Exploration efficace d'espace de conception pour la synthèse de haut niveau dynamique et spéculative

Dylan Leothaud, Jean-Michel Gorius, Simon Rokicki, Simon Rokicki, Steven Derrien

Compas'2024 : Parallélisme/ Architecture / Système
IETR/LS2N/Nantes-U - Nantes - France, 2-5 juillet 2024

Exploration efficace d'espace de conception pour la synthèse de haut niveau dynamique et spéculative

Dylan Leothaud, Jean-Michel Gorius, Simon Rokicki, Steven Derrien*

Université de Rennes,
Laboratoire IRISA - 263 Avenue Général Leclerc
35000 Rennes - France
prenom.nom@irisa.fr

Résumé
Les outils de synthèse de haut niveau (HLS, ou *High-Level Synthesis*) permettent de générer des circuits à partir de programmes C/C++ sans passer par une spécification via des langages de type HDL. Ces outils sont très efficaces pour des programmes ayant un flot de contrôle régulier, mais rencontrent des difficultés à extraire du parallélisme lorsque le flot de contrôle devient trop complexe. L'utilisation de mécanismes d'exécution dynamique et spéculative permet de lever cette limitation, mais au prix d'un surcoût en surface important. Déterminer précisément où, quand, et comment utiliser ces mécanismes est donc un enjeu crucial pour ce type d'approche. Dans cet article, nous proposons une technique d'exploration de cet espace de conception capable de trouver des solutions efficaces dans des espaces de plusieurs centaines de milliers de solutions.

Mots-clés : Synthèse de haut niveau ; Ordonnancement dynamique ; Spéculation ; Exploration d'espace de conception

1. Introduction

La synthèse de haut niveau (HLS, ou *High-Level Synthesis*) permet de produire une description de circuit directement à partir d'un programme décrivant son comportement. Ce type de technique est de plus en plus utilisé, en particulier pour la conception d'accélérateurs sur FPGA. Les outils actuels sont capables de produire des circuits très performants pour des noyaux de calculs réguliers (boucles imbriquées, etc.) mais peinent à traiter efficacement des noyaux où le flot de contrôle est plus complexe. Ces limitations découlent de l'utilisation d'un ordonnancement statique des opérations, qui raisonne en pire cas en ne se basant que sur les informations disponibles à la compilation.
Il est possible de lever ces limitations en exploitant des ordonnancements dynamiques [1, 7, 8, 15, 16, 4, 18] voire spéculatifs [9, 14, 12, 16] qui permettent d'améliorer significativement les performances des circuits produits. Ces gains en performance peuvent cependant se traduire par un surcoût en surface inutilement élevé. Ce dernier est dû à un recours systématique à ces approches, y compris là où elles n'apportent que peu de gains. Déterminer quand, où et comment tirer parti de ces approches est donc un problème crucial, d'autant plus difficile à

*. Univ Rennes, Inria, CNRS, IRISA
Ce travail est financé par le projet ANR LOTR (ANR-23-CE25-0016)

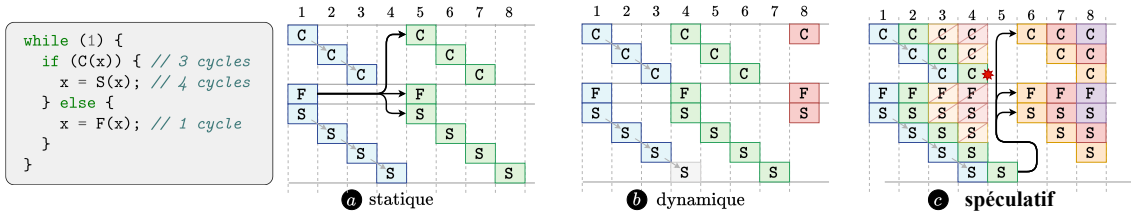

FIGURE 1 – Une vue d'ensemble des techniques d'ordonnancement sans contrainte de ressource utilisées pour la synthèse de haut niveau.

traiter que l'espace des solutions devient rapidement gigantesque, et expose des compromis complexes entre surface et performance.
Notre approche aborde cette étape d'exploration via une combinaison de techniques d'analyse statique et de profilage. Elle repose sur une représentation du programme mettant en évidence les opportunités d'ordonnancement dynamique et spéculatif [13, 9, 12]. Plus précisément, nos contributions sont les suivantes :

— Un algorithme d'exploration d'espace de conception, permettant une exploration rapide grâce à des heuristiques de recherche et des techniques d'élagage efficace.
— Une évaluation expérimentale de notre approche sur différents exemples, accompagnée d'une discussion qualitative et quantitative des résultats obtenus.

