Parcours par liste de chemins : une nouvelle classe de mécanismes de suivi de flot SIMT

Sylvain Collange, Nicolas Brunie

Compas'2017 : Parallélisme/ Architecture / Système
Sophia Antipolis, France, du 27 au 30 juin 2017

Parcours par liste de chemins : une nouvelle classe de mécanismes de suivi de flot SIMT

Sylvain Collange, Nicolas Brunie

Inria Centre de recherche Rennes – Bretagne Atlantique
sylvain.collange@inria.fr
Kalray
nicolas.brunie@kalray.eu

Résumé
Le modèle d'exécution SIMT employé dans les GPU synchronise l'exécution de groupes de threads afin d'exécuter leurs instructions communes sur des unités SIMD. Ce modèle nécessite des mécanismes matériels ou logiciels pour gérer la divergence et la reconvergence de contrôle entre threads. Une nouvelle classe de tels algorithmes émerge dans la littérature depuis quelques années. Nous présentons une classification de ces techniques sur la base de leur caractéristique commune, un parcours de graphe à base de liste. Nous comparons le coût de mise en œuvre sur FPGA de deux variantes du processeur Simty, l'une basée sur un tel mécanisme de reconvergence à base de liste triée et l'autre sur un mécanisme d'arbitrage entre compteurs de programme. La liste triée permet un passage à l'échelle significativement meilleur à partir de 8 threads par warp.

Mots-clés : GPU, SIMT, FPGA

---

1. Introduction

Les processeurs graphiques (GPU) sont désormais établis dans le rôle de processeurs parallèles généralistes, notamment pour le calcul scientifique et l'apprentissage automatique. Leur succès repose en grande partie sur leur modèle d'exécution *Single-Instruction Multi-Threading* (SIMT). Suivant ce modèle, le programmeur écrit un code *Single Program, Multiple Data* (SPMD), sous la forme d'un programme exécuté par de nombreux threads. Le matériel exécute ce code en mode *Single Instruction Multiple Data* (SIMD) en groupant les threads en *warps* et en synchronisant les threads de chaque warp pour qu'ils exécutent la même instruction au même moment. Les modèles de programmation des GPU s'unifient progressivement avec les modèles de programmation parallèles pour CPU, que ce soit au niveau langage à travers CUDA C et C++, OpenMP ou au niveau plate-forme avec HSA [13]. Cependant, les jeux d'instructions GPU conservent une différence clé avec les jeux d'instructions « traditionnels ». En effet, ceux-ci doivent communiquer à la microarchitecture la structure explicite du graphe de flot de contrôle. L'unification des jeux d'instructions constitue la prochaine étape à franchir pour rapprocher les modèles de programmation CPU et GPU.
Le problème clé dans le cadre d'un jeu d'instruction unifié consiste à gérer la divergence et la convergence de branchement lorsque les threads d'un warp suivent des chemins différents dans le code du programme, en se basant uniquement sur des instructions de branchements conditionnels ou indirects classiques [2, 5, 6, 7].

Nous présentons Section 2 un état de l'art des méthodes à base de listes qui se sont répandues dans la littérature GPU ces dernières années. Nous décrivons ensuite Section 3 le mécanisme de tables de contextes triées mis en œuvre dans le cœur Simty et comparons le coût de mise en œuvre des tables de contexte avec celui de l'arbitrage entre PC.

2. Gérer la divergence et la convergence

Depuis notre dernier état de l'art à Compas [2, 5], plusieurs nouvelles méthodes de suivi SIMT ont été mises en œuvre ou proposées dans la littérature. Parmi celles-ci, nous présentons ici une taxonomie des méthodes à base de liste.

Nomenclature Un *chemin* est caractérisé par un compteur de programme (PC) et le sous-ensemble des threads d'un warp qui possèdent ce PC. Le contexte d'exécution associé à un warp peut être représenté soit par le vecteur des PC individuels de chaque thread du warp, soit par une liste de chemins disjoints complète. Par disjoint et complet, nous entendons qu'à chaque instant, chaque thread du warp est présent dans un et un seul chemin. Au cours de l'exécution, le nombre de chemins de la liste peut varier entre un et le nombre de threads du warp.

