GNU Radio et CorteXlab: Tour d'horizon

Cyrille Morin

Maracas team, CITI Lab, Inria

28/11/23

GNU Radio

Composition

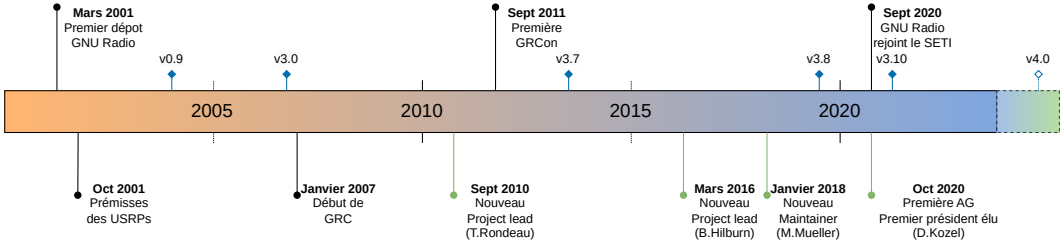
Éléments principaux

- Coeur (C++) d'interfaçage entre blocs de calcul (Buffers, scheduler)
- Ordonnancement laissé à l'OS: Un thread par bloc
- Blocs de calcul en C++ ou Python. Possibilité d'en écrire de nouveaux, personnalisés
- Glue en Python (récemment aussi C++) pour créer et controller des chaines de blocs (flowgraphs)
- Interface graphique (GRC) pour créer et paramétrer des flowgraphs
- Grande bibliothèque de blocs standards ($\backsim 500$)

Philosophie

- Traitement hautement modulaire
- Possibilité de concevoir une chaine de traitement sans écrire de code
- Ajout facile de nouveaux modules/blocs
- Fonctionnement sur une large gamme d'équipements
- Priorise le fait d'être versatile

Historique


Mars 2001
Premier dépot
GNU Radio
v0.9
v3.0
Sept 2011
Première
GRCon
v3.7
v3.8
Sept 2020
GNU Radio
rejoint le SETI
v3.10
v4.0
2005
2010
2015
2020
Oct 2001
Prémisses
des USRPs
Janvier 2007
Début de
GRC
Sept 2010
Nouveau
Project lead
(T.Rondeau)
Mars 2016
Nouveau
Project lead
(B.Hilburn)
Janvier 2018
Nouveau
Maintainer
(M.Mueller)
Oct 2020
Première AG
Premier président élu
(D.Kozel)

Écosystème

Développement open-source

- Code sur GitHub (https://github.com/gnuradio)
- GNU GPL v3.0
- Développement principalement volontaire, quelques financements ponctuels
- Documentation sur wiki qui s'étoffe

Communauté

- Populations mondiale, mais pôles de participation aux USA et Europe (UK, FR, DE)
- Nombreux champs d'application: télécom, radar, astronomie, ...
- Mailing list et Chat (Matrix) très actifs (Plusieurs mails, dizaines de messages par jour)
- Conférence annuelle (GRCon) aux USA, et GREuDays (depuis quelques années) en Europe
- Présence régulière d'industriels de SDR (NI/Ettus, Analog devices, Epiq...)

Intégration matérielle

Blocs dédiés SDR

- Fournis par Ettus pour les USRPs
- Fournis par ADI pour les PlutoSDR (récemment ajouté à la bibliothèque standard)
- Blocs/Modules plus génériques: OsmoSDR, Soapy
- Possibilité pour d'autre vendeurs de créer leur module

Traitement des données

- Volk: bibliothèque de noyaux optimisés pour le calcul vectoriel sur différentes architectures CPU
- Blocs de traitement sur FPGA avec RFNoC (Ettus, USRPs)
- Quelques modules externes utilisent GPU (fosphor,...)
- Usage d'accélérateurs encore assez rare (mais très demandé)

Évolutions à venir

Passage à une version 4.0

- 3.x depuis 2006
- Grande dette technologique (architecture, scheduler, dépendances, ...)
- 5 dernières années, poussée pour alléger cette dette (3.8: python2/xml, 3.9: swig, 3.10: boost)
- Une évolution profonde est nécessaire

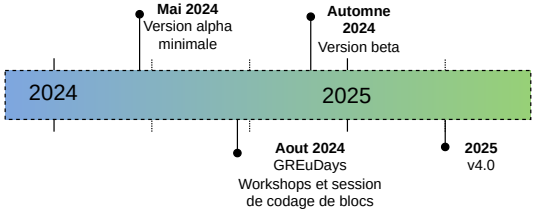
Développement convergent

- Depuis 2022, un processus de design, prototypage par des membres coeur de la communauté
- Arrivée du GSI/FAIR (DE) avec des ressources humaines importantes et spécialisées
- Motive et permet une réécriture complète de la base de code

Évolutions à venir

Éléments principaux

- Codage de blocs simplifié (moins de boilerplate)
- Utilisation de concepts C++ modernes (C++23, voire 26)
- Grande modularité des composants de base, Scheduler (N bloc/thread, voire mieux), Buffers, ...
- Usage d'instructions SIMD simplifié (voire automatisé) pour les développeurs
- Runtime distribuée (entre ordinateurs, accélérateurs, ...)
- Licence LGPL envisagée (Déjà fait pour Volk)