La structure de cet article est le suivant : la Section 2 offre un aperçu général de l'ordonnancement dynamique et spéculatif. La Section 3 détaille notre approche, tandis que la Section 4 évalue expérimentalement notre méthode et la compare aux travaux existants.

2. Contexte

Cette section rappelle certains concepts liés à la HLS, plus particulièrement les techniques d'ordonnancement d'opérations. Elle donne aussi une vue d'ensemble du flot de compilation SpecHLS et de sa représentation interne.

2.1. Ordonnancement en synthèse de haut niveau

L'efficacité des circuits obtenus via la HLS repose sur la capacité de l'outil à extraire du parallélisme, par exemple via du pipeline de boucle. Cette optimisation permet de recouvrir l'exécution de plusieurs itérations, permettant de dériver des chemins de données pipelinés. Un ordonnancement pipeliné se caractérise par sa cadence (II, ou *Initiation Interval*), qui correspond au nombre de cycles séparant le début de deux itérations. Cette cadence est contrainte par la profondeur des dépendances de calcul entre itérations. La valeur minimale de II pour une boucle est appelée *RecMII*, et peut-être calculée via un algorithme de complexité polynomiale en le nombre d'opérations de la boucle [17].
La valeur obtenue est cependant une valeur pire cas, qui peut se révéler très pessimiste pour des boucles avec un flot de contrôle complexe ou des accès mémoires avec des motifs d'accès irréguliers. Cela est illustré par le programme de la Figure 1, dans lequel l'exécution de l'itération suivante dépend du résultat des calculs de l'itération courante. Dans ce cas, l'ordonnanceur statique considère le pire cas possible, ce qui conduit à un II de 4, comme représenté dans la partie ⓐ de la Figure 1.
Les ordonnancements dynamique et spéculatif permettent quant à eux la conception de circuits plus efficaces. Une exécution dynamique s'adapte au comportement réel du circuit, comme illustré dans la partie ⓑ de la Figure 1. Une exécution spéculative fait une hypothèse sur le ré-

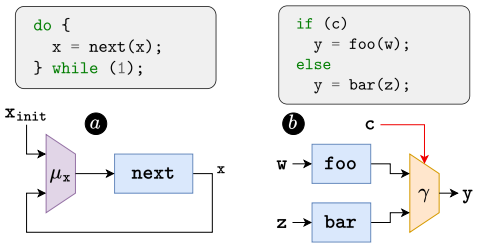

FIGURE 2 – Opérateurs de la représentation Gated-SSA. Les nœuds μ ⓐ représentent les têtes de boucle, les nœuds γ ⓑ représentent les jonctions de flot de contrôle.

sultat du calcul de l'itération actuelle afin de débuter la suivante. Si l'hypothèse s'avère exacte, l'exécution spéculative aura alors permis de démarrer les itérations suivantes en avance. En revanche, lorsque l'hypothèse n'est pas vérifiée, il devient nécessaire d'annuler les itérations qui ne devraient pas encore avoir débuté avant de redémarrer l'exécution spéculative. Une trace d'exécution spéculative est représentée sur la partie ⓒ de la Figure 1. Plusieurs travaux ont proposé d'intégrer ces mécanismes d'exécution dans des outils de HLS.

Certains outils génèrent des circuits dynamiques à partir d'une approche basée sur les flots de données [15], où chaque composant du circuit est dynamique et le flot de contrôle est représenté par des échanges de jetons. Cette approche induit un surcoût important en utilisation de ressources [5]. Pour minimiser ce surcoût il est possible de retirer le caractère dynamique du circuit lorsque cela n'améliore pas les performances [2, 3, 5].

D'autres approches consistent à transformer le programme en amont d'un outil de HLS classique afin d'intégrer une exécution dynamique ou spéculative au pipeline de boucle statique [9, 12]. SpecHLS [12] est un compilateur source-à-source qui utilise cet méthode. Il permet de concevoir un circuit dynamique ou spéculatif à partir d'un programme C/C++ et d'un outil de HLS commercial.

2.2. Énoncé du problème

SpecHLS utilise comme représentation intermédiaire des programmes une variante de la représentation Gated-SSA (ou *GSSA*) [19, 20]. La représentation GSSA est basée sur la représentation SSA, et spécialise des nœuds φ, soit en nœuds μ pour les têtes de boucles, soit en nœuds γ à la fin des structures conditionnelles. Les jonctions γ encodent également la condition du choix de l'origine de la valeur. Deux exemples de représentation GSSA et leurs programmes associés sont illustrés sur la Figure 2.

