Modèle d'Ising, Méthode MCMC et Calcul Haute Performance

Alexis Evaristo
Hector Baril

30 janvier 2026

Polytech Lille, Informatique, Statistique et Intelligence Artificielle

Table des matières

Chapitre 1

Introduction Générale

Le modèle d'Ising est un modèle probabiliste utilisé en physique statistique pour modéliser le ferromagnétisme. Ce phénomène macroscopique est le produit d'interactions microscopiques entre les spins d'un matériau. L'objectif de ce travail est de simuler ce modèle de manière approchée en utilisant une méthode de Monte-Carlo par chaînes de Markov (MCMC) et de mettre en évidence le phénomène de transition de phase.

1.1 Définition du Modèle

On considère une boîte de côté $2N+1$ centrée à l'origine dans le réseau $\mathbb{Z}^2$, notée Λ_N :

$$\Lambda_N = \{(i,j) \in \mathbb{Z}^2 : -N \leq i,j \leq N\} \tag{1.1}$$

Une configuration de spins est une application $\sigma : \Lambda_N \to \{-1,+1\}$, où la valeur $\sigma(x)$ représente le spin au site x. L'énergie d'une telle configuration, ou Hamiltonien, est définie par la somme des interactions entre spins voisins :

$$H(\sigma) = -\sum_{x \sim y} \sigma(x)\sigma(y) \tag{1.2}$$

Cette énergie est minimale lorsque les spins voisins sont identiques. La probabilité d'occurrence d'une configuration dépend de la température T et est donnée par la mesure de Gibbs :

$$P_{N,T}(\sigma) = \frac{1}{Z_{N,T}} e^{-\frac{1}{T}H(\sigma)} \tag{1.3}$$

1.2 Problématique et Transition de Phase

Un défi majeur réside dans le calcul de la constante de normalisation $Z_{N,T}$. La formule explicite n'étant pas calculable, la simulation directe est impossible pour de grandes valeurs de N. L'enjeu est alors d'utiliser des méthodes algorithmiques pour estimer l'espérance du spin à l'origine et observer si ce dernier reste influencé par les conditions aux bords du domaine. Lorsque cette influence persiste malgré l'augmentation de la taille de la boîte, on parle de transition de phase.

L'étude de ces systèmes à grande échelle nécessite des ressources de calcul importantes, justifiant une approche orientée vers le calcul haute performance.

1.3 Organisation du Rapport

Ce rapport s'articule autour de cinq chapitres :

— **Chapitre 1 : Introduction Générale**, qui pose les bases théoriques du modèle d'Ising.

— **Chapitre 2 : Méthode MCMC**, détaillant l'algorithme de simulation par chaînes de Markov.

— **Chapitre 3 : Implémentation**, présentant la mise en œuvre logicielle et les premiers résultats.

— **Chapitre 4 : Parallélisation avec CUDA**, consacré à l'accélération des calculs sur processeur graphique.

— **Chapitre 5 : Conclusion**, synthétisant les observations sur la transition de phase et les performances obtenues.

Chapitre 2

Méthode MCMC

L'objectif de ce chapitre est de détailler la mise en œuvre de la méthode de Monte-Carlo par chaînes de Markov (MCMC) pour simuler le modèle d'Ising. Comme il est impossible de simuler directement la mesure de probabilité $P_{N,T}$ en raison de la complexité du calcul de la constante de normalisation $Z_{N,T}$,l'utilisation d'une approche itérative dans le but d'atteindre la distribution stationnaire souhaitée est la méthode a adopté.

2.1 Algorithme de Metropolis-Hastings

L'algorithme de Metropolis-Hastings permet de construire une chaîne de Markov dont la mesure stationnaire est la mesure de Gibbs.

