LOG8490 - Génie Logiciel Quantique

Hiver 2025

TP2 - IMPLÉMENTATION DE L'ALGORITHME DE GROVER

Vincent Anctil - 2148974

Mathieu Grenier - 2080754

Soumis à :

Heng Li

19 mars 2025

Table des matières

Résumé

Le rapport suivant présente l'implémentation de l'algorithme de Grover pour une chaîne de 5 qubits. Pour y arriver, nous avons développé l'oracle quantique nécessaire pour inverser la phase de la chaîne cible ***10110***. Ensuite, pour appliquer la deuxième étape de l'algorithme qui est l'inversion autour de la moyenne, nous avons construit la phase de diffusion qui sera appliquée au circuit. Avec ces deux implémentations, il est donc possible de construire l'algorithme de Grover. Dans ce rapport, vous verrez dans l'ordre suivant : le contexte et les motivations derrière ce travail, l'explication approfondie de chacune des étapes de notre implémentation, le rapport et l'analyse de nos résultats, pour finir avec la conclusion qui résume les points clés du rapport et un compte-rendu de la contribution de chaque membre.

Introduction

Plusieurs motivations premières sont présentes dans ce rapport. D'abord, le but de ce laboratoire est de se familiariser avec l'algorithme de Grover, qui est un algorithme de recherche qui trouve avec une forte probabilité l'entrée unique qui produit une sortie ciblée. Pour y arriver, il est essentiel de comprendre comment fonctionne un oracle quantique et de l'implémenter efficacement. Ensuite, une autre motivation est de développer l'entièreté de l'algorithme et de l'exécuter sur un ordinateur quantique réel. Dans le rapport, vous verrez la divergence des résultats lorsque le circuit est exécuté dans une simulation comparativement à un ordinateur quantique par l'entremise de l'infonuagique. Enfin, chacune des étapes de l'algorithme ont été travaillées ce qui a concrètement renforcé notre compréhension de la théorie vue en classe.

Description de l'approche

Implémentation de l'oracle (inversion de phase)

L'oracle a été implémenté en utilisant le principe de retour de phase. Les qubits fonctionnels sont placés dans l'état $|+\rangle$ et un qubit auxiliaire est placé dans l'état $|-\rangle$. En appliquant une porte CNOT, ayant comme contrôle les qubits fonctionnels et comme cible le qubit auxiliaire, il y a un retour de phase. C'est-à-dire que la phase de l'état spécifique qui enclenche la porte CNOT aura une phase négative. Cette solution peut être démontrée par le circuit suivant. Dans le circuit suivant, la fonction *f(x)* est égale à 1 lorsque *x* est égale à l'état cible. Dans notre cas, cet état est le "10110". L'oracle à implémenter doit donc appliquer XOR *f(x)* au qubit auxiliaire.

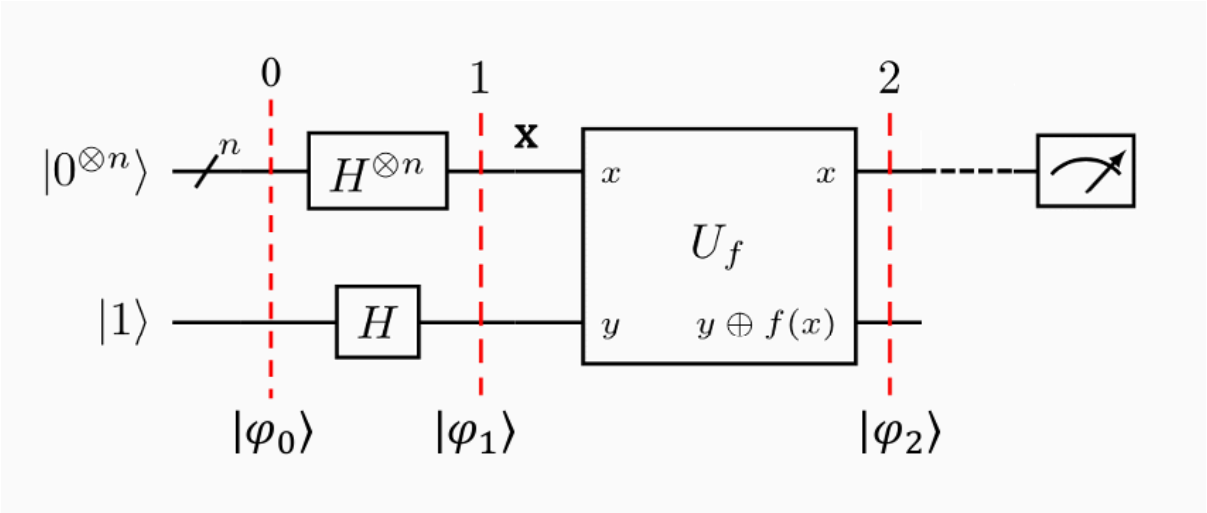


Figure 1 : Schéma de l'oracle dans l'algorithme de Grover

Afin d'implémenter ceci, la porte CNOT est utilisée, car elle est équivalente à une porte XOR. Celle-ci va appliquer le XOR si et seulement si la condition des qubits de contrôles est rencontrée. Dans notre cas, celle-ci est:

- $q_0 = 0$
- $q_1 = 1$
- $q_2 = 1$
- $q_3 = 0$
- $q_4 = 1$

Ce qui équivaut à 10110.

Dans le circuit suivant, l'oracle est représenté par la porte CNOT. Les portes hadamard et la porte NOT sont utilisées pour placer le système dans l'état approprié.