Le problème du suivi de branchements en SIMT se ramène à un parcours du graphe de flot de contrôle. Le processeur suit un des chemins de chaque warp, que nous appelons chemin actif. Les autres chemins sont mémorisés dans une liste. La divergence de contrôle consiste à bifurquer un chemin en deux nouveaux chemins. L'un des deux chemins est sauvegardé dans la liste, tandis que l'exécution se poursuit sur l'autre chemin. La convergence consiste en une fusion du chemin actif avec un chemin de la liste, ou entre deux chemins de la liste ayant le même PC.

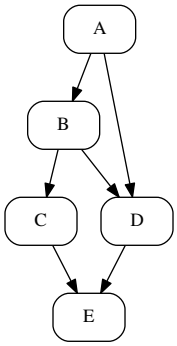
Ordres de parcours Les différentes politiques de gestion de la divergence et reconvergence peuvent être exprimées comme différents ordres de parcours de graphe. La figure 1 compare les différents ordres de parcours obtenus pour un code du type `if(A && B) C; else D; E;`. L'ensemble des threads de chaque chemin est représenté par un vecteur de bits ou *masque* : le bit `i` du vecteur est à 1 si et seulement si le thread `i` est inclus dans l'ensemble. Nous abordons tour à tour chacun de ces ordres de parcours.

2.1. Parcours en profondeur

L'ordre le plus simple consiste à parcourir le graphe en profondeur (Figure 1b). Lorsque le flot de contrôle est structuré, le parcours en profondeur suit les niveaux d'imbrication de structure de contrôle. Ainsi, les convergences ont lieu dans l'ordre inverse des divergences. Le parcours en profondeur revient à considérer la liste des chemins comme une pile.

Le parcours en profondeur laisse le choix de l'ordre de parcours entre les deux chemins de chaque divergence. Nous pouvons adapter à cet effet la politique d'ordonnancement de la technique à base de pile que nous avions proposée à Sympa 2009 [7]. Afin de faciliter la reconvergence, les branchements divergents sont traités différemment suivant leur direction pour donner la priorité au chemin de PC minimal. Pour un branchement divergent vers l'avant (saut vers un PC supérieur), le chemin correspondant à la cible du saut est ajoutée à la pile et l'exécution se poursuit à l'instruction suivante. Pour un branchement divergent vers l'arrière, le chemin de l'instruction suivante est sauvegardé, et c'est à la cible du saut que l'exécution continue. Lors d'un branchement uniforme vers l'avant, le nouveau PC est comparé au PC du chemin du haut de la pile. S'il est supérieur, le haut de la pile est échangé avec le chemin actif (par exemple au passage de C à D Figure 1b). Lorsque le PC est égal à celui du sommet de la pile, ce dernier est dépilé et fusionné avec le chemin actif pour assurer la reconvergence (cas de D).

Bien que cet algorithme emploie une pile d'adresses et de masques comme les techniques à



(a) Graphe de flot de contrôle

A 1111		
B 1011	(D 0100)	
C 1000	(D 0011)	(D 0100)
D 0111	(E 1000)	
E 1111		

(b) Parcours en profondeur

A 1111		
D 0100	(B 1011)	[E 0000]
B 1011	[E 0100]	
C 1000	(D 0011)	[E 0100]
D 0011	[E 1100]	
E 1111		

(c) Parcours par activité minimale

A 1111	
B 1011	(D 0100)
D 0111	(C 1000)
C 1000	(E 0111)
E 1111	

(d) Parcours en largeur

A 1111	
B 1011	(D 0100)
C 1000	(D 0111)
D 0111	(E 1000)
E 1111	

(e) Parcours par PC minimal

A 1111		
B 1011	D 0100	
C 1000	D 0011	(E 0100)
E 1111		

(f) Parcours parallèle

FIGURE 1 – Exemples d'ordres de parcours illustrant l'état de la liste à chaque étape. Le thread 0 parcourt dans l'ordre les blocs ABCE du graphe, le thread 1 ADE et les threads 2 et 3 parcourent ABDE. Les entrées sauvegardées dans la liste sont notées entre parenthèses, et les points de synchronisation entre crochets.

base de pile issues du projet Pixar Chap [16] employés dans certains GPU actuels, l'idée de base est fondamentalement différente : la pile de chemins représente ici une liste de directions futures, plutôt qu'un état global.

