Implémentation FPGA optimisée de Q-Learning

MAROUANE BEN-AKKA, CAMEL TANOUGAST, CAMILLE DIOU, HARRY RAMENAH

Université de Lorraine, LCOMS, F-57000, Metz, FRANCE

Résumé – Cet article présent un accélérateur matériel dédié aux algorithmes d'apprentissage par renforcement Q-Learning, basé sur une architecture optimisée de gestion interne des données de la Matrice-Q en utilisant des mémoires intégrées LUTRAM fonctionnant en mode FIFO écriture synchrone/lecture asynchrone. Comparativement à des travaux antérieurs, les résultats d'implémentation sur cible FPGA Xilinx Zynq UltraScale+ MPSoC ZCU104 montrent que l'architecture proposée, traitant des données matrice Q de taille 16 bits, un nombre d'actions Z=4 et différents scénarios selon un nombre d'états, nécessite une consommation dynamique maximale de 46 mW, une réduction de ressources jusqu'à 37 % de LUTRAM et 76 % de FF pour une fréquence maximale de 168MHz. L'architecture a été validée dans l'application du jeu de Labyrinthe (environnement de taille 2x3) avec une fréquence maximale de 74 MHz tout en nécessitant uniquement 202 slices LUT et 102 slices register avec la technologie FPGA Virtex-6 ML605.

Abstract – This paper proposes an optimized configurable logic architecture-based Q-Learning hardware accelerator. This architecture ensures an efficiency storage management of inner Q-Matrix data by using embedded LUTRAM memories configured as FIFO mode with synchronous writing/asynchronous reading. Comparing to previous works, the implementation of the proposed architecture in the Xilinx FPGA Zynq UltraScale+ MPSoC ZCU104 performing 16 bits data-length Q-Matrix processing with a number of actions Z=4 and different scenarios in terms of number of states, only requires a maximal dynamic power consumption of 46 mW while reducing down to 37% of LUTRAM and 76% of FF at a frequency processing of 168MHz. Similarly, considering the FPGA Virtex-6 ML605, the proposed architecture is validated for a scenario of maze game guiding a robot in an arena (Grid World 2*3) and operating at a maximum frequency of 74 MHz while requiring only 202 slice LUTs and 102 slice registers.

1 Introduction

Les algorithmes d'apprentissage automatique par renforcement (*Reinforcement Learning* – RL) sont de plus en plus utilisés dans différents domaines [3-10] et leurs implémentations matérielles restent un défi pour assurer le traitement volumineux de données sous contraintes temporelles pour des applications embarquées [1]. Q-Learning est un algorithme d'apprentissage RL de type hors stratégie (« off-policy »), permettant à un agent d'apprendre à effectuer des actions pour maximiser ou optimiser des récompenses obtenues, et ceci de façon autonome [2]. L'implémentation FPGA de Q-Learning repose sur une parallélisation soit du nombre d'états N ou des actions Z, et sur la base de stockages des valeurs d'état-action $Q(s, a)$ (Matrice-Q) dans des mémoires RAM [11-12]. Cet article présente un accélérateur matériel Q-Learning basé sur une architecture FPGA optimisée exploitant une parallélisation des nombres d'actions Z par une gestion efficace des phases lecture/écriture des mémoires (LUTRAM) pour stocker les valeurs d'état-action $Q(s, a)$. Cette gestion lecture/écriture des mémoires de stockage des valeurs de la Matrice-Q assure une optimisation des ressources matérielles utilisées, une réduction de la puissance dynamique consommée tout en améliorant la fréquence maximale de fonctionnement.

Ce papier est organisé comme suit. La section 2 présente l'algorithme Q-Learning. La section 3 décrit l'architecture d'accélération matérielle adaptée à Q-Learning. La section 4 présente et compare les résultats d'implémentation. La section 5 présente les performances d'implémentation de l'architecture Q-Learning proposée dans le cas du jeu de labyrinthe. La section 6 donne une conclusion et des perspectives.

2 Technique de Q-Learning

Q-Learning est un algorithme RL qui apprend une fonction de valeur notée Q [13]. Cette fonction estime la valeur d'action-état $Q(s, a)$, qui est une somme de récompenses. Dans un environnement discret de N états et Z actions possibles, les valeurs action-état $Q(s, a)$ sont stockées dans une matrice de taille $N \times Z$. Lorsque cette fonction de valeurs Q actions-états est connue ou apprise par l'agent, la stratégie optimale peut être construite en sélectionnant l'action « a » qui maximise la valeur $Q(s, a)$ quand l'agent se trouve dans l'état « s ». La figure 1 montre le pseudo code utilisé pour exécuter l'algorithme Q-Learning.

