Afrique SCIENCE *14(5) (2018) 65 - 74*
ISSN 1813-548X, http://www.afriquescience.net

Implémentation d'une primitive MPI Spawn pour le déploiement dynamique de tâches matérielles dans un système multiprocesseurs à mémoire distribuée sur puce reconfigurable

Roland Christian GAMOM NGOUNOU EWO[1*], Bertrand GRANADO[2] et Bertrand Hilaire FOTSIN[3]

[1] *Uniersité de Douala, ENSET, Laboratoire Génie Informatique et Automation (GIA), BP 1872 Douala, Cameroun*
[2] *Sorbonne Universités, UMR 7606 LIP6 Paris, France*
[3] *Université de Dschang, Département de physiques, Unité de Recherche de Matière Condensée, d'Electronique et de Traitement du Signal (URMACETS), BP 67, Dschang, Cameroun*

* Correspondance, courriel : *gamomchristian@yahoo.ca*

Résumé

Cet article décrit une nouvelle version de MPI-HCL, une bibliothèque de fonctions comprenant des primitives de communications par passage de message pour la réalisation d'applications parallèles, sur des systèmes multiprocesseur sur puce reconfigurable (MP-RSoC). La primitive MPI *Spawn* utilisable pour créer des tâches à la volée a été ajoutée pour permettre une activation ou une désactivation dynamique des tâches constituantes de l'application parallèle. Nous avons, implémenté en langage VHDL, le protocole de communication par accès mémoire distant (RMA) décrit par le standard MPI version 2. Cette implémentation est utilisée pour la réalisation des primitives de communications entre les tâches matérielles. Nous montrons dans ce travail qu'avec la nouvelle primitive MPI Spawn, la bibliothèque MPI-HCL permet aux concepteurs VHDL de créer des modèles pour déployer des applications parallèles. Avec la primitive MPI spawn, MPI-HCL supporte le passage à l'échelle, accepte des tâches matérielles hétérogènes et contribue à réduire l'énergie consommé par le MP-RSoC. La plateforme de démonstration a été testée sur une carte de développement pour FPGA Xilinx Artix-7.

Mots-clés : *RMA, MPI, FPGA, MP-RSoC, tâche matérielle, application parallèle.*

Abstract

The implementation of MPI Spawn primitive for dynamic deployment of hardware tasks on a multiprocessors system in a reconfigurable chip using distributed memory

In this work we describe a new version of MPI-HCL[1], our middleware including message passing primitives for parallel applications, on reconfigurable systems on chip (MP-RSoC). The MPI Spawn primitive for dynamic task creation has been added to enable dynamic activation or deactivation of parallel application tasks. Using VHDL language, we implemented the remote memory access protocol (RMA) defined by the MPI-2 standard for the realization of communication primitives between hardware tasks. We show in this work that with the new MPI Spawn primitive, the MPI-HCL library allows VHDL designers to create a platform that simplifies the deployment of parallel applications while supporting scalability, heterogeneity of hardware tasks and reduction of energy consumption in the MP-RSoC. The demonstration platform has been tested on a development board for Xilinx Artix-7 FPGA.

Keywords : *RMA, MPI, FPGA, MP-RSoC, Spawn, hardware task, deployment tools, parallel application.*

1. Introduction

Le MP-RSoC (Multi Processing Reconfigurable System On Chip) est un système complexe comprenant le matériel et le logiciel avec possibilité d'exécuter plusieurs tâches ou fonctions en parallèle. Avec l'augmentation de la densité des portes logiques contenues dans des circuits intégrés programmable de type FPGA (*Field Programmable Gate Array*), il devient possible d'y déployer des MP-RSoC complexes. Plusieurs équipements modernes comme les automobiles, les satellites, les robots industriels sont gérés par un système embarqué complexe ayant des contraintes élevées en performance, consommation, flexibilité, évolutivité et encombrement afin de satisfaire les besoins de l'utilisateur. Ce dernier ne se focalise plus sur les fonctions et les services rendus par l'équipement, mais il prend de plus en plus en comptes les autres caractéristiques que nous avons citées plus haut afin de fixer son choix sur un équipement. Un exemple d'application embarquée exigeante en performance de calcul est la vision autonome temps réel, utilisée dans les véhicules sans conducteur [1]. En effet la vision autonome temps réel met en œuvre plusieurs tâches (conversion des couleurs, détection de contour, identification, tri, classification, etc.) qui fonctionnent en parallèles et qui communiquent entre-elles. Les développeurs de systèmes embarqués ont besoin d'une plateforme MP-RSoC pour faire tourner une telle application c'est pourquoi plusieurs projets de recherches ont pour but de proposer de tels plateformes.

