Implémentations sur MPSoC FPGA d'un algorithme d'affectation pour le suivi d'objets

Denis SHEMONAEV[1,2] Bertrand LE GAL[1] Christophe JEGO[1] Anthony BESSEAU[2]

[1]Laboratoire IMS, UMR 5218, 351 Cours de la libération, 33405 Talence cedex, France

[2]EMG2, 15 Avenue de Norvège, 91140 Villebon-sur-Yvette, France

Résumé – L'étape d'affectation est une étape clé pour le suivi multi-objets. Afin de garantir une exécution temps réel sur des cibles embarquées, il est important de choisir un algorithme efficace et en adéquation avec l'architecture cible. Dans cet article, nous présentons une étude comparative de l'implémentation d'un algorithme d'affectation par enchères pour des problèmes asymétriques [2] sur une cible de type SOM (System On Module) intégrant un processeur ARM et un circuit FPGA. Cette implémentation se distingue de l'état de l'art [6, 11] par le contexte de suivi multi-objets sur cible embarquée. En effet le contexte implique des problèmes d'affectation relativement limités ainsi que par le choix d'un algorithme plus performant. Les résultats obtenus montrent que notre implémentation logicielle est, dans la plupart des cas, adéquate pour valider le critère temps réel.

Abstract – An assignment algorithm is a key step for multi-object tracking. In order to guarantee real-time execution on embedded targets, it is important to choose an efficient algorithm that matches with the target architecture. In this paper, we present a comparative study of the implementation of an auction-based assignment algorithm for asymmetric problems [2] on a System On Module (SOM) target integrating an ARM processor and an FPGA circuit. This implementation differs with previous works [6, 11] by the context of multi-object tracking on an embedded target, which implies relatively limited assignment problems as well as by the choice of a more efficient algorithm. The results obtained show that our software implementation is, in most cases, sufficient to validate the real-time criterion.

1 Introduction

Le suivi d'objets ou de personnes est une problématique actuelle dans de nombreuses applications. Les avancées sur le sujet sont intrinsèquement liées aux progrès de l'intelligence artificielle, en particulier la détection. Il existe plusieurs applications directes de cette technologie, notamment le contrôle qualité dans une chaîne de production ou la détection de comportements suspects dans les lieux publics. Pour ces deux applications, la fonction de suivi d'objets ou de personnes est aussi essentielle que la détection à proprement parler. Du suivi d'objets émerge une problématique essentielle, qui est l'affectation de nouvelles détections aux objets déjà suivis.

Le problème d'affectation a été formulé sous la forme d'un algorithme qui vise à affecter un ensemble d'agents à des tâches afin de maximiser le gain global. Parmi les algorithmes les plus utilisés on retrouve l'algorithme Hongrois [8], l'algorithme LAPJV [7] ainsi que l'algorithme par enchères [1]. Le choix de l'algorithme à employer peut-être orienté par les contraintes d'intégration propres au système à concevoir. En effet, les applications citées précédemment peuvent nécessiter une proximité entre le système et le capteur, et cela, pour des raisons de sécurité ou des contraintes d'intégration. Actuellement, les circuits FPGA de type Zynq offrent un compromis intéressant entre puissance de calcul, capacité d'intégration et consommation énergétique. Dans ce cadre, l'algorithme opérant par enchères [1] possède une prédisposition à la parallélisation et peut donc pleinement être exploité par le circuit FPGA. Afin de se conformer à la formulation de l'algorithme par enchères, les tâches seront désormais appelées objets.

Dans cet article, nous allons présenter une étude comparative de l'implémentation de l'algorithme par enchères sur circuit FPGA et sur cible CPU embarqué pour résoudre le problème d'affectation dans le suivi d'objets. Afin de prendre en compte l'ensemble des problèmes existants, l'algorithme retenu est une variante de l'algorithme par enchères pour les problèmes asymétriques [2], que nous avons adapté pour le cas où le nombre d'agents est supérieur au nombre de tâches.

L'algorithme et nos implémentations logicielles et matérielles ont été validés et évalués sur carte à l'aide d'un jeu de tests généré à partir de l'ensemble de données MOT15 [9]. Les différents sujets ont été extraits des vidéos à l'aide d'un détecteur de la famille YOLOv4 [4]. Les résultats obtenus sont comparés aux résultats d'une version modifiée de l'algorithme Hongrois [5], employé dans l'algorithme SORT [3].

