Chapitre 1

Extensions spécialisées avec 3 opérandes

Ce chapitre étend les travaux présentés dans le chapitre précédent, utilisant des opérations classiques à deux opérandes en entrées. Comme cela a été mentionné, ce choix, parfaitement valide au demeurant, reste la principale limitation bridant la conception d'instructions adaptées au décodage de CCE. Dans cette partie, nous évaluons les gains qu'il est possible d'atteindre en levant cette contrainte architecturale. En effet, des cœurs RISC-V plus complexes permettent l'accès à 3 données pour exécuter, par exemple, des opérations de type MAC (Multiply And Accumulate). La première partie du chapitre expose les différentes caractéristiques architecturales de ces cœurs pouvant opérer sur trois registres d'entrée simultanément. Nous présentons également le cœur RISC-V sélectionné et la stratégie mise en œuvre pour insérer nos instructions dans l'ISA RISC-V.

Dans un second temps, nous présentons les différentes extensions mises au point et nous mettons en valeur les différences avec les résultats obtenus dans le chapitre précédent. Pour terminer, une évaluation de l'impact de ces nouvelles extensions est réalisée. Les gains en termes de temps d'exécution sont exposés et mis en perspective des surcoûts matériels engendrés. Ces expérimentations, comparables à celles réalisées pour les instructions à 2 registres, sont étendues pour les configurations SIMD sur 64 bits.

Contents

1.1 Modèle d'architecture ciblé

Les cœurs de processeurs généralistes sont en principe conçus de manière à pouvoir délivrer 2 opérandes par cycle d'horloge à l'unité arithmétique et logique (UAL). Ce choix de conception est intrinsèquement lié à l'ISA des processeurs de type RISC où les instructions exécutées manipulent au plus 2 deux opérandes provenant des registres et/ou une valeur immédiate contenue dans un champ de l'instruction. Cependant, certains cœurs de processeurs pensés pour exécuter plus efficacement des applications de traitement du signal possèdent un troisième port de lecture pour la file de registres, afin de permettre l'exécution d'instructions de type MAC (Multiply And Accumulate). L'accès à une troisième opérande permet l'intégration d'opérations plus complexes au sein de l'UAL, dans le but de réduire le nombre d'instructions à réaliser, et par voie de conséquence, de diminuer le temps d'exécution des applicatifs ainsi que la taille de leur binaire. Ces nouvelles opportunités ont toutefois un impact négatif sur la complexité matérielle de l'architecture, car elles nécessitent l'ajout d'un réseau de multiplexage supplémentaire dans la file de registres ou bien une duplication des bancs mémoire dans le but de délivrer cette donnée supplémentaire à l'UAL. De plus, un format d'instruction supplémentaire doit être géré dans le décodeur d'instructions pour autoriser l'usage de cet troisième opérande. L'augmentation de la complexité matérielle induite et le faible usage d'une troisième opérande dans l'UAL pour un bon nombre d'applications généralistes, limitent l'intégration de cette approche la majorité des cœurs RISC-V.

Dans le cadre de l'étude présentée dans les chapitres précédents, nous avons identifié et intégré des instructions possédant au maximum deux opérandes en entrée, notée dans ce chapitre **2R**. Cette contrainte architecturale a limité certains choix de conception et n'a offert que peu d'opportunités pour exploiter au mieux les codes LDPC-NB et les turbo codes. Dans ce chapitre, cette contrainte est allégée en considérant que l'UAL, et donc les instructions, sont en mesure de manipuler trois données/registres (*rS1, rS2, rS3*). La contrainte sur la sortie de l'UAL, elle reste inchangée, elle ne peut produire au maximum qu'une seule donnée (*rD*).

Dans cette section, nous présentons une évolution des extensions conçues afin d'accélérer l'exécution des algorithmes de décodage des codes correcteurs d'erreurs. L'utilisation

Champ	**Bits**
Opcode	[6:0]
Registre destination	[11:7]
Fonction 3	[14:12]
Registre source 1	[19:15]
Registre source 2	[24:20]
Fonction 2	[26:25]
Registre source 3	[31:27]

Table 1.1 – Format 32 bits standardisé pour les instructions de type R4 (3 registres d'entrées) dans les spécifications RISC-V : RV-FDQ

d'une troisième opérande en entrée a nécessité une remise à plat des extensions proposées précédemment, pour aboutir à une solution efficace : certaines instructions à deux entrées ont été fusionnées, et d'autres ont aussi été créées et/ou modifiées afin de maintenir un flot d'exécution cohérent.

Dans la spécification de base de l'ISA RISC-V, il n'existe pas de format standardisé permettant d'encoder directement 3 registres sources dans une instruction de type RV32/64I. Afin de rester compatible avec cette ISA et ainsi de continuer à bénéficier des environnements de compilation, nous avons décidé d'utiliser le format proposé dans les extensions RV-FDQ (simple, double et quadruple précision en virgule flottante). Ces dernières proposent un format 32 bits spécifique pour les opérations de type FMADD (Fused Multiplication-Accumulate) tel que cela est décrit dans le tableau 1.1.

Le détournement de ce format existant pour les opérations flottantes permet de réduire le surcoût matériel lié à l'introduction de nos extensions dans le décodeur d'instructions. Afin de maintenir la compatibilité des cœurs avec l'extension F, nous avons sélectionné des *opcodes* prévus pour l'ajout d'instructions spécialisées. Ces derniers sont utilisés uniquement pour des instructions spécialisées additionnelles à deux ou trois opérandes et ont pour valeur $\times 0b, \times 3b, \times 5b, \times 7b$, conformément à la spécification RISC-V. Il est à noter que les instructions manipulant seulement un ou deux registres utilisent comme précédemment le type R.

Les cœurs de RISC-V que nous avions auparavant sélectionnés possèdent des architectures internes simples et ne supportent pas par défaut les extensions F, D ou Q qui nécessitent une file de registres à 3 sorties pour exécuter les instructions de type FMA. En conséquence, leurs décodeurs d'instructions n'étaient pas prévus pour décoder les instructions de type R et de générer les signaux de contrôle adéquats. En conséquence, pour