2.2. Parcours en profondeur avec minimisation de la profondeur de pile

L'architecture AMD Graphics Core Next 2 traite principalement la divergence en logiciel. Elle offre néanmoins une gestion matérielle de la divergence pour certains cas de flot de contrôle irréductible [1]. Le compilateur délimite une région à une entrée et une sortie au moyen d'une instruction *Fork* à l'entrée et d'une instruction *Join* à la sortie. À l'exécution, les branchements divergents à l'intérieur du graphe sont traités de la façon suivante : (a) le chemin emprunté par la majorité des threads est sauvegardé dans la pile, et (b) l'exécution se poursuit sur le chemin suivi par la minorité. Lorsque l'instruction *Join* est atteinte, l'entrée du dessus de la pile est dépilée et l'exécution se poursuit sur le chemin correspondant. Une fois la pile vide, l'exécution continue après l'instruction *Join* avec le masque initial.

Le choix du chemin emprunté par la minorité garantit qu'à chaque étape, le nombre de threads actifs est au moins divisé par deux. Le nombre d'entrées nécessaires dans la pile est donc borné par $\log_2(n)$ pour n threads par warp, contre n dans le cas d'un ordonnancement générique à base de liste. Cette optimisation est particulièrement importante ici car la pile est réalisée au moyen de registres, lesquels sont alloués de manière statique pour chaque branchement, même ceux qui ne divergent pas à l'exécution. En contrepartie, ce mécanisme parcourt le sous-graphe de flot de contrôle comme un arbre : il ne permet aucune reconvergence partielle avant d'atteindre l'instruction Join en sortie. Dans l'exemple de la figure 1c, le bloc D est ainsi parcouru

deux fois. De fait, cette technique n'offre pas de bénéfice concernant la reconvergence sur des graphes irréductibles, et exclue de facto les boucles non-triviales.

2.3. Parcours en largeur

Le parcours en profondeur du graphe de flot de contrôle peut amener à des interblocages sur des programmes qui opèrent des synchronisations entre threads. En particulier, lorsqu'un thread acquiert un verrou sur un chemin alors qu'un thread sur un autre chemin effectue une boucle d'attente active sur ce verrou, le parcours en profondeur suivra la boucle d'attente indéfiniment sans libérer le verrou. Un parcours en largeur (figure 1d) assure une équité entre les chemins et permet d'éviter un tel interblocage [9]. Il revient à considérer la liste des chemins comme une file. Néanmoins, le parcours en largeur nécessite toujours de disposer de points de reconvergence « sûrs ».
Un brevet qui pourrait correspondre à l'architecture NVIDIA Volta propose un ordre de parcours hybride [8]. Il parcourt en profondeur les chemins correspondant à du flot de contrôle structuré, puis en largeur les chemins non-structurés.