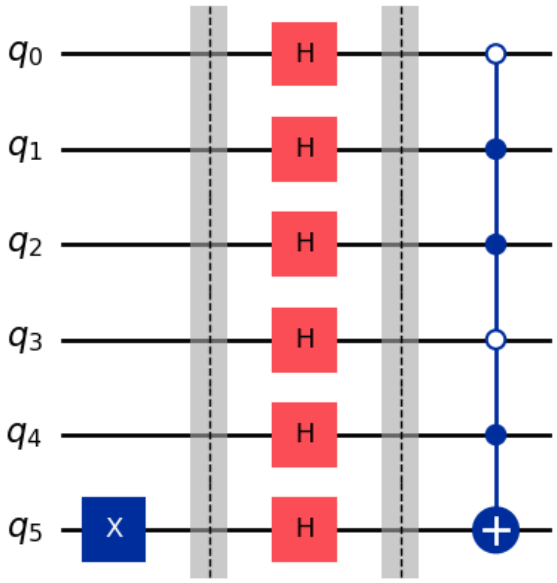


Figure 2 : Visualisation de l'oracle dans l'algorithme de Grover

Afin de tester la performance de l'oracle, on visualise les amplitudes des différents états. Comme on peut voir, l'état 10110 a été inversé tandis que les autres sont restés positifs.

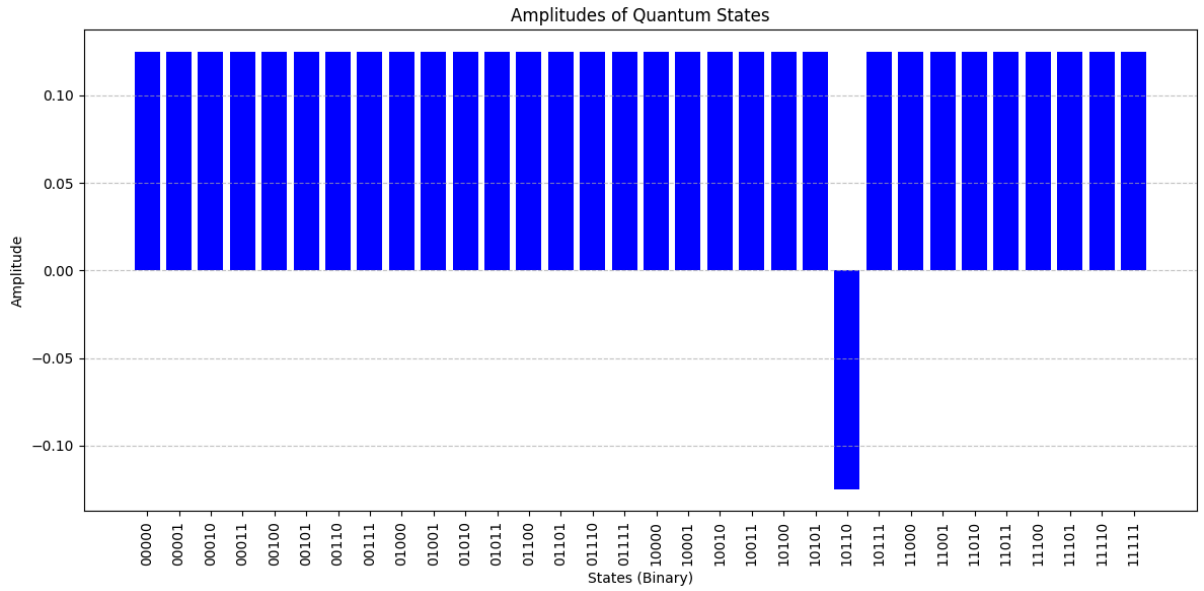


Figure 3 : Tableau de la simulation de l'oracle

La représentation matricielle de l'oracle peut être calculée en utilisant la notation ket-bra et la table de vérité du circuit.

$$U = \sum_{i=0}^{N-1} |entrée_i\rangle\langle sortie_i|$$

Comme tous les entrées et sorties sont identiques à part pour l'état affecté par le NOT, on peut dire qu'il s'agit d'une matrice identité à part pour l'état $|101101\rangle$ devenant l'état $|101100\rangle$ et l'état $|101100\rangle$ devenant l'état $|101101\rangle$.

```
def naive_oracle(qc: QuantumCircuit, ctrl_state: str, total_qubit_count: int, functional_qubit_count: int):
    auxilary_bit_index = total_qubit_count - 1
    functional_qubit_indexes = list(range(functional_qubit_count))
    qc.mcx(functional_qubit_indexes, auxilary_bit_index, ctrl_state=ctrl_state)
```

Figure 4 : Code de l'oracle

Implémentation de la diffusion (inversion autour de la moyenne)

L'étape qui suit l'implémentation de l'oracle quantique est celle de la diffusion. Cette étape est définie par l'inversion autour de la moyenne. Pour ce faire, nous avons appliqué une porte hadamard sur tous les qubits. Cette étape transforme tous les qubits dans une superposition égale de $|0\rangle$ et $|1\rangle$. Elle est essentielle pour préparer les qubits aux manipulations autour de l'amplitude. Ensuite, la porte X est aussi appliquée à tous les qubits puisqu'elle permet de changer le signe des amplitudes de manière contrôlée. Le milieu de l'algorithme est l'opération centrale de la diffusion. La porte hadamard est appliquée au dernier qubit afin de le préparer à l'inversion de phase traditionnelle. Suivant celle-ci se trouve la porte multi-contrôlée X (MCX), qui inverse l'état du qubit cible si tous les qubits de