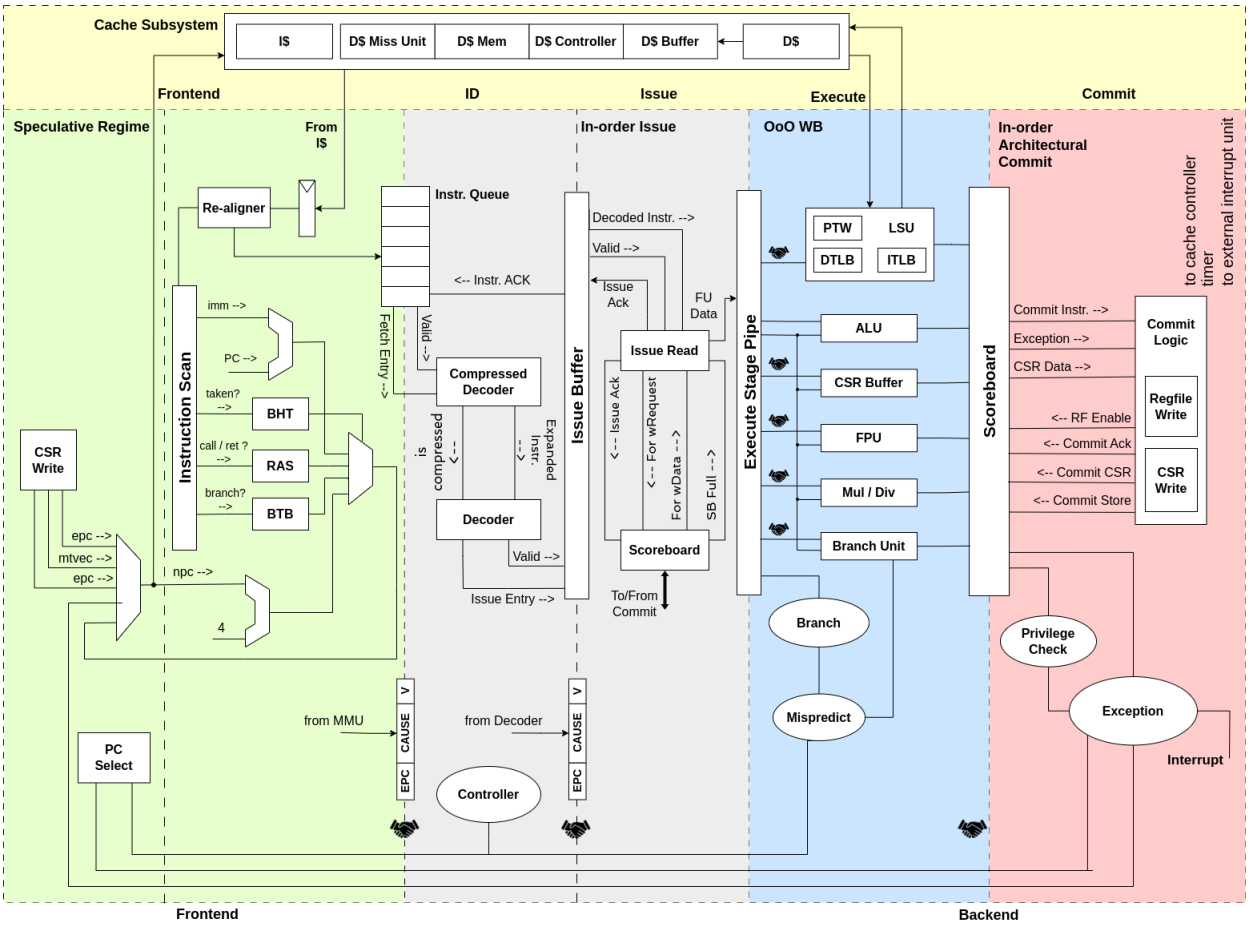

Figure 1.1 – Schéma de l'architecture du cœur CVA6.

éviter des temps d'ingénierie chronophages, nous avons décidé de changer de cible architecturale. Parmi les cœurs RISC-V *high-end* supportant ces extensions, nous pouvons citer par exemple les cœurs BOOM [1], CVA6 [2], SHAKTI [3] et ROCKET [4]. Ces derniers, en plus de couvrir un nombre important d'extensions de l'ISA possèdent aussi un mode de fonctionnement 64 bits.

Les SoC dans lesquels ils sont intégrés possèdent des mémoires caches ainsi que bien d'autres périphériques leur permettant de faire démarrer un noyau Linux. Pour des raisons techniques, nous avons sélectionné le cœur CVA6 développé par l'Open Hardware Group. Ce cœur, développé par ETH Zürich et poussé dans des projets étudiants par Thales, possède pour principal avantage d'être déployable sur les cartes de prototypage Digilent Genesys 2 que nous avions à disposition.

Le cœur CVA6 possède une architecture interne composée de 6 étages de pipeline,

Système évalué	**CVA6 cœur**	**CVA6 cœur**	**CVA6 cœur**	**CVA6 SoC**	**CVA6 SoC**	**CVA6 SoC**
Chemin de données	64 bits	64 bits	64 bits	64 bits	64 bits	64 bits
Entrées dans l'UAL	2	3	3	2	3	3
ISA	RV64I	RV64I	RV64I	RV64I	RV64I	RV64I
Extensions RV	MAC	MAC	MACFD	MAC	MAC	MACFD
LUT	34576	34695	46615	59495	59624	71551
Flip-Flop	19806	19929	23954	41870	41993	46006
Blocs DSP	16	16	27	16	16	27
Blocs RAM	72	72	72	87	87	87

Table 1.2 – Complexités matérielles des cœurs CVA6 avec SoC sur cible Kintex 7 lorsque 2 ou 3 opérandes sont accessibles dans l'UAL. Résultats post-placement routage avec fréquence de fonctionnement fixée à 50 MHz.

tel que cela est schématisé dans la figure 1.1. Il implante l'ISA de base RV64I, supporte nativement les extensions M/C/F et A de la spécification RISC-V et intègre un mécanisme de prédiction de branchement efficace. En interne, en fonction de la configuration du cœur, ce dernier peut opérer sur des données codées sur 32 bits ou 64 bits. Ce dernier format de représentation des données, choisi avant la synthèse logique du cœur, permet d'appliquer un facteur de parallélisation deux fois supérieur lors des modes de fonctionnement SIMD. Toutefois, il est à noter que tout comme les cœurs RISC-V précédemment étudiés, le cœur CVA6 ne gère pas les accès mémoire aux données non-alignées. Les informations fournies dans le tableau 1.2 permettent de quantifier la complexité du cœur CVA6 ainsi que du SoC dans lequel il est inclus. Les résultats présentés sont issus des rapports post-placement et routage pour un FPGA de type Xilinx Kintex 7 lorsqu'une fréquence de fonctionnement de 50 MHz est fixée.

1.2 Évolution des extensions

Dans cette partie, nous allons succinctement présenter les évolutions des extensions précédemment proposées. Pour des raisons de lisibilité et de redondance, les descriptifs complets des instructions sont reportés en annexes.

L'utilisation de trois opérandes d'entrée permet de concevoir des instructions plus complexes en chaînant, par exemple, plus de ressources matérielles. Afin d'identifier les nouvelles instructions pour chaque famille de codes correcteurs d'erreurs, nous sommes repartis des spécifications algorithmiques et des listings assembleurs issus de nos descriptions logicielles initiales. Ceci, pour ne pas être influencé par des choix et des décisions

	Mnémonique	Mode SISD		Mode SIMD inter-trames	
		2R	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_abs_pi8	✓	✓	✓	✓
	i8_add_sat127_pi8	✓	✓	✓	✓
	i8_min_pi8	✓	✓	✓	✓
	i8_sub_sat127_pi8	✓	✓	✓	✓
	i8_cmpeq_pi8	✓		✓	
	i8_invB_Aneq1_pi8	✓		✓	
	i8_max_pi8	✓		✓	
	i8_xorA_signB_pi8	✓		✓	
	i8_sign_pi8			✓	✓
Instr à 3 reg.	i8_invC_xorA_signB_pi8		✓		✓
	i8_minC_maxAB_pi8		✓		✓
	i8_blendC_cmpeqAB_pi8		✓		✓
Total (2R|3R)		8	7(4|3)	9	8(5|3)

Table 1.3 – Instructions proposées pour l'accélération du décodage des codes LDPC en utilisant 2 ou 3 opérandes.

issues de la première phase de nos travaux.