Alors que les concepteurs d'applications embarqués ciblant les GP-GPU disposent d'outils de développement d'applications parallèles tel que NVIDIA CUDA C/C++ [2] qui s'est imposé au fil du temps [3], les concepteurs d'applications pour cible FPGA ont besoin d'outils facilitant le développement d'applications parallèles. Pour réaliser une application parallèle sur cible FPGA, les concepteurs d'applications sont amenés à mettre en œuvre eux-mêmes les communications entre les différentes tâches de leur application. Le concepteur de chaque application parallèle sur cible FPGA a le choix entre utiliser un outil de synthèse automatique d'application à partir d'une modélisation de haut niveau ou d'utiliser un langage de description matérielle (HDL) afin de coder lui-même l'application parallèle [4]. Pour répondre au besoin d'automatisation de la génération de code HDL, les performances des outils de synthèse automatisés de type ESL (*Electronic Synthesis Language*), ou HLS (*High Level Synthesis*) sont de plus en plus améliorées [5]. Des outils tels que autoESL de Xilinx devenu Vivado HLS [6], permettent de synthétiser en RTL une application décrite en langage C/C++. Par contre, ces outils ne prennent pas totalement en charge l'aspect multitâche d'une application : le concepteur qui utilise le langage VHDL, doit lui-même compléter le code généré, pour obtenir les résultats souhaités. Cette dernière phase est pénible lorsque le nombre de tâches parallèles dépassent la dizaine.

Les projets FOSFOR et BORPH [7, 8] proposent des plateformes génériques pour la réalisation et le déploiement d'applications pour MP-RSoC. Ces plateformes partiellement automatisées permettraient de réduire le temps de développement de l'application tout en offrant l'efficacité d'un développement manuel. Les plateformes génériques et les outils que nous avons étudiées [4 - 9], ont pour limites la technologie du FPGA utilisé, l'obligation d'utiliser un bus ou un *softcore* particulier, elles ont également la contrainte d'utiliser un système d'exploitation non standard pour mettre en œuvre l'application. Développer une application sur une plateforme avec des outils et des fonctionnalités non standards est pénible pour les développeurs car ils doivent apprendre de nouveaux outils puis très souvent ils doivent eux-mêmes créer ou personnalisés plusieurs fonctionnalités dont ils ont besoin et cela augmente le temps de prototypage de l'application. La plateforme nommée MATIP que nous proposons, utilise le langage normalisé VHDL ainsi que la bibliothèque standardisée de programmation d'applications parallèle MPI. A partir de la version 2, la bibliothèque MPI a reçu une extension de fonctions de type RMA (*Remote Memory Access*) qui facilitent le transfert d'informations entre des tâches parallèles [10] ; MATIP implémente un sous-ensemble de ces fonctions pour réaliser les communications entre tâches parallèles. Les outils de déploiement dynamique d'application tel que PlanAhead

et GoAhead [11] facilitent la reconfiguration dynamique d'applications sur cible FPGA, toutefois la phase de conception étant séparée de la phase de déploiement, le processus de développement de l'application est rallongé et laborieux lorsque le nombre de tâches matérielles à reconfigurer dépassent la dizaine. Des projets comme FOSFOR, BORPH, MCAPI [7 - 9] proposent de gérer l'application pour MP-RSoC à l'aide d'un système d'exploitation (HW-OS), toutefois un système d'exploitation ajoute un surcoût en termes de ressources utilisées aux différentes tâches matérielles. L'objectif de cet article est de présenter une nouvelle primitive nommée MPI_comm_spawn qui complète un ensemble de primitives qui facilitent la conception d'applications parallèles pour MP-RSoC, de mesurer les temps de réponses de ces primitives, et de comparer le surcoût en utilisation de ressources avec ceux affichés par d'autres plateformes MP-RSoC. Le reste de cet article est présenté ainsi qu'il suit : dans la section 2 nous présentons l'architecture de MATIP, notre plateforme MP-RSoC et expliquons les principes de réalisation de la nouvelle primitive ; dans la section 3, nous présentons les performances de la nouvelle primitive et rappelons ceux des primitives développées lors de précédents travaux ; dans la section 4, nous comparons nos résultats à ceux de quelques plateformes existantes et soulignons l'intérêt de notre plateforme ; La section 5 est consacrée à la conclusion de cet article.