2.4. Parcours par ordre de priorité

Le parcours par priorité consiste à trier la liste des chemins selon un ordre basé sur son contenu. L'architecture DWF réalise une liste de chemins dont les threads sont identifiés par leur numéro [11]. Cette représentation généralise les masques en permettant à un chemin d'inclure des threads de plusieurs warps [1]. L'article décrivant DWF considère plusieurs politiques de priorité, dont la majorité des threads, la minorité et le PC minimal.
L'architecture Maven considère une politique dite *1-stack* basée sur le PC minimal. Un tri systolique de la liste de chemins est réalisé au moyen d'un registre à décalage [15]. Cette réalisation est possible car l'architecture considérée ne gère qu'un seul warp. Une variante dite *2-stack* modifie l'ordre de tri pour traiter plus efficacement les boucles avec plusieurs branchements arrière. En contrepartie, *2-stack* ne traite pas les boucles imbriquées.
Le choix du PC minimum ne permet pas d'arbitrer entre des threads dans différentes fonctions. Nous avons proposé à Compas 2011 de donner la priorité au niveau d'appel de fonction maximal, puis au PC minimal [5]. Cette politique est réalisée dans Simty et évaluée section 3.
Le tri par PC garantit que la convergence s'effectue entre des chemins adjacents dans la liste. Ainsi, il suffit de comparer le chemin actif avec la tête de la liste pour détecter la reconvergence, sans avoir besoin d'opérer une recherche associative.
Un brevet de QUALCOMM décrit un tri similaire par niveau d'appel de fonction et par PC [4], tandis qu'un brevet d'ARM présente une réalisation matérielle pour une liste de deux chemins triée par PC [12].

2.5. Parcours en parallèle

Le parcours de graphe fournit du parallélisme qui peut être exploité. Plusieurs chemins actifs peuvent être parcourus simultanément ou alternativement, afin de masquer les latences ou augmenter le débit d'exécution. Des tentatives en ce sens ont combiné une pile de masques classique avec un ensemble de chemins parcourus en parallèle [17, 18]. Cependant, l'emploi d'une pile avec état limite soit le potentiel d'exécution en parallèle, soit la reconvergence [14].
En revanche, une liste de chemins permet de passer simplement des chemins entre l'état actif à l'état passif. La reconvergence reste néanmoins un problème, en particulier sur du flot de contrôle non structuré. Par exemple, le bloc D est parcouru deux fois sur la figure 1f.

1. Suivant notre nomenclature, Warp désigne un groupe de threads assemblé de manière statique, ce qui diffère de la terminologie de l'article [11].

L'architecture SBI avec contraintes base la reconvergence sur un intervalle d'adresses : la reconvergence a lieu avec tous les chemins compris dans un intervalle de PC indiqué par une instruction de reconvergence [3]. La motivation est que le code binaire d'un niveau de structure de contrôle est généralement connexe. L'architecture Multi-path se base sur une table de reconvergence [10]. La reconvergence est alors limitée aux régions à une entrée et une sortie, et la taille de la table de reconvergence n'est pas bornée. L'architecture HARP représente les chemins avec les numéros de threads qui les composent comme DWF, en offrant davantage de souplesse dans l'exécution en parallèle [14]. La reconvergence est basée sur une recherche associative dans une table de barrières de reconvergence.

3. Comparaison entre arbitrage et liste triée

Nous évaluons les coûts respectifs du mécanisme d'arbitrage entre PC [2, 5] et des tables de chemins [3]. Nous avons réalisé les deux mécanismes en RTL dans le processeur Simty [6]. Simty est un processeur SIMT multi-warp configurable, dont le pipeline à 10 étages est présenté Figure 2. Nous détaillons tour à tour les deux réalisations de suivi de flot SIMT.

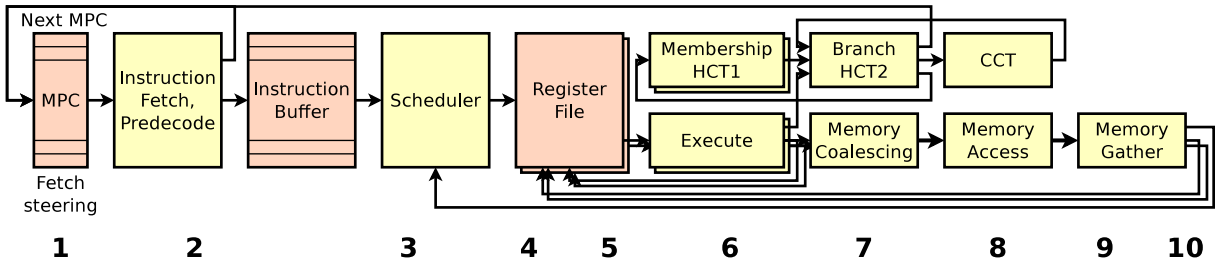


FIGURE 2 – Vue d'ensemble du pipeline Simty. La partie dédiée à la gestion des branchements est détaillée Figure 3.

