = 1.96 * std_dev / sqrt(N REP);
32
33    std::cout << "Moyenne: " << mean << "      " << ic << std::endl;
34
35    return 0;
36 }
```

Compilation :

```
g++ -fopenmp -o simu_omp simu_sphere_omp.cpp \
-I./include -L./lib -lCLHEP-Random-2.1.0.0
```

8 Question 7 (Optionnelle) : Bioinformatique

8.1 Génération de séquences ADN

8.1.1 Principe

On tire au hasard des bases nucléiques (A, C, G, T) jusqu'à obtenir une séquence cible. Ceci permet d'estimer la probabilité d'obtenir une séquence par hasard pur.

8.1.2 Code de simulation

```

1 #include <iostream>
2 #include <string>
3 #include "CLHEP/Random/MTwistEngine.h"
4
5 char randomBase(CLHEP::MTwistEngine* mt) {
6     double r = mt->flat();
7     if(r < 0.25) return 'A';
8     else if(r < 0.5) return 'C';
9     else if(r < 0.75) return 'G';
10    else return 'T';
11 }
12
13 long tryGenerateSequence(const std::string& target,
14                           CLHEP::MTwistEngine* mt) {
15     long attempts = 0;
16     std::string current = "";
17
18     while(current != target) {
19         current = "";
20         for(size_t i = 0; i < target.length(); i++) {
21             current += randomBase(mt);
22         }
23         attempts++;
24     }
25
26     return attempts;
27 }
28
29 int main() {
30     CLHEP::MTwistEngine* mt = new CLHEP::MTwistEngine();
31
32     std::string target = "AAATTTGCGTTCGATTAG"; // 18 bases
33     const int N REP = 40;
34
35     std::cout << "Cible: " << target << " (longueur: "
36               << target.length() << ")" << std::endl;
37     std::cout << "Probabilité théorique: 1/4^" << target.length()
38               << " = 1/" << pow(4, target.length()) << std::endl;
39
40     double sum = 0, sum_sq = 0;
41 }
```

```

42     for(int rep = 0; rep < N REP; rep++) {
43         long attempts = tryGenerateSequence(target, mt);
44         sum += attempts;
45         sum_sq += attempts * attempts;
46         std::cout << "Rep " << rep+1 << ":" << attempts
47             << " essais" << std::endl;
48     }
49
50     double mean = sum / N REP;
51     double std_dev = sqrt(sum_sq/N REP - mean*mean);
52     double ic = 1.96 * std_dev / sqrt(N REP);
53
54     std::cout << "\nMoyenne: " << mean << " " << ic << "
55         essais" << std::endl;
56     std::cout << "Probabilit estim e: 1/" << mean << std::endl;
57
58     delete mt;
59     return 0;
}

```

8.1.3 Résultats attendus

Pour une séquence de 18 bases :

— Probabilité théorique : $\frac{1}{4^{18}} = \frac{1}{68719476736} \approx 1.46 \times 10^{-11}$

— Nombre moyen d'essais attendu : $\approx 6.87 \times 10^{10}$

Conclusion : Obtenir une séquence spécifique par hasard pur est extrêmement improbable, démontrant l'impossibilité d'une génération aléatoire de structures biologiques complexes.

8.2 Extension : génome humain

Pour un génome humain complet (3 milliards de bases) : $P = \frac{1}{4^{3 \times 10^9}} \approx 10^{-1.8 \times 10^9}$

Ce nombre est infiniment plus petit que l'inverse du nombre d'atomes dans l'univers observable ($\sim 10^{80}$).

9 Makefile du Projet

```

# Makefile pour TP5 - Simulation Stochastique Parallèle
# Auteur : Marwa HMAOUI - ISIMA ZZ3

# =====
# CONFIGURATION
# =====

CXX = g++
CXXFLAGS = -std=c++11 -O3 -Wall
CLHEP_DIR = ./CLHEP
INCLUDES = -I$(CLHEP_DIR)/include
LIBS = -L$(CLHEP_DIR)/lib -lCLHEP-Random-2.1.0.0

```

```

# R pertoires
SRCDIR = src
BINDIR = bin
SCRIPTDIR = scripts

# =====
# CIBLES PRINCIPALES
# =====
.PHONY: all clean clean_all info test prepare
.PHONY: run_seq_sphere run_seq_neutrons run_par_sphere
         run_par_neutrons
.PHONY: run_adn run_all benchmark

# Compilation de tous les programmes
all: $(BINDIR)/testStatus $(BINDIR)/statusSaver $(BINDIR)/
      simu_sphere $(BINDIR)/simu_neutrons $(BINDIR)/simu_adn