Algorithm 1 Algorithme de Metropolis

```
X = x_0
for k = 1 to Nombre_Iterations do
    i = X
    choisir j avec probabilité q_{i,j}
    ρ = (π_j q_{j,i}) / (π_i q_{i,j})
    if ρ > 1 then
        X = j
    else
        Choisir U uniformément sur (0,1), et si U < ρ alors X = j
    end if
end for
```

2.2 Calcul du rapport d'acceptation ρ

La probabilité d'accepter une nouvelle configuration σ' à partir d'une configuration σ est définie par le ratio de leurs probabilités respectives selon la mesure de Gibbs :

$$\rho = \frac{P_{N,T}(\sigma')}{P_{N,T}(\sigma)} = \frac{\frac{1}{Z_{N,T}} e^{-\frac{1}{T}H(\sigma')}}{\frac{1}{Z_{N,T}} e^{-\frac{1}{T}H(\sigma)}} \tag{2.1}$$

On observe immédiatement que la constante de normalisation $Z_{N,T}$ se simplifie. En utilisant les propriétés de l'exponentielle, le rapport devient :

$$\rho = e^{-\frac{1}{T}(H(\sigma')-H(\sigma))} = e^{-\frac{\Delta H}{T}} \tag{2.2}$$

C'est cette simplification cruciale qui permet de contourner l'impossibilité de calculer $Z_{N,T}$.

2.3 Algorithme de Metropolis-Hastings pour le modèle d'Ising

Le principe repose sur la modification locale et successive des spins de la configuration.

Algorithm 2 Algorithme de Metropolis pour le modèle d'Ising

Initialiser une configuration $\sigma \in \Sigma_N$ (par exemple, de manière aléatoire)
for $k = 1$ **to** $Nombre\_Iterations$ **do**
    Choisir un site x uniformément au hasard dans la boîte Λ_N
    Proposer une configuration σ' identique à σ sauf au site x : $\sigma'(x) = -\sigma(x)$
    Calculer la variation d'énergie : $\Delta H = H(\sigma') - H(\sigma)$
    Calculer le rapport d'acceptation : $\rho = \exp\left(-\frac{\Delta H}{T}\right)$
    Générer une variable aléatoire $u \sim \mathcal{U}([0, 1])$
    if $u < \rho$ **then**
        Accepter la transition : $\sigma \leftarrow \sigma'$
    else
        Conserver la configuration actuelle : $\sigma \leftarrow \sigma$
    end if
end for

2.4 Optimisation : Variation locale de l'énergie

L'un des avantages majeurs de l'algorithme de Metropolis appliqué au modèle d'Ising est qu'il n'est pas nécessaire de calculer l'énergie totale $H(\sigma)$ de la grille à chaque itération.

Puisque les configurations σ et σ' ne diffèrent qu'en un seul site x, la variation d'énergie ΔH ne dépend que des interactions entre le site x et ses voisins directs $y \sim x$. L'énergie associée uniquement au site x est :

$$H_{site}(x) = -\sigma(x) \sum_{y \sim x} \sigma(y) \tag{2.3}$$

Lors d'un basculement de spin ($\sigma(x) \to -\sigma(x)$), la variation d'énergie est simplement :

$$\Delta H = H(\sigma') - H(\sigma) = 2\sigma(x) \sum_{y \sim x} \sigma(y) \tag{2.4}$$

Cette propriété est fondamentale pour la performance du programme car :

— Elle évite un calcul global coûteux de l'énergie de toute la boîte Λ_N à chaque étape.
— Elle réduit la complexité du calcul de la transition à une opération locale impliquant seulement 4 voisins.
— Elle permet d'effectuer des millions de tentatives de basculement par seconde, ce qui est nécessaire pour atteindre l'équilibre thermodynamique.

Chapitre 3

Implémentation

L'implémentation du modèle d'Ising bidimensionnel s'est faite dans le langage C, pour des raisons de performances. Cette section détaille les choix de structures de données et les algorithmes utilisés pour simuler le ferromagnétisme et les transitions de phase, conformément au modèle théorique défini dans le sujet.