3.1. Arbitre

La solution à base d'arbitre maintient un registre physique de PC par thread au moyen d'une table indexée par le numéro de warp qui renvoie le vecteur des PC des thread du warp. Le niveau d'appel de fonction maximum ou PC minimum est sélectionné par réduction à base d'un arbre de comparateurs et multiplexeurs. Notre implémentation se base sur l'arbitre développé par Nicolas Brunie pour cible ASIC [2]. Nous n'avons pas retenu l'option d'un arbitre unifié que nous avions initialement proposée pour des raisons de simplicité d'intégration, l'arbitrage mémoire et l'arbitrage des branchements intervenant à des étages différents du pipeline de Simty.

3.2. Liste triée

Dans la version à base de liste, l'unité de suivi des chemins est pipelinée sur 3 étages (Figure 3). Elle est composée de deux tables de chemins « chauds » (*Hot Context Table*, HCT), une unité de compaction et tri (*Context Compact-Sort*, CCS), et une table de chemins « froids » (*Cold Context Table*, CCT). La CCT est dimensionnée pour gérer le pire cas d'un unique thread par chemin. À partir du chemin actif, l'unité de branchement/mémoire crée jusqu'à deux nouveaux chemins a et b. Ces chemins et c venant de la seconde HCT sont triés et éventuellement fusionnés par l'unité CCS, pour produire entre 1 et 3 chemins $x < y < z$. L'étage de *fetch* est redirigé vers le

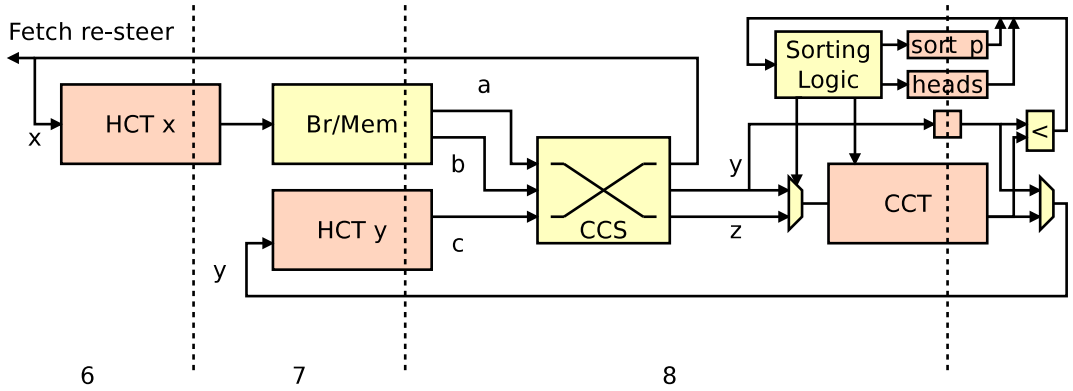


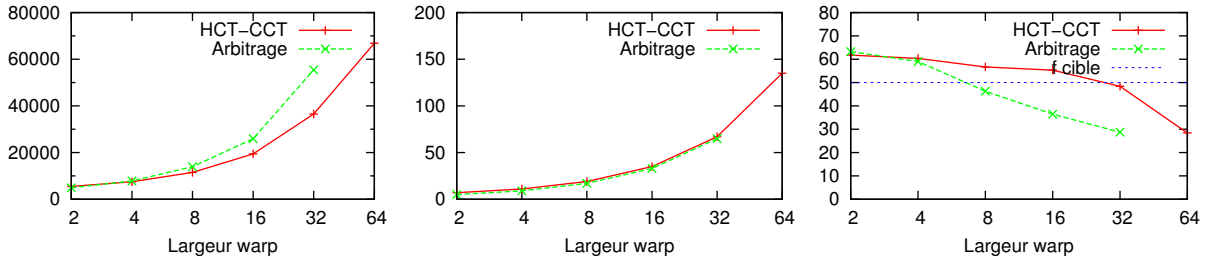
FIGURE 3 – Détail du suivi de chemins par HCT/CCT