contrôle sont dans l'état |1⟩. C'est cette porte qui effectue l'inversion autour de la moyenne en marquant les états qui sont au-dessus de la moyenne des amplitudes, en inversant leur phase. Pour compléter l'inversion, la porte hadamard sur le dernier qubit est essentielle. Finalement, il est important d'appliquer une deuxième fois les portes X sur tous les qubits afin d'annuler l'effet des premières portes et d'une dernière série de portes hadamard pour terminer l'inversion autour de la moyenne, ce qui amplifie l'état cible.

```python
def naive_diffuser(qc: QuantumCircuit, functional_qubit_count: int):
    for qubit in range(functional_qubit_count):
        qc.h(qubit)

    for qubit in range(functional_qubit_count):
        qc.x(qubit)

    control_qubit_indexes = list(range(functional_qubit_count - 1))
    final_functional_qubit_index = functional_qubit_count - 1

    qc.h(final_functional_qubit_index)
    qc.mcx(control_qubit_indexes, final_functional_qubit_index)
    qc.h(final_functional_qubit_index)

    for qubit in range(functional_qubit_count):
        qc.x(qubit)

    for qubit in range(functional_qubit_count):
        qc.h(qubit)
```

Figure 5 : Code du diffuseur

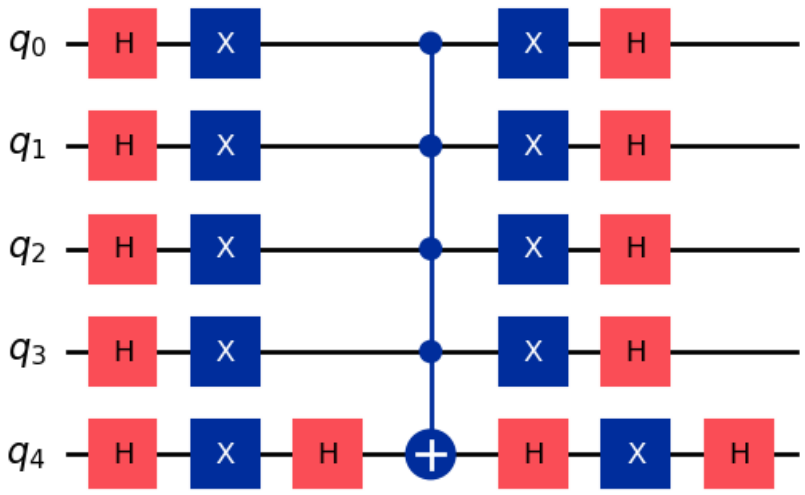
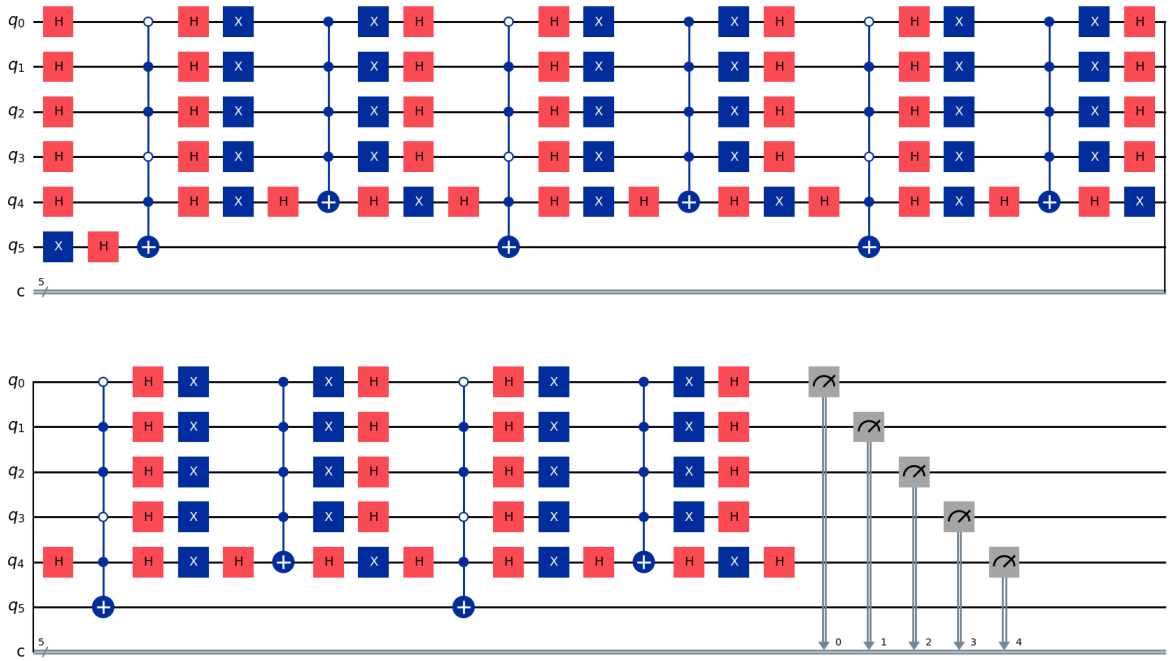


Figure 6 : Visualisation du diffuseur implémenté

Implémentation de l'algorithme de Grover

L'algorithme de grover a été implémenté en répétant $\sqrt{N}$ fois l'exécution de l'oracle suivit du diffuseur, où *N* est égal au nombre d'états de bases possibles.



