Accélération matérielle pour le traitement de trafic sur FPGA

Tristan Groleat, Sandrine Vaton, Matthieu Arzel

Accélération matérielle pour le traitement de trafic sur FPGA

Tristan Groléat[1] and Sandrine Vaton[1] and Matthieu Arzel[1]

[1] *Télécom Bretagne - Technopôle Brest-Iroise - CS 83818 - 29238 Brest Cedex 3 - France*

Les débits augmentant en cœur de réseau comme dans l'accès, les algorithmes de traitement de trafic doivent fonctionner à des vitesses toujours plus élevées. Or il est souvent impossible de traiter des débits de l'ordre de 10 à 100 Gb/s sur des ordinateurs conventionnels. Les FPGAs sont des puces programmables à très bas niveau, particulièrement adaptées pour atteindre ce genre de vitesses. Au travers de deux applications courantes dans les réseaux : la classification et la génération de trafic, cet article montre la méthode de mise en place et l'intérêt de l'accélération matérielle pour le traitement de trafic.

Keywords: Traitement de trafic, FPGA, accélération matérielle, classification, génération de trafic

1 Introduction

Le traitement de trafic est une tâche de base dans les réseaux. Lors du développement de nouveaux algorithmes dans ce domaine, il faut s'assurer qu'ils pourront fonctionner à des débits adaptés au type de trafic qu'ils ont à traiter. Actuellement, des débits de l'ordre de 10 à 100 Gb/s sont fréquents. Or la section 2 va montrer qu'une implémentation logicielle est très limitée en termes de débit et de fiabilité. C'est pourquoi cet article s'intéresse aux FPGAs, des puces programmables à très bas niveau, qui permettent d'apporter une accélération matérielle aux algorithmes de traitement de trafic. C'est une solution efficace et raisonnablement peu coûteuse pour prouver le passage à l'échelle de nouveaux algorithmes proposés.

L'utilité des FPGAs sera expliquée à travers deux exemples. Le premier (section 3) sera la classification de trafic basée sur un algorithme d'apprentissage supervisé : les *Support Vector Machines* (SVM) [CV95]. L'objectif est d'attribuer aux paquets reçus une classe (pair à pair, navigation web, *streaming* vidéo…) à 10 Gb/s. Comme la classification nécessite de nombreux calculs, l'architecture massivement parallèle des FPGAs sera mise en avant.

Le second exemple est la génération de trafic (section 4). Un générateur a été développé capable d'envoyer des paquets avec des propriétés spécifiées à 10 Gb/s. Son code est ouvert et disponible en ligne [tel13]. Ce générateur est très flexible et précis. La flexibilité est en partie permise par la possibilité de reconfigurer les FPGAs, et la précision par celle de contrôler leur fonctionnement au cycle d'horloge (5,3 ns) près.

2 L'accélération matérielle

La grande flexibilité du développement logiciel et la disponibilité d'ordinateurs puissants font penser que tout peut être fait en logiciel. Cependant l'architecture des ordinateurs, avec de nombreuses couches logicielles, ne permet pas un contrôle complet du trafic, en particulier des débits et délais inter-paquets. Pour remédier à ce problème, N. Bonelli et al. [BDPGP12] ont modifié le fonctionnement du noyau de leur système et développé une nouvelle socket qui permet un meilleur contrôle du trafic. Leur générateur de trafic atteint ainsi les 10 Gb/s sur un ordinateur puissant, mais pas s'il génère des petits paquets (64 octets : la taille minimum pour Ethernet).

Ces limites montrent l'intérêt de l'accélération matérielle dés que l'on cherche à aller à plus de 10 Gb/s ou à faire des traitement complexes. Les FPGAs sont une solution possible. Il existe principalement 2 cartes spécialisées dans le traitement de trafic : la NetFPGA 10G [net12], avec 4 interfaces à 10 Gb/s et la

Combov2 d'INVEA TECH [IT12], avec 2 à 4 interfaces à 10 Gb/s. La programmation d'un FPGA se fait au niveau des portes logiques, au cycle d'horloge près. Ceci permet des traitements massivement parallèles et un contrôle très précis des délais.