Dans la représentation GSSA d'un programme, un nœud γ est une opportunité de spéculation [9]. SpecHLS permet de concevoir des circuits spéculant sur un ensemble d'hypothèses $(\gamma_{i,j})$ selon laquelle γ_i prendrait toujours l'entrée j. Par la suite on appel cet ensemble d'hypothèses une configuration de γ. Il est courant que les programmes mènent à plusieurs centaines de milliers de configuration de γ. On ne peut donc pas tous les parcourir pour trouver la meilleure configuration. L'enjeu est alors de choisir efficacement une configuration de γ afin de déterminer où et comment spéculer dans un circuit.

3. Approche proposée

Cette Section décrit l'algorithme d'exploration d'espace de conception utilisé ainsi que les heuristiques de recherche et les optimisations d'élagage de cet espace. Pour déterminer *où* et *comment* spéculer dans un circuit, nous proposons un algorithme d'exploration d'espace de conception utilisant une combinaison d'analyse statique et d'heuristiques de recherche. Notre

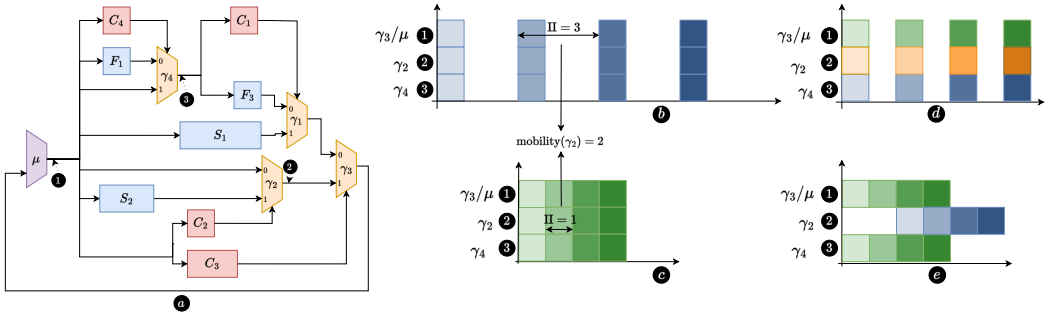

FIGURE 3 – Exemple de circuit pour notre exploration. La partie ⓐ représente la forme Gated-SSA du circuit. La partie ⓑ représente une trace d'exécution du circuit déroulé quatre fois. La partie ⓒ donne une trace d'exécution où chaque γ est contrôlé par un oracle qui choisit la première valeur disponible. Dans la partie ⓓ, γ_1 et γ_4 sont contrôlés par des oracles et γ_2 choisit toujours la première entrée. La partie ⓔ représente une trace déroulée où γ_1, γ_3 et γ_4 sont contrôlés par des oracles.

approche consiste à étendre progressivement une configuration de γ avec des hypothèses de spéculation supplémentaires, permettant ainsi de construire un arbre d'exploration dont on peut élaguer certaines branches. Parmi toutes les configurations de γ existantes, nous souhaitons trouver l'ensemble des configurations respectant les propriétés suivantes : i) le RecMII atteint est inférieur ou égale à un II cible (généralement II $= 1$), ii) la probabilité d'apparition de la configuration est supérieure à un seuil défini par l'utilisateur, et iii) la configuration est dite minimale, c'est-à-dire, retirer une hypothèse sur l'entrée choisie d'un γ fait passer le RecMII au-dessus du II cible.
Pour un circuit avec n nœuds γ binaires, le nombre de configurations possible est de 3^n. En effet, pour chaque nœud γ, il est possible de spéculer soit sur une de ses deux entrées, soit de ne pas spéculer. Le nombre de configurations possibles étant très important, il n'est généralement pas possible de toutes les explorer. Nous devons alors être capables de réduire la taille de cet arbre d'exploration autant que possible.

Probabilité d'apparition d'une configuration
Dans un circuit spéculatif, une mauvaise prédiction ralentit l'exécution du circuit. Une probabilité faible de mauvaise prédiction est importante pour garantir l'efficacité du circuit. Pour cela, nous exécutons le programme en amont afin d'obtenir des informations de profilage sur les entrées choisies par les nœuds γ, et pouvoir calculer la probabilité d'apparition d'une configuration de γ donnée. Ajouter une hypothèse à une configuration de γ la contraint davantage et diminue sa probabilité d'apparition. Cette propriété nous permet donc d'élaguer notre arbre d'exploration lorsque la probabilité d'apparition est inférieure au seuil.