Figures 7 et 8 : Visualisation de l'algorithme comportant l'oracle et le diffuseur

Exécution de simulation classique avec AER

La simulation classique a été faite à l'aide du *AerSimulator* avec la méthode *statevector*. Cela permet de simuler le circuit sans bruit afin d'évaluer ses performances. Puisque la méthode *statevector* permet de calculer l'état quantique à chaque étape, elle nous a permis de simuler l'exécution complète du système quantique, sans bruit ni décohérence. Cela signifie que le résultat obtenu devrait être exact s'il était produit dans un environnement idéal, sans les imperfections des ordinateurs quantiques réels.

```
qc = naive_grover('10110')

aer_sim = AerSimulator(method='statevector')
qobj = transpile(qc, aer_sim)
result = aer_sim.run(qobj).result()

counts = result.get_counts()
plot_histogram(counts)
```

Figure 9 : Exécution de la simulation avec AER

Exécution de simulation d'un ordinateur quantique

La simulation d'un ordinateur quantique a été utilisée à l'aide du backend *ibm_sherbrooke*. Cela permet de voir les performances du circuit avant d'utiliser un vrai ordinateur quantique.

Une optimisation de niveau 3 a été utilisée afin d'avoir le circuit le plus court possible, afin de réduire l'effet de décohérence et de bruit dû à un circuit ayant une trop grande profondeur.

```
# Run our circuit on the backend.
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler

# Replace 'YOUR_API_TOKEN' with your actual IBM Quantum API token
token='dfda9ea30f8353753911497de4aaec89e6e4f3a5e7a88149c16fbc2983ed9c8c93cbc6a8c7385930d2af8383b71cc0d6c613562e3d3a4cdc6c05302da8a92b0b'
QiskitRuntimeService.save_account(
  token=token,
  channel="ibm_quantum", # `channel` distinguishes between different account types
  overwrite=True
)

service = QiskitRuntimeService()
```

Figure 10 : Configuration du service IBM et du jeton API

```
qc = naive_grover('10110')

# Specify the backend to use
backend = service.backend("ibm_sherbrooke")
sim_backend = AerSimulator.from_backend(backend)

qobj = transpile(qc, sim_backend, optimization_level=3)
result = sim_backend.run(qobj, shots = 4096).result()
counts = result.get_counts()
print(counts)
plot_histogram(counts)
```

Figure 11 : Simulation avec ibm_sherbrooke

Exécution sur un ordinateur quantique

L'exécution sur un ordinateur quantique a été faite sur l'ordinateur étant le moins occupé au moment du laboratoire. Cela signifie que le circuit est envoyé via infonuagique à un processeur quantique physique. Dans notre cas, nous avons utilisé *ibm_brisbane*. Encore une fois, une optimisation de niveau 3 a été utilisée afin d'avoir le circuit le plus court possible. Cette exécution a permis de tester les performances des algorithmes quantiques dans un environnement réel.

```
# choose the least busy QPU
real_backend = service.least_busy(simulator=False, operational=True)
print(real_backend)
print(real_backend.status())
```

```
<IBMBackend('ibm_brisbane')>
<qiskit_ibm_runtime.models.backend_status.BackendStatus object at 0x00000264DA806430>
```

```
qc = naive_grover('1011')

qobj = transpile(qc, real_backend)

# Define Sampler: https://docs.quantum.ibm.com/guides/get-started-with-primitives
sampler = Sampler(mode=real_backend)

# Run calculation
job = sampler.run([qobj])

result = job.result()
```

Figure 12 : Exécution sur l'ordinateur quantique d'ibm brisbane

Rapport et discussion

Résultat de la simulation classique avec AER

La simulation classique démontre que l'état cible a été privilégié, démontrant ainsi la performance de l'algorithme de Grover. Il est normal que d'autres états aient été observés, car le système est en état de superposition, donc la probabilité d'observer un autre état n'est pas nul. Ainsi, la grande majorité des tirs ont été sur la cible, mais d'autres valeurs ont été obtenues.