3.1 Représentation Mémoire de la Grille

L'objet mathématique qui reprèsente ce modèle est une matrice carrée de côté $L = 2N + 1$ centrée en 0. Informatiquement, ce modèle est implémenté par la structure suivante : `grille` (fichier `grille.h`)

- La taille N (le demi-côté).
- Un tableau dynamique `sigma` pour stocker les spins. La linéarisation de la matrice 2D permet un stockage contigu en mémoire. Pour un site de coordonnées (i, j) avec $0 \leq i, j < 2N + 1$, l'accès se fait par l'index $i \times \text{taille} + j$.
- L'énergie totale actuelle $H(\sigma)$ de la configuration.

Les spins $\sigma(x)$ prennent les valeurs définies par les constantes macro `POS` (+1) et `NEG` (-1).

3.2 Calcul de la variation d'énergie

La mise en œuvre de l'algorithme de Metropolis repose sur une opération critique : le calcul de la variation d'énergie ΔH. Cette fonction est appelée à chaque itération de la simulation. Son efficacité est donc le facteur déterminant de la rapidité globale du programme.

3.2.1 Approche Naïve : Disjonction de Cas Géométriques

Initialement, le calcul de la variation d'énergie a été implémenté en distinguant manuellement chaque position possible du spin sur la grille. En effet, un spin situé au centre possède quatre voisins, un spin sur un bord en possède trois, et un spin dans un coin n'en possède que deux.

Cette approche utilise une structure conditionnelle complexe pour éviter les accès hors limites dans le tableau :

Listing 3.1 – Version naïve de calculVariationVoisin

```c
int calculVariationVoisin(struct grille * grille, int i, int j) {
    int H = 0;
    if (i > 0 && j > 0 && i < DIM-1 && j < DIM-1) {
        // Cas standard : centre de la grille
        H -= grille->sigma[i][j] * grille->sigma[i][j+1] * 2; // Droite
        H -= grille->sigma[i][j] * grille->sigma[i+1][j] * 2; // Bas
        H -= grille->sigma[i][j] * grille->sigma[i][j-1] * 2; // Gauche
        H -= grille->sigma[i][j] * grille->sigma[i-1][j] * 2; // Haut
    }
    else {
        // Gestion complexe des bordures et des coins
        if ( i == 0 ) {
            if ( j == 0 ) { // Coin haut-gauche
                H -= grille->sigma[i][j] * grille->sigma[i][j+1]*2;
                H -= grille->sigma[i][j] * grille->sigma[i+1][j]*2;
            }
            else if (j == DIM-1) { // Coin haut-droite
                H -= grille->sigma[i][j] * grille->sigma[i+1][j]*2;
                H -= grille->sigma[i][j] * grille->sigma[i][j-1]*2;
            }
            else { // Bordure haute
                H -= grille->sigma[i][j] * grille->sigma[i][j+1]*2;
                H -= grille->sigma[i][j] * grille->sigma[i+1][j]*2;
                H -= grille->sigma[i][j] * grille->sigma[i][j-1]*2;
            }
        }
        // ... La suite du code suit cette logique de branchement
    }
    return -H;
}
```

Bien que logiquement correcte, cette implémentation est difficile à maintenir et peu performante sur des architectures modernes.

3.2.2 Refactorisation et Optimisation

Pour optimiser ce calcul, une approche plus élégante consite à définir des "zones" géométriques. Celles-ci composent chaque direction (Haut, Bas, Gauche, Droite) indépendamment en vérifiant simplement la validité de l'indice avant l'accès.