Ordonnancement du circuit déroulé
Afin de construire un arbre d'exploration de taille raisonnable, nous proposons des heuristiques reposant sur un ordonnancement du circuit déroulé. Cette technique consiste à construire un ordonnancement pour circuit acyclique, équivalent au circuit initial dupliqué où les dépendances inter-itérations sont remplacées par des dépendances à la copie précédente du circuit. Le nombre de copies doit être suffisant pour permettre de simuler le régime permanent du circuit. Lors de cet ordonnancement, certains nœuds γ ne dépendent plus que d'une seule de leur entrée, pour simuler une exécution spéculative. D'autres nœuds γ sont contrôlés par un *oracle*, et démarrent leur exécution dès qu'une de leurs entrées sont disponibles. Des exemples d'or-

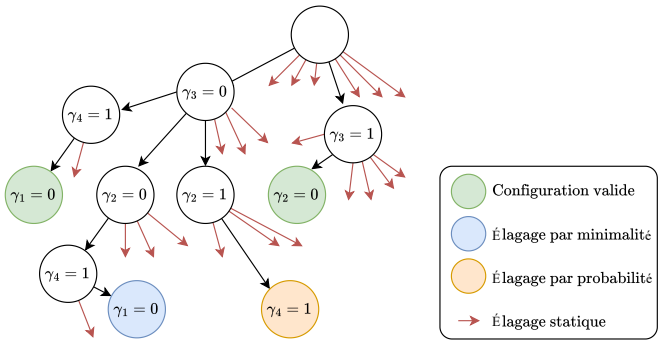

FIGURE 4 – Arbre d'exploration pour l'exemple de la Figure 3, avec, par ordre croissant de mobilité, γ_3, γ_2, γ_4, et γ_1.

donnancements déroulés sont représentés sur la Figure 3. Cet ordonnancement déroulé nous permet de construire une heuristique et une optimisation :

Mobilité des γ. Lors d'un ordonnancement déroulé, le nombre de cycles entre deux exécutions successives d'un nœud γ est une métrique du nombre maximum de cycles d'exécution pouvant être gagnés en cas de spéculation. On appelle ce nombre de cycles la mobilité du nœud γ. Choisir en priorité les nœuds γ ayant une grande mobilité comme cible de spéculation aura donc plus de chances de diminuer le RecMII. On trie alors les nœuds γ par mobilité décroissante.

Élagage lorsque la cible est inatteignable. Dans un ordonnancement déroulé, un nœud γ contrôlé par un oracle commence au moins aussi tôt que s'il était fixé à une de ses entrées. Dans une configuration de γ donnée, lorsque certains nœuds γ sont spéculé et d'autres ne le sont pas, il est possible de générer un ordonnancement dans lequel les nœuds γ spéculés sont fixés à leur valeur spéculée, tandis que les non spéculés sont contrôlés par des oracles. Dans cet ordonnancement déroulé, si un nœud μ, a un nombre de cycles entre deux itérations supérieur à la cible de II, il sera impossible d'atteindre cette cible en étendant cette configuration. On peut alors arrêter l'exploration de cette branche.

Un extrait d'arbre d'exploration est représenté sur la Figure 4.

4. Validation expérimentale et discussion

Cette Section évalue notre approche pour la taille des espaces d'exploration et le temps nécessaire à l'exploration. Nous donnons également les résultats de la HLS pour nos exemples afin de montrer que notre approche permet de trouver des solutions correctes malgré l'exploration partielle de l'espace de conception.
Notre approche a été intégrée à SpecHLS, un flot de compilation basé sur l'infrastructure GeCoS [10] et dépend de l'ordonnanceur du projet CIRCT [6]. L'outil de synthèse de haut niveau utilisé en face arrière du flot est Vitis HLS 2021.1, le FPGA ciblé est le XC7A200 et nous visons une fréquence de 100MHz. Notre II cible est de 1 et le seuil de probabilité pour l'élagage est de 10%. Nous avons choisi un ensemble d'exemples provenant de travaux précédents [1, 12, 3, 11] pour lesquels le choix d'une configuration de nœuds γ menant à $\text{II} = 1$ est difficile. Nos exemples sont :

— **RISC-V** : Un processeur dont le jeu d'instruction est RV32I.
— **Superscalar** : Une version du processeur RISC-V dont le corps de boucle a été déroulé deux fois, permettant ainsi d'exécuter deux instructions par itération.