Algorithme Q-learning

1. Paramètres de l'algorithme : taux d'apprentissage $0 < \alpha \leq 1$, $0 \leq \gamma \leq 1$, $\varepsilon > 0$
2. Initialiser à zéro $Q(s, a)$ pour tout $s \in S, a \in A(s)$.
3. Boucle for pour chaque épisode :
4. Initialiser S_t à la valeur de l'état initial de l'application
5. Boucle for pour chaque pas de l'épisode
6. Choisi a_t à l'état S_t en utilisant la stratégie $\varepsilon - greedy\ de\ Q$
7. Effectue l'action a_t, observer r_t, S_{t+1}
8. $Q_{new}(S_t, a_t) = Q_{old}(S_t, a_t) + \alpha(r_t + \gamma \max Q(S_{t+1}, A) - Q_{old}(S_t, a_t)$
9. $S_t \leftarrow S_{t+1}$
10. Répète jusqu'à S_t est l'état final de l'épisode.

Figure 1. Pseudo-code de l'algorithme Q-Learning.

L'algorithme Q-Learning initialise les paramètres (taux d'apprentissage α, facteur d'actualisation γ et stratégie ε) et les valeurs action-état $Q(s, a)$ pour tous les états (S_t) et les actions possibles (a_t) effectuées permettant une récompense immédiate r_t. Ensuite, l'agent effectue une action a_t en utilisant la stratégie $\varepsilon - greedy$

(Stratégie d'exploration-exploitation de l'environnement) permettant de mettre à jour la valeur $Q(S_t, a_t)$. Ce processus est répété jusqu'à ce que l'agent arrive à l'étatfinal de l'épisode.

3 Architecture de l'accélérateur Q-Learning

La figure 2 présente l'architecture proposée de l'accélérateur Q-Learning basée sur la parallélisation des nombres d'actions Z. Cette architecture permet la gestion des communications et du stockage des données de la Q-Matrice en utilisant des mémoires LUTRAM conçues en mode écriture synchrone/lecture asynchrone. Ainsi, la Matrice-Q est stockée dans Z LUTRAMs. Nous avons un bloc LUTRAM par action. Chaque RAM contient une colonne entière de la Matrice-Q. Le nombre d'emplacements mémoire correspond aux nombres d'états N. L'adresse de lecture correspond à l'état S_{t+1}, et l'adresse d'écriture à l'état actuel S_t. Les signaux d'activations des LUTRAMs sont générés par un bloc *décodeur* en fonction de l'entrée a_t afin de sélectionner la valeur $Q(S_t, a_t)$ à mettre à jour. Les sorties des LUTRAMs correspondent aux lignes de la Matrice-Q $Q(S_{t+1}, A)$, où A est l'ensemble des actions possibles. $Q(S_t, a_t)$ est obtenu en retardant les sorties des blocs mémoires, et en sélectionnant la valeur de la sortie LUTRAM correspondante par un multiplexage commandé par l'entrée a_t. Le bloc *Max* (basé sur un arbre de comparateurs binaires [15]) prend en entrée $Q(S_{t+1}, A)$ afin de déterminer le $\max_a Q(S_{t+1}, A)$. Le bloc *Q-Updater* implémente l'équation de Q-Learning (1) selon la technique du point fixe [13] pour générer la valeur action-état $Q_{new}(S_t, a_t)$ qui sera stockée dans l'emplacement correspondant de la LUTRAM.