Cette refactorisation permet de réduire drastiquement la taille du code :

Listing 3.2 – Version optimisée et refactorisée

```
int calculVariationVoisin(int *spins, int i, int j) {
    int s = spins[i * DIM + j]; // Spin cible
    int voisins = 0;

    if (i > 0)       voisins += spins[(i-1) * DIM + j]; // Haut
    if (i < DIM - 1) voisins += spins[(i+1) * DIM + j]; // Bas
    if (j > 0)       voisins += spins[i * DIM + (j-1)]; // Gauche
    if (j < DIM - 1) voisins += spins[i * DIM + (j+1)]; // Droite

    return 2 * s * voisins;
}
```

3.3 Conditions aux Bords et Parallélisation

Afin d'étudier la transition de phase et la cassure de symétrie :

— **Conditions aux bords** : La fonction `init_grille_border` permet de fixer les spins situés sur le pourtour de la boîte Λ_N à une valeur fixe (+1 ou -1), influençant ainsi l'orientation globale du système à basse température.

— **Parallélisation** : Pour obtenir des statistiques robustes sur l'état du spin central $\sigma(0)$, L'utilisation de la bibliothèque `pthread` est nécessaire puisqu'elle permet de réaliser M simulations indépendantes (définies par le `NOMBRE_THREADS`). De plus, ces simultations sont lancés en parallèle. Un mutex protège les compteurs globaux `nbPos` et `nbNeg` lors de l'agrégation des résultats finaux.

Chapitre 4

Parallélisation des calculs avec CUDA

L'étude statistique du modèle d'Ising, particulièrement pour identifier une transition de phase, nécessite de simuler un grand nombre de grilles indépendantes sur de nombreuses itérations. Sur un processeur classique (CPU), ces calculs sont effectués de manière séquentielle, ce qui limite considérablement la taille des systèmes étudiés. Ce chapitre présente l'implémentation d'une solution massivement parallèle utilisant l'architecture NVIDIA CUDA.

4.1 Environnement et Configuration

La simulation a été développée et testée sur l'environnement **Google Colab**, utilisant un GPU NVIDIA T4. Cet environnement permet d'exploiter des milliers de cœurs de calcul simultanément. Le code source s'appuie sur le compilateur `nvcc` et la bibliothèque `curand` pour la génération de nombres aléatoires sur GPU.

Les paramètres de notre simulation sont définis comme suit :

— `N_SIM` (16 384) : Nombre de simulations indépendantes lancées en parallèle.
— `N` (50) : Taille du réseau, générant une grille de $(2N + 1) \times (2N + 1)$ sites.
— `NbFor` (1 000 000) : Nombre de tentatives de basculement de spins par simulation.
— `T` (2.26) : Température de simulation, proche de la température critique théorique.

4.2 Stratégie de Parallélisation

La stratégie retenue consiste à affecter une simulation complète à chaque **thread** (unité de calcul) du GPU. Contrairement à une parallélisation interne à une seule grille, cette méthode permet d'éviter les problèmes de synchronisation entre les spins d'un même réseau et maximise l'occupation du GPU.

4.2.1 Gestion de la Mémoire

Avant de lancer le calcul, l'allocation dynamique dans la mémoire globale du GPU (*Device Memory*) est nécessaire. Puisque chaque thread doit disposer de son propre espace pour stocker sa grille de spins afin d'éviter tout conflit d'écriture.

```
1 size_t mem_size = (size_t)N_SIM * DIM * DIM * sizeof(int);
2 gpuErrchk(cudaMalloc(&d_spins, mem_size));
```

Pour une configuration de 16 384 simulations d'une grille de 101×101, l'allocation représente environ 670 Mo de mémoire vidéo.

4.3 Le Kernel Metropolis-Hastings

Le *kernel* est la fonction principale exécutée sur le GPU. Dans notre code, il est nommé `metropolis_hastings`.