CorteXlab

CorteXlab

En bref

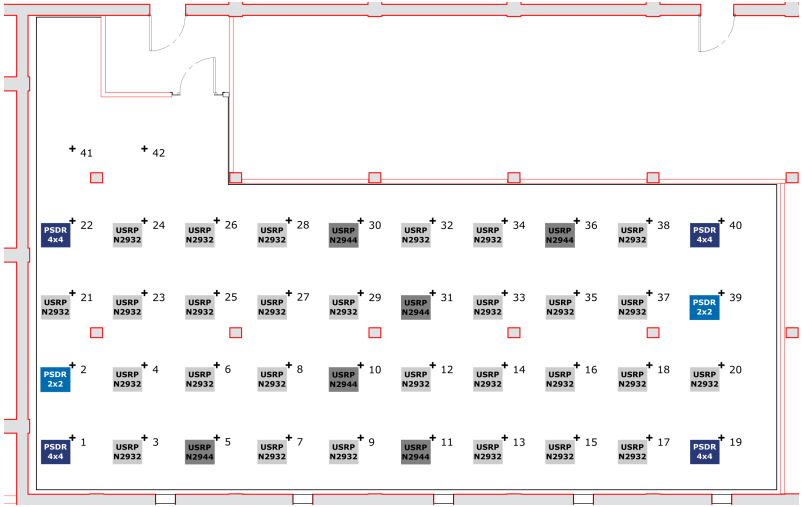
- CorteXlab: COgnitive Radio Testbed and EXperimentation LAB
- Lieu de recherche et test pour la couche physique
- Ouvert librement à tous: Académie, Industrie, ...
- Accès total à distance
- Usage similaire à un cluster de calcul
- GNU Radio principalement utilisé (mais pas indispensable)

Historique

- 2012: Montage du projet au sein de l'Équipex FIT (Future Internet of Things)
- 2013: Construction, création de l'infrastructure
- 2014: Mise en production

La salle

La salle


41
42
PSDR 4x4 22
USRP N2932 24
USRP N2932 26
USRP N2932 28
USRP N2944 30
USRP N2932 32
USRP N2932 34
USRP N2944 36
USRP N2932 38
PSDR 4x4 40
USRP N2932 21
USRP N2932 23
USRP N2932 25
USRP N2932 27
USRP N2932 29
USRP N2944 31
USRP N2932 33
USRP N2932 35
USRP N2932 37
PSDR 2x2 39
PSDR 2x2 2
USRP N2932 4
USRP N2932 6
USRP N2932 8
USRP N2944 10
USRP N2932 12
USRP N2932 14
USRP N2932 16
USRP N2932 18
USRP N2932 20
PSDR 4x4 1
USRP N2932 3
USRP N2944 5
USRP N2932 7
USRP N2932 9
USRP N2944 11
USRP N2932 13
USRP N2932 15
USRP N2932 17
PSDR 4x4 19

Éléments radio

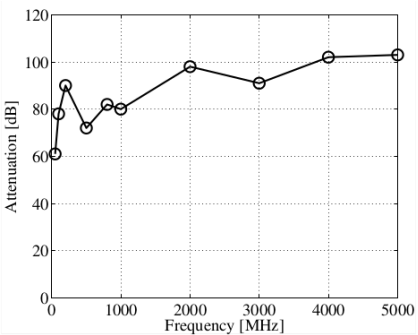


Figure: Atténuation par rapport à l'extérieur

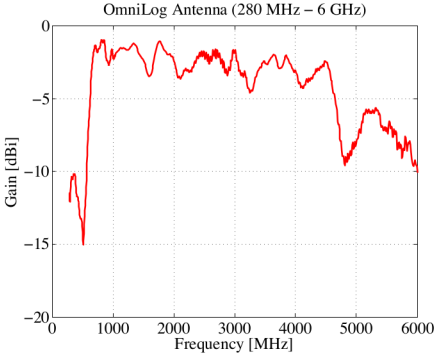


Figure: Antennes utilisées

USRP-2932

Features

- 400 MHz - 4.4 GHz
- BW: Up to 20 MHz at 16 bits
- BW: Up to 40 MHz at 8 bits
- OCXO crystal (25 ppb)
- Synchronisation externe d'horloge possible
- 30 exemplaires

USRP-2944

Features

- 10 MHz - 6 GHz
- BW: Up to 160 MHz at 14 bit $>$ 500 MHz
- BW: Up to 84 MHz at 14 bit $<$ 500 MHz
- 2 canaux full duplex
- TCXO crystal (2.5 ppm)
- Synchronisation externe d'horloge possible
- 6 exemplaires

Noeuds de calcul

Features

- Un PC dédié par SDR (Connexion directe Ethernet)
- Intel Core i7 - 4 Cores @ 2.3 GHz
- 8 GB Ram
- Fanless
- Réseau Ethernet 1Gb entre tous les PCs et la Gateway