Il résulte de cette analyse qu'en plus des instructions manipulant trois opérandes d'entrées, notées **3R**, il est nécessaire de conserver des instructions à une ou deux opérandes. En effet, pour certaines opérations complexes qui pourraient être factorisées, certaines valeurs intermédiaires doivent aussi être stockées en mémoire pour une utilisation ultérieure. C'est le cas, par exemple, du calcul de la valeur absolue (*i8_abs_pi8*), et des additions/soustractions saturées (*i8_add_sat127_pi8* et *i8_sub_sat127_pi8*). L'extension du jeu d'instructions que nous avons ainsi développé, est une combinaison entre un sous-ensemble des instructions précédemment identifiées et de nouvelles instructions. Les instructions nouvellement définies, ainsi que celles issues du travail présenté dans le chapitre 3 sont résumées pour les codes LDPC dans le tableau 1.3. Dans ce tableau, les colonnes notées **2R** font référence aux extensions proposées dans le chapitre précédent, tandis que les colonnes labellisées **3R** décrivent les extensions nécessaires ici (qu'elles possèdent 2 ou 3 opérandes).

On observe pour les codes LDPC que trois nouvelles instructions à 3 registres ont été proposées pour le mode d'exécution scalaire. Ces dernières, décrites dans le tableau 1.4, sont utilisées de la manière suivante :

— *i8_invC_xorA_signB_pi8* : Permet de simplifier le calcul de parité en couplant le calcul du signe du LLR et la mise à jour de la valeur de parité.

Instruction	Mnémonique	Implantation
Maximum puis minium	i8_minC_maxAB_pi8 rD,rS1,rS2,rS3	rD := MIN8(MAX8(rS1,rS2),rS3)
Calcul du minimum et mise à jour	i8_blendC_cmpeqAB_pi8 rD,rS1,rS2,rS3	mask := (rS1==rS2) ?0xff:0x00 min_t := rS1 & ∼mask min_u := rS3 & mask rD := min_t |min_u
Génération du message sortant du noeud de parité	i8_invC_xorA_signB_pi8 rD,rS1,rS2,rS3	rD := rS1 ∧(rS2 ⩾0) ?rS3 : -rS3

Table 1.4 – Spécification des nouvelles instructions à 3 opérandes pour le décodage des codes LDPC.

Instruction	Mnémonique	Implantation
Fonction G de l'arbre de décodage SC	i8_Gx_pi8 rD,rS1,rS2,rS3	Addsat := sat127rS1 + rS2 Subsat := sat127rS2 - rS1 rD :=(rS3 == 0) ? Addsat : Subsat

Table 1.5 – Spécification de l'instruction pour la fonction **G** pour le décodage des codes polaires.

— *i8_minC_maxAB_pi8* : Réalise la mise à jour de la valeur du *min2* de l'algorihtme Min-Sum en une seule instruction.

— *i8_blendC_cmpeqAB_pi8* : Met en œuvre un masque logique pour sélectionner la valeur de *min1* ou de *min2* lors de la génération du LLR en sortie.

La mise au point de ces trois instructions spécialisées pour le mode scalaire a permis de remplacer quatre instructions précédemment déployées (**2R**). Il est à noter que ces instructions à 3 opérandes permettent aussi de supprimer l'usage d'instructions arithmétiques et logiques RV32I qui étaient mises en œuvre dans les cœurs de calcul [1]. Pour les architectures SIMD inter-trames, les résultats obtenus sont similaires à ceux obtenus précédemment.

Contrairement au cas des codes LDPC où les nouvelles instructions **3R** sont une combinaison des instructions précédemment identifiées, pour les codes polaires l'extension à 3 opérandes a aussi permis d'optimiser la fonction **G** dont l'équation est fournie dans le chapitre 3. En effet, celle-ci ne pouvait être accélérée auparavant. La possibilité d'utiliser 3 opérandes en entrée de l'UAL permet d'inclure l'ensemble des traitements à réaliser au sein d'une seule et unique instruction, tel que cela est indiqué dans le tableau 1.5.

L'analyse des données présentées dans le tableau 1.6 permet de noter que l'utilisation de cette nouvelle instruction permet de se passer des instructions *i8_setMask_Aeq1_pi8*, *i8_add_sat127_pi8* et *i8_sub_sat127_pi8* et ainsi de réduire le nombre de modifications

1. Ces dernières n'apparaissent pas dans le tableau

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_setMask_Aeq1_pi8			✓		✓	
	i8_add_sat127_pi8	✓		✓		✓	
	i8_sub_sat127_pi8	✓		✓		✓	
	i8_clrA_Bneq0_pi8	✓	✓	✓	✓	✓	✓
	i8_Rx_pi8	✓	✓	✓	✓	✓	✓
	i8_Fx_pi8	✓	✓	✓	✓	✓	✓
Instr. à 3 reg.	i8_Gx_pi8		✓		✓		✓
Total (2R|3R)		6	4(3|1)	6	4(3|1)	6	4(3|1)
Extension pour l'algorithme Fast-SC(F-SC)							
Instr. à 2 reg.	i16_add_accA_loB_pi8			✓	✓		
	i16_add_accA_hiB_pi8			✓	✓		
	i8_sign_pi16			✓	✓		
	i8_Hxor_pi8					✓	✓
	i8_Hadd_pi8					✓	✓
Total (2R|3R)		6	4(3|1)	9	7(6|1)	9	6(5|1)

Table 1.6 – Instructions proposées pour l'accélération du décodage des codes polaires (algorithmes SC et F-SC) en utilisant 2 ou 3 opérandes.

Instruction	**Mnémonique**	**Implantation**
Minimum entre C et addition saturée de A et B	u8_min_addsat_pu8	rD := Min(Sat63(rS1+rS2), rS3)
Comparaison égale ou supérieure de A et B retourne les 8 LSB de C	u8_cmpge_AndB_lo8C_pu8	rD := (rS1 >= rS2) ? rS3[7:0]:0x00

Table 1.7 – Nouvelles instructions spécifiques pour les codes LDPC-NB avec 3 registres sources.

à apporter au décodeur d'instructions. Afin d'accélérer le décodage des codes polaires utilisant l'algorithme SC, seules 4 instructions sont maintenant requises. Cette diminution d'un facteur 1.5× du nombre d'instructions spécialisées, par rapport à l'approche **2R** va permettre de réduire significativement le nombre d'instructions à exécuter. Le tableau 1.6, met en évidence que les instructions présentées afin d'améliorer les opérations d'élagage de leur côté, ne bénéficient pas de ce 3$^{\text{ème}}$ opérande. Ainsi, l'accélération de l'élagage est toujours supportée par 5 instructions **2R**. En procédant à une analyse similaire à celle que nous venons de décrire, un nouveau jeu d'instruction est ainsi identifié et reporté dans le tableau 1.7 pour la famille de code LDPC-NB. Celle-ci consiste en l'assemblage de plusieurs instructions spécifiques proposées précédemment et de nouvelles.