4.3.1 Génération Aléatoire Parallèle

L'un des défis majeurs de la parallélisation est d'assurer que chaque thread génère une séquence de nombres aléatoires différente. Pour se faire `curand_init` initialise un état propre à chaque thread en utilisant son identifiant unique (`tid`) comme *offset* :

```
int tid = blockIdx.x * blockDim.x + threadIdx.x;
curand_init(seed, tid, 0, &state);
```

4.3.2 Calcul Local de la Variation d'Énergie

Chaque thread exécute sa boucle de Metropolis de manière autonome. Pour optimiser le calcul, le thread utilise la fonction `calculVariationVoisin`. Celle-ci calcule la différence d'énergie ΔH en ne regardant que les voisins immédiats du spin choisi (haut, bas, gauche, droite), sans jamais recalculer l'énergie totale de la grille :

```
__device__ int calculVariationVoisin(int *spins, int i, int j) {
    int s = spins[i * DIM + j];
    int voisins = 0;
    if (i > 0)       voisins += spins[(i-1) * DIM + j];
    if (i < DIM - 1) voisins += spins[(i+1) * DIM + j];
    if (j > 0)       voisins += spins[i * DIM + (j-1)];
    if (j < DIM - 1) voisins += spins[i * DIM + (j+1)];
    return 2 * s * voisins;
}
```

4.3.3 Acceptation et Rollback

L'algorithme de Metropolis est implémenté avec un mécanisme de *rollback*. Le thread inverse le spin (`voisin_grille`), calcule la variation d'énergie, puis, si le changement est défavorable ($\Delta H > 0$), il décide selon une probabilité $\exp(-\Delta H/T)$ s'il doit annuler le basculement ou non.

4.4 Réduction des Résultats par Opérations Atomiques

Une fois les 1 000 000 d'itérations terminées, chaque thread examine le spin au centre de sa grille (position `N, N`). Puis, des opérateurs atomiques se chargent de sauvegarder les résultats de toutes les simulations dans deux compteurs globaux (`count_pos` et `count_neg`), sans conflit de variables partagées.

```
if (spins[N * DIM + N] == POS) atomicAdd(count_pos, 1);
else atomicAdd(count_neg, 1);
```

L'utilisation de `atomicAdd` est indispensable ici : elle garantit que même si des milliers de threads tentent d'incrémenter le compteur au même instant, chaque unité sera correctement comptabilisée sans perte d'information.

4.5 Synthèse des Performances

Grâce à cette implémentation CUDA, le temps de calcul est drastiquement réduit. Le GPU T4 permet de traiter les 16 milliards de tentatives de basculement de spins (16 384 simulations $\times$ 1 000 000 itérations) en quelques secondes. Cette efficacité permet d'obtenir une estimation précise de l'espérance du spin central, donnée par la formule :

$$E[\sigma(0)] = \frac{h_{pos} - h_{neg}}{N_{SIM}} \tag{4.1}$$

Cette valeur est ensuite transférée du GPU vers le CPU via `cudaMemcpy` pour être affichée à l'utilisateur.

Chapitre 5

Résultats Numériques

Ce chapitre présentera les résultats obtenus par l'application de l'algorithme de Metropolis-Hastings au modèle d'Ising en deux dimensions. L'objectif est d'étudier l'influence de la température T sur la magnétisation du spin central dans une configuration à bords fixes (conditions de Dirichlet).

5.1 Étude statistique sur une grille de taille modérée ($N = 20$)

La première série de simulations comportait une grille de dimension $L \times L$ avec $L = 2N + 1 = 41$ sites et les spins situés aux frontières de la grille étaient fixés de manière permanente à l'état $\sigma_{bord} = +1$ (POS). Chaque simulation consistait en 10^6 pas de Monte-Carlo par site, garantissant une exploration suffisante de l'espace des phases pour cette échelle.

5.1.1 Résultats obtenus

Le tableau 5.1 récapitule l'espérance du spin central $\langle\sigma_0\rangle$, ainsi que le décompte des états finaux positifs (P) et négatifs (N) sur l'ensemble des échantillons simulés.

