# Faculté des sciences

# Université de Sherbrooke

# Projet de session

Par

Mathieu Beaudoin — 17 088 376

Michael Veilleux — 17 062 900

Destiné à : Daniel-Junior Dubé

# IFT 630

# Processus concurrents et parallélisme

Introduction

L’objectif de notre projet était d’explorer le concept de « blockchain », apporté par le fondateur du bitcoin et créateur de la première « blockchain », en 2009, Satoshi Nakamoto. Nous l'avons choisi parce que ce concept utilise le parallélisme, et aussi un univers de la cryptographie que nous voulions explorer.

En surface, une chaîne de bloc est une technologie de stockage et de transmission d’informations sans organes de contrôle. Similaire à un répertoire, chaque bloc contient une transaction, traçable depuis sa création. C’est une base de donnée peer-to-peer (réseaux d’ordinateurs interconnectés), ou chaque nouvelle information, générée à l’aide d’un nouveau bloc depuis ses successeurs, est vérifiée par l’entièreté du réseau et destinée à ne jamais être modifiée ou supprimée. Le registre, étant synchronisé sur un réseau d’ordinateurs, n’a pas d’administrateurs ni stockage de données, et doit se baser sur un algorithme de consensus pour se fier à un ajout de bloc.

Le concept de « miner » une cryptomonnaie est le fait de regarder les transactions de plusieurs blocs dans le « blockchain » dans une tranche de temps, pour vérifier qu'il n’a pas été falsifié avec le concept du double spending (utiliser deux fois le même montant). Un bloc contient une clé publique accessible à tous et une clé privée visible par le propriétaire. Comme le « blockchain » est accessible à tous, le travail d’un mineur est de vérifier la demande de transaction d’un utilisateur, accumuler et valider, pour ensuite ajouter le bloc. Le principe de la vérification est que l’utilisateur génère une valeur à l’aide d’un message et de sa clé privée. Puis il utilise le message et la clé publique et crée une valeur, vérifiant que les deux correspondent et prouvant que le bloc n’a pas été illégalement modifié. Pour ajouter un bloc, un mineur doit donner une preuve cryptographique sous forme de hash en le passant dans une fonction de hachage. Il est impossible de prédire le hash, donc on doit faire le calcul en boucle jusqu’à ce qu’on atteigne le nombre de zéro valide, en incrémentant une valeur qu’on appelle « nonce », qui est concaténée.

Le domaine de la cryptomonnaie est de nos jours très populaire, principalement parce qu’elle est une monnaie alternative, qui ne peut être déclinée. Une récompense est donnée à la personne ayant ajouté un bloc (par le même fait trouvé la solution au puzzle, où la probabilité est égale à la puissance de calcul fournie divisée par la puissance de calcul total). Le paiement peut commencer après avoir analysé 1 MB de données et il survient lorsqu’un mineur est le premier à donner une réponse valide à un problème (deviner un nombre hexadécimal de 64 caractères égaux ou inférieurs à la cible). Pour ce faire, un mineur doit avoir un grand « hashrate » par secondes. La plupart des mineurs de cryptomonnaie utilisent des « mining pool » afin d’optimiser leurs gains. Cela permet de contribuer au minage d’un bloc et d’être payé par une fraction de cryptomonnaie selon le « hashrate fournis/hashrate total ». Lorsque le bloc est miné (ou qu’un autre « pool/mineur » l’a miné), on change de bloc. Il y a plusieurs types de communications entre pools et mineur, un populaire est Stratum : il se charge d’attribuer les calculs par mineurs sans dupliquer de travail.  
  
 La cryptomonnaie générée de façon graduelle et qui est accessible par tous, suit le même fonctionnement que la vraie monnaie. Par contre, il y a un concept intéressant, appelé le « Halfing », où la somme totale donnée aux mineurs réduit de moitié. Cela survient environ aux quatre ans (à tous les 210 000 blocs). Présentement, pour Bitcoin, la somme est de 12,5 bitcoins par 10 minutes et devrait se reproduire en 2020.

