Parallélisation automatique de chaînes de tâches pour la Radio Logicielle

Diane Orhan

Compas'2023 : Parallélisme / Architecture/ Système
LISTIC/USMB - Annecy France, 4-7 juillet 2023

Parallélisation automatique de chaînes de tâches pour la Radio Logicielle

Diane Orhan*

Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400 Talence, France
diane.orhan@inria.fr

Résumé
La Radio Logicielle désigne l'implémentation d'éléments de transmission radio sur des architectures programmables au lieu des traditionnels circuits intégrés dédiés. Elle a aussi l'avantage de faciliter le prototypage de nouveaux standards et de réduire le coût et le temps de la mise sur le marché. Les architectures programmables, notamment CPUs, doivent pouvoir atteindre les débits des architectures dédiées pour les standards actuels, en utilisant des techniques de parallélisation. Il est possible d'exploiter plusieurs niveaux de parallélisation comme l'approche SIMD des CPUs mais aussi le parallélisme à l'échelle du graphe de tâches d'une chaîne de traitements numériques. Pour ce dernier type de parallélisme, il est possible d'exploiter les propriétés des tâches de traitement. Ainsi, si une tâche est sans état interne, alors il est possible d'appliquer du parallélisme de tâches à travers leur réplication. Combiné avec le parallélisme de pipeline, il est possible d'imaginer des heuristiques permettant pour une chaîne donnée et un nombre de ressources limités d'atteindre le débit de traitement le plus élevé. Dans cet article, nous proposons un modèle définissant le problème pour une chaîne de communications lorsque ces deux types de parallélismes sont exploités, ainsi qu'une expérimentation sur une chaîne de Radio Logicielle existante.

1. Introduction

Les communications numériques sont traditionnellement implémentées sur du matériel dédié avec des objectifs de haut débit et de faible latence. Cependant, les standards de communications, comme le standard de réseau de téléphonie mobile 5G, évoluent et sont versatiles avec de nombreuses configurations possibles et des variations sur les spécifications de latence ou débit [14]. De plus, la conception de nouveaux circuits dédiés à l'ensemble des configurations est coûteuse et implique un développement important avant la mise sur le marché [26].
Afin de répondre à ces contraintes, il est nécessaire d'avoir des solutions matérielles flexibles qui peuvent être programmables. Le domaine de la Radio Logicielle répond à ce besoin-là [7] puisqu'elle consiste en l'implémentation des composants d'une chaîne de communications numériques de manière logicielle sur du matériel reconfigurable comme sur des DSP (*Digital Signal Processor*) [20], des FPGA (*Field Programmable Gate Array*) [23] ou encore des CPU (*Central Processing Unit*) [17, 21]. Les GPU ne sont pas aussi compétitifs du point de vue de leur latence résultant de la copie des données en entrée et sortie du GPU [17]. Nous nous concentrons sur

*. Le texte a été relu par Olivier Aumage, Denis Barthou, Christophe Jego, Laercio Lima-Pilla.

les CPUs dont les performances, en exploitant le parallélisme, sont similaires à celles de circuits dédiés [21].
Il est possible de tirer parti de l'aspect multicœur des CPUs en pipelinant [6] les opérations à effectuer. Pour ce faire, les différentes opérations sont réparties sur plusieurs cœurs de calcul tout en respectant les dépendances. Ainsi, plusieurs trames de données de communications numériques peuvent être traitées simultanément sur des opérations différentes. Cette approche permet d'augmenter le débit de traitement des trames de données même si elle a le défaut de détériorer la latence. Il est également possible d'exploiter un autre aspect d'une chaîne de communications numériques. En effet, les éléments d'une chaîne, ou tâches, possèdent ou non un état interne de par leur nature. Les tâches sans état interne ont la capacité de pouvoir être répliquées. La réplication permet de faire le même traitement sur plusieurs trames en parallèle, entraînant une augmentation de débit [8].
Optimiser le pipeline d'une chaîne de communications numériques sur une architecture multicœur revient à partitionner la chaîne en groupes de tâches consécutives (que l'on appelle étages), à répliquer ou non des étages (étages constitués uniquement de tâches réplicables donc sans état interne) puis à assigner les étages aux cœurs de calculs. Il existe aujourd'hui peu de solutions automatiques. Or cela implique du travail supplémentaire pour les concepteurs de chaîne de communications numériques. L'objectif est donc de maximiser le débit, et également de minimiser le nombre de ressources de manière automatique.
La Section 2 de cet article présente l'état de l'art associé au contexte de cette étude. Le problème est défini dans la Section 3 et la Section 4 décrit l'idée de l'algorithme et le cadre d'évaluation des heuristiques développées pour répondre à la problématique. Enfin, la Section 5 détaille l'avancée actuelle des travaux de recherche avant de conclure.