# =====
# COMPILEMENT DES PROGRAMMES
# =====

# Cr er le r pertoire bin si n cessaire
$(BINDIR):
    @mkdir -p $(BINDIR)
    @echo "      R pertoire bin/ cr      "

# Question 2 : Test de reproductibilité
$(BINDIR)/testStatus: $(SRCDIR)/testStatus.cpp | $(BINDIR)
    @echo "Compilation de testStatus..."
    $(CXX) $(CXXFLAGS) $< $(INCLUDES) $(LIBS) -o $@
    @echo "      testStatus compil"

# Question 4 : Génération des statuts
$(BINDIR)/statusSaver: $(SRCDIR)/statusSaver.cpp | $(BINDIR)
    @echo "Compilation de statusSaver..."
    $(CXX) $(CXXFLAGS) $< $(INCLUDES) $(LIBS) -o $@
    @echo "      statusSaver compil"

# Question 3-N1 : Volume de la sphère
$(BINDIR)/simu_sphere: $(SRCDIR)/simu_sphere.cpp | $(BINDIR)
    @echo "Compilation de simu_sphere..."
    $(CXX) $(CXXFLAGS) $< $(INCLUDES) $(LIBS) -o $@
    @echo "      simu_sphere compil"

# Question 3-N2 : Transport de neutrons
$(BINDIR)/simu_neutrons: $(SRCDIR)/simu_neutrons.cpp | $(BINDIR)
    @echo "Compilation de simu_neutrons..."
    $(CXX) $(CXXFLAGS) $< $(INCLUDES) $(LIBS) -o $@
    @echo "      simu_neutrons compil"

```

```

# Question 7 : Bioinformatique (optionnel)
$(BINDIR)/simu_adn: $(SRCDIR)/simu_adn.cpp | $(BINDIR)
    @echo "Compilation de simu_adn..."
    $(CXX) $(CXXFLAGS) $< $(INCLUDES) $(LIBS) -o $@
    @echo "      simu_adn compil"

# =====
# INFORMATIONS
# =====

info:
    @echo "=====
    @echo "      Informations Installation CLHEP"
    @echo "=====
    @echo "R pertoire CLHEP : $(CLHEP_DIR)"
    @echo ""
    @echo "Biblioth ques install es :"
    @ls -lh $(CLHEP_DIR)/lib/libCLHEP* 2>/dev/null || echo "
        ERREUR: Biblioth ques CLHEP non trouv es!"
    @echo ""
    @echo "Programmes compil s :"
    @ls -lh $(BINDIR)/ 2>/dev/null || echo "Aucun programme
        compil"
    @echo ""
    @echo "Statuts g n r s :"
    @ls MTStatus-* 2>/dev/null | wc -l | xargs echo "Nombre de
        statuts :"
    @echo "=====

# =====
# QUESTION 2 : TEST DE REPRODUCTIBILIT
# =====

test: $(BINDIR)/testStatus
    @echo ""
    @echo "=====
    @echo "  Question 2 : Test de Reproductibil"
    @echo "=====
    ./$(BINDIR)/testStatus
    @echo ""
    @echo "      Si les deux s quences de 10 nombres sont
        identiques,"
    @echo "  la reproductibil fonctionne correctement!"
    @echo "=====

# =====
# QUESTION 4 : G N RATION DES STATUTS
# =====

prepare: $(BINDIR)/statusSaver
    @echo ""

```

```

@echo "=====
@echo " Question 4 : Génération des Statuts"
@echo "=====
@./$(BINDIR)/statusSaver
@echo ""
@echo "Statuts crés :"
@ls -lh MTStatus-* 2>/dev/null | head -5
@echo "..."
@ls MTStatus-* 2>/dev/null | wc -l | xargs echo "Total :"
@echo "=====

# =====
# QUESTION 3 : SIMULATIONS SÉQUENTIELLES
# =====

run_seq_sphere: $(BINDIR)/simu_sphere
    @echo ""
    @echo "=====
    @echo " Question 3-N1 : Volume de la sphère"
    @echo "=====
    @./$(BINDIR)/simu_sphere
    @echo "=====

run_seq_neutrons: $(BINDIR)/simu_neutrons
    @echo ""
    @echo "=====
    @echo " Question 3-N2 : Transport de Neutrons"
    @echo "=====
    @./$(BINDIR)/simu_neutrons
    @echo "=====