Cet article est organisé de la manière suivante. La section 2 se focalise sur la présentation du contexte, notamment une discussion sur les algorithmes d'affectation existants dans la littérature, puis la présentation de l'algorithme par enchères. La section 3 aborde l'implémentation de l'algorithme sur un circuit FPGA. Puis la section 4 présente les expérimentations réalisées sur CPU ainsi que sur FPGA. Enfin, la section 5 permet de conclure sur l'ensemble de ces travaux et de dresser diverses perspectives.

2 Contexte de l'étude

La partie principale d'un algorithme de suivi d'objets est l'affectation de nouvelles détections aux objets déjà existants. L'algorithme de suivi SORT, s'intéresse à l'aire de l'intersection de deux boites englobantes et leur union afin de déterminer leur compatibilité. Pour ce faire, nous avons utilisé la métrique *IOU* pour l'affectation :

$$IOU = \frac{\text{intersection}}{\text{union}}$$

Cette métrique permet de quantifier la proximité de deux boites englobantes indépendamment du rapport hauteur/largeur. À chaque nouvelle image traitée, une matrice est construite avec les valeurs d'*IOU* calculées entre les nouvelles détections et les objets suivis, l'utilisation d'un algorithme d'affectation s'impose. Par défaut, SORT utilise une version modifiée de l'algorithme Hongrois. Il est à souligner que le passage de l'algorithme Hongrois à l'algorithme par enchères pour l'affectation dans l'algorithme SORT n'a pas d'impact sur les performances sur la base de données MOT15.

2.1 Les avantages de l'algorithme par enchères

Les algorithmes communément utilisés pour l'affectation sont l'algorithme Hongrois et l'algorithme LAPJV [7, 8]. Cependant, leurs formulations sont difficilement parallélisables et leurs complexités calculatoires sont polynomiales. Nous pouvons également noter que les performances temporelles de leurs implémentations logicielles opérant sur des matrices éparses sont souvent en dessous des performances de l'algorithme par enchères [7]. Or, dans un contexte de suivi d'objets, les matrices obtenues sont majoritairement éparses. Le manque de parallélisme explique pourquoi, à notre connaissance, il n'existe pas dans la littérature d'implémentation matérielle des algorithmes Hongrois et LAPJV. Par ces observations, notre choix s'est tourné vers l'algorithme par enchères.

Des travaux sur l'implémentation d'un algorithme d'affectation par enchères sur circuit FPGA existent. Zhu et Zhang [11] ont proposé une plateforme d'accélération matérielle pour l'algorithme par enchères permettant d'obtenir une amélioration des performances d'un facteur 10 par rapport à une implémentation CPU pour des problèmes de grande taille (≥ 500). Leur architecture matérielle se base sur un algorithme par enchères exécutant une unique passe *forward*, car ils considéraient un problème d'affectation avec m agents et n objets ($m \leq n$). Il est important de noter que ce n'est pas toujours le cas dans un contexte de suivi d'objets où il est courant d'avoir $m > n$. Récemment, dans [6] une architecture matérielle pour la résolution de problèmes d'affectation peu denses a été détaillée, dans un contexte de suivi d'objets utilisant un filtre PMBM. Une représentation éparse ainsi que la gestion de la spéculation permet d'obtenir un gain de 50 au niveau des performances en comparaison avec [11]. Là encore, l'implémentation est également basée sur une version *forward* de l'algorithme.

2.2 Présentation de l'algorithme par enchères

Considérons l'ensemble $\mathbf{A}$, l'ensemble des coûts des objets j pour les agents i et l'ensemble $\mathbf{R}$ l'ensemble des solutions, contenant l'affectation des objets j aux agents i.

Pour un problème d'affectation asymétrique avec m agents et n objets, l'objectif est d'assigner un agent i à un objet j, ayant une valeur d'assignation $a_{ij} \in \mathbf{A}$, de sorte à maximiser le bénéfice total. Chaque objet j possède un prix p_j et chaque agent i a un bénéfice π_i. L'exécution de l'algorithme permet d'obtenir un vecteur résultat $r \in \mathbf{R}$, qui à un agent i associe un objet j tel que :

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} \cdot r_i \qquad \text{est maximale.} \tag{1}$$

Afin de garantir une affectation optimale à $m\epsilon$ près, d'après [2], les conditions suivantes doivent être satisfaites :

$$\pi_i + p_j \geq a_{ij}, \qquad \forall (i,j) \in \mathbf{A}. \tag{2}$$

$$\pi_i + p_j = a_{ij}, \qquad \forall (i,j) \in \mathbf{R} \tag{3}$$

$$p_j \leq \min_{k \text{ objets assignés dans } \mathbf{R}} p_k, \quad \forall j \text{ non assignés dans } \mathbf{R} \tag{4}$$