2. État de l'art

2.1. Partitionnement

Le problème traité s'apparente à un problème de partitionnement pour lequel il existe des algorithmes dans certain cas particulier.
Le *pipelined workflow scheduling* [8] désigne l'ordonnancement d'applications traitant des streams continus de données, ou trames dans le cadre de la Radio Logicielle, de même taille. C'est notamment le cas des chaînes de transmission et réception radio. Il est possible de distinguer plusieurs types de parallélismes dans ce type d'application : le **parallélisme de tâches**, c'est-à-dire l'exécution concurrentielle de tâches indépendantes pour la même trame. Le **parallélisme de pipeline** désigne le cas où deux tâches indépendantes sont exécutées simultanément pour des trames différentes. Le **parallélisme de réplication** a lieu lorsque deux instances d'une même tâche sans état interne traitent deux trames différentes. Enfin, le **parallélisme de données** est utilisé pour les tâches dont le traitement inhérent peut être parallélisé. Chacun de ces types de parallélismes a un impact spécifique sur les métriques de l'ordonnancement. Le parallélisme de réplication permet d'améliorer le débit comme le parallélisme de pipeline. Ce dernier implique néanmoins une augmentation de la latence ou temps de traitement totale pour une trame. Le parallélisme de tâche permet, quant à lui, de minimiser la latence.
L'algorithme de Nicol [27] décrit un algorithme optimal de partitionnement pour l'équilibrage de charge de chaîne de tâches. La chaîne peut être découpée en plusieurs étages de pipeline en fonction du nombre de ressources disponibles de manière à minimiser la latence (parcours de la chaîne pour une donnée) tout en limitant le nombre de ressources utilisées. Cet algorithme repose sur le fait de trouver la valeur optimale du temps d'exécution maximal d'un étage à l'aide d'une recherche dichotomique. Cet algorithme considère uniquement la technique de

parallélisme de pipeline.

2.2. Bibliothèques de Radio Logicielle

Il existe des suites logicielles dédiées à la conception et à la réalisation de systèmes de Radio Logicielle et au traitement du signal. Chacune possède sa propre façon de réaliser l'ordonnancement des tâches de traitement. Les langages présentés dans le paragraphe suivant s'orientent en particulier pour des applications de *streaming*, c'est-à-dire avec un flux continu de données, organisé en trames de données.

La plupart de ses langages se basent sur un modèle de calcul *synchronous dataflow* (SDF) [18]. Dans ce modèle, le graphe de tâches est constitué de blocs dit synchrones et sont caractérisés par le fait que le nombre de données produites et consommées est connu en avance. Ce modèle garantit de pouvoir ordonnancer statiquement, à la compilation, avec un nombre fini de ressources. Cependant, ce modèle ne considère pas la possibilité aux tâches d'avoir un état interne, c'est-à-dire de garder une mémoire des traitements passés. contrairement au modèle adopté dans ce papier.

GNU Radio [3] est la bibliothèque open-source la plus répandue [6]. Elle fournit à la fois un environnement de simulation pour le prototypage et des outils permettant l'implémentation d'une chaîne sur matériel générique en passant par une interface graphique facilitant le prototypage. Selon Bloessl, Müller et Hollick [10], le fonctionnement de l'ordonnancement dans GNU Radio est le *Thread Per Block scheduling*, chaque bloc, qui est assimilé à une tâche, est associé à un thread. L'ordonnancement en lui-même est laissé à l'ordonnanceur du système d'exploitation, à savoir Linux ou a d'autre runtime comme CEDR [22] qui permet d'envisager des architectures hétérogènes. Ce mode de fonctionnement a plusieurs inconvénients qui peuvent impacter les performances comme le fait de ne pas exploiter le cache [10]. Aussi, lorsque l'ordonnanceur doit gérer plusieurs threads par ressources, cela crée des *overheads* [24]. La prochaine version de GNU Radio 4.0 envisage d'autoriser d'avoir plus d'un seul bloc par threads [24]. L'algorithme proposé dans ce papier pourrait alors y être appliqué.