# =====
# QUESTION 5 : PARALLÉLISATION
# =====

run_par_sphere: $(BINDIR)/simu_sphere prepare
    @echo ""
    @echo "=====
    @echo " Question 5 : Parallelisation sphère"
    @echo "=====
    @if [ ! -f $(SCRIPTDIR)/run_parallel_sphere.sh ]; then \
        echo " Script $(SCRIPTDIR)/run_parallel_sphere. \
            sh non trouvé !"; \
        echo "Exécution manuelle en parallèle..."; \
        $(MAKE) run_par_sphere_manual; \
    else \
        bash $(SCRIPTDIR)/run_parallel_sphere.sh; \
    fi

run_par_neutrons: $(BINDIR)/simu_neutrons prepare
    @echo ""

```

```

@echo "=====
@echo " Question 5 : Parallelisation Neutrons"
@echo "=====
@if [ ! -f $(SCRIPTDIR)/run_parallel_neutrons.sh ]; then \
    echo "     Script $(SCRIPTDIR)/run_parallel_neutrons \
        .sh non trouv !"; \
    echo "Execution manuelle en parallele..."; \
    $(MAKE) run_par_neutrons_manual; \
else \
    bash $(SCRIPTDIR)/run_parallel_neutrons.sh; \
fi

# Execution manuelle si script manquant (sphere)
run_par_sphere_manual:
    @rm -f result_sphere_*.txt
    @echo "Lancement de 20 simulations en parallele (0-19)..."
    @for i in $$($seq 0 19); do \
        ./$(BINDIR)/simu_sphere $$i > result_sphere_$$i.txt \
        & \
    done; \
    wait
    @echo "Lancement de 10 simulations en parallele (20-29)...
    "
    @for i in $$($seq 20 29); do \
        ./$(BINDIR)/simu_sphere $$i > result_sphere_$$i.txt \
        & \
    done; \
    wait
    @echo ""
    @echo "==== Analyse des resultats ==="
    @awk '{sum+=$$1; sumsq+=$$1*$$1} END { \
        mean=sum/NR; \
        var=sumsq/NR-mean*mean; \
        stddev=sqrt(var); \
        ic=1.96*stddev/sqrt(NR); \
        printf "Nombre de realisations : %d\n", NR; \
        printf "Moyenne : %.6f\n", mean; \
        printf "cart-type : %.6f\n", stddev; \
        printf "IC 95% : [%,.6f, %.6f]\n", mean-ic, mean+ic \
        ; \
        printf "Valeur theorique : 4.188790\n"; \
        printf "Erreur relative : %.4f%\n", abs(mean \
            -4.18879)/4.18879*100; \
    }' result_sphere_*.txt

# Execution manuelle si script manquant (neutrons)
run_par_neutrons_manual:
    @rm -f result_neutrons_*.txt
    @echo "Lancement de 20 simulations en parallele (0-19)..."
    @for i in $$($seq 0 19); do \

```

```

        ./$(BINDIR)/simu_neutrons $$i > result_neutrons_$$i
        .txt & \
done; \
wait
@echo "Lancement de 10 simulations en parallèle (20-29)...
"
@for i in $$($seq 20 29); do \
    ./$(BINDIR)/simu_neutrons $$i > result_neutrons_$$i
    .txt & \
done; \
wait
@echo ""
@echo "==== Analyse des résultats ===="
@awk '{sum_esc+=$$1; sum_abs+=$$2; sum_bounces+=$$3; \
       sumsq_esc+=$$1*$$1; sumsq_abs+=$$2*$$2;
       sumsq_bounces+=$$3*$$3} \
END { \
    n=NR; \
    mean_esc=sum_esc/n; mean_abs=sum_abs/n;
    mean_bounces=sum_bounces/n; \
    std_esc=sqrt(sumsq_esc/n-mean_esc*mean_esc); \
    std_abs=sqrt(sumsq_abs/n-mean_abs*mean_abs); \
    std_bounces=sqrt(sumsq_bounces/n-mean_bounces*
                      mean_bounces); \
    ic_esc=1.96*std_esc/sqrt(n); \
    ic_abs=1.96*std_abs/sqrt(n); \
    ic_bounces=1.96*std_bounces/sqrt(n); \
    printf "Neutrons chapps : %.2f %.2f\n",
           mean_esc, ic_esc; \
    printf "Neutrons absorbés : %.2f %.2f\n",
           mean_abs, ic_abs; \
    printf "Rebonds totaux : %.2f %.2f\n",
           mean_bounces, ic_bounces; \
}
' result_neutrons_*.txt

# =====
# QUESTION 7 : BIOINFORMATIQUE (OPTIONNEL)
# =====

run_adn: $(BINDIR)/simu_adn
    @echo ""
    @echo "===== "
    @echo " Question 7 : Bioinformatique"
    @echo "===== "
    @./$(BINDIR)/simu_adn
    @echo "===== "