TABLE 1 – Utilisation de ressources et taille des espaces d'exploration

Benchmark	Type	Surface					Taille de l'espace				Temps
		II	T_{clk} (ns)	LUT	FF	DSP	Nœuds γ	Référence	Heuristique	SOTA [18]	d'exploration
FPU	Référence	3	7.90	353	283	2	21	30.9B	116k	10.9k	1055s
	Spéculatif	1	17.6	2 885	1 105	2					
SpMM	Référence	6	6.27	256	442	4	12	708k	60	1.35k	6s
	Spéculatif	2	6.71	1 097	1 377	4					
RISC-V CPU	Référence	5	8.891	1 499	361	0	16	752M	1.46k	11.3M	263s
	Spéculatif	1	14.35	1630	953	0					
Superscalar	Référence	4	7.16	3502	2688	0	16	178M	1.19k	5.74M	177s
	Spéculatif	2	10.451	6865	5682	0					

— **FPU** : Une FPU permettant des opérations d'addition et de multiplication. Ces opérations sont plus efficaces lorsque les opérandes ont le même exposant.
— **SpMM** : Une multiplication de matrices creuses.

La Table 1 présente nos résultats. Elle montre que notre approche réduit considérablement la taille de l'espace de conception de jusqu'à six ordres de grandeur. En comparaison, la colonne *SOTA* représente la taille de l'espace exploré suivant l'approche de Szafarczyk et *al.* [18]. Les résultats de la HLS, montrent que le II après spéculation est inférieur à celui sans spéculation. En revanche, on remarquera que deux exemples n'atteignent pas $II = 1$, et que la fréquence des autres exemples est inférieure à 100MHz, principalement en raison d'une mauvaise estimation des latences des opérations dans notre outil. Une réévaluation de ces latences est nécessaire pour obtenir un ordonnanceur plus proche de la vérité terrain.

Szafarczyk et *al.* [18] proposent une analyse de compilateur sur le graphe de flot de contrôle (GFC) permettant d'introduire du dynamisme dans un programme pour la HLS. Leur analyse repose sur un algorithme glouton itérant sur l'ensemble des blocs de base de chaque cycle élémentaire du GFC. Contrairement à notre approche, les auteurs ne proposent pas de concevoir des circuits spéculatifs, et le nombre de blocs de base sur l'ensemble des cycles élémentaires peut augmente significativement pour des exemples complexes.

Dynamatic [15, 14] est un outil permettant la conception de circuits dynamiques et spéculatifs. Cet outil délègue la gestion de la spéculation mémoire et la résolution des dépendances à une *load-store queue* externe. Notre approche traite la spéculation mémoire de la même façon que les autres spéculations et l'intègre dans notre exploration. De plus, nos travaux permettent une découverte plus fine des opportunités de spéculation, menant donc à un espace de conception plus grand, qui nécessite un traitement adapté.

5. Conclusion

L'introduction d'un support pour la conception de circuits spéculatifs à partir d'outils de synthèse de haut niveau permet d'obtenir automatiquement tout une gamme d'accélérateurs matériels efficaces. L'exécution spéculative permet d'étendre les capacités des outils de HLS aux programmes ayant un flot de contrôle complexe. Cependant, déterminer où et comment appliquer cette spéculation met en lumière de nouveaux problèmes d'exploration d'espaces de conception. Cet article propose une approche d'exploration qui intègre des analyses statiques et du profilage pour explorer rapidement de vastes espaces, sans dépendre de simulations RTL, souvent lentes. Notre approche permet notamment d'explorer rapidement l'espace de conception nécessaire à la génération d'un cœur RISC-V superscalaire à exécution dans l'ordre à partir d'une description algorithmique dudit comportement du processeur.