Toutefois, du fait de la complexité accrue de l'algorithme de décodage (en logiciel),

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	u8_addu_sat64_pu8	✓	✓	✓	✓	✓	✓
	u8_minu_pu8	✓	✓	✓	✓	✓	
	u8_subu_sat64_pu8	✓	✓	✓	✓	✓	✓
	u8_maxu_pu8			✓	✓		
	u8_cmpge_pu8			✓			
	u8_hminu_pu8					✓	✓
	u8_Aandb_lo8B_pu8			✓			
Instr. à 3 reg.	u8_min_addsat_pu8		✓		✓		✓
	u8_cmpge_AandB_lo8C_pu8				✓		
Total (2R|3R)		3	4(3|1)	5	6(4|2)	4	4(3|1)

Table 1.8 – Instructions proposées pour l'accélération du décodage des codes LDPC-NB en utilisant 2 ou 3 opérandes.

il est difficile d'extraire des instructions spécifiques pertinentes et compatibles avec les contraintes voulues. En effet, le nombre d'accès mémoires et la gestion des pointeurs mémoire complexifient l'analyse du code. Les instructions pertinentes identifiées sont les suivantes :

— *u8_min_addsat_pu8* : permet une concaténation des instructions d'additions saturées, puis génère le minimum entre ce résultat et la troisième entrée.

— *u8_cmpge_AandB_lo8C_pu8* : effectue une comparaison entre A et B puis renvoie les 8 premiers bits de C ou bien 0 selon le résultat de la comparaison.

Comme pour les codes LDPC, l'utilisation de ces nouvelles instructions vont théoriquement aboutir à une réduction du nombre total d'instructions exécutées dans l'algorithme. Une différence importante entre les versions inter et intra-trame est l'utilisation d'un minimum horizontal. Une majorité des instructions à 2 registres sources de notre extension sont conservées. Les nouvelles instructions à 3 registres permettent, quand à elles, d'améliorer la latence dans la fonction de calcul des nœuds de parité, à l'aide de la fonction *u8_min_addsat_pu8*. Enfin, l'instruction *cmpge_AandB_lo8C_pu8* permet de simplifier l'étape de recherche de l'index de la valeur minimale du vecteur de données inter-trames.

Enfin, en ce qui concerne les turbo codes, les résultats sont grandement améliorés grâce à l'utilisation d'une architecture **3R**. En effet, cette famille de codes bénéficie grandement des instructions à 3 registres sources, comme montré dans 1.10. Par exemple, les calculs nécessaires au parcours d'un treillis LTE de l'algorithme BCJR font appel à des opéra-

	Mnémonique	Mode SISD		Mode SIMD inter-trames		Mode SIMD intra-trame	
		2R	**3R**	**2R**	**3R**	**2R**	**3R**
Instr. à 2 reg.	i8_max_pi8	✓	✓	✓	✓	✓	✓
	i8_scale_pi8	✓	✓	✓	✓		
	i8_sign_pi8			✓	✓		
	i8_add_pi8			✓	✓	✓	✓
	i8_sub_pi8			✓	✓	✓	✓
	i8_add_srl1_pi8			✓	✓		
	i8_srl1_pi8			✓	✓		
	i8_shuffle_pi8					✓	✓
	i8_add_div2_pi8					✓	✓
	i8_sub_div2_pi8					✓	✓
	i8_sat_sub_pi8					✓	✓
	i8_scale_add_pi8					✓	✓
Instr. à 3 reg.	i8_accumax_pi8		✓		✓		
	i8_maxpm_pi8		✓		✓		✓
	i8_accupp_pi8		✓		✓		✓
	i8_accump_pi8		✓		✓		✓
Total (2R|3R)		2	6(2|4)	7	11(7|4)	8	11(8|3)

Table 1.9 – Instructions proposées pour l'accélération du décodage des turbo codes en utilisant 2 ou 3 opérandes.

Instruction	**Mnémonique**	**Implantation**
Addition accumulée	i8_accupp_pi8 rD,rS1,rS2,rS3	rD := rS1 + rS2 + rS3
Addition et soustraction accumulée	i8_accump_pi8 rD,rS1,rS2,rS3	rD := rS1 - rS2 + rS3
Maximum accumulée	i8_accumax_pi8 rD,rS1,rS2,rS3	rD := MAX8(MAX8(rS1+rS2),rS3)
Maximum entre A+B et C-B	i8_maxpm_pi8 rD,rS1,rS2,rS3	rD := MAX8(rS1 + rS2 , rS3 - rS2)

Table 1.10 – Nouvelle Instruction spécifique pour les turbo codes avec 3 registres sources.

tions utilisant 3 données. Il s'agit de la concaténation d'opérations simples : addition, soustraction et maximum. On retrouve ces nouvelles instructions dans la table ci-dessous 1.9.

NOTA BENE : Les instructions *i8_add_div2_pi8*, *i8_sub_div2_pi8*, *i8_sat_sub_pi8* et *i8_scale_add_pi8* ne figurent pas dans les chapitres précédents.

En effet, celles-ci ont été identifiées après l'écriture de ces chapitres, et conservent le type **2R** (ou 2 registres sources). Puisque leur application est totalement intégrée à ce chapitre, elles sont ainsi présentées ici. Il est important de noter la forte comptabilité entre les instructions **3R** pour les différents modes de parallélisation, ce qui permet de diminuer d'autant la complexité du jeu d'instructions dédié. La latence finale est également optimisée grâce au nombre d'instructions spécialisées mises en œuvre. La version scalaire

SISD bénéficie de 2 instructions, tandis que les architectures **3R**, ont à disposition entre 6 et 11 instructions spécialisées. Les résultats de leur implantation matérielle sont présentés dans les sections suivantes.

1.3 Impact des nouvelles extensions de l'ISA

Afin de mesurer l'impact de ces nouvelles extensions, plusieurs campagnes de tests ont été effectuées. Deux métriques ont été mesurées pour évaluer les performances de nos propositions :

— Le nombre de cycles d'horloges nécessaires pour exécuter les algorithmes de décodage.

— L'augmentation de la complexité matérielle du cœur CVA6 et l'impact sur sa fréquence de fonctionnement.

Ces évaluations ont été réalisées en sélectionnant la configuration de base du cœur CVA6 (RV64I). L'évaluation du temps d'exécution des différents algorithmes de décodage a été obtenu par simulation. Celles-ci ont été réalisées avec l'aide de l'outil *verilator* (version 4.110) qui, à partir de la description en SystemVerilog du cœur CVA6, permet de réaliser des simulations *Bit Accurate* et *Cycle Accurate*. Cette stratégie, a été privilégiée pour éviter des temps de synthèse logique prohibitifs. En effet, il faut compter environ une heure pour réaliser une synthèse et le placement routage du SoC contenant le cœur CVA6.

1.3.1 Évaluation des extensions scalaires

Cette section présente les résultats obtenus pour les décodeurs de CCE étudiés lorsqu'ils opèrent sur des données scalaires, i.e. sans parallélisation SIMD. Les instructions spécialisées traitent des données de 8 bits. Le tableau 1.11 présente une synthèse des résultats obtenus pour l'ensemble des algorithmes de décodage évalués. Les résultats rapportés ici sont homogènes : l'ensemble des travaux présentés dans les chapitres 4 et 5 ont été rejoués pour l'architecture CVA6. Dans le tableau 1.11, sont indiqués les temps nécessaires à l'exécution des versions basiques des décodeurs (❶, Baseline), des versions étendues précédentes (❷, **ISA 2R**) et des versions présentées dans ce chapitre (❸, **ISA 3R**). Dans ce tableau, sont aussi indiqués les gains relatifs entre les différentes versions (❶→❸, ❷→❸) afin de simplifier l'analyse.