TABLE 5.1 – Magnétisation du spin central en fonction de la température ($N = 20$)

Température (T)	Espérance $\langle\sigma_0\rangle$	Total Positifs (P)	Total Négatifs (N)
0.10	1,000000	196 608	0
1,00	0,999237	196 533	75
2,00	0,908545	156 348	7 492
2,20	0,779785	145 800	18 040
2,27	**Seuil critique (T_c)**	–	–
2,50	0,101332	81 199	66 257
2,70	0,011909	74 606	72 850
2,72	0,003893	74 015	73 441
3,00	-0,007216	73 196	74 260

5.1.2 Analyse de la transition de phase

L'analyse de ces données met en évidence deux régimes distincts, séparés par la température critique d'Onsager ($T_c \approx 2,269$) :

- **Phase ordonnée ($T < T_c$) :** Pour $T = 0,1$ et $T = 1,0$, l'espérance est quasi unitaire. L'énergie d'interaction J domine l'agitation thermique, et l'influence des bords positifs se propage de proche en proche jusqu'au centre de la grille. On observe une transition progressive : à $T = 2,2$, l'espérance chute à $0,779$, signe que les fluctuations thermiques commencent à concurrencer l'ordre imposé par les frontières.
- **Zone de transition ($T \approx T_c$) :** Entre $T = 2,2$ et $T = 2,5$, on observe une brisure brutale de la magnétisation. À $T = 2,5$, l'espérance n'est plus que de $0,101$. Bien que la température soit supérieure à T_c, le résidu positif témoigne de l'effet de taille finie : la grille est suffisamment restreinte pour que le centre subisse encore l'influence des parois.
- **Phase désordonnée ($T > T_c$) :** Au-delà de $T = 2,7$, l'espérance converge vers zéro (aux incertitudes statistiques près). Le système entre dans un régime paramagnétique où l'entropie domine. La longueur de corrélation ξ devient inférieure à la distance séparant le centre des bords ($N = 20$).

5.2 Étude statistique sur une grille de grande taille ($N = 100$)

5.2.1 Problématique du changement d'échelle

Le passage à une grille de taille supérieure ($N = 100$, soit $201 \times 201 = 40\,401$ sites) soulève des difficultés numériques majeures liées à la dynamique de l'algorithme de Metropolis.

Lors des essais préliminaires à basse température ($T = 0,1$) avec 20×10^6 pas, une absence de convergence fut observée. Par le fait que l'espérance restait proche de zéro malgré des conditions aux limites favorisant l'état positif.

Ce phénomène s'explique par le fait que le temps nécessaire pour que l'information (l'ordre des bords) se propage jusqu'au centre croît selon une loi de puissance de la taille L du système. Avec 20×10^6 pas pour $40\,401$ sites, chaque spin n'est sollicité qu'environ 500 fois, ce qui est insuffisant pour coordonner un système de cette envergure.

Pour pallier ce ralentissement, le nombre d'itérations devait être porté à 250×10^6 **pas** pour les simulations sur grille $N = 100$, autrement il n'est pas possible d'observer de convergence totale. Remarquons que dans certain cas, même avec un millard de pas, il n'y a pas de convergence. Le seuil de 250×10^6 **pas** a été fixé arbitrairement lors d'observations à l'œil nue.

Nous avons testé le système à deux températures distinctes, sous des conditions de bords positives. Les résultats obtenus pour $8\,192$ simulations parallèles sont consignés dans le tableau suivant :

Température (T)	Positifs (P)	Négatifs (N)	Espérance (E)
0.1	8159	33	0,991943
1.0	7979	213	0,947998
2.0	5525	2667	0.348877
3,0	4061	4131	-0.008545

TABLE 5.2 – Résultats des simulations pour N=100 et 250.10^6 itérations

À basse température ($T = 0.1$), l'espérance du spin central est extrêmement proche de 1 ($0,9919$). Cela démontre que le système est fortement ordonné : malgré la distance séparant le centre des bords de la boîte (100 sites), l'influence des conditions aux limites positives est quasi-totale.