StreamIt est également une bibliothèque qui fournit un langage et un environnement de compilation pour le développement d'une chaîne de Radio Logicielle [28]. Contrairement à GNU Radio, StreamIt possède un propre algorithme d'ordonnancement, intégré dans son compilateur [19]. Le compilateur est capable de partitionner le graphe de tâches d'une chaîne de Radio Logicielle en fusionnant les blocs (parallélisme de pipeline), lorsqu'il a moins de ressources que de tâches, ou en décomposant les blocs (parallélisme de réplication) lorsqu'il y a plus de cœurs que de tâches. Son objectif est d'avoir autant de blocs que de ressources disponibles en utilisant un algorithme de *Mixed Integer Linear Programming* [16]. Tout comme avec GNU Radio, cette stratégie d'ordonnancement utilise toutes les ressources mises à disposition par l'architecture cible en essayant de répartir plus équitablement la charge. L'approche de l'algorithme proposé dans ce papier quant à lui propose de maximiser les performances tout en minimisant le nombre de ressources utilisées.

Array-OL est un autre langage spécifique au traitement du signal et notamment aux streaming vidéos, c'est-à-dire des structures de données complexes [18]. Tout comme StreamIt, l'ordonnancement est fait par le compilateur avec son propre algorithme [11]. Celui-ci concentre plutôt ses efforts sur le parallélisme de données contrairement aux approches décrites plus haut.

Enfin, **AFF3CT** est une bibliothèque, dans un premier temps, dédiée au codage canal [6, 1] et un aussi simulateur qui permet de modéliser une chaîne et/ou de faire du prototypage. L'utilisateur peut développer en C++ ou en python grâce un wrapper [5]. Il est possible d'exploiter du parallélisme dans le traitement d'une trame par une tâche grâce aux instructions vectorisées via la bibliothèque MIPP [12, 4]. Il est également possible de faire du parallélisme inter-tâches

en répliquant des tâches, celles qui n'ont pas d'état interne, et de pouvoir ainsi les exécuter simultanément sur deux trames différentes. Dans ce cas, deux instances d'une même tâche peuvent effectuer le même traitement de données différentes en parallèle. Alors que dans le cas du parallélisme de pipeline, chaque étage, composé de tâches différentes, fait le traitement de données différentes. Il revient à l'utilisateur de décider quelle tâche répliquer ou non, voire de dupliquer un étage entier de tâches clonables, mais aussi de choisir quelles tâches sont pipelinées. L'utilisateur peut également associer un thread particulier à un étage. Cela signifie que l'ordonnancement des tâches revient à l'utilisateur. L'algorithme décrit plus tard dans ce papier répond à ce problème en effectuant l'ordonnancement de manière automatique.

3. Définition du problème

3.1. Modèle

Il est possible de modéliser le problème de parallélisation automatique de tâches pour la Radio Logicielle à l'aide d'un ordonnancement de flux pipeliné en y rajoutant certaines contraintes [8]. Le flux de tâches considéré est une chaîne de N tâches $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$. Elles sont exécutées séquentiellement, c'est-à-dire τ_i est exécutée après τ_{i-1}. Une tâche τ_i a un temps d'exécution w_i, appelé poids. Les temps d'échanges des données et de synchronisation d'étage de pipeline sont négligés et n'apparaissent pas dans ce modèle. Il est possible de séparer les tâches en deux groupes $\mathcal{T}_F$ et $\mathcal{T}_L$ selon si elles possèdent ou non un état interne.
Le système est constitué de P processeurs homogènes, uniformes et interconnectés. L'objectif de performance principal est d'optimiser le débit inverse T. Le second objectif est de minimiser le nombre de processeurs assignés.
Une chaîne peut être découpée en sous-séquences définies par un couple (s_i, p_i) appelé étage où s_i est la i^e sous-séquence de la chaîne composée de n_i tâches et p_i le nombre de processeurs alloués à la i^e sous-séquence avec $\sum_{i=1}^{k} n_i = N$ et $\sum_{i=1}^{k} p_i \leq P$ où k est le nombre d'étages dans la chaîne. Un étage s_i est dit sans état interne si toutes les tâches qui le composent sont sans état interne. À l'inverse, s_i est dit avec état interne si au moins une des tâches qui le composent possède un état interne. Seul un étage sans état interne peut profiter de plusieurs processeurs via la réplication de tâches sur plusieurs processeurs.
Le temps d'exécution d'un étage est défini comme suit :