		LDPC OMS	SC	**polaire** F-SC	**LDPC-NB** MS	**turbo** MLM
❶ Baseline	cycles	95955	454637	239292	1715625	1585822
❷ SISD ISA 2R	cycles	66013	271609	125008	1357945	938893
❸ SISD ISA 3R	cycles	56514	257901	114207	1086453	819757
Gain (❶→❸)	cycles	39441	196736	125085	629172	766065
Gain (❶→❸)	(%)	41.1%	43.2%	52.2%	36.6%	48.3%
Gain (❷→❸)	cycles	9499	13708	10801	271492	119136
Gain (❷→❸)	(%)	14.3%	5.0%	8.6%	19.9%	12.6%
Débit ❶	Kbits/s	141	900	1711	7	390
Débit ❷	Kbits/s	206	1508	3276	9	659
Débit ❸	Kbits/s	240	1588	3586	12	755
Accélération ❶ → ❸		1.7×	1.8×	2.1×	1.6×	1.9×

Table 1.11 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration 8 bits SISD avec une fréquence de fonctionnement de 50 MHz.

Si l'on s'intéresse tout d'abord aux décodeurs LDPC, on peut noter une réduction du nombre de cycles d'horloge d'environ 14% entre la version précédente (ISA **2R ❷**) et la version actuelle (ISA **3R ❸**). Ces gains sont liés aux instructions à 3 opérandes d'entrée que nous avons développées. Par rapport à la version d'origine du décodeur, les 7 instructions spécialisées que nous avons ajoutées ont permis de réduire le temps d'exécution de 41% sur le cœur CVA6.

Pour les codes polaires, les gains sont plus limités lorsque l'on considère l'algorithme de décodage SC. En effet, le passage de la version ❷ à ❸ n'engendre qu'une modeste réduction de 5% du temps d'exécution. Cette dernière n'ajoute qu'une seule instruction depuis la version ❷ avec l'instruction *i8_Gx_pi8* appliquant la fonction **G** de l'arbre de décodage. Cela permet de s'affranchir de l'utilisation des instructions d'addition et soustraction saturées ainsi que de la sélection conditionnelle. Toutefois, lors de la compilation, le compilateur GCC introduit des instructions supplémentaires de conversions entre les formats int8_t ↔ int32_t lors de l'utilisation de l'instruction **i8_Gx_i8** à 3 entrées. La pénalité liée à l'ajout de ces instructions d'extension de signe des opérandes est moins présente pour le décodeur élaguant l'arbre de décodage (F-SC). Ainsi, une réduction de 8.6% est mesurée entre les versions ❷ et ❸. L'accélération atteint même 52.2% lorsque l'on compare la version ❶ et ❸.

Les facteurs d'accélérations mesurés pour le décodage des codes LDPC-NB sont plus

importants. En effet, entre les versions ❷ et ❸ l'accélération avoisine les 20%. Ce facteur d'accélération est issu de l'utilisation de la nouvelle instruction *u8_min_addsat_pu8* qui est mise en œuvre à la fois dans la fonction de mise à jour des messages des nœuds de variables vers les nœuds de parités ainsi que dans le calcul des nœuds de parités. Ce gain important s'explique par la faible accélération issue des instructions proposées dans ❷. L'accélération obtenue par rapport à la solution initiale ❶ est de $\approx 37\%$.

Enfin, si l'on concentre notre attention sur les turbo décodeurs, une amélioration de 48% est obtenue par la configuration ❸ par rapport à la configuration initiale ❶. Les 4 instructions (ainsi que les instructions **2R**) ajoutées dans ❸ par rapport à la configuration ❷ ont permis une réduction supplémentaire de $\approx 12\%$. Les gains obtenus sont limités à ces niveaux à cause des accès mémoire (*load*, *store* et arithmétique des pointeurs) qui deviennent prépondérants par rapport aux instructions arithmétiques.

Les dernières lignes du tableau 1.11 présentent le débit normalisé des décodeurs, pour une fréquence de fonctionnement à 50 MHz. Cette fréquence est liée à l'implantation matérielle du cœur CVA6 sur le FPGA Kintex-7 présent sur la carte Digilent Genesys 2. Il est à noter que les instructions ajoutées dans le cœur CVA6 n'ont pas d'impact sur la fréquence maximale de fonctionnement. Les résultats présentés mettent en exergue que les configurations ❸ des décodeurs offrent des gains substantiels grâce aux instructions spécialisées proposées.

Afin d'estimer le coût matériel associé à l'implantation des ces instructions dans le cœur testé, les UAL correspondant à chaque configuration sont isolées et comparées à la version de l'UAL standard du cœur. Les surcoûts, exprimés en LUT, sont présentés dans la figure 1.2.

La figure 1.2 présente ainsi les résultats obtenus. Pour les codes LDPC, la configuration **2R** implante 8 instructions, soit 1 instruction supplémentaire par rapport à la configuration **3R** avec 7 instructions. Toutefois, le coût varie de presque 50% entre ces 2 versions. CE gain provient de la simplification du multiplexeur de sortie de l'UAL. Dans le cas des codes polaires, les algorithmes SC et Fast-SC (F-SC) partagent les mêmes implantations d'instructions en matériel pour les configurations **2R** et **3R**, ainsi les résultats observés sont similaires. L'implantation de 4 (**3R**) au lieu de 6 instructions (**3R**), permet de réduire l'empreinte matérielle associée de $\approx 30\%$. Cette observation est également valable avec les codes LDPC-NB. Enfin, pour les turbo codes, 6 instructions sont implantées pour la configuration (**3R**), contre seulement 2 (**2R**). On constate ainsi que le coût matériel augmente fortement et est plus que doublé (137%). Il est à noter que toutes ces instructions

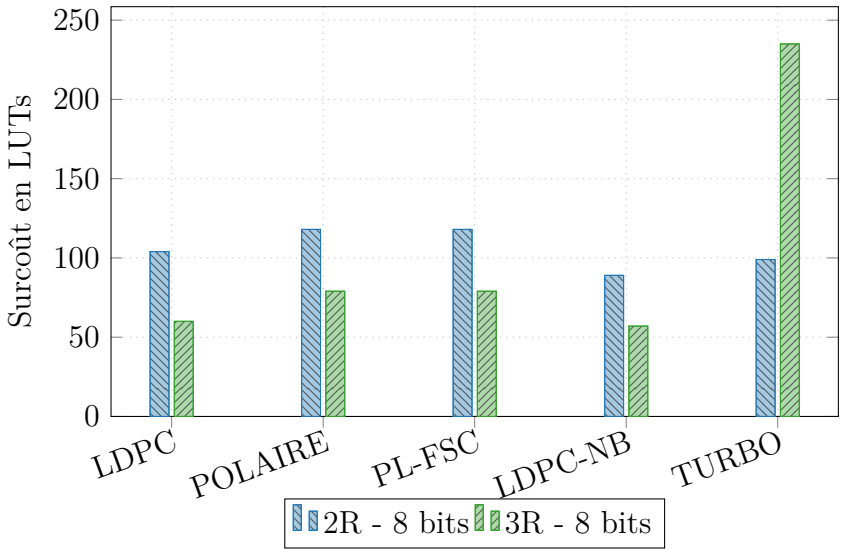

Figure 1.2 – Surcôut des instructions spécialisées SISD dans l'UAL du cœur CVA6.

sont implantées sans conséquences sur la fréquence de fonctionnement du cœur.