Afin de satisfaire la condition 4, un scalaire λ est introduit de sorte que :

$$p_j \geq \lambda, \quad \forall j \text{ objets assignés dans } \mathbf{R} \tag{5}$$

L'algorithme se déroule en deux temps : (1) la phase de mise de la part des agents (*forward*) et (2) la phase de mise de la part des objets (*backward*). Elles permettent de converger vers un compromis, afin que chaque agent soit satisfait de son affectation à une valeur ϵ près. La première phase *forward*, décrite dans l'algorithme 1, permet d'affecter un ensemble d'agents non assignés I^u à un ensemble d'objets J. Si lors de cette étape un objet est déjà assigné à un autre agent, alors celui-ci rejoint l'ensemble I^u. La seconde phase *backward*, décrite dans l'algorithme 2, permet d'associer les objets non associés à des agents lors de la phase *forward* à l'aide d'un fonctionnement symétrique à celle-ci. Ces étapes sont réitérées jusqu'à ce que tous les agents soient associés à un objet, dans le cas où $m \leq n$, ou que tous les objets soient associés à un agent, dans le cas où $m > n$. Enfin, une dernière étape *backward* est exécutée afin de valider les hypothèses initiales.

Algorithme 1 : Étape dite *forward*

Entrées : Matrice de bénéfices $\mathbf{A}$, agents i, objets j, prix p, profits π, seuil λ
Sorties : affectation r, prix p, profits π

```
pour chaque agent non assigné i^u faire
    benefices = A(i^u) - p
    val1 = max1(benefices)
    val2 = max2(benefices)
    col1 = indice_colonne(val1)
    mise = A(i^u, col1) − val2 + ϵ
    p[col1] = max(λ, mise)
    π[i^u] = val2
    si λ ≤ mise alors
        si r[i^u] est non vide alors
            col_precedente = r[i^u]
            supprimer_affectation(col_precedente)
        fin
        r[i^u] = col1
    fin
fin
```

3 Implémentations temps réel

Le contexte de cette étude étant lié à l'intégration de fonctionnalités de suivi d'objets ou de personnes sur des systèmes embarqués proches du capteur, nous avons étudié et évalué l'implémentation de l'algorithme par enchères sur une plateforme SOM (System On Module) KRIA développée par Xilinx. La carte KV260 intègre un FPGA de type Zynq UltraScale+ alliant un processeur Arm® Cortex®-A53 à quatre coeurs physiques (PS) et une partie logique (PL).

Algorithme 2 : Étape dite *backward*

Entrées : Matrice de bénéfices $\mathbf{A}$, agents i, objets j, prix p, profits π, seuil λ
Sorties : affectation r, prix p, profits π

1 **pour chaque** *objet non assigné* j^u **faire**
2 benefices = $\mathbf{A}(j^u)$ - π
3 val1 = max1(benefices)
4 val2 = max2(benefices)
5 ligne1 = indice_ligne(val1)
6 **si** $val1 \geq \lambda + \epsilon$ **alors**
7 p[ligne1] = $\max\{\lambda, val2 - \epsilon\}$
8 **si** *il existe* $r[i]$ *tel que* $i = ligne1$ **alors**
9 ligne_precedente = $r[i]$
10 supprimer_affectation(ligne_precedente)
11 **fin**
12 r[ligne1] = j^u
13 **sinon**
14 p[ligne1] = $val1 - \epsilon$
15 **si** *il existe* l *avec* $p_l < \lambda$ *tel que* $l > n - m$ **alors**
16 $\lambda = \min_{l \geq n-m} \{p_l\}$
17 **fin**
18 **fin**
19 **fin**

Afin de comparer les performances au niveau des temps de résolution de l'algorithme d'affectation, deux implémentations distinctes ont été réalisées, une logicielle (C/C++) et une matérielle.

3.1 Implémentation logicielle

Nous avons d'abord procédé à une implémentation purement logicielle du processus d'affectation, dérivée d'une description assez linéaire de la formulation algorithmique présentée précédemment. Une phase d'optimisation visant tout d'abord à augmenter la localité des données et à minimiser la complexité calculatoire a été effectuée. Puis, le cœur ARM de la plateforme KRIA étant superscalaire et afin d'améliorer le parallélisme d'instructions (ILP) nous avons utilisé les *templates* disponibles en C++ afin de générer des codes logiciels spécifiques aux données à traiter. Cela permet ainsi au compilateur logiciel par exemple de dérouler des boucles et supprimer des instructions de contrôle.