$$w(s_i, p_i) = \begin{cases} \sum_{\tau \in s_i} w_\tau & \text{si } s_i \cap \mathcal{T}_F \neq \emptyset,\ p_i \geq 1, \\ \frac{1}{p_i} \sum_{\tau \in s_i} w_\tau & \text{si } s_i \cap \mathcal{T}_F = \emptyset,\ p_i \geq 1, \\ \infty & \text{sinon} \end{cases} \tag{1}$$

Le débit inverse est quant à lui défini par le temps maximum d'exécution de tous les étages :

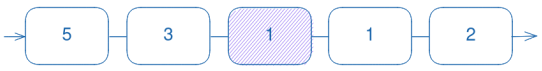
$$T = \max_{i=1,..,k} w(s_i, p_i) \tag{2}$$

L'objectif est donc de trouver un découpage en étages minimisant le débit inverse T. Si l'ensemble des tâches sont avec état interne, alors, cela revient au problème de partitionnement de chaîne [27] dont les solutions sont connues. À l'inverse, si l'ensemble des tâches sont sans état interne, la solution optimale est de répliquer l'ensemble de la chaîne sur les ressources disponibles [9].

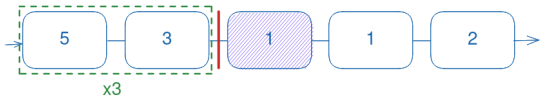
3.2. Exemples d'illustration

Soit la chaîne présentée dans la Figure 1a. Elle est composée de cinq tâches sur lesquelles sont inscrits leur temps d'exécution (5, 3, 1, 1, et 2), considéré comme entier. La tâche hachurée re-

présente une tâche avec état interne, les autres tâches sont sans état interne. Le débit inverse est minoré par le temps d'exécution de la tâche avec état interne la plus longue, c'est-à-dire $T \geq 1$ dans cet exemple. Il est possible de déterminer intuitivement quel sera le meilleur ordonnance-



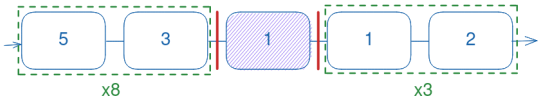
(a) Exemple de chaîne comprenant 4 tâches sans état interne et 1 tâche avec état interne en hachurée.



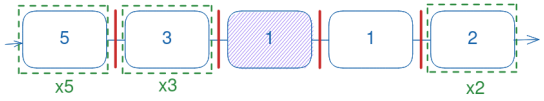
(b) Solution optimale pour 4 processeurs.

FIGURE 1 – Ordonnancement d'une chaîne sur 4 processeurs.

ment en fonction du nombre de processeurs P. L'opération de duplication d'étage est indiquée, comme sur la Figure 1b, par un cadre en pointillé autour de l'étage ou de la tâche concernée. Les différents étages de pipeline sont séparés par une barre verticale. La Figure 1b présente la configuration optimale pour $P = 4$ processeurs. Intuitivement, les deux tâches du début sont à regrouper et à dupliquer puisqu'elles sont les plus contraignantes de la chaîne. Dans cette configuration, les poids valent respectivement pour le premier étage, $w(s_1, 3) = 8/3$ et pour le deuxième étage $w(s_2, 1) = 4$. Ce dernier étage est le plus lent, c'est donc lui qui cadence le traitement. Ainsi le débit inverse vaut $T = 4$. Pour $P = 3$, seul le poids du premier étage changerait avec $w(s_1, 2) = 8/2 = 4$, la solution est donc identique pour 3 ou 4 processeurs au niveau du débit inverse.



(a) Solution isolant les tâches sans état et avec état interne.



(b) Solution isolant les tâches une par une.

FIGURE 2 – Ordonnancement possible d'une chaîne sur 12 processeurs.