3 La classification de trafic

3.1 Algorithme

L'algorithme de classification implémenté est léger en comparaison de la *Deep Packet Inspection*, mais nécessite quand même de faire des calculs à haut débit pour traiter 10 Gb/s. Il a été montré que l'utilisation des SVMs est efficace [WZA06]. La classification se fait par flot TCP ou UDP, les 3 propriétés utilisées étant les tailles des 3 premiers paquets non vides des flots [EGS09].

L'algorithme SVM fonctionne en deux phases : l'apprentissage puis la classification. Lors de l'apprentissage, une trace étiquetée est utilisée pour calculer un modèle SVM. Ce modèle contient en particulier des vecteurs support, utilisés ensuite pour classifier les vecteurs inconnus. Dans cet article, seule l'implémentation de la phase de classification est étudiée. L'apprentissage est fait au préalable.

Nous avons développé un prototype logiciel grâce à la librairie ouverte LibSVM. L'objectif était de vérifier la précision de la classification et la vitesse de l'algorithme. La trace étiquetée utilisée pour les tests a été fournie par un laboratoire d'Ericsson. Elle contient 39 056 flots dans 8 classes différentes, et il faudrait traiter 27 805 flots par seconde pour pouvoir la classifier à 10 Gb/s.

Le prototype donne environ 98 % de flots placés dans la bonne classe, quand l'apprentissage est fait sur $1/5$ de la trace et la classification sur les $4/5$ restants. Au niveau de la vitesse, sur un processeur 2.66 GHz 6-core Xeon X5650 et 12 Go de RAM DDR3, en utilisant LibSVM avec OpenMP pour le support du multi-processus, on obtient un traitement de 4 655 flots par seconde, soit 17 % des pré-requis. Ceci ne prend même pas en compte la nécessaire étape de traitement des paquets avant la classification.

3.2 Implémentation matérielle

Pour classifier un flot, l'algorithme SVM consiste à calculer une fonction "noyau" entre le vecteur à classifier x et chaque vecteur support x_i. La fonction "noyau" la plus utilisée est appelée RBF (*Radial Basis Function*) : $K(x_i,x) = exp\left(-\gamma \parallel x_i - x \parallel^2\right)$.

Une fois cette fonction calculée pour chaque vecteur support, un mécanisme de sommes puis de vote permet de choisir la classe du flot.

L'intérêt de l'implémentation matérielle est de permettre une parallélisation massive des calculs. Ici :

- le calcul de $\parallel x_i - x \parallel^2$ nécessite 3 soustractions (les vecteurs ont 3 dimensions), 3 calculs de carrés puis une somme. Ceci peut se faire en parallèle : les 3 soustractions suivies des carrés en même temps, et enfin la somme.
- Le calcul de l'exponentielle doit se faire autant de fois qu'il y a de vecteurs support, soit 3 160 fois ici. La surface du FPGA ne permet pas de faire les 3 160 calculs en même temps. On mettra cependant le maximum d'unités de calcul en parallèle, divisant d'autant le temps mis pour chaque classification.
- Les autres calculs (sommes, mécanisme de vote...) dépendent les uns des autres et ne peuvent pas être faits en parallèle. Un *pipeline* sera cependant créé : chaque unité de calcul travaille en même temps, mais sur des données différentes. Par exemple dans le calcul de la norme : il faut calculer une soustraction, puis un carré, et enfin une somme. En même temps que la somme est effectuée sur le vecteur de support 1, le carré est effectué sur le 2 et la soustraction sur le 3. Ainsi un nouveau calcul peut être commencé à chaque coup d'horloge.