Bibliographie

1. Alle (M.), Morvan (A.) et Derrien (S.). – Runtime Dependency Analysis for Loop Pipelining in High-Level Synthesis. – In *Proceedings of the 50th Annual Design Automation Conference*, *DAC '13*, DAC '13, New York, NY, USA, 2013. Association for Computing Machinery.
2. Cheng (J.), Josipovic (L.), Constantinides (G. A.), Ienne (P.) et Wickerson (J.). – Combining Dynamic & Static Scheduling in High-level Synthesis. – In *Proceedings of the 2020 ACM/-SIGDA International Symposium on Field-Programmable Gate Arrays*, *FPGA '20*, FPGA '20, p. 288–298, New York, NY, USA, 2020. Association for Computing Machinery.
3. Cheng (J.), Josipović (L.), Constantinides (G. A.), Ienne (P.) et Wickerson (J.). – DASS : Combining Dynamic & Static Scheduling in High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, n3, 2022, pp. 628–641.
4. Cheng (J.), Wickerson (J.) et Constantinides (G. A.). – Probabilistic Scheduling in High-Level Synthesis. – In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 195–203, 2021.
5. Cheng (J.), Wickerson (J.) et Constantinides (G. A.). – Finding and finessing static islands in dynamically scheduled circuits. – In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 89–100, 2022.
6. CIRCT : Circuit IR Compilers and Tools. – `https://github.com/llvm/circt`.
7. Cong (J.), Lau (J.), Liu (G.), Neuendorffer (S.), Pan (P.), Vissers (K.) et Zhang (Z.). – FPGA HLS Today : Successes, Challenges, and Opportunities. *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, n4, août 2022, pp. 51 :1–51 :42.
8. Dai (S.), Zhao (R.), Liu (G.), Srinath (S.), Gupta (U.), Batten (C.) et Zhang (Z.). – Dynamic hazard resolution for pipelining irregular loops in high-level synthesis. – In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, *FPGA '17*, FPGA '17, p. 189–194, New York, NY, USA, 2017. Association for Computing Machinery.
9. Derrien (S.), Marty (T.), Rokicki (S.) et Yuki (T.). – Toward Speculative Loop Pipelining for High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, n11, novembre 2020, pp. 4229–4239.
10. Floc'h (A.), Yuki (T.), El-Moussawi (A.), Morvan (A.), Martin (K.), Naullet (M.), Alle (M.), L'Hours (L.), Simon (N.), Derrien (S.), Charot (F.), Wolinski (C.) et Sentieys (O.). – GeCoS : A framework for prototyping custom hardware design flows. – In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 100–105, 2013.
11. Gorius (J.-M.), Rokicki (S.) et Derrien (S.). – Design Exploration of RISC-V Soft-Cores through Speculative High-Level Synthesis. – In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pp. 1–6, 2022.
12. Gorius (J.-M.), Rokicki (S.) et Derrien (S.). – SpecHLS : Speculative Accelerator Design Using High-Level Synthesis. *IEEE Micro*, vol. 42, n5, 2022, pp. 99–107.
13. Gorius (J.-M.), Rokicki (S.) et Derrien (S.). – A Unified Memory Dependency Framework for Speculative High-Level Synthesis. – In *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction*, *CC 2024*, CC 2024, p. 13–25, New York, NY, USA, 2024. Association for Computing Machinery.
14. Josipović (L.), Guerrieri (A.) et Ienne (P.). – Speculative Dataflow Circuits. – In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, *FPGA '19*, FPGA '19, p. 162–171, New York, NY, USA, 2019. Association for Computing Machinery.
15. Josipović (L.), Ghosal (R.) et Ienne (P.). – Dynamically Scheduled High-level Synthesis. – In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate*

Arrays, *FPGA '18*, FPGA '18, pp. 127–136, New York, NY, USA, février 2018. Association for Computing Machinery.

16. Lapotre (V.), Coussy (P.), Chavet (C.), Wouafo (H.) et Danilo (R.). – Dynamic branch prediction for high-level synthesis. – In *2013 23rd International Conference on Field programmable Logic and Applications*, pp. 1–6, 2013.
17. Leiserson (C. E.) et Saxe (J. B.). – Retiming synchronous circuitry. *Algorithmica*, vol. 6, n1, 1991, pp. 5–35.
18. Szafarczyk (R.), Nabi (S. W.) et Vanderbauwhede (W.). – Compiler Discovered Dynamic Scheduling of Irregular Code in High-Level Synthesis. – In *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 1–9, Los Alamitos, CA, USA, sep 2023. IEEE Computer Society.
19. Tu (P.) et Padua (D.). – Efficient building and placing of gating functions. – In *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, *PLDI '95*, PLDI '95, p. 47–55, New York, NY, USA, 1995. Association for Computing Machinery.
20. Tu (P.) et Padua (D.). – Gated SSA-Based Demand-Driven Symbolic Analysis for Parallelizing Compilers. – In *Proceedings of the 9th International Conference on Supercomputing*, *ICS '95*, ICS '95, p. 414–423, New York, NY, USA, 1995. Association for Computing Machinery.