À $T = 1$, bien que l'agitation thermique augmente, l'espérance reste très élevée ($0,9479$).

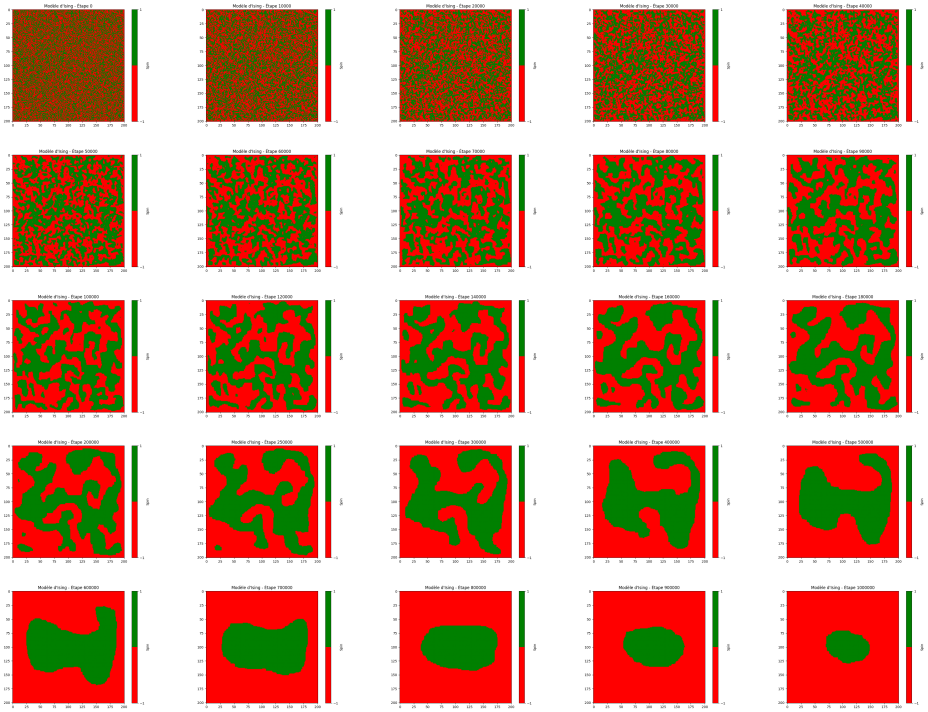

FIGURE 5.1 – Simulation de 250 millions de pas avec T = 0.1 et des bords positifs

Pour cette simulation, il est possible d'observer beaucoup de mouvement lors des premiers pas ainsi qu'une convergence certaine. Mais il est important de noter que même après 1 million de changements acceptés, le système n'a toujours pas fini sa transition.

5.2.2 Évaluation de la Puissance de Calcul

L'aspect le plus remarquable de ces essais réside dans la charge de calcul traitée par le GPU. L'exécution de ces simulations a présenté les caractéristiques suivantes :

- **Temps d'exécution** : Chaque série de simulations a duré environ 52 minutes.
- **Volume de calcul** : Avec 8 192 threads traitant chacun 250 millions de pas, le GPU a effectué un total de $2,048 \times 10^{12}$ (plus de 2 000 milliards) de tentatives de basculement de spins par session.

— **Empreinte mémoire** : Le stockage des grilles de spins et des états des générateurs aléatoires a nécessité l'allocation de $2,5$ Go de mémoire vive (VRAM) sur la carte graphique.

Une telle masse de données et de calculs aurait été inenvisageable sur un processeur classique en un temps raisonnable. La parallélisation CUDA a permis d'explorer des échelles de temps (250 millions de pas) et d'espace ($N = 100$) nécessaires pour obtenir une précision numérique de l'ordre de 10^{-6} sur l'espérance du spin, validant ainsi la robustesse de notre approche de calcul haute performance.