3.2 Implémentation matérielle

Nous avons ensuite réalisé une implémentation matérielle dont l'architecture est illustrée par la Figure 1. Elle est composée d'un ensemble d'éléments de calcul implémentant les lignes 2 à 6 des algorithmes 1 et 2, configurables avant synthèse. Pour alimenter ces $\mathbf{Q}$ éléments de calcul durant l'étape *forward*, la mémoire contenant les prix est partitionnée. Pour l'étape *backward* traitant les données en colonne, nous avons ajouté une mémoire spécifique qui contient la transposée de la matrice d'affectations. Afin d'implémenter cette architecture, nous avons exploité les capacités offertes par l'outil de synthèse de haut niveau Vitis HLS, afin d'obtenir de la généricité et de la flexibilité au niveau de la configuration des accélérateurs. Dans le but d'obtenir des performances élevées, une attention particulière a été portée sur la façon de décrire l'algorithme ainsi que sur l'insertion de directives de synthèse pertinentes. Le format de données utilisé par défaut est l'entier sur 32 bits. Le transfert de données est géré par un bus AXI.

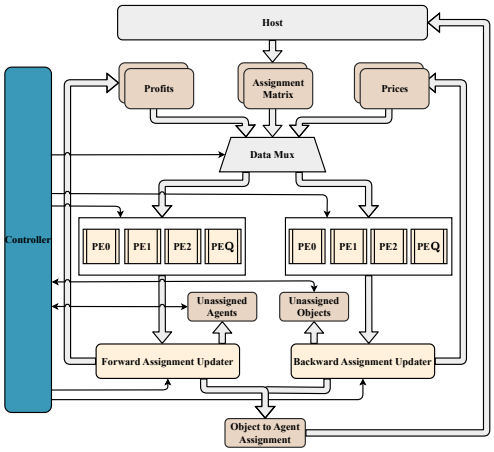

FIGURE 1 : Architecture matérielle de l'accélérateur

4 Évaluation des performances

Afin de comparer les performances des deux implémentations et estimer l'impact sur une plateforme embarquée, trois types d'expériences ont été menées. Les deux premières ont été effectuées sur la plateforme KRIA KV260. La version logicielle du processus d'affectation a été exécutée sur le processeur ARM Cortex-A53 dont la fréquence de fonctionnement est fixée à 2.30 GHz. La version matérielle est déployée dans la PL du FPGA et inter-connectée avec la mémoire à l'aide d'un DMA AXI. Le processeur supervise l'activation de l'accélérateur et de la gestion du DMA. Dans cette configuration, les temps de transferts ont été retranchés pour homogénéiser la comparaison. La troisième expérimentation a été effectuée sur un ordinateur portable incluant un processeur Intel Core i7-10750H fonctionnant à 2.60GHz afin d'estimer l'écart de performances avec la plateforme embarquée. Les jeux de test utilisés dans un premier temps sont composés de 1000 matrices denses générées aléatoirement. La densité des matrices correspond à un cas extrême dans notre contexte applicatif.

4.1 Données aléatoires

Les temps d'exécution moyens de ces différentes configurations sont données dans la Figure 3. Il est important de noter que le temps d'exécution est fortement dépendant des données. Nous observons à partir de ces résultats que, pour un contexte de suivi vidéo, le temps de traitement est bien inférieur à 30 FPS. L'exécution sur CPU x86 est plus rapide que l'exécution sur un cœur CPU ARM ou au sein du circuit FPGA. Par ailleurs, pour de matrices de taille 8x8 et 32x32,

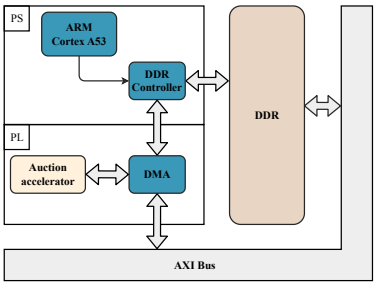

FIGURE 2 : Architecture de test mise en œuvre

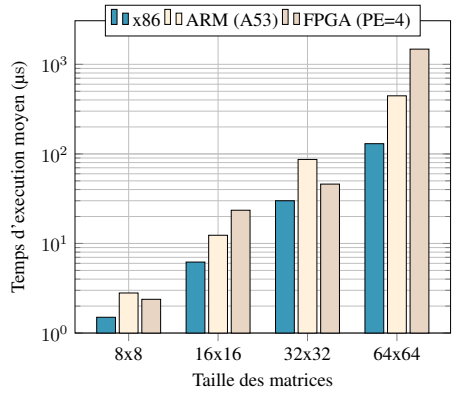

FIGURE 3 : Temps d'exécution des différentes implémentations sur circuit FPGA et CPU.

l'exécution sur CPU ARM est plus rapide que l'implémentation sur circuit FPGA. Ce constat s'inverse pour les autres expérimentations. En analysant le code assembleur x86 / ARM, nous avons constaté que notre implémentation logicielle exploite extrêmement bien les jeux d'instructions de ces deux CPU, particulièrement les instructions `cmov`, évitant l'exécution de sauts conditionnels. Par ailleurs, l'utilisation des *templates* qui permettent une exécution plus efficace, additionné à la différence entre les fréquences de fonctionnement et la faible parallélisation actuelle (PE=4) de l'architecture matérielle, explique pourquoi notre accélérateur sur circuit FPGA est généralement moins performant que les implémentations logicielles sur CPU x86 et CPU ARM.