La Figure 2 présente d'autres solutions pour la même chaîne avec $P = 12$ processeurs. La première solution décrite dans la Figure 2b vise à ramener l'ensemble de tâches à un temps d'exécution valant 1 pour atteindre le débit inverse minimum. La stratégie de la Figure 2a est différente : les tâches sont rassemblées des étages selon leur état interne. Puis un nombre suffisant de processeurs leur est attribué pour pouvoir aboutir à $T = 1$. A priori, ces deux approches sont optimales par rapport au débit inverse ou au nombre de processeurs. Cependant, si la latence est prise en compte, la seconde configuration est meilleure avec une latence de 3 contre une latence de 5 pour la première configuration. Il est donc possible de trouver plusieurs solutions optimales pour les variables d'optimisation.
Étant donné les techniques de parallélisation envisagées, pipeline et duplication, les règles définies permettent d'établir le comportement de ces tâches vis-à-vis de ces approches. Ce modèle permet donc d'envisager le développement d'heuristiques d'ordonnancement.

4. Description de l'algorithme et évaluation

4.1. Algorithme

L'algorithme développé se fait en deux parties avec les fonctions PROBE et SOLVE qui cherchent à partitionner une chaîne donnée de façon à minimiser lé débit inverse (maximiser le débit) et à minimiser le nombre de ressources utilisées.
La fonction PROBE prend en entrée la chaîne concernée, le nombre de ressources P et un débit inverse T cible. Pour celui-ci, cette fonction cherche s'il est possible de faire un partitionnement respectant T en utilisant au plus P ressources. L'algorithme découpe des sous-séquences de poids le plus élevé tout en restant inférieur à T avec le moins possible de ressources, en itérant éventuellement plusieurs fois pour trouver cette configuration. À la fin de cette fonction, la chaîne est partitionnée. Pour vérifier si ce partitionnement est valide, il suffit que le nombre de ressources utilisées pour le découpage soit inférieur à P. Dans le cas contraire, la solution trouvée n'est pas valide.
La fonction SOLVE utilise la fonction PROBE pour savoir si pour un T donné, il est possible de trouver une solution optimale respectant le nombre de ressources. SOLVE cherche le T optimal en effectuant une recherche binaire sur celui-ci.

4.2. Évaluation

L'évaluation d'algorithmes ou d'heuristiques se fait sur des cas d'utilisation réels et implémenté notamment dans AFF3CT.
La première phase a pour but de déterminer l'ordonnancement d'une chaîne pour un nombre de ressources P, un niveau de bruit σ et un *interframe* donné. Le niveau de bruit permet de qualifier la qualité du signal reçut et influe sur la durée des traitements de communications numériques. La première étape de cette phase consiste à exécuter la chaîne séquentiellement. Ainsi, il est possible de récupérer le temps d'exécution moyen des tâches d'une chaîne de communications numériques pour un niveau de bruit spécifique. Il est alors possible d'appliquer l'heuristique développée qui génère en sortie un graphe de dépendance de tâches avec l'ordonnancement, un fichier C++ avec l'implémentation associée et plusieurs graphes de comparaison des métriques (débit, latence) en fonction du nombre de processeurs pour des heuristiques de référence, l'algorithme abordé plus haut notamment. Le fichier généré peut être exécuté en *multithreadé*, il est alors possible d'analyser la latence et le débit réel de l'implémentation choisie vis-à-vis des simulations. Des différences peuvent apparaître à cause des synchronisations d'étage de pipeline, ou encore par rapport à d'autres implémentations pour le même niveau de bruit.
La chaîne de communication choisie pour faire le test est la chaîne du standard de communication DVB-S2, pour Digital Video Broadcasting - second generation. Cette chaîne est utilisée notamment pour la transmission satellitaire de contenu de vidéo [15] et est implémentée dans la bibliothèque AFF3CT [2]. Grâce à la diversité de codages canal et de modulations que propose ce standard, il peut être utilisé notamment pour du service de broadcast (BS, Broadcast Services), la transmission TV (TV, Digital multiprogramme Television) ou de la TV haute définition (HDTV High Definition Television). L'évaluation porte plus particulièrement sur la chaîne de réception, car le décodeur qu'elle contient est la tâche la plus longue. Cassagne et al. [13] fournissent un point de comparaison, la même chaîne y est parallélisée et pipelinée à la main pour une architecture spécifique avec 35 ressources. Pour pouvoir la comparer à l'implémentation faite par l'algorithme, les performances de la chaîne sont testées sur l'architecture choisie pour l'évaluation.
Seule la chaîne de réception est évaluée, c'est, en effet, dans cette partie que se trouve la tâche