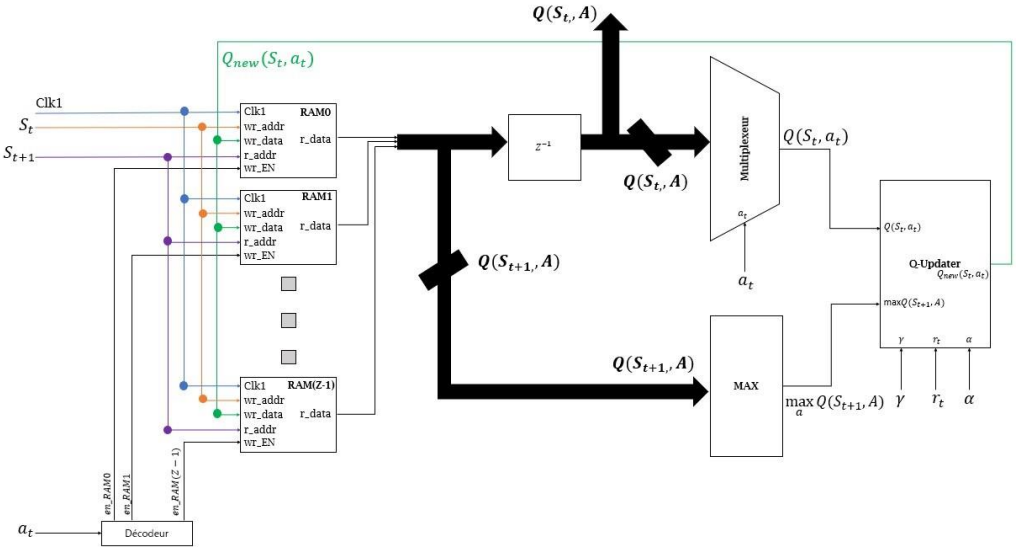

Figure 2. Architecture de l'accélérateur Q-Learning.

$$Q_{new}(S_t, a_t) = Q_{old}(S_t, a_t) + \alpha(r_t + \gamma \max_a Q(S_{t+1}, A) - Q_{old}(S_t, a_t)) \quad (1)$$

Afin de réduire la complexité de calcul et les ressources matérielles, la technique de calcul du point fixe est implémentée par des multiplicateurs approximatifs basés sur des registres à décalage [16]. L'utilisation des multiplicateurs approximatifs permet d'éviter l'utilisation de blocs DSP, et donc de minimiser à la fois les ressources logiques et la consommation de puissance dynamique.

4 Résultats d'implémentation et comparaison

Les résultats d'implémentation de l'architecture Q-Learning en termes des nombres de LUT, LUTRAM (blocs mémoires), registres (FF) et blocs DSP sont décrits dans le tableau 1 pour la technologie Xilinx FPGA Ultra Scale + ZCU104, en considérant une taille des valeurs de la Matrice-Q de 16 bits, un nombre d'actions Z = 4, et les nombres d'états N = 8, 16, 32, 64, 128 et 256.

Tableau 1. Résultats d'implémentations et comparaison.

	N	LUT	LUT RAM	FF	DSP	CLK (MHz)	PWR (mW)
Arch. Proposée	8	223	40	64	0	161	35
	16	215	40	64	0	167	36
	32	208	40	64	0	166	36
	64	239	80	64	0	168	36
	128	387	160	64	0	166	45
	256	565	320	64	0	166	46
Réf. [12]	8	179	64	250	3	152	58
	16	178	64	252	3	152	58
	32	180	64	254	3	149	60
	64	204	96	256	3	153	57
	128	333	160	258	3	158	80
	256	513	320	272	3	156	88

Comparativement aux travaux antérieurs [12], l'architecture accélérateur Q-Learning proposée assure une fréquence maximale supérieure (minimum 161 MHz) et ne dépendant pas du nombre d'états N. Bien que l'architecture proposée nécessite un surcout de 19 % de LUT, elle permet une diminution de 16 % à 37 % de LUTRAM pour N = 8, 16, 32 et 64, et respectivement une diminution de 74 % à 76 % de registres (FF). Le nombre de LUTRAMs dépend de la taille $N \times Z$ de la Matrice-Q. Le nombre de registres est fonction de la taille des valeurs de la Matrice-Q et du nombre d'actions Z (nombre FF = 16 * 4). La figure 3 montre que l'architecture proposée permet une réduction minimum de la consommation de puissance de 39 %.

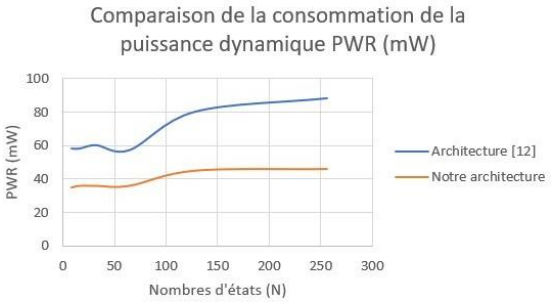

Figure 3. Comparaison de la consommation de puissance dynamique PWR (mW).

5 Implémentation de l'accélérateur Q-learning dans un environnement applicatif.

Afin de comparer les performances de l'architecture proposée, nous l'avons intégrée dans un environnement correspondant au jeu du Labyrinthe avec un scénario dans lequel un robot ou agent se déplace dans une arène en cherchant à atteindre par analyse une région précise. La figure 4 présente l'architecture adaptée à l'agent Q-Learning dans le cas d'un environnement 2x3.