1.3.2 Évaluation des extensions SIMD inter-trames

Dans les expérimentations précédentes, nous nous sommes focalisés sur les décodeurs manipulant des données scalaires. Nous allons maintenant étudier l'impact de ces nouvelles extensions lors de l'exécution de décodeurs bénéficiant d'une parallélisation inter-trames. Le CVA6 étant un cœur 64 bits, nous avons décidé de tester deux formats de parallélisation SIMD : 32 et 64 bits. Le premier traite quatre informations codées sur 8 bits chacune (**int8_t×4**), tandis que le second étend ce traitement à 8 données (**int8_t×8**). Les résultats de ces expérimentations sont résumés dans le tableau 1.12.

Si l'on s'intéresse dans un premier temps aux performances des décodeurs LDPC manipulant des données codées sur **int8_t×4**, on observe une réduction de 12% lors du passage ❷→❸. Cette observation est cohérente par rapport aux résultats des expérimentations SISD (tableau 1.11). Pour la version manipulant des données **int8_t×4**, le gain mesuré est plus faible avec 9.2%. Ceci est causé par le temps d'entrelacement des données en amont et en aval de l'étape de décodage, qui consomme 2 fois plus de cycles d'horloge. Toutefois, comme le nombre de données manipulées est 2 fois plus élevé, avec un nombre de cycles d'horloge à peu près équivalent, le débit final est ainsi doublé. Concernant les autres codes correcteurs, lorsqu'ils manipulent des données codées sur 32 bits, on remarque que les gains liés au passage de ❷→❸ sont plus importants que ceux relevés

inter-trames			**LDPC** OMS	SC	**polaire** F-SC	**LDPC-NB** MS	**turbo** MLM
int8	❶ Baseline	cycles	95955	454637	239292	1715625	1585822
int8_t×4	❷ ISA 2R	cycles	61443	373854	215509	1292853	1051247
(32 bits)	❸ ISA 3R	cycles	54027	292678	167095	969990	804471
	Gain (❶→❸)	cycles	41928	161959	72197	745635	781351
	Gain (❶→❸)	(%)	43.7%	35.6%	30.2%	43.5%	49.2%
	Gain (❷→❸)	cycles	7416	81176	48414	322863	246776
	Gain (❷→❸)	(%)	12.0%	21.7%	22.4%	25.0%	23.4%
	Débit ❶	Kbits/s	141	900	1711	7	390
	Débit ❷	Kbits/s	885	4382	7602	39	2356
	Débit ❸	Kbits/s	1006	5598	9805	52	3078
int8_t×8	❹ ISA 2R	cycles	57898	388220	243988	1359186	1207724
(64 bits)	❺ ISA 3R	cycles	52546	335032	213560	1015731	1035964
	Gain (❶→❺)	cycles	43409	119605	25732	699894	549858
	Gain (❶→❺)	(%)	45.2%	26.3%	10.8%	40.8%	34.6%
	Gain (❷→❺)	cycles	5352	53188	30428	343455	171760
	Gain (❷→❺)	(%)	9.2%	13.7%	12.4%	25.7%	14.2%
	Débit ❶	Kbits/s	141	900	1711	7	390
	Débit ❹	Kbits/s	1879	8440	13430	75	4101
	Débit ❺	Kbits/s	2070	9780	15343	100	4781
Accélération							
	❶ → ❸	32 bits	7.1×	6.2×	5.7×	7.1×	7.9×
	❶ → ❺	64 bits	14.6×	10.9×	9.0×	13.5×	12.2×
	❸ → ❺	32 → 64 bits	2.1×	1.7×	1.6×	1.9×	1.6×

Table 1.12 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration SIMD inter-trames avec une fréquence de fonctionnement fixée à 50 MHz.

lors des expérimentations en SISD. Ils varient de 20% à 26% en fonction des familles de code, montrant l'intérêt de ces extensions d'instructions. Des gains moins marqués sont observés pour les décodeurs manipulant des données de type **int8_t×8**. En effet, pour les données **int8_t×8**, GCC introduit fréquemment des instructions d'extension de signe (32 bits → 64 bits) qui sont inutiles dans nos cas d'usage. Ces dernières impactent négativement le temps d'exécution des décodeurs. L'utilisation de nos instructions à 3 opérandes en mode **int8_t×8** permet de supprimer des données temporaires, et donc une partie de ces extensions de signe associées, augmentant de facto le facteur d'accélération.

L'ajout des extensions 32 bits et 64 bits apporte des gains notables en termes d'accélération du temps de calcul. Toutefois, l'implantation des opérateurs matériels au sein de l'UAL du cœur CVA6 engendre aussi une augmentation de la complexité matérielle.

La figure 1.3 présente le surcoût effectif de ces extensions (**3R**) lorsqu'elles sont insérées dans le cœur CVA6. Il est à noter que, comme précédemment, l'ajout de ces extensions en matériel n'a pas eu d'influence sur la fréquence de fonctionnement maximale du cœur après placement et routage par l'outil Vivado.

Si l'on s'intéresse dans un premier temps aux extensions proposées pour les décodeurs LDPC, on observe que les surcoûts matériels sont équivalents (5% supérieur) en mode **2R** et **3R**. De fait, même si plusieurs instructions **2R** ont été fusionnées pour produire des instructions **3R**, une instruction **2R** supplémentaire a été ajoutée. Le surcoût matériel pour les codes polaires est quant à lui plus faible (4%) que celui de l'extension **2R**. L'implantation des codes polaire avec l'algorithme FAST-SC mobilise 3 instructions supplémentaires en mode **2R** et **3R** afin d'accélérer les nœuds de l'arbre élagué. Cette différence permet un début d'explication pour la différence de coût $\approx 25\%$ observée en comparaison avec les codes polaires simples. Toutefois, il y a aussi une différence entre les 2 modes précédemment observés. Pour les codes LDPC-NB et les turbo codes, l'ajout de 2 et 4 instructions respectivement dans l'extension **3R** ont engendré une augmentation respective de $\approx 2\%$ et $\approx 77\%$ du coût en LUT.

Ces coûts matériels, qui peuvent sembler importants, sont dans un premier temps à mettre en perspective par rapport au coût matériel du cœur CVA6 (tableau 1.2) qui nécessite 34576 LUT dans sa configuration 64 bits. Ainsi, l'augmentation de la complexité du cœur est comprise entre 1% et 3% en fonction de l'extension considérée. Ces surcoûts peuvent être considérés comme marginaux lorsqu'on les compare aux facteurs d'accélération de $9\times$ à $14\times$ obtenus (tableau 1.12).

1.3.3 Évaluation des extensions SIMD intra-trame

Cette dernière section est consacrée à l'étude des extensions proposées pour le second schéma de parallélisation SIMD. Une partie des instructions est commune avec celles mises en œuvre pour la parallélisation inter-trames. Toutefois, d'autres instructions impliquant des réductions horizontales ont été nécessaires. L'objectif de ce schéma est de réduire la latence de traitement des décodeurs. Pour y parvenir, il est nécessaire d'exploiter un parallélisme calculatoire variable et/ou plus faible car les descriptions algorithmiques, malgré le fait qu'elles soient plus complexes, sont moins favorables.