de décodage, qui est la plus calcul intensive. La configuration choisie est le mode de communication MODCOD 2 avec un niveau de bruit de 4dB, un nombre d'interframe de 16 et le nombre d'itérations du décodeur est fixé à 10.
L'architecture cible est la suivante : 2×18(Intel ® Xeon ™ Platinum 8168 CPU), une vitesse de processeur de 2600 MHz, 128 Go de Ram, le mode *Turbo Boost* est désactivé tout comme l'*hyperthreading*.
Les figures 3 et 4 présentent respectivement le débit et la latence en fonction du nombre de processeurs pour la chaîne de réception dans le cas de la simulation, i.e. la valeur théorique trouvé par l'algorithme, de la valeur effective testée sur l'architecture (*benchmark*) et la valeur effective de la version faite à la main (*baseline*). La configuration optimale est atteinte pour 18 ressources contre 35 ressources utilisées par la baseline. Aussi, le débit réel atteint pour 18 ressources est de 43 Mbps soit plus du double du débit de la baseline qui atteint 20 Mbps. La latence réelle trouvée pour 18 ressources est également meilleure, car étant de 25000 μs contre 88000 μs pour la baseline. Il est possible d'observer des sursauts de latence à la figure 4 qui correspondent à l'ajout d'un étage de pipeline dans la configuration. Un autre aspect est que la valeur réelle est inférieure de 8 % à la valeur théorique qui atteint 47 Mbps. Le coût de mouvement de données pourrait être la source de cet écart, cependant le parallélisme de pipeline fait qu'il y a du recouvrement de calcul. Le surcoût de synchronisation semble être un meilleur candidat pour expliquer cette différence.

5. État des travaux et conclusion

Les travaux réalisés se sont focalisés dans un premier temps sur le développement d'une heuristique permettant de mieux appréhender la problématique [25], un simulateur python est associé à cette heuristique et permet de connaître quelles seraient les performances d'une chaîne de communication numérique dans le cadre du modèle adopté. Les travaux actuels sont dirigés vers la réalisation d'un algorithme permettant de trouver la configuration optimale qui répond à la problématique définie en 3. L'évaluation effectuée est satisfaisante quant au gain qu'elle permet d'avoir du point de vue du débit, de la latence et aussi du nombre de ressources utilisées. Il s'agit actuellement de mieux comprendre les différences entre les résultats théoriques de simulation et les performances obtenues expérimentalement. Aussi, il serait intéressant d'effectuer ces mêmes évaluations de performances sur d'autres chaînes existantes. La suite immédiate de ces travaux porte sur la parallélisation de plusieurs chaînes sur une même architecture avec comme objectif de maximiser le débit agrégé des différentes chaînes et en même temps de respecter le débit minimal exigé pour chaque chaîne. Suite à cela, les problématiques abordées précédemment pourront être entendues aux architectures hétérogènes.
Nous avons vu qu'il existe un besoin de paralléliser les graphes de tâches des chaînes de communications numériques dans le domaine de la Radio Logicielle afin d'obtenir des débits compatible avec le standard voulu. En exploitant plusieurs types de parallélisme, de pipeline et de tâches, il est possible d'obtenir un algorithme de découpage optimal de la chaîne qui demande encore d'être implémenté pour des chaînes de communications numériques. Il est possible d'aller plus loin sur la parallélisation en complexifiant le modèle adopté pour les travaux futurs.