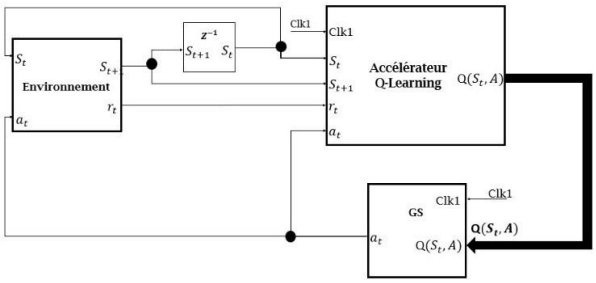

Figure 4. Application à un environnement de l'architecture Q-Learning.

La figure 5 représente l'environnement correspondant à une grille 2*3 avec S1 pour état initiale et S6 pour état final. Dans cet environnement, l'agent dispose de quatre actions possibles (Monter = 0, Descendre = 1, Gauche = 2, Droite = 3). L'agent reçoit une récompense de -500 s'il touche les bordures extérieures, sinon 0. Par contre, l'agent reçoit une récompense de 100 quand il arrive à l'état final. L'agent reçoit l'état S_{t+1} et la récompense r_t du bloc *environnement*. L'état S_t est obtenu en retardant par un registre l'état S_{t+1}. L'action a_t est générée par le bloc *Générateur de Stratégie* (GS) en fonction des valeurs Q-Matrice stockées dans l'architecture proposée *accélérateur Q-Learning*. Le bloc GS intègre une stratégie de type $\varepsilon - greedy$ à base d'un générateur aléatoire pour assurer à l'agent l'exploration de l'environnement avec une probabilité de ε, et d'exploiter sa connaissance avec une probabilité de $1 - \varepsilon$ [11].

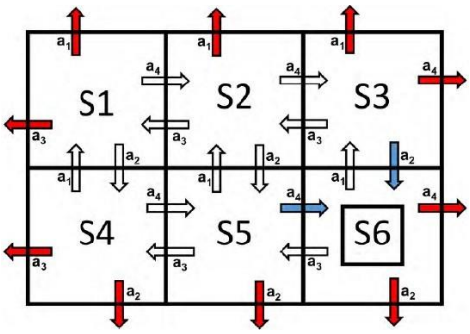

Figure 5. Application à un environnement 2 * 3 [11].

Pour une comparaison aux travaux antérieurs [11-12], les résultats d'implémentations sont donnés dans le tableau 2 en considérant la technologie Virtex-6 FPGAML605 et une taille des valeurs de la Matrice-Q égale à16 bits.

Tableau 2 Résultats implémentations pour un environnement applicatif d'une grille 2* 3.

	N	DSP	REG	LUT	CLK (MHz)
Architecture proposée	8	/	102	202	74
	16		103	202	73
	32		104	204	76
	64		105	234	71
	128		106	388	73
	256		107	568	68
Réf. [12]	8	3	186	148	72
	16		188	142	74
	32		190	147	72
	64		192	191	74
	128		194	304	71
	256		196	512	72
Réf. [11]	6	34	548	1734	24
	12	58	1029	3387	22
	20	90	1670	5594	20
	30	130	2470	8872	20
	56	234	4552	15526	15
	132	370	10533	70311	13

Comparativement à [11], l'architecture proposée permet une réduction de 81% du nombre de registres (FF) et de 88 % du nombre de LUT avec une fréquence de fonctionnement trois fois plus rapide. Par rapport à [12], l'architecture proposée n'utilisant pas de blocs DSP nécessite un surcoût de 26 % du nombre de LUT, mais une réduction de 45% du nombre de registres (FF).

6 Conclusion

Cet article propose une architecture d'accélération matérielle optimisée adaptée à l'algorithme Q-

Learning. L'architecture proposée repose sur des blocs mémoires en mode FIFO avec écriture synchrone/ lecture asynchrone. L'architecture conçue est adaptative et peut être déployée pour divers environnements avec différents nombres d'états et d'actions possibles. Les résultats d'implémentation sur un FPGA Ultra Scale+ ZCU 104 pour le cas d'une taille de données de 16 bits de la Matrice-Q, d'un nombre d'actions Z = 4 et différents nombres d'états de l'environnement (N) montrent une fréquence maximale supérieure à 161MHz et une consommation maximale de la puissance dynamique de 46 mW. Comparativement aux architectures Q-Learning présentées dans la littérature, l'architecture proposée présente un très bon compromis performance, ressources logiques et consommation de puissance. En effet, l'utilisation de multiplicateurs approximatifs au lieu de blocs DSPs et une gestion des blocs de mémorisation des valeurs de la matrice Q en configuration écriture synchrone/ lecture asynchrone permettent une réduction de la consommation d'énergie et des ressources logiques plus adaptées pour les systèmes embarqués. En perspective, une évolution de l'architecture proposée est envisagée pour l'intégration de l'algorithme double Q-learning.