Objectifs

Notre projet de départ était de concevoir un programme qui implément des fonctions de minage pour Ethereum, mais la complexité du projet dépassait énormément ce que nous avions anticipé. La difficulté de devoir s’intégrer à un « mining pool » et aussi le fait que nous n’avons pas trouvé l’information nécessaire à miner cette cryptomonnaie de façon compacte (signature, type de « hash » utilisé, etc) nous a fait décider de simplement simuler le processus de minage d’une cryptomonnaie. De plus, le temps requis était trop élevé. En effet, en nous basant sur des projets déjà existants, nous avons vu qu’il fallait créer une structure de « blockchain », un serveur « peer-to-peer », une connexion à un « mining pool stratum » (qui répartit les tâches de calcul, ce que nous voulions faire nous-mêmes), des validations au niveau des « peers » et données et beaucoup d’autres choses.

Nous avons passé plusieurs heures et mêmes journées à explorer les différents aspects de « blockchain » et de cryptomonnaie, soit par la lecture à ce sujet sur divers sites et en regardant plusieurs vidéos qui approfondissaient certains concepts. Ce qui nous a amené à rejeter notre première hypothèse, parce que nous n’avons pas réussi à miner une vraie cryptomonnaie et par le même fait aucun gain monétaire n’a été engendré. Par contre, après toutes ces recherches et lectures, nous avons élevé notre compréhension au sujet de « blockchain » et de la cryptomonnaie ce qui était un des objectifs de ce projet.

Description du travail

Notre simulation se concentrait sur la partie de hachage d’une chaîne de caractères et sur l’algorithme qui permettait de trouver le bon « nonce ». Cet algorithme prend l’index du bloc en cours de minage, le « timestamp » de création, le « hash » du bloc précédent et un « nonce » incrémenté dans une boucle, « hash » le résultat en sha256 pour le signer. La signature d’un hash varie selon la monnaie. Souvent, il est question d’avoir un certain nombre de 0 au début du « hash ». Dans notre implémentation, ce nombre de 0 est indiqué par une constante du nom de « difficulty ». Par défaut il était à 5. Plus on incrémente ce nombre de 0, plus il est long pour le programme pour trouver le bon « nonce ».

Outils utilisés  
Pour la conception du programme, nous avons utilisé Ubuntu 18.04, avec un éditeur de texte. Pour compiler la partie CUDA, nous avons fait un « makefile » qui prenait en compte toutes les versions de CUDA pour rétrocompatibilité. Pour le reste, nous exécutions avec g++. Lors de l’appel à CUDA, nous avons créé un processus enfant du fichier exécutable, en passant le « hash » et la difficulté en paramètre au « main ». Les fichiers SHA256 ont été pris d’internet, car ce sont des algorithmes génériques. D’autres fichiers ont été inspirés de solution sur internet et adaptés à notre situation.

Résultats

Pour nos tests de performance, nous avons créé 10 blocs un après l’autre. À chaque création de blocs nous minions le bloc pour avoir un « hash » signé. Pour chaque niveau de difficulté, le nombre de calculs se multipliait par un facteur de 16. Nous avons choisi une difficulté de 5 par défaut, car ce n’était pas trop long, mais assez pertinent pour voir une bonne différence entre nos tests. Nous avons effectué tous nos tests sur la même machine pour garder les données fiables. Nous avons effectué le test de 10 blocs avec difficulté de 1 à 8, un total de 10 fois. Nous avons fait la moyenne des 10 exécutions du programme pour chaque type d’exécution et chaque difficulté, soit séquentielle, « multithread » et en utilisant le GPU CUDA pour paralléliser. Voici un petit tableau qui montre les résultats obtenus.