GPU server

- 2x Intel Xeon 4114 @ 2.20GHz (40 threads)
- 2x GTX1080Ti GPU
- 64 GB Ram
- 2x 10 Gb Ethernet avec le reste de la plateforme

Usage

Connexion

- Création gratuite de compte en ligne
- Connexion au serveur d'interface via SSH
- Réservation de créneaux depuis une interface web dédiée
- Déploiement et récupération de résultats depuis son HomeDir sur le serveur d'interface

Scénario d'expérience décrit en yaml

- Durée
- Noeuds impliqués
- Image Docker à utiliser (hébergée en ligne)
- Commandes à lancer dans les containers

Tests DVB-S2 Aff3ct

Quelques adaptations nécessaires

- Vieux CPUs: besoin de compiler avec architecture SSE4.1 (pas d'AVX disponible)
- Limitation à 4 threads (et désactivation du thread pinning)
- Rajout d'une fonction rshiftr pour SSE4.1 (uniquepent présent pour AVX2)
- Radios USRP: remplacement de boost::thread par std::thread (probablement une différence de version de dépendances)
- Images docker précompilées disponibles (Pour juste Aff3ct, ou la chaine DVB-S2)

Signal to Noise Ratio			Bit Error Rate and Frame Error Rate					Thoughput & Time	
Sigma	Es/N0	Eb/N0	FRA	BE	FE	BER	Fer	Sim throu	ET
0.3607	5.85	3.40	100	19585	100	1.38e-02	1.00e+00	0.160	00h00'08
0.3565	5.95	3.50	122	16489	100	9.50e-03	8.20e-01	0.291	00h00'05
0.3524	6.05	3.60	196	13485	100	4.83e-03	5.10e-01	0.390	00h00'07
0.3484	6.15	3.70	535	12390	100	1.63e-03	1.87e-01	0.719	00h00'10
0.3444	6.25	3.80	4395	11015	100	1.76e-04	2.28e-02	1.492	00h00'41

Figure: Stats obtenues en simulation sur l'un des PCs de CorteXlab (n 36), avec les paramètres de référence de QPSK_8_9_freq_000_delay_40.txt

Tests DVB-S2 Aff3ct

Quelques adaptations nécessaires

- Vieux CPUs: besoin de compiler avec architecture SSE4.1 (pas d'AVX disponible)
- Limitation à 4 threads (et désactivation du thread pinning)
- Rajout d'une fonction rshiftr pour SSE4.1 (uniquepent présent pour AVX2)
- Radios USRP: remplacement de boost::thread par std::thread (probablement une différence de version de dépendances)
- Images docker précompilées disponibles (Pour juste Aff3ct, ou la chaine DVB-S2)

Signal to Noise Ratio		Bit Error Rate and Frame Error Rate					Thoughput & Time	
Es/N0	Eb/N0	FRA	BE	FE	BER	Fer	Sim throu	ET
11.43	8.99	24624	0	0	2.85e-09	4.06e-05	11.836	00h00'29

Figure: Stats obtenues en simulation entre deux noeuds voisins de CorteXlab (16=>14), avec les paramètres d'exemple et 20Msps

Évolutions prévues: PEPR Réseaux du Futur

Besoins

- Ordinateurs vieillissants: Plus de puissance de calcul
- SDRs: Plus de cartes modernes, avec plus de MIMO
- Flexibilité: Accéder à plusieurs radios à la fois
- Accélérateurs: Intéger d'autres équipements que CPU (GPU, DPU,...)
- Mesures: Accéder à de nouvelles métriques (ex: puissance consommée)
- Interconnexion: Permettre des expérimentations multi-site, avec Cloud (SLICES-FR)

Évolutions prévues

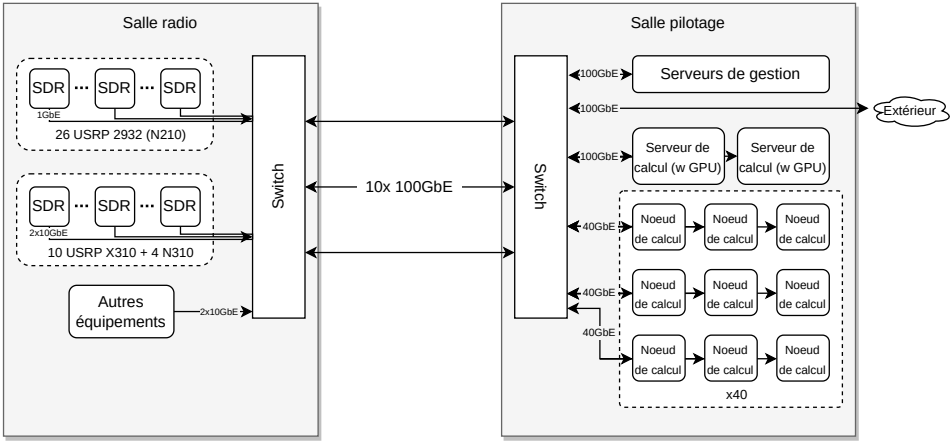


Figure: Architecture prévue

Fin

Merci de votre attention