Prototypage de radio logicielle à base du language Julia

Corentin Lavaud, Robin Gerzaguet, Matthieu Gautier, Olivier Berder

Prototypage de radio logicielle à base du language Julia

Corentin Lavaud*, Robin Gerzaguet*, Matthieu Gautier* and Olivier Berder*
* Univ Rennes, CNRS, IRISA
prenom.nom@irisa.fr

Abstract

Nous proposons une nouvelle approche de prototpyage rapide et efficace à partir du langage Julia. Les radios logicielles sont des architectures radio-fréquences qui permettent de capturer et de traiter un signal electromagnétique. Du fait de leur flexibilité et leur capacité de reconfiguration, les radios logicielles sont des outils précieux largement déployés pour des contextes très divers. Puisque la majorité de la chaine de traitement se fait en software il convient de choisir un langage de programmation qui garantisse cette flexibilité. Les stratégies classiques s'appuient sur des langages bas-niveau (C/C++), pour garantir les performances d'exécution (au détriment de la simplicité de réalisation) ou des approches haut-niveau (Python) pour offrir une grande capacité d'abstraction (au détriment des performances). Dans cet article, nous introduisons une nouvelle méthodologie basée sur Julia qui adresse ce problème du "double langage". Nous proposons un ensemble d'outils pour piloter des radios logicielles et nous démontrons par l'intermédiaire d'un benchmark que les performances obtenues avec l'approche Julia sont très intéressantes.

1 Introduction

Le principe de radio logicielle a été introduit il y a trente ans par Mitola où le paradigme de la radio logicielle est proposé pour la première fois. Il s'agit de proposer des architectures de radios dont la partie analogique est réduite à son strict minimum. L'immense majorité des traitements se font en *software*, ce qui garantit une plus grande flexibilité et la capacité de reconfigurer le même terminal pour différentes applications. De ce paradigme émerge la nécessité de proposer une méthodologie de conception spécifique, et des architectures radio fréquence génériques capables d'adresser des bandes de fréquences multiples.

De fait, de nombreuses radios logicielles ont été proposées: basée sur des architectures génériques (*Generic purpose processor*) ou spécialisés (ASIC ou FPGA). Les radios logicielles modernes sont capables de réaliser en temps réel des taches complexes, dans un périmètre applicatif croissant (cybersécurité, systèmes de communication,...).

La manière dont on programme ces cibles logicielles reste toutefois libre: il est souvent nécessaire de combiner briques logicielles et co-processeurs hardware (pour les taches contraignantes en terme de débit//latence). Le langage utilisé pour programmer ces radios a donc un fort impact. D'un côté, il est tentant de maximiser les performances via des approches bas-niveau (C/C++) mais le temps de développement peut être rédhibitoire. A contrario, les langages haut-niveaux (Python//Matlab) seront plus adaptés au prototypage et à l'exploitation logicielle mais les performances ne seront pas au rendez-vous. Ainsi, déployer une application sur radio logicielle se fait souvent en deux temps: une exploration via langage haut-niveau et une ré-écriture des briques logicielles en langage bas-niveau pour atteindre un degré acceptable de performance.

Le langage Julia recemment proposé [1] se positionne par rapport à cette problématique du double langage: La syntaxe est très proche de celle d'un langage haut-niveau (approche de *scripting*) mais le code est compilé à la volée (via LLVM) permettant d'avoir d'excellentes performances d'exécution. Dans cet article, nous présentons un écosystème logiciel (e.g. *AbstractSDRs.jl*) [2] permettant de s'interfacer avec différentes radio logicielles en langage Julia et on compare les performances obtenus pour une application ciblée rédigée en C++ en Python et en Julia. On montre que les performances obtenues en Julia sont tout à fait similaires à celle du C++ tout en garantissant une syntaxe plus concise, flexible et facilement adaptable et améliorable (vectorisation, ...).

2 Ecosystème développé

Le système proposé se base sur une interface de programmation unifiée capable de s'interfacer avec différentes architectures de radio logicielles. Chaque architecture est gérée par un sous-module spécifique et un module de gestion maître est dédié à la répartition dynamique (*Multiple Dispatch*). Ce type de paquet encapsulé garantit à la fois une grande flexibilité (chaque sous-module peut être modifié indépendamment pourvu qu'il respecte les lignes directrices de l'interface maître) et une facilité d'extension (ajout d'autres sous-module pour élargir la gamme des radios supportées). Le paquet

proposé est *open source* et la version courante se situe sous Github [3]. Via l'approche proposée, il est possible de configurer et d'échanger des données avec des USRP (*Universal Software Radio Peripheral*) d'Ettus Research, de l'ADALM-Pluto d'Analog Device et des RTL-SDR. Il est également possible de spécifier une interface Ethernet générique ce qui permet de communiquer avec des radios logicielles basées sur des SoC (*System On Chip*) telle que la USRP E310 de Ettus Research (Julia étant deployable sur les processeurs l'ARM v7) ou de deployer des topologies réseaux en arbre.