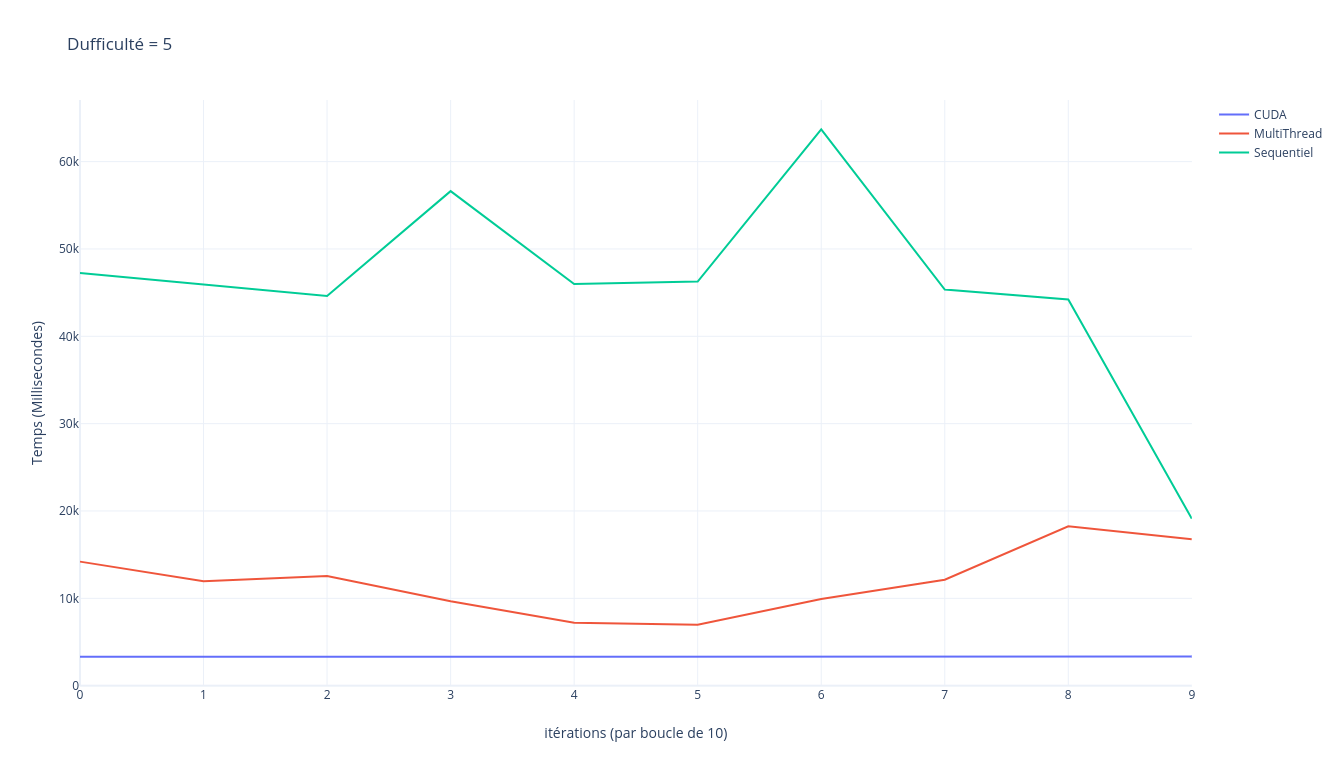
Temps moyen d’exécution selon le type d’exécution

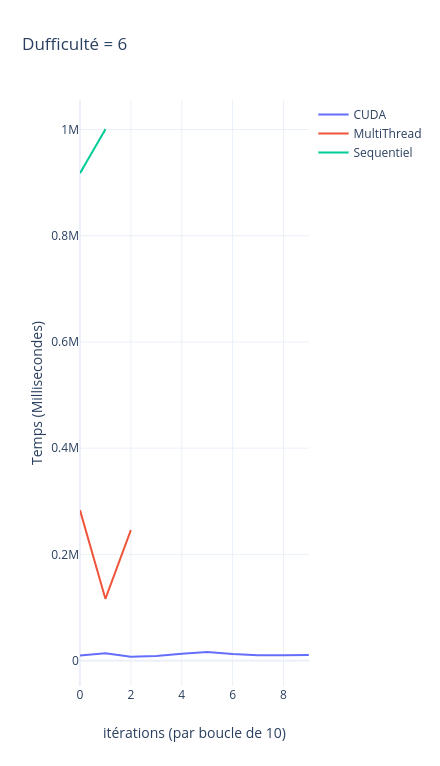
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type d’exécution | Temps moyen des 10 tests | | | |
| Difficulté | 4 | 5 | 6 | 7 |
| Séquentiel | 3946,6 | 45806.6 ms | 917 721\*\* | TLDT |
| Multi Thread | 803.2 | 11963.5 ms | 215 078 333\*\* | TLDT |
| CUDA | 2905.3e | 3325.3 ms | 11205.6 ms | 135 119\*\* |