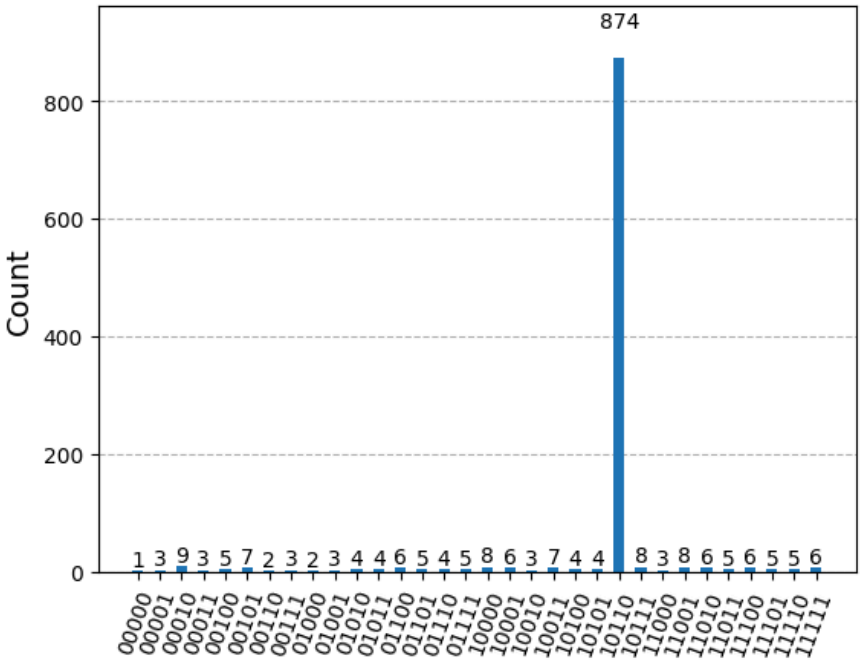


Figure 13 : Tableau des résultats de la simulation classique

Résultat de la simulation d'un ordinateur quantique

Les résultats suivants démontrent la performance de l'algorithme sur une simulation d'ordinateur quantique. Le simulateur est incapable de trouver l'état privilégié avec une chaîne de 5 caractères. Cela est probablement dû au bruit et au temps de décohérence qui est augmenté à cause de la complexité du circuit. Comme le circuit est très profond, plus de portes quantiques sont appliquées, plus il y aura du bruit. En réduisant à 4 caractères, on voit que l'algorithme est capable de trouver l'état cible. Il y a donc une corrélation directe entre la performance de l'algorithme et le nombre d'états possibles. En réduisant à 3 caractères, la différence entre l'état cible et les autres états est encore plus grande.

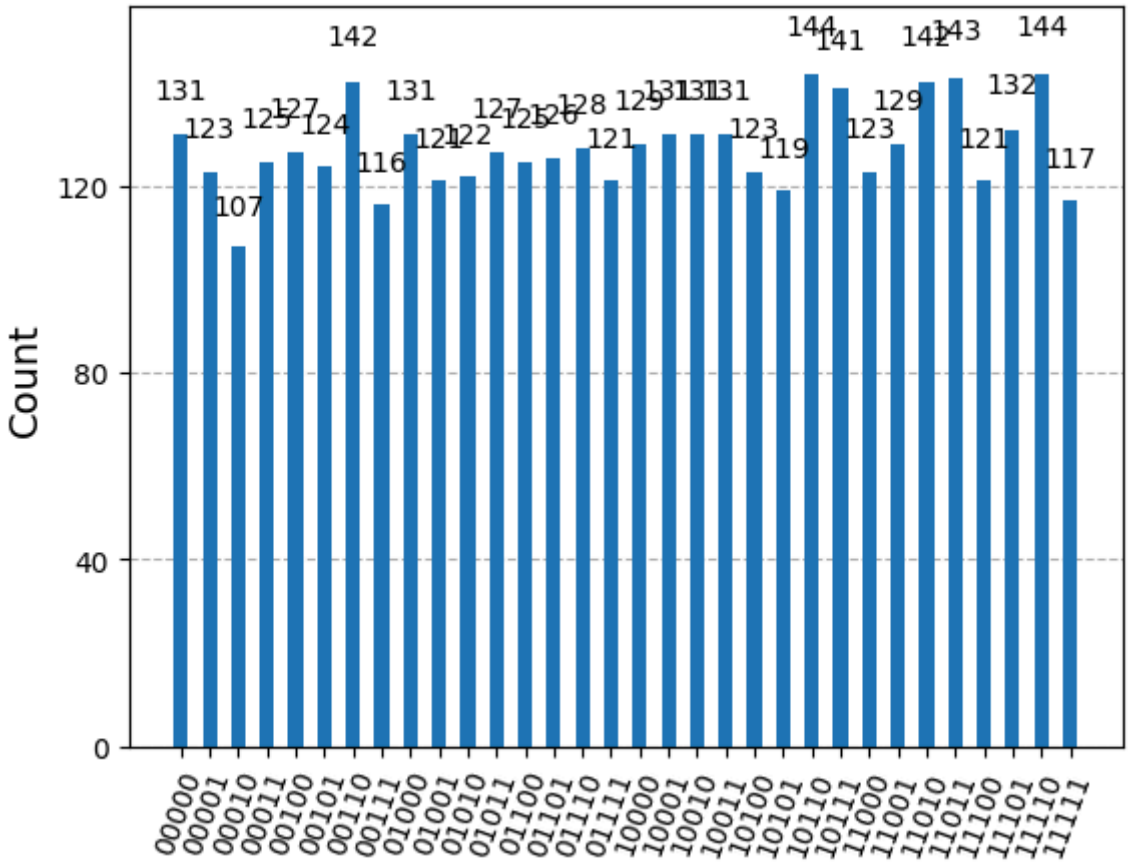


Figure 14 : Tableau des résultats de la simulation avec 5 qubits sur l'ordinateur quantique avec état cible *10110*