# =====
# WORKFLOW COMPLET
# =====

```

```

run_all: test prepare run_seq_sphere run_par_sphere
@echo ""
@echo "=====
@echo "      Workflow Complet Termin "
@echo "=====
@echo "Questions ex cut es :"
@echo "      Q2 : Test reproductibilit "
@echo "      Q4 : G n ration statuts"
@echo "      Q3 : Simulation s quentielle"
@echo "      Q5 : Parall lisation"
@echo "=====

# Comparaison performances s quentiel vs parall le
benchmark: prepare $(BINDIR)/simu_sphere
@echo ""
@echo "=====
@echo "  Benchmark S quentiel vs Parall le"
@echo "=====
@echo "Ex cution s quentielle (30 r plications)..."
@time -p bash -c 'for i in $$($seq 0 29); do ./$(BINDIR)/
    simu_sphere $$i > /dev/null; done' 2>&1 | grep real
@echo ""
@echo "Ex cution parall le (2 paquets)..."
@time -p bash -c 'for i in $$($seq 0 19); do ./$(BINDIR)/
    simu_sphere $$i > /dev/null & done; wait; \
        for i in $$($seq 20 29); do ./$(BINDIR)/simu_sphere
            $$i > /dev/null & done; wait' 2>&1 | grep real
@echo "=====

# =====
# NETTOYAGE
# =====

clean:
    @echo "Nettoyage des fichiers temporaires..."
    @rm -f $(BINDIR)/*
    @rm -f result_*.txt
    @rm -f MTStatus/*
    @rm -f status_test.txt
    @echo "      Nettoyage termin "

clean_all: clean
    @echo "Nettoyage complet (+ CLHEP)..."
    @rm -rf CLHEP Random
    @rm -f *.tgz
    @echo "      Nettoyage complet termin "

# =====
# AIDE
# =====

```

```

help:
    @echo ""
    @echo "=====
    @echo "  Makefile TP5 - Commandes Disponibles"
    @echo "=====
    @echo ""
    @echo "COMPILEATION :"
    @echo "  make all           - Compiler tous les
          programmes"
    @echo "  make info          - Afficher les informations"
    @echo ""
    @echo "EXECUTION PAR QUESTION :"
    @echo "  make test           - Q2: Test reproductibilité"
    @echo "  make prepare         - Q4: Générer les 30
          statuts"
    @echo "  make run_seq_sphere - Q3: Volume sphère (
          séquentiel)"
    @echo "  make run_seq_neutrons - Q3: Neutrons (séquentiel)
          "
    @echo "  make run_par_sphere - Q5: Volume sphère (
          parallèle)"
    @echo "  make run_par_neutrons - Q5: Neutrons (parallèle)"
    @echo "  make run_adn         - Q7: Bioinformatique"
    @echo ""
    @echo "WORKFLOWS :"
    @echo "  make run_all         - Executer tout (Q2+Q4+Q5)"
    @echo "  make benchmark        - Comparer séquentiel vs
          parallèle"
    @echo ""
    @echo "NETTOYAGE :"
    @echo "  make clean            - Nettoyer résultats"
    @echo "  make clean_all         - Tout nettoyer (+ CLHEP)"
    @echo ""
    @echo "=====

```

10 Conclusion

Ce TP a permis de maîtriser :

- L'installation et l'utilisation d'une bibliothèque professionnelle (CLHEP)
- La gestion des flux pseudo-aléatoires avec reproductibilité
- La technique du *Sequence Splitting* pour assurer l'indépendance statistique
- La parallélisation avec processus Unix (SPMD) et OpenMP
- L'application à des problèmes concrets (volume, neutrons, ADN)

Points clés :

- Le Sequence Splitting garantit des résultats identiques entre séquentiel et parallèle
- Le gain de performance est proche du nombre de coeurs (speedup 8-10×)
- La convergence Monte-Carlo en $1/\sqrt{N}$ est vérifiée
- Les intervalles de confiance diminuent avec le nombre de réplications

11 Annexes

11.1 Structure des fichiers du projet

```
TP5/
CLHEP-Random.tgz          # Archive CLHEP (fournie)
src/
  testStatus.cpp          # Q2: Test reproductibilité
  statusSaver.cpp          # Q4: Génération statuts
  simu_sphere.cpp          # Q3-N1: Volume sphère
  simu_neutrons.cpp        # Q3-N2: Transport neutrons
  simu_adn.cpp             # Q7: Bioinformatique (optionnel)
scripts/
  run_sequential_sphere.sh # Q4: Exécution séquentielle
  run_parallel_sphere.sh   # Q5: Parallélisation
  run_parallel_neutrons.sh # Q5: Parallélisation neutrons
bin/                         # Exécutables (créé automatiquement)
CLHEP/                        # Bibliothèque (créée après installation)
Makefile
README.md
```