Annexes

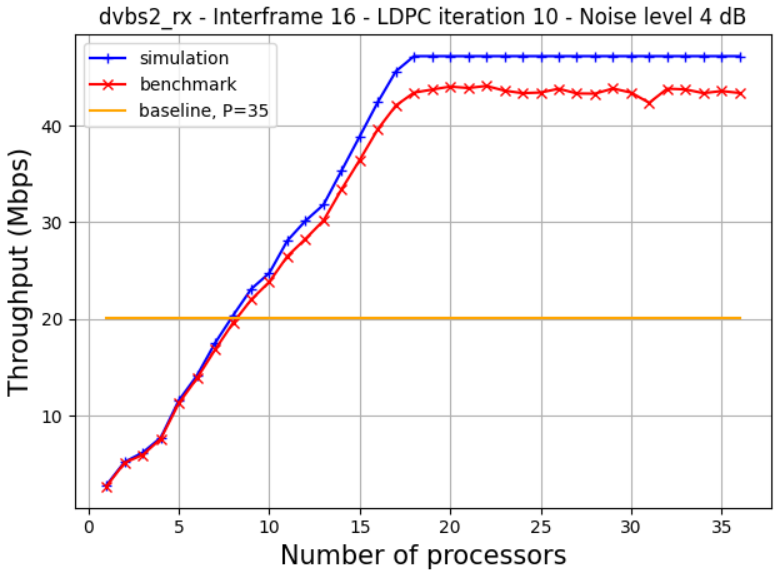


FIGURE 3 – Débit (Mb/s) en fonction du nombre de processeurs pour la chaîne de réception de DVBS2

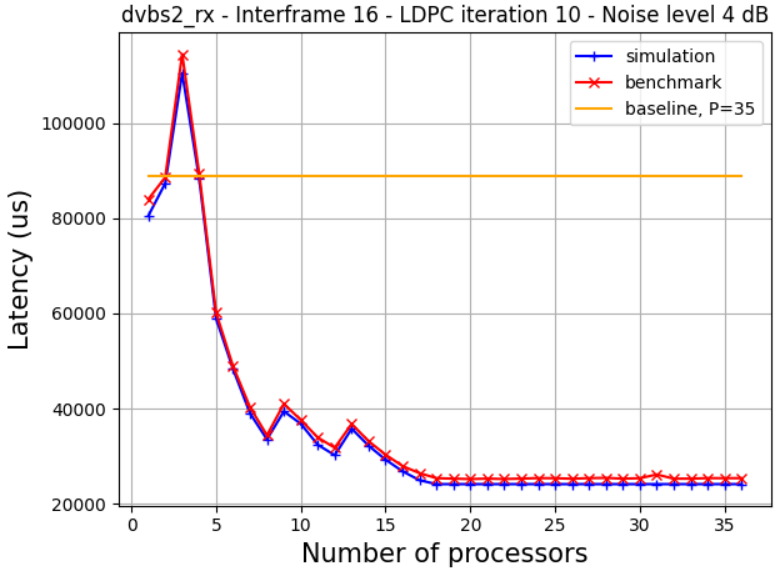


FIGURE 4 – Latence (µs) en fonction du nombre de processeurs pour la chaine de réception de DVBS2