De cette manière, des centaines d'opérations sont exécutées à chaque coup d'horloge. Cependant les opérations mathématiques gérées par le FPGA sont plus basiques que sur les processeurs actuels. En particulier les calculs sur des flottants sont très coûteux. C'est pourquoi toutes les valeurs sont passées en virgule fixe. La dynamique de chaque valeur est donc très limitée, et il faut choisir sur combien de bits chaque paramètre est codé pour ne pas trop diminuer la précision des résultats. Une simulation développée en logiciel permet de vérifier ici qu'après optimisation, la précision se maintient à environ 97 %.

Les multiplications sont aussi très coûteuses en matériel. C'est pourquoi les calculs de carrés et d'exponentielles sont effectués grâce à des mémoires en lecture seule (ROMs) : l'adresse de lecture est l'argument

de la fonction et la valeur lue est son résultat. Par exemple, pour le carré, on stocke à l'adresse 3 la valeur 9. Pour que ces mémoires soient les plus petites possibles, les propriétés mathématiques des fonctions (symétrie du carré et $e^{a+b} = e^a \times e^b$) sont mises à profit.

Des alternatives aux ROMs existent pour calculer des fonctions particulières. Par exemple pour l'exponentielle, nous pourrions utiliser l'algorithme CORDIC qui calcule une approximation du résultat par des additions et des décalages de bits.

3.3 Résultats

Grâce à la parallélisation, si n est le nombre de vecteurs support et p le nombre d'unités de calcul que l'on arrive à placer sur le FPGA, la classification d'un flot nécessite $23+5+n/p+2\log_2(p)$ cycles d'horloge. 23 est la longueur du *pipeline* et 5 le nombre d'opérations fixes dues au vote final. $2\log_2(p)$ opérations servent à réunir les résultats des différentes unités de calcul.

Les synthèses (opération permettant de traduire le code en circuit à programmer sur le FPGA) faites pour la carte NetFPGA 10G indiquent pour 8 unités de calcul une occupation de la surface du FPGA de 70 % et une fréquence maximale de fonctionnement de 165 MHz, ce qui donne plus de 290 000 flots par seconde. C'est plus de 60 fois plus qu'en logiciel.

Des résultats détaillés ont été publiés [GAV12]. Ces résultats ne sont que pour l'unité de classification. Le module de traitement des paquets prend de la place aussi, mais ce n'est pas le sujet de cet article. Le classificateur complet gère quand même des débits bien supérieurs aux 10 Gb/s requis.

4 La génération de trafic

La section précédente montre l'intérêt du FPGA pour faire des calculs massivement parallèles et traiter du trafic en temps réel. Cette section s'intéresse à la gestion des paquets eux-même.

4.1 Fonctionnement

L'objectif ici est de développer un générateur de trafic au code source ouvert avec plusieurs caractéristiques : fonctionnement à au moins 10 Gb/s, caractéristiques de trafic (débit, en-têtes, type de paquet...) paramétrables, extensible à tous types de trafic, et avec un coût abordable.

Des générateurs de trafic sur FPGA existent déjà [SSG11], mais à plus bas débit et moins configurables. Le coût abordable est permis par la carte NetFPGA 10G, dont le prix est réduit pour les académiques.

Pour pouvoir atteindre des débits élevés, il faut limiter au maximum la communication avec l'ordinateur, trop lent. C'est pourtant lui qui peut apporter la flexibilité demandée. Le générateur ne communiquera donc avec l'ordinateur qu'au début, lors d'une phase de configuration.

La configuration se fera par le concept de flux. Un flux est un ensemble de paquets qui partagent le même squelette, et des modificateurs qui permettent de changer certaines choses dans ce squelette : champs aléatoires, champs qui s'incrémentent, générateurs de données de longueur variable...Un flux devra pouvoir atteindre 10 Gb/s, même en générant des petits paquets.

4.2 Implémentation matérielle

L'implémentation matérielle du générateur se fait autour d'un bus de données de 64 bits de large, ce qui donne un débit de 12 Gb/s pour la carte Combo. Ce bus intègre un mécanisme permettant à chaque module d'arrêter les données si nécessaire.