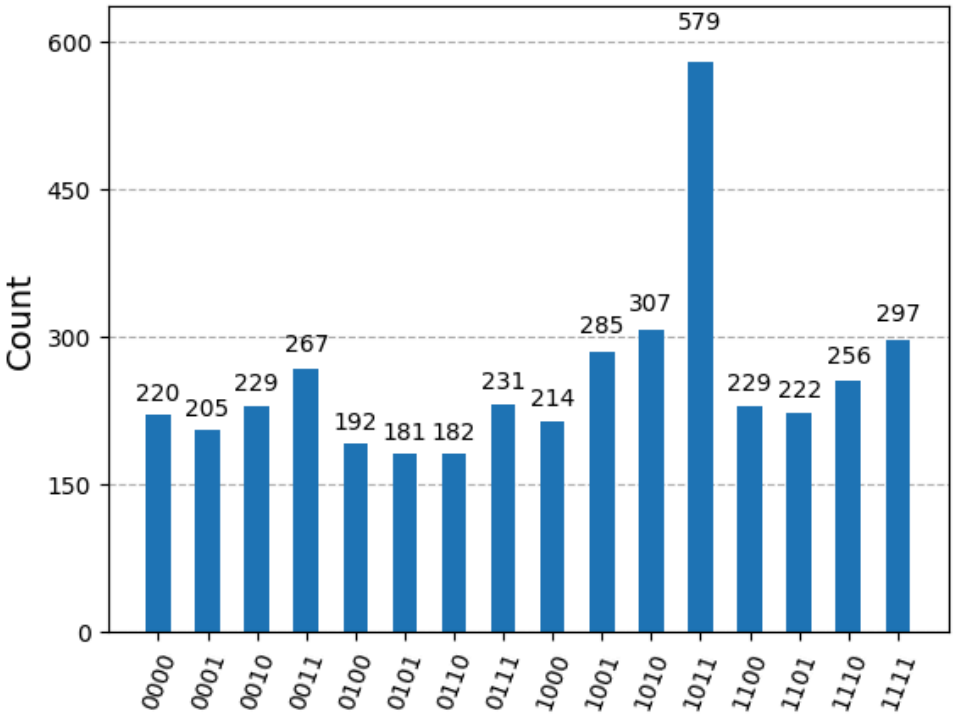


Figure 15 : Tableau des résultats de la simulation avec 4 qubits sur l'ordinateur quantique avec état cible *1011*

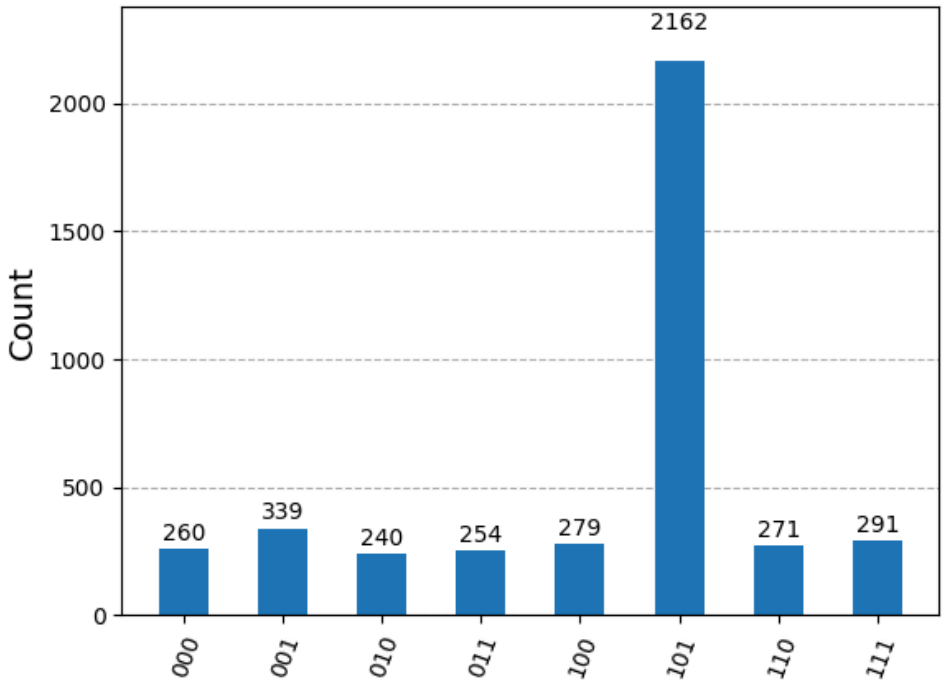


Figure 16 : Tableau des résultats de la simulation avec 3 qubits sur l'ordinateur quantique avec état cible *101*

Afin de tester l'hypothèse que la décohérence est dû à la profondeur de l'algorithme, on teste l'algorithme avec un nombre plus petit d'itérations. Ici, seulement une itération est utilisée, donc le circuit est beaucoup plus court.