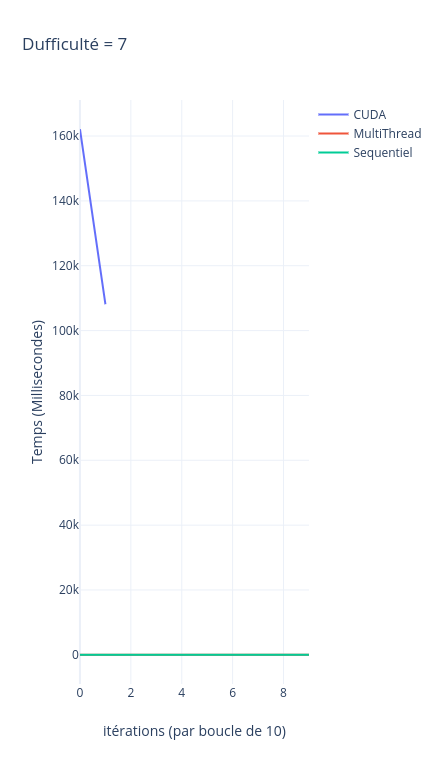
TLDT\* = Too Long Didn`t Test  
\*\* = Par souci de temps nous n’avons pas effectué ce test 10 fois.

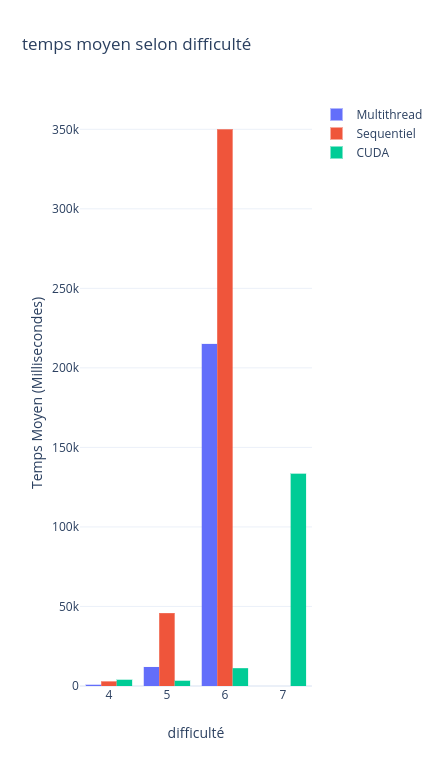
Graphiques pour les visuels

Nous avons fait des graphiques avec les résultats obtenus, mais pour obtenir les résultats exacts voir TempsBenchMark.txt









Analyse des résultats

À l’aide de notre simulation, nous avons pu confirmer notre deuxième hypothèse, qui était que la parallélisation du minage de blocs améliore grandement nos résultats dans les cas où il y a beaucoup de calculs à faire. En effet, si on regarde le dernier graphique, on peut voir que pour une difficulté à 6, CUDA est nettement plus rapide que « multithread », qui lui est beaucoup plus rapide que séquentiel (qui est supposé être près de 1000 secondes). Par contre, pour une difficulté de moins de 4, l’initialisation des fonctionnalités CUDA prend plus de temps qu’un calcul séquentiel.

Difficultés/problèmes rencontrés

Nous avons rencontré beaucoup de problèmes durant le développement de notre programme. Surtout en ce qui concerne la simulation CUDA. Au début, nous compilions les cpp sur une vm Ubuntu. Mais cette vm ne supportait pas CUDA. Alors nous avons essayé de le compiler avec Microsoft Visual Studio. Sur Windows le problème rencontré était dans la communication interprocessus, parce que nous voulions créer un nouveau processus qui exécutait un programme CUDA et recevoir ce processus par « IPC message passing ». En cherchant une solution, nous avons trouvé et installé Windows Subsystem for Linux, en pensant que la compilation allait utiliser les headers Windows et Linux en même temps. Malheureusement, ça ne fonctionnait pas comme ça. Par contre, avec ce nouvel outil, nous pouvions faire fonctionner nvcc et CUDA sur Ubuntu. Malheureusement, les gpu de nos ordinateurs portables n’étaient pas assez puissants, alors nous sommes allés au laboratoire d’informatique poursuivre le projet. Certains ordinateurs du laboratoire avaient des problèmes avec leur Ubuntu et leur drivers NVIDIA. Nous avons trouvé la machine DINF-1023-16 qui fonctionnait parfaitement. Nous avons utilisé cette machine pour tous nos tests.

Conclusion

Nous avons aimé créer un projet par nous-mêmes et l’expérience à été enrichissante en connaissances autant la rédaction du rapport que la conception du programme. Nous croyons que nos objectifs ont été atteints malgré le changement d’envergure du projet initial. Notre travail pourrait faire preuve d’améliorations au niveau du code (optimisation). Il pourrait aussi avoir une meilleure interface graphique. Nous aurions pu mettre des données plus réalistes qui sont utilisées dans les cryptomonnaies courantes.