Les performances temporelles des décodeurs ainsi optimisés à l'aide des extensions présentées dans ce chapitre sont exposées dans le tableau 1.13, à l'exception des décodeurs LDPC qui à cause de leur nécessité de réaliser des accès non alignés en mémoire n'ont pu

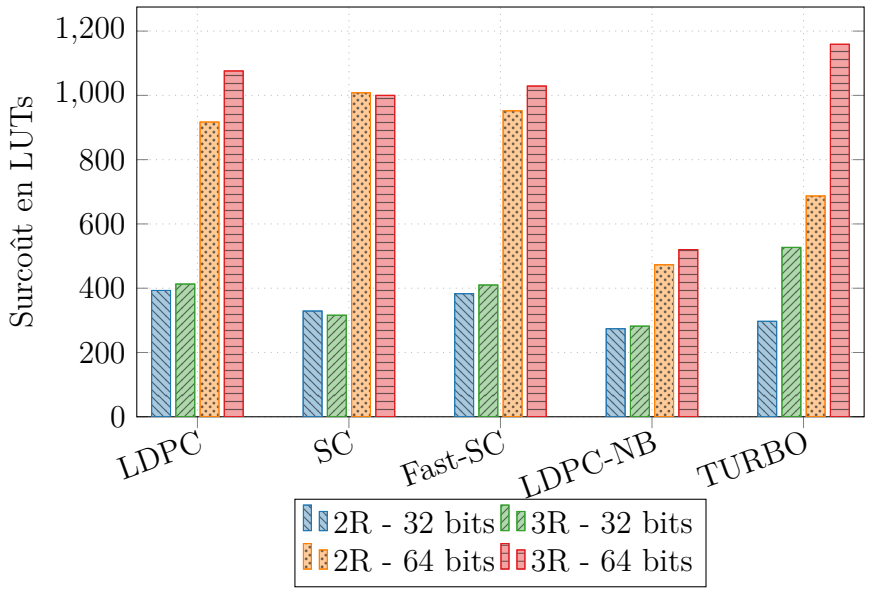

Figure 1.3 – Surcoût matériel induit par les extensions nécessaires à la parallélisation inter-trames en configuration 32 bits et 64 bits.

être évalués.

Les données issues du tableau 1.13 comparent les performances obtenues par les nouvelles extensions à **3R** par rapport à celles du chapitre précédent. Si l'on s'intéresse dans un premier temps au décodage des codes polaires et plus précisément au décodeur SC, on remarque que le schéma de parallélisation SIMD en 32 bits manipulant 4 données en parallèle permet de réduire de 76% le temps d'exécution par rapport à l'approche naïve (❶, Baseline). Par rapport à la version **2R**, l'utilisation de **3R** apporte un gain supplémentaire de 23%. Ces ordres de grandeur se retrouvent pour les expérimentations réalisées pour l'algorithme F-SC.

Les gains obtenus pour le décodage des turbo codes sont plus conséquents. Par rapport à la version naïve, le temps d'exécution de la version **3R** est réduit de 90%. Les nouvelles extensions mises au point grâce à la 3$^{\text{ème}}$ opérande ont permis de réduire de 45% le temps d'exécution de la version **2R**.

Les gains observés pour les décodeurs LDPC-NB atteignent 27% et 27.1% lorsque comparés au décodeur de base. Théoriquement, l'utilisation de 8 données concurrentes en parallèle (**int8_t×8**) devrait améliorer significativement la latence de décodage par rapport à 4 données (**int8_t×4**), contrairement à ce qui est observé. Ces résultats sont principalement liés à l'inadéquation de la stratégie de parallélisation vis-à-vis des opérations mises en oeuvre dans l'algorithme de décodage, et du format des données par

intra-trame				**polaire**	**LDPC-NB**	**turbo**
			SC	F-SC	MS	MLM
int8	❶ Baseline	cycles	454637	239292	1715625	1585822
int8_t×4	❷ ISA 2R	cycles	139961	55031	1270968	269167
(32 bits)	❸ ISA 3R	cycles	108701	42684	927114	148194
	Gain (❶→❸)	cycles	345936	196608	788511	1437628
	Gain (❶→❸)	(%)	76.0%	82.1%	46.0%	90.6%
	Gain (❷→❸)	cycles	31260	12347	343854	120973
	Gain (❷→❸)	(%)	22.3%	22.4%	27.0%	44.9%
	Débit ❶	Kbits/s	900	1711	7	390
	Débit ❷	Kbits/s	2926	7443	10	2300
	Débit ❸	Kbits/s	3768	9596	14	4178
int8_t×8	❹ ISA 2R	cycles	102298	24172	1264971	139404
(64 bits)	❺ ISA 3R	cycles	80894	20660	921355	67483
	Gain (❶→❺)	cycles	373743	218632	794270	1518339
	Gain (❶→❺)	(%)	82.2%	91.3%	46.3%	95.7%
	Gain (❹→❺)	cycles	21404	3512	343616	71921
	Gain (❹→❺)	(%)	20.9%	14.3%	27.1%	51.5%
	Débit ❶	Kbits/s	900	1711	7	390
	Débit ❹	Kbits/s	4003	16945	10	4441
	Débit ❺	Kbits/s	5063	19825	14	8175
Accélération						
	❶ → ❸	32 bits	4.2×	5.6×	1.9×	10.7×
	❶ → ❺	64 bits	5.6×	11.6×	1.9×	23.7×
	❹ → ❺	32 → 64 bits	1.3×	2.1×	1×	2.2×

Table 1.13 – Comparaison des performances des cœurs usant d'instructions à 2 et 3 entrées, configuration SIMD intra-trame avec une fréquence de fonctionnement fixée à 50 MHz.

rapport à la largeur de l'unité SIMD. En effet, le format $\mathbb{GF}$=16 nécessiterait des instructions SIMD sur 16×8 bits pour accélérer les multiplications et les divisions dans $\mathbb{GF}$.

Les facteurs d'accélération entre les versions originales des décodeurs et des implantations (❶ → ❸), présentés dans le tableau 1.13 soulignent ici aussi l'intérêt des extensions proposées.

Lorsque l'on s'intéresse à l'extension de la taille des données manipulées par nos extensions SIMD de 32 bits à 64 bits on observe une accélération variant d'un facteur 1× → 2×. Des facteurs d'accélération de ≈ 2× sont obtenus pour certains codes, tandis que pour d'autres les accélérations sont moins favorables, voir bien inférieures à celles obtenues dans le cas de la parallélisation inter-trames. Cela s'explique par un parallélisme de calcul plus difficilement exploitable en mode intra-trame. Par exemple, concernant le décodage

SC des codes polaires, le parallélisme de calcul varie lors du décodage. Ainsi, au niveau des feuilles de l'arbre, seules des opérations scalaires sont réalisées. En effet, durant une partie du processus de décodage, le doublement des unités de calculs est sans conséquence, limitant de fait l'accélération globale. Le facteur d'accélération du décodeur LDPC-NB est bien inférieur aux attentes. En effet, les accès non réguliers à la mémoire lors des calculs des CNs induisent des pénalités qui annulent les gains liés à nos extensions.