Le système d'encapsulation permet de deployer une même application pour différentes radios logicielles. Ainsi, nous avons pu également proposer quelques couches applicatives *open source* qui se basent sur le paquet *AbstractSDRs.jl* comme un analyseur de spectre configurable (*AbstractSDRsSpectrum.jl*) ou un récepteur radio FM (avec *AbstractSDRsFMReceiver.jl*).

3 Benchmark et performances

3.1 Propriétés et itérations des codes

Nous calculons un débit après traitement par l'intermédiaire d'une simulation Monte Carlo avec 20 itérations indépendantes de 10 secondes. Le traitement choisit est le calcul d'une moyenne glissante d'un module carré d'une transformée de Fourier rapide dont les propriétés sont décrites dans [2].

On considère deux versions: une initiale L0 et une optimisée Lx. L0 correspond au prototypage (qui reste complexe pour le C++). Pour l'optimisation Lx, nous avons sequentiellement i) supprimé les vérifications des conditions de bornage des boucles ii) déroulé les boucles puis choisit des conteneurs bas-niveaux pour les buffers, iii) nous avons enfin vectorisé les instructions processeurs avec l'utilisation massive d'instructions SSE et d'AVX. Pour le langage python, nous avons opté pour une compilation à la volée avec Numba-JIT.

3.2 Benchmark avec une radio logicielle

Nous représentons sur la Figure 1 le débit de sortie de l'algorithme réalisé dans les différents langages par rapport au débit d'entrée configuré sur la radio logicielle X310. Les performances les plus mauvaises sont obtenues avec le langage Python en version L0 comme attendu. Julia sans aucune optimisation offre de meilleures performances que Python (avec une syntaxe très proche) mais reste loin des performances du C++. Au niveau L0 aucun langage n'atteint la borne des 200 MS/S. Avec le meilleur niveau d'optimisation, le C++ et Julia atteignent la borne désirée, et ce n'est pas le cas du Python (malgré la compilation JIT). A noter que le travail d'optimisation est bien plus simple à réaliser en Julia via notamment l'utilisation puissante des macros. Ceci permet d'accélérer la transition entre la phase de prototypage rapide et l'éventuelle version plus efficace du code.

4 Conclusion

Dans cet article, nous avons présenté une nouvelle méthodologie pour adresser efficacement des radios logicielles dans le langage Julia. Ce langage présente des propriétés intéressantes pour résoudre la problématique du double langage (ré-écrire du code dans un langage bas-niveau pour améliorer les performances). Nous proposons un écosystème logiciel *open source*; *AbstractSDRs.jl*; qui permet de de s'interfacer et de piloter différentes radios logicielles directement en Julia. Avec l'approche proposée, il est donc possible de prototyper avec des radios logicielles tout en garantissant d'excellentes performances, notamment en débit. Pour illustrer les avantages de notre approche, un benchmark compare les performances de l'approche Julia avec celle d'un langages haut-niveau (Python) et bas-niveau (C++). On montre que Julia permet des performances très similaires à celle du C++ tout en permettant une grande flexibilité (syntaxe proche du Python) et une forte capacité d'optimisation (via les macros).

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[2] C. Lavaud, R. Gerzaguet, M. Gautier, and O. Berder, "AbstractSDRs: Bring down the two-language barrier with Julia Language for efficient SDR prototyping," in *IEEE Embedded Systems Letters (ESL)*, 2021.

[3] Julia Telecom, "AbstractSDRs - Common API for Software Defined Radio ," 2020, https://github.com/JuliaTelecom/AbstractSDRs.jl.

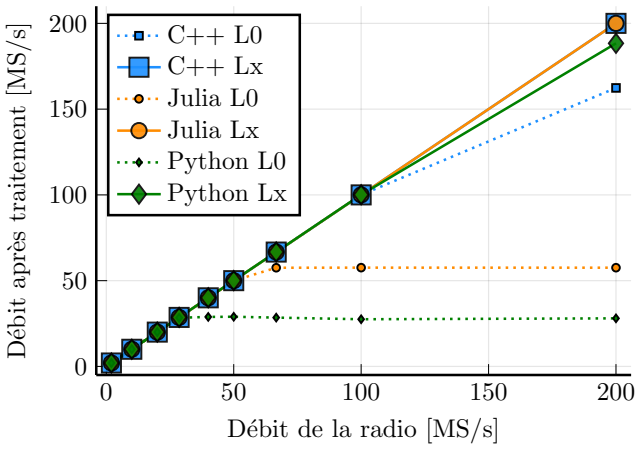

Figure 1. Benchmark pour le code initial et le code optimisé avec une X310.