PC du chemin x. Si les trois chemins sont valides, z est inséré dans la CCT. Si seul un chemin est valide, y est extrait de la CCT.
Nous avons considéré deux versions de la CCS. Une première version, à base d'un réseau de tri composé de trois niveaux successifs de comparateurs et multiplexeurs, s'est révélée avoir un délai excessif pour un tri en un cycle. Nous avons développé une seconde version à base de comparaisons d'ordre et d'égalité en parallèle entre paires puis sélection par multiplexeurs à trois entrées commandés par des fonctions booléennes des résultats des comparaisons. La seconde version est celle que nous avons retenue pour Simty et que nous considérons dans l'évaluation section 3.3.
Le maintien de deux HCT n'est pas strictement nécessaire au bon fonctionnement de l'unité, mais permet de minimiser le temps de résolution des branchements. En effet, le chemin de la seconde HCT assure un rôle de suppléant, qui prend le relai immédiatement en cas de destitution du chemin principal. La CCT est ainsi hors du chemin critique du calcul du nouveau PC.
La CCT est triée en continu par une machine à états multiplexée temporellement entre les warps, lorsqu'aucune insertion ou extraction n'a lieu. Chaque entrée est comparée tour à tour avec le chemin y de la seconde HCT. Lorsque l'ordre n'est pas respecté, les entrées sont échangées au cycle suivant [2].

3.3. Évaluation

Nous comparons les résultats de synthèse après placement-routage pour un FPGA Altera Cyclone IV EP4CE115 sur la figure 4. La surface est exprimée en éléments logiques (LE) et en blocs de SRAM de 128 $\times$ 32 bits (M9K). La fréquence ciblée est 50 MHz. Le placement-routage de la version avec arbitrage échoue pour 64 threads par warp, avec une estimation de surface à la synthèse de l'ordre de 91000 LE.
À partir de 8 threads par warp, le surcoût de l'arbitrage entre PC devient significatif, en particulier en surface de logique. En effet, outre la logique combinatoire des comparateurs, les vecteurs de PC nécessitent des mémoires larges et peu profondes qui sont réalisées avec des bascules individuelles sur cette génération de FPGA. À l'inverse, les HCT et CCT sont des mémoires étroites bien adaptées aux ressources du FPGA.
La fréquence estimée chute également en dessous de la fréquence cible de 50 MHz du fait de la latence de l'arbitrage. Cependant, nous n'avons pas rééquilibré le pipeline : en pratique, il faudrait ajouter un au plusieurs étages à l'arbitration du PC pour préserver le temps de

2. Si une insertion ou extraction a lieu entre temps, l'échange est annulé afin de garantir l'intégrité de la CCT.



(a) Surface (LE) (b) Block RAM (M9K) (c) Fréquence (MHz)

FIGURE 4 – Comparaison des résultats de synthèse après placement-routage du cœur Simty entre un suivi de la divergence à base de HCT et CCT et à base d'arbitrage, pour 8 warps.

cycle. Ce faisant, la latence de l'arbitrage n'affecterait que la pénalité de mauvaise prédiction de branchement, laquelle a peu d'impact sur une architecture massivement multi-threadée [2]. Le coût en surface de l'ensemble du contrôle de la CCT (gestion des insertions, extractions et tri) va de 316 LE pour 2 threads/warp à 633 LE pour 64 threads/warp, soit un surcoût respectif de de 6% et 1% à l'échelle du cœur. Le prix à payer pour maintenir la CCT triée est donc contenu, et sans commune mesure avec celui de l'arbre de réduction de la technique à base d'arbitrage.