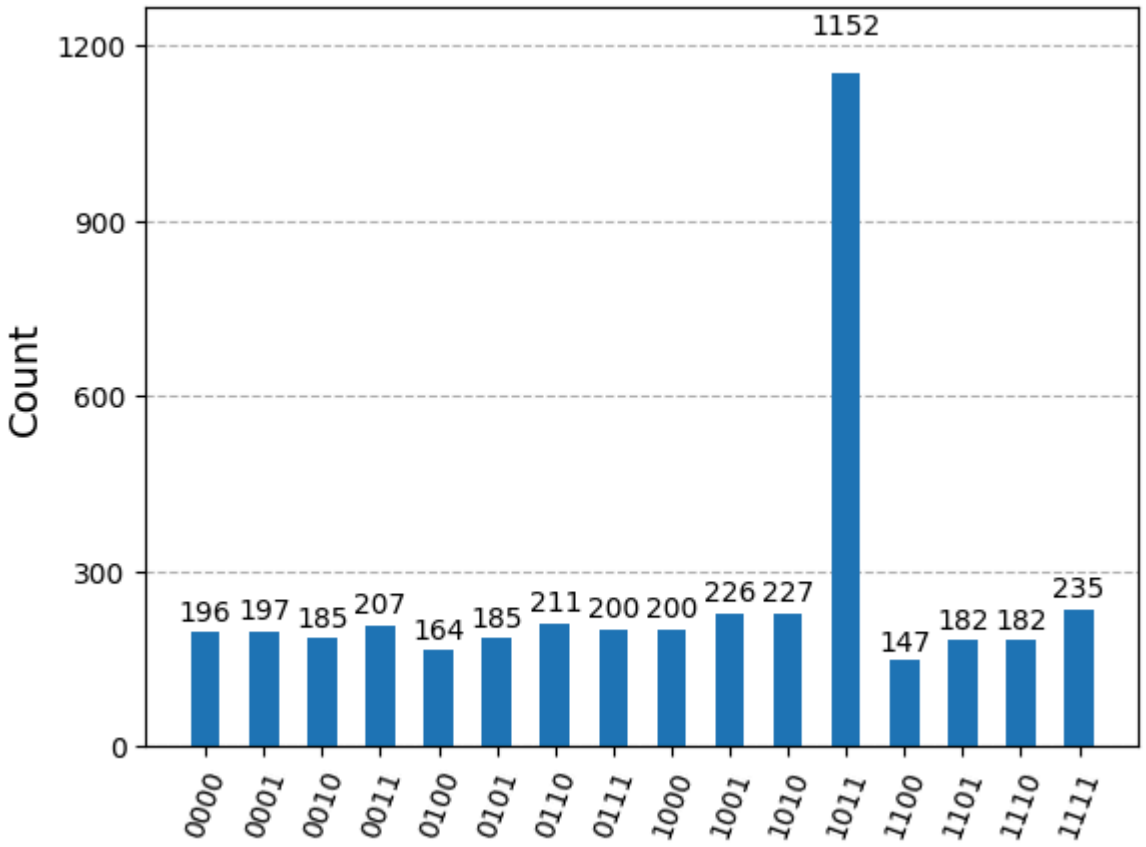


Figure 16 : Tableau des résultats de la simulation avec 4 qubits sur l'ordinateur quantique avec état cible *101* avec un circuit plus court

On peut voir que les résultats sont beaucoup mieux qu'initialement, prouvant ainsi que la décohérence est dû à la profondeur du circuit.

Résultat de l'exécution sur un ordinateur quantique

Lorsqu'il s'agit de l'exécution sur un vrai ordinateur quantique, le bruit a un effet encore plus grand sur la performance de l'algorithme. Pour les chaînes de 4 et 5 caractères, l'algorithme est incapable de cibler correctement la solution. Pour les chaînes de 3 caractères, une différence marquée apparaît entre l'état cible et les autres états. Ceci est probablement dû au bruit, car le circuit utilisé a une trop grande profondeur. Il y a donc plus de portes quantiques à exécuter, ce qui augmente aussi le temps d'exécution. Le temps d'exécution doit être le plus bas possible, car le temps de décohérence est le temps nécessaire aux qubits pour perdre leur *vérité*. Ainsi, les résultats deviennent aléatoires après un certain temps.

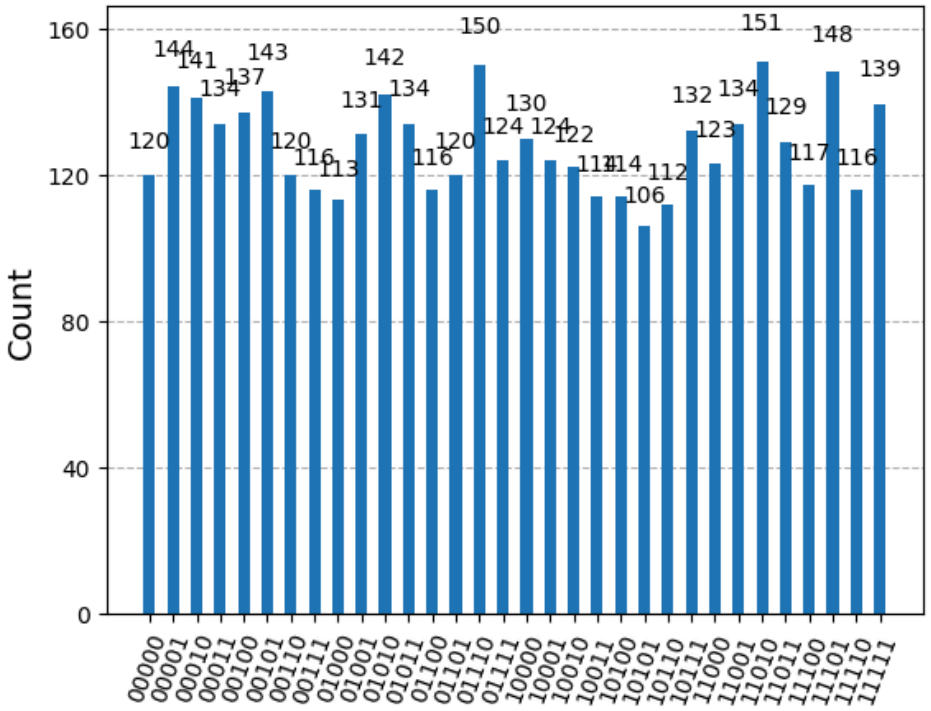


Figure 17 : Tableau des résultats avec 5 qubits sur l'ordinateur quantique avec état cible *10110*