Chaque flux configuré est généré par un bloc appelé générateur de flux, présent sur le FPGA. Les générateurs de flux fonctionnent en parallèle et les paquets générés sont mélangés dans le trafic final. Il est impossible de configurer plus de flux qu'il n'y a de générateurs.

Chaque flux est composé d'une série de modificateurs les uns derrière les autres. Pour supporter les 10 Gb/s, les modificateurs forment un *pipeline* : ils fonctionnent tous en même temps, mais sur des données différentes. Chaque modificateur a un rôle précis : l'un génère du trafic aléatoire dans certains champs, l'autre module le débit en arrêtant le trafic pendant un délai donné, le suivant recalcule le champ FCS Ethernet qui valide la validité des données...

Lors de la configuration, les modificateurs sont activés et configurés, ou désactivés. Ceci permet de paramétrer les caractéristiques du trafic. Si les modificateurs existants ne permettent pas de générer le trafic voulu, il est facile d'en développer un nouveau, ce qui rend le générateur extensible.

4.3 Résultats

Le générateur est encore en développement, son code est ouvert et disponible en ligne [tel13]. Son architecture permet d'atteindre sans difficultés les 10 Gb/s. Il est même possible de faire sortir les générateurs de flux sur des interfaces différentes pour atteindre 20 Gb/s.

Même si le générateur est un peu moins souple qu'un générateur logiciel, il est quand même configurable et extensible. Et la tenue des débits ne pose pas de problèmes particuliers, contrairement à une implémentation logicielle. Les limites sont dues à la taille du FPGA, qui limite le nombre des flux simultanés, et au nombre de modificateurs déjà disponibles.

5 Conclusion

Les FPGAs permettent de prototyper de manière assez flexible et à un coût abordable des algorithmes de traitement de trafic fonctionnant à plusieurs dizaines de Gb/s. INVEA-TECH travaille même actuellement sur une nouvelle carte supportant 100 Gb/s.

Nous avons montré à travers deux exemples que l'accélération matérielle a un intérêt, que ce soit pour faire des opérations mathématiques pour analyser du trafic reçu en temps réel, ou pour travailler directement au niveau des paquets et générer sans goulot d'étranglement du trafic à très haut débit.

Bien que le développement sur FPGA soit moins flexible que le développement logiciel, il est possible de faire communiquer le FPGA avec un ordinateur pour rendre certains blocs configurables.

L'étape supplémentaire d'adaptation de l'algorithme au matériel, décrite pour la classification SVM, permet de réfléchir aux opérations élémentaires requises par l'algorithme, et est en réalité utile pour tout travail d'optimisation des performances. Pour aller plus loin, le code développé pour le FPGA peut aussi servir de base pour concevoir des puces plus spécialisées comme les ASIC (circuit intégré spécifique à une application) pour augmenter les débits et la miniaturisation.

Références

[BDPGP12] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi. Flexible high performance traffic generation on commodity multi–core platforms. *Traffic Monitoring and Analysis*, pages 157–170, 2012.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[EGS09] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support Vector Machines for TCP traffic classification. *Computer Networks*, 53(14) :2476–2490, 2009.

[GAV12] Tristan Groleat, Matthieu Arzel, and Sandrine Vaton. Hardware acceleration of SVM-based traffic classification on FPGA. In *IWCMC*, pages 443–449. IEEE, 2012.

[IT12] Invea-Tech. COMBO cards. `http://www.liberouter.org/hardware.php?flag=2`, May 2012. [Online].

[net12] NetFPGA 10G. `http://netfpga.org/10G_specs.html`, 2012. [Online].

[SSG11] M. Sanlı, E.G. Schmidt, and H.C. Güran. FPGEN : A fast, scalable and programmable traffic generator for the performance evaluation of high-speed computer networks. *Performance Evaluation*, 68(12) :1276–1290, 2011.

[tel13] Open-source hardware traffic generator. `https://github.com/tristan-TB/hardware-traffic-generator`, 2013. [Online].

[WZA06] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 2006.