Bibliographie

1. Aff3ct - a fast forward error correction toolbox, github.com/aff3ct/aff3ct (visité le 24.03.2023).
2. Dvb-s2 sdr transceiver, github.com/aff3ct/dvbs2 (visité le 24.03.2023).
3. Gnu radio – the free and open software radio ecosystem, github.com/gnuradio/gnuradio (visité le 05.04.2023).
4. Myintrinsics++ (mipp), github.com/aff3ct/mipp (visité le 24.03.2023).
5. Python to aff3ct, github.com/aff3ct/py_aff3ct (visité le 24.03.2023).
6. A.Cassagne et al. – A dsel for low latency software-defined radio on multicore cpus. – In *CASES'21 : ACM International Conference on Compilers, Architecture, and Synthesis for Embeded Systems*, New York, NY USA, October 2021. ACM.
7. Akeela (R.) et Dezfouli (B.). – Software-defined radios : Architecture, state-of-the-art, and challenges. *Computer Communications*, vol. 128, 2018, pp. 106–125.
8. Benoit (A.), Çatalyürek (U. V.), Robert (Y.) et Saule (E.). – A survey of pipelined workflow scheduling : Models and algorithms. *ACM Comput. Surv.*, vol. 45, n4, aug 2013.
9. Benoit (A.) et Robert (Y.). – Complexity results for throughput and latency optimization of replicated and data-parallel workflows. *Algorithmica*, vol. 57, n4, 2010, pp. 689–724.
10. Bloessl (B.), Müller (M.) et Hollick (M.). – Benchmarking and Profiling the GNU Radio Scheduler. – In *9th GNU Radio Conference (GRCon 2019)*, Huntsville, AL, September 2019. GNU Radio Foundation.
11. Boulet (P.). – Formal semantics of array-ol, a domain specific language for intensive multi-dimensional signal processing. 01 2008.
12. Cassagne (A.), Aumage (O.), Barthou (D.), Leroux (C.) et Jego (C.). – Mipp : a portable c++ simd wrapper and its use for error correction coding in 5g standard. – pp. 1–8, 02 2018.
13. Cassagne (A.), Léonardon (M.), Tajan (R.), Leroux (C.), Jégo (C.), Aumage (O.) et Barthou (D.). – A flexible and portable real-time dvb-s2 transceiver using multicore and simd cpus. – In *2021 11th International Symposium on Topics in Coding (ISTC)*, pp. 1–5, 2021.
14. ETSI 3GPP - TS 38.212. – Multiplexing and channel coding (r. 15)., 2018.
15. ETSI EN 302 307 V1.2.1. – Digital video broadcasting (dvb); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (dvb-s2), 2009.
16. Gayen (N.), Ax (J.), Flasskamp (M.), Klarhorst (C.), Jungeblut (T.), Tang (M.) et Kelly (W.). – Scalable mapping of streaming applications onto mpsocs using optimistic mixed integer linear programming. – In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 348–352, 2018.
17. Giard (P.), Sarkis (G.), Leroux (C.), Thibeault (C.) et Gross (W. J.). – Low-latency software polar decoders. *J. Signal Process. Syst.*, vol. 90, n5, may 2018, p. 761–775.
18. Glitia (C.), Dumont (P.) et Boulet (P.). – Array-ol with delays, a domain specific specification language for multidimensional intensive signal processing. *Multidimensional Systems and Signal Processing*, vol. 21, 06 2010.
19. Gordon (M. I.), Thies (W.), Karczmarek (M.), Lin (J.), Meli (A. S.), Lamb (A. A.), Leger (C.), Wong (J.), Hoffmann (H.), Maze (D.) et Amarasinghe (S.). – A stream compiler for communication-exposed architectures. *SIGPLAN Not.*, vol. 37, n10, oct 2002, p. 291–303.
20. Karlsson (A.), Sohl (J.), Wang (J.) et Liu (D.). – epuma : A unique memory access based parallel dsp processor for sdr and cr. – In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 1234–1237, 2013.
21. Le Gal (B.) et Jego (C.). – High-throughput multi-core ldpc decoders based on x86 proces-

sor. *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, n5, 2016, pp. 1373–1386.

22. Mack (J.), Gener (S.), Akoglu (A.), Holtom (J.), Chiriyath (A.), Chakrabarti (C.), Bliss (D.), Krishnakumar (A.), Goksoy (A.) et Ogras (U.). – Gnu radio and cedr : Runtime scheduling to heterogeneous accelerators. – In *Proceedings of the GNU Radio Conference*volume 7, 2022.
23. Maheshwarappa (M. R.), Bowyer (M.) et Bridges (C. P.). – Software defined radio (sdr) architecture to support multi-satellite communications. – In *2015 IEEE Aerospace Conference*, pp. 1–10, 2015.
24. Morman (J.), Lichtman (M.) et Müller (M.). – The future of gnu radio : Heterogeneous computing, distributed processing, and scheduler-as-a-plugin. – In *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, pp. 180–185, 2022.
25. Orhan (D.). – *Ordonnancement automatique et parallèle du flux de données appliqué à la radio logicielle et notamment au logiciel AFF3CT*. – Thèse, Université de Bordeaux (UB), France, juin 2022.
26. Palkovic (M.), Declerck (J.), Raghavan (P.), Dejonghe (A.) et Van der Perre (L.). – Dart - a high level software-defined radio platform model for developing the run-time controller. – In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1617–1620, 2011.
27. Pınar (A.) et Aykanat (C.). – Fast optimal load balancing algorithms for 1d partitioning. *Journal of Parallel and Distributed Computing*, vol. 64, n8, 2004, pp. 974–996.
28. William Thies, Michal Karczmarek (S. A.). – Streamit : A language for streaming applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002.