4.2 Données MOT15

Afin de mieux considérer le contexte du suivi d'objets, nous avons également effectué des expérimentations sur des matrices plus réalistes obtenues à partir de détections de personnes sur la base de données MOT15. Le détecteur d'objets choisi est YOLOv4-tiny [4], entrainé sur la base de données COCO [10]. Après étude des matrices IOU obtenues avec SORT, il s'avère que les matrices sont principalement carrées avec comme taille maximale 12 x 12. Contrairement aux expérimentations précédentes, ces matrices sont éparses (moy. $\approx$ 70%). Les résultats obtenus sur des matrices 12x12 et 8x8 sont présentés en Figure 4. Pour ces données réelles, on constate que l'architecture matérielle que nous avons réalisée sur circuit FPGA surclasse les implémentations logicielles d'un facteur 3.

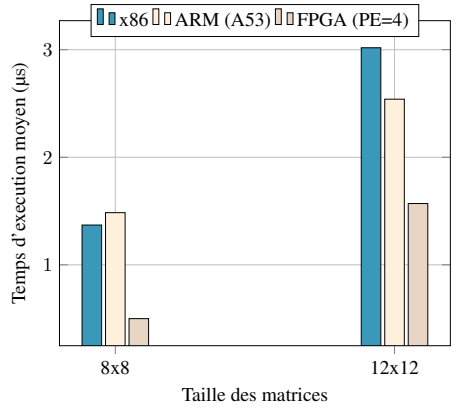

FIGURE 4 : Temps d'exécution sur des données MOT15

5 Conclusion

Dans cet article, nous avons présenté des implémentations sur MPSoC FPGA d'un algorithme d'affectation par enchères pour un contexte de suivi vidéo. Nos implémentations sont plus génériques que les implémentations présentées dans [6, 11]. En effet, elles contiennent une phase *backward* permettant de résoudre des matrices d'affectation asymétriques. Les résultats obtenus sur des matrices carrées denses générées aléatoirement montrent que pour des matrices de petite taille et un nombre de PE égal à 4, le choix d'une implémentation sur circuit FPGA n'est pas toujours pertinente. Cependant, pour des matrices réelles extraites de MOT15, l'implémentation sur circuit FPGA permet un gain de performances d'un facteur 3. Une écriture méticuleuse de l'algorithme initial en C ainsi que les performances des compilateurs actuels permettent une bonne adéquation algorithme/architecture sur CPU x86 et ARM. Nos futurs travaux viseront à incorporer cet algorithme d'affectation par enchère pour optimiser un algorithme de détection et de suivi d'objets plus global, traitant plusieurs flux vidéos simultanément sur le MPSoC de la plateforme NATVision.

Références

[1] D. P. BERTSEKAS : The auction algorithm : A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(1), 1988.

[2] D. P. BERTSEKAS *et al.* : A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications*, 1(3), 1992.

[3] A. BEWLEY *et al.* : Simple Online and Realtime Tracking. *In IEEE International Conference on Image Processing (ICIP)*, 2016.

[4] A. BOCHKOVSKIY *et al.* : YOLOv4 : Optimal Speed and Accuracy of Object Detection, 2020.

[5] D. F. CROUSE : On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4), 2016.

[6] E. R. JELLUM *et al.* : Solving Sparse Assignment Problems on FPGAs. *ACM Transactions on Architecture and Code Optimization*, 2022.

[7] R. JONKER et A. VOLGENANT : A shortest augmenting path algorithm for dense and sparse linear assignment problems. 1987.

[8] H. W. KUHN : The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2), 1955.

[9] L. LEAL-TAIXÉ *et al.* : MOTChallenge 2015 : Towards a benchmark for multi-target tracking. *arXiv :1504.01942 [cs]*, 2015.

[10] T. Y. LIN *et al.* : Microsoft COCO : Common Objects in Context, 2015.

[11] P. ZHU *et al.* : An FPGA-based acceleration platform for auction algorithm. *In IEEE International Symposium on Circuits and Systems*, 2012.