L'impact des extensions proposées pour l'accélération des décodeurs intra-trame sur la complexité matérielle a aussi été évaluée. La figure 1.4 présente le surcoût des extensions en termes de LUT issues des rapports post-placement et routage. Contrairement aux expérimentations réalisées en mode inter-trames, on peut observer une plus grande disparité entre les surcoûts des extensions **2R** et **3R**. Cela est lié au fait que la possibilité d'utiliser 3 opérandes a permis de concevoir plus d'instructions spécialisées. C'est particulièrement le cas, par exemple, pour les turbo codes où l'utilisation d'une troisième opérande a permis de créer 4 instructions spécialisées supplémentaires. Cela explique le doublement du coût de l'extension **3R** par rapport à son pendant **2R**. Toutefois, même si cette observation devrait également se vérifier en mode 64 bits, l'outil Vivado réalise des optimisations durant les étapes post-placement et routage qui tempèrent cette tendance. Un phénomène équivalent est observé pour l'extension dédiée aux code LDPC-NB où l'extension **2R** est plus onéreuse que la version **3R** en configuration 32 bits. Enfin, si l'on compare le surcoût des extensions dédiées au mode intra-trame, par rapport à celles pour le mode inter-trames, on note qu'il est du même ordre de grandeur, à l'exception des instructions pour les turbo codes ; ces dernières sont environ 20% plus onéreuses. Comme nous l'avons précédemment évoqué, ces coûts matériels qui peuvent sembler importants ne représentent que de 1% à 3% de la complexité du CVA6.

1.4 Conclusion

Dans ce chapitre, nous avons présenté de nouvelles extensions permettant d'accélérer les algorithmes de décodage des CCE. Contrairement aux précédentes extensions, ces dernières bénéficient d'un accès possible à un troisième opérande. Cette opportunité a permis de proposer de nouvelles instructions en vue de minimiser le temps d'exécution des algorithmes. Les différentes extensions ont été déclinées sous plusieurs formes afin de permettre d'évaluer à la fois un traitement scalaire, et SIMD, des données. De plus, grâce à l'utilisation d'un cœur RISC-V plus complexe possédant un chemin de données 64 bits,

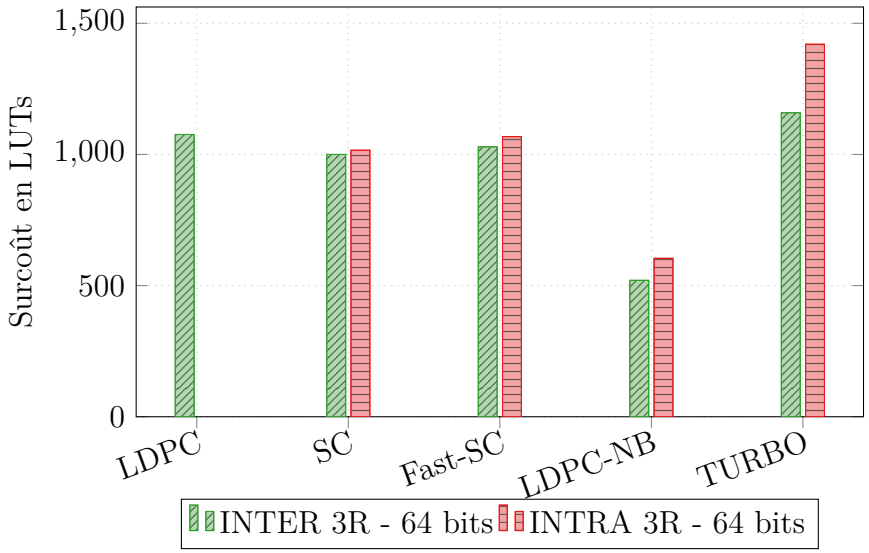

Figure 1.4 – Surcoût matériel induit par les extensions nécessaires à la parallélisation intra-trame en configuration 32 bits et 64 bits.

nous avons pu étudier les options de passage à l'échelle des extensions proposées.

Le tableau 1.14 propose une synthèse des facteurs d'accélération qui ont été mesurés pour les schémas de parallélisation SIMD. Les facteurs d'accélérations qui sont rapportés ont été calculés par rapports aux codes logiciels n'exploitant que le jeu d'instructions de base des cœurs RISC-V. Comme le met en évidence ce tableau, l'ensemble des extensions proposées offre des facteurs d'accélération élevés tandis que le nombre d'instructions ajouté à l'ISA est relativement faible. Par exemple, pour les codes polaires, le passage de 3 opérandes d'entrées à permis de concevoir 1 instruction qui permet d'atteindre une accélération avoisinant 10.7× (64 bits, mode inter-trames). Des facteurs d'accélération plus important ont été obtenus pour le décodage des turbo codes (23.5×) au prix d'un nombre d'instructions spécialisées supérieur (11 instructions en mode 64 bits pour une parallélisation intra-trame). L'utilisation d'un troisième opérande d'entrée pour les opérations spécialisées a permis, en moyenne, d'augmenter le facteur d'accélération de 1.32× (32%) par rapport aux extensions **2R**. Le choix de la stratégie de parallélisation impacte fortement sur les niveaux de performance atteignables. En effet, alors que la stratégie intra-trame permet d'atteindre un facteur d'accélération de 25.5× pour les turbo codes, elle ne permet d'obtenir que 1.7× pour le décodage des codes LDPC-NB où l'approche inter-trames offre 13.5×.

D'un point de vue matériel, les extensions **3R** ont des complexités matérielles comparables aux extensions **2R**. En effet, le coût lié à l'ajout de nouvelles instructions est

CCE		**2R**	**3R**	**2R**	**3R**
SIMD		**int8_t×4**	**int8_t×4**	**int8_t×8**	**int8_t×8**
Bits		**32**	**32**	**64**	**64**
LDPC	inter-trames	885 Kbps	1006 Kbps	1879 Kbps	2070 Kbps
OMS		(**6.2**×)	(7.1×)	(**13.2**×)	(14.6×)
polaire	inter-trames	4382 Kbps	5506 Kbps	8440 Kbps	9633 Kbps
SC		(4.9×)	(6.2×)	(9.4×)	(10.8×)
	intra-trame	2926 Kbps	3768 Kbps	4003 Kbps	5063 Kbps
		(3.2×)	(4.2×)	(4.4×)	(5.6×)
polaire	inter-trames	7602 Kbps	9625 Kbps	13430 Kbps	15212 Kbps
Fast-SC		(4.4×)	(5.7×)	(7.8×)	(9×)
	intra-trame	7443 Kbps	9596 Kbps	16495 Kbps	19825 Kbps
		(4.3×)	(5.6×)	(9.9×)	(11.6×)
LDPC NB	inter-trames	39 Kbps	50 Kbps	75 Kbps	94 Kbps
MS		(5.3×)	(7.0×)	(10.1×)	(13.5×)
	intra-trame	10 Kbps	13 Kbps	10 Kbps	14 Kbps
		(1.3×)	(1.8×)	(1.3×)	(1.8×)
TURBO	inter-trames	2356 Kbps	3078 Kbps	4101 Kbps	4781 Kbps
MLM		(6.0×)	(7.9×)	(10.5×)	(12.2×)
	intra-trame	2300 Kbps	4178 Kbps	4441 Kbps	8175 Kbps
		(5.9×)	(**10.7**×)	(11.4×)	(**23.5**×)

Table 1.14 – Accélération observé des débits par rapport à la Baseline.

en partie compensée par les gains réalisés par la fusion d'autres instructions. La fusion d'instructions engendrant par exemple une réduction du coût du multiplexeur sélectionnant les résultats en sortie de l'UAL. Les résultats basés sur l'utilisation du cœur CVA6, obtenus post-placement routage, mettent en évidence le faible impact des extensions sur le coût global du cœur. En effet, ces dernières induisent une augmentation limitée de 1% à 3% du coût du cœur en termes de LUT. Enfin, pour toutes les expérimentations que nous avons réalisées, aucune pénalité concernant la fréquence maximale de fonctionnement du cœur n'a été observée.