4. Conclusion

Les mécanismes de suivi de la divergence SIMT à base de liste sont une alternative prometteuse aux piles de masques issues des architectures SIMD des années 1980-1990 qui sont employés dans les GPU actuels. Les techniques à base de liste combinent les avantages des piles de masque, à savoir faible coût et faible latence, avec ceux de l'arbitrage entre PC indépendants : compatibilité avec les jeux d'instructions généralistes, robustesse, et gestion du flot de contrôle arbitraire tel qu'exceptions et interruptions.

Bibliographie

1. AMD. – *Southern Islands Series Instruction Set Architecture*, Aug 2012.
2. Brunie (N.) et Collange (S.). – Reconvergence de contrôle implicite pour les architectures SIMT. *Revue des Sciences et Technologies de l'Information - Série TSI : Technique et Science Informatiques*, vol. 32, n2, février 2013, pp. 153–178.
3. Brunie (N.), Collange (S.) et Diamos (G.). – Simultaneous Branch and Warp Interweaving for Sustained GPU Performance. – In *39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 49 – 60, Portland, OR, United States, 2012.
4. Chen (L.). – Executing subroutines in a multi-threaded processing system. – US Patent 9,229,721, jan 2016.
5. Collange (S.). – Une architecture unifiée pour traiter la divergence de contrôle et la divergence mémoire en SIMT. – In *SYMPosium en Architectures*, p. 02, Saint-Malo, France, mai 2011.
6. Collange (S.). – Un processeur SIMT généraliste synthétisable. – In *Conférence d'informatique en Parallélisme, Architecture et Système (Compas)*, Lorient, France, juillet 2016.
7. Collange (S.), Daumas (M.), Defour (D.) et Parello (D.). – Étude comparée et simulation d'algorithmes de branchements pour le GPGPU. – In *SYMPosium en Architectures nouvelles de machines (SYMPA)*, 2009.
8. Diamos (G. F.), Johnson (R. C.), Grover (V.), Giroux (O.), Choquette (J. H.), Fetterman

(M. A.), Tirumala (A. S.), Nelson (P.) et Krashinsky (R. M.). – Execution of divergent threads using a convergence barrier. – US Patent App. 14/798,265, jul 2015.

9. ElTantawy (A.) et Aamodt (T. M.). – MIMD synchronization on SIMT architectures. – In *49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.
10. ElTantawy (A.), Ma (J. W.), O'Connor (M.) et Aamodt (T. M.). – A scalable multi-path microarchitecture for efficient GPU control flow. – In *International Symposium on High Performance Computer Architecture (HPCA)*, 2014.
11. Fung (W. W.), Sham (I.), Yuan (G.) et Aamodt (T. M.). – Dynamic warp formation : Efficient MIMD control flow on SIMD graphics hardware. *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 6, n2, 2009, p. 7.
12. Holm (R.) et Mansell (D. H.). – Scheduling program instructions with a runner-up execution position. – US Patent 9,436,473, sep 2016.
13. Hwu (W.-m. W.). – *Heterogeneous System Architecture : A new compute platform infrastructure*. – Morgan Kaufmann, 2015.
14. Lashgar (A.), Khonsari (A.) et Baniasadi (A.). – HARP : Harnessing inactive threads in many-core processors. *ACM TECS*, vol. 13, n3s, 2014.
15. Lee (Y.), Avizienis (R.), Bishara (A.), Xia (R.), Lockhart (D.), Batten (C.) et Asanović (K.). – Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. – In *ACM SIGARCH Computer Architecture News*volume 39, pp. 129–140. ACM, 2011.
16. Levinthal (A.) et Porter (T.). – Chap - a SIMD graphics processor. – In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, *SIGGRAPH '84*, SIGGRAPH '84, pp. 77–82, 1984.
17. Meng (J.), Tarjan (D.) et Skadron (K.). – Dynamic warp subdivision for integrated branch and memory divergence tolerance. *SIGARCH Comput. Archit. News*, vol. 38, n3, 2010, pp. 235–246.
18. Rhu (M.) et Erez (M.). – The dual-path execution model for efficient GPU control flow. – In *International Symposium on High Performance Computer Architecture (HPCA2013)*, pp. 591–602. IEEE, 2013.