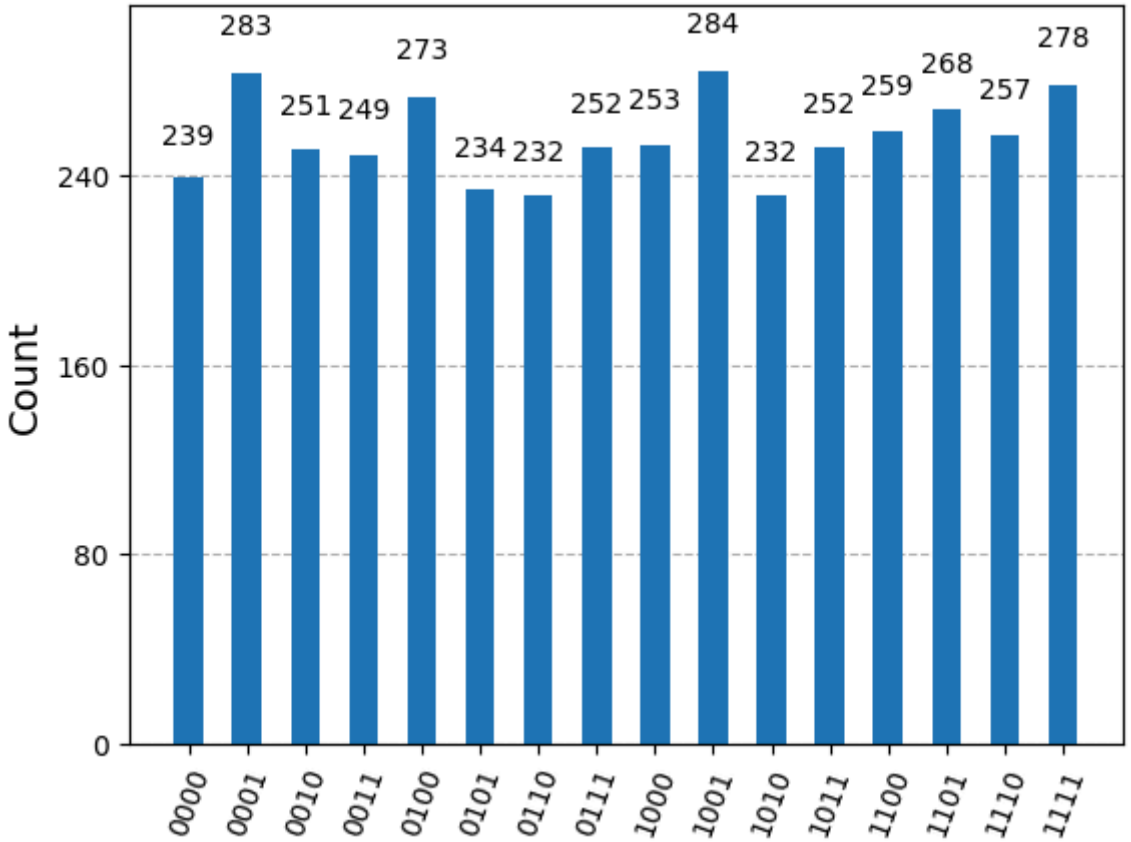


Figure 18 : Tableau des résultats avec 4 qubits sur l'ordinateur quantique avec état cible *1011*

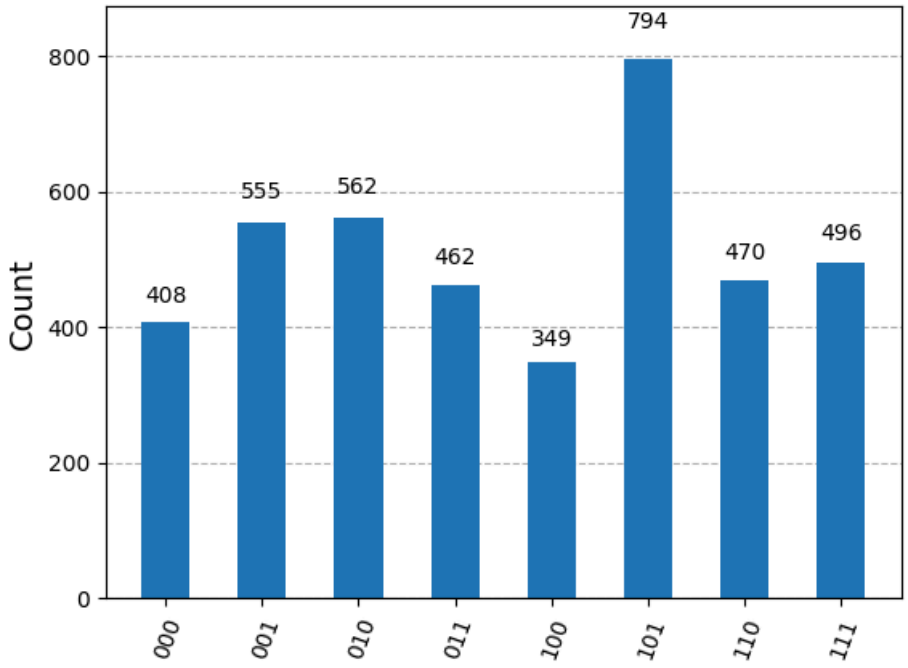


Figure 19 : Tableau des résultats avec 3 qubits sur l'ordinateur quantique avec état cible *101*

La même technique de test de profondeur a été utilisée en réduisant le nombre d'itérations.

Conclusion

Ce travail pratique nous a permis d'implémenter et de tester l'algorithme de Grover pour une recherche de chaîne binaire. Les simulations classiques ont validé notre algorithme, composé de l'oracle et de diffuseur, mais les simulations et exécutions sur ordinateurs quantiques réels ont révélé les défis du bruit et de la décohérence. La profondeur du circuit a un impact direct sur la précision des résultats, avec des performances dégradées pour des chaînes plus longues. En effet, plus le circuit est complexe, plus il est sensible aux erreurs induites par le bruit et la décohérence. Nous avons donc utilisé un degré plus grand d'optimisation, ce qui a atténué ces effets. Cette expérience a souligné l'importance de la pratique pour comprendre les limitations et le potentiel des ordinateurs quantiques.

Contribution de l'équipe

Nous avons tous les deux contribué de manière équivalente dans le projet. Que ce soit en présence durant les séances de laboratoires ou à distance, de l'implémentation de l'algorithme de Grover à la rédaction du rapport, nous avons travaillé ensemble pour livrer un travail respectable et honnête.