Références

[1] B. Mahesh, *Machine Learning Algorithms -A Review*. 2019. doi: 10.21275/ART20203995.

[2] R. S. Sutton et A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.

[3] M. Matta *et al.*, « A Reinforcement Learning-Based QAM/PSK Symbol Synchronizer », *IEEE Access*, vol. 7, p. 124147-124157, 2019, doi: 10.1109/ACCESS.2019.2938390.

[4] A. He *et al.*, « A Survey of Artificial Intelligence for Cognitive Radios », *IEEE Trans. Veh. Technol.*, vol. 59, nº 4, p. 1578-1592, mai 2010, doi: 10.1109/TVT.2010.2043968.

[5] C. Wei, Z. Zhang, W. Qiao, et L. Qu, « Reinforcement-Learning-Based Intelligent Maximum Power Point Tracking Control for WindEnergy Conversion Systems », *IEEE Trans. Ind. Electron.*, vol. 62, nº 10, p. 6360-6370, oct. 2015,doi: 10.1109/TIE.2015.2420792.

[6] S. Levine, C. Finn, T. Darrell, et P. Abbeel, « End- to-End Training of Deep Visuomotor Policies ».

[7] A. Konar, I. Goswami Chakraborty, S. J. Singh, L.C. Jain, et A. K. Nagar, « A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot », *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 43, nº 5, p. 1141-1153, sept. 2013, doi: 10.1109/TSMCA.2012.2227719.

[8] J.-L. Lin, K.-S. Hwang, W.-C. Jiang, et Y.-J. Chen, « Gait Balance and Acceleration of a Biped Robot Based on Q-Learning », *IEEE Access*, vol. 4, p.2439-2449, 2016, doi: 10.1109/ACCESS.2016.2570255.

[9] Y. Deng, F. Bao, Y. Kong, Z. Ren, et Q. Dai, « Deep Direct Reinforcement Learning for Financial Signal Representation and Trading », *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, nº 3, p. 653-664, mars 2017, doi: 10.1109/TNNLS.2016.2522401.

[10] J. Zhu, Y. Song, D. Jiang, et H. Song, « A New Deep-Q-Learning-Based Transmission Scheduling Mechanism for the Cognitive Internet of Things »,*IEEE Internet Things J.*, vol. 5, nº 4, p. 2375-2385, août 2018, doi: 10.1109/JIOT.2017.2759728.

[11] L. M. D. Da Silva, M. F. Torquato, et M. A. C. Fernandes, « Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA », *IEEE Access*, vol. 7, p. 2782-2798, 2019, doi: 10.1109/ACCESS.2018.2885950.

[12] S. Spanò *et al.*, « An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm », *IEEE Access*, vol. 7, p. 186340-186351, 2019, doi: 10.1109/ACCESS.2019.2961174.

[13] C. J. C. H. Watkins et P. Dayan, « Q-learning », *Mach. Learn.*, vol. 8, nº 3, p. 279-292, mai 1992, doi: 10.1007/BF00992698.

[14] A. K. Panda, P. Rajput, et B. Shukla, « FPGA Implementation of 8, 16 and 32 Bit LFSR with Maximum Length Feedback Polynomial Using VHDL », in *2012 International Conference on Communication Systems and Network Technologies*, mai 2012, p. 769-773. doi: 10.1109/CSNT.2012.168.

[15] B. Yuce, H. F. Ugurdag, S. Gören, et G. Dündar, « Fast and Efficient Circuit Topologies forFinding the Maximum of n k-Bit Numbers », *IEEE Trans. Comput.*, vol. 63, nº 8, p. 1868-1881, août 2014, doi: 10.1109/TC.2014.2315634.

[16] M. R. Pillmeier, M. J. Schulte, et E. G. W. Iii, « Design alternatives for barrel shifters », in *Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, SPIE, déc. 2002, p. 436-447. doi: 10.1117/12.452034.