2. Matériel et méthodes

Le FPGA est un circuit intégré contenant de la logique reconfigurable utilisable pour implémenter des processeurs, des mémoires, des bus de communications, et d'y faire fonctionner des programmes et des modules de traitement numériques hétérogènes encore appelé IP *(Intellectual Proprety)*. Concevoir un MP-RSoC qui est un système comprenant plusieurs unités matérielles interconnectées par des bus et généralement organisé en couches demande de suivre une méthodologie rigoureuse. La conception du MP-RSoC MATIP a suivi une méthodologie dite *meet- in-the middle ;* l'architecture de la plateforme a été organisée en trois couches [12] puis les composants de chaque couche ont été réalisés en décrivant puis en assemblant des modules élémentaires. Nous avons réalisé la couche réseau qui est basée sur un crossbar reconfigurable ayant entre 2 et 16 ports, selon la valeur des paramètres d'instanciation. La couche communication regroupe l'ensemble des primitives de communication, et la couche application qui encapsule les tâches utilisées par l'application parallèle. L'application de démonstration simule un transfert de type ping-pong ; la tâche de rang 0 envoie une trame d'octets à trois autres tâches puis ces tâches renvoient la trame précédemment reçue à la tâche de rang 0. Les tests présentés dans cet article ont été effectués sur un FPGA Xilinx Artix-7 xc7A100T, toutefois il est possible de déployer MATIP sur un FPGA d'une autre famille ou sur un FPGA d'un autre fabricant. La couche de communication appelée MPI-HCL fait l'objet d'une amélioration ; elle est présentée dans la suite de cette section.

2-1. Architecture de la plateforme MATIP

Le standard MPI décrit un protocole pour la programmation d'applications parallèles de type SPMD (Single program multiple data) ou MPMD (Multiple Program Multiple Data) basé sur le paradigme de passage de message et propose un ensemble de primitives à implémenter pour construire une bibliothèque MPI. Nous avons retenu ce standard pour notre architecture pour les trois raisons suivantes :

1. Dans les environnements de programmation multiprocesseurs, le paradigme de communication par passage de message supporte mieux le passage à l'échelle sur le plan matériel et permet des temps de mise au point des applications plus réduits que le paradigme à mémoire partagée lorsque le nombre de tâches parallèles dépassent la dizaine [13].
2. MPI est un standard ouvert et répandu dans la communauté des développeurs d'applications parallèles
3. Peu d'implémentations d'une version MPI pour MP-RSoC existent à ce jour, aucune ne met en œuvre la fonction MPI_comm_spawn à notre connaissance.

Notre architecture est organisée en trois couches ***(Figure 1)***: une couche pour l'interconnexion, une couche pour les communications et la troisième couche pour l'application. La couche d'interconnexion est un réseau sur puce (NoC) qui est réalisé à partir d'un *crossbar*.

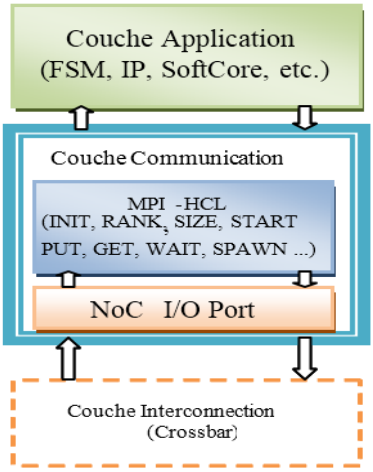

Figure 1 : *Couches de la plateforme MATIP*

Ce réseau sur puce (NoC) est conçu pour être configuré en fonction des besoins de la plateforme. Un paramètre générique permet de définir le nombre de ports maximum du NoC au moment de la synthèse de la plateforme. La couche de communication est constituée du composant MPI-HCL ; il contient la version matérielle des primitives MPI qui permettent :

— d'initialiser un environnement MPI (MPI_init, MPI_comm_rank),
— d'activer des fenêtres mémoire pour le stockage des messages distants (MPI_Win_start, MPI_Win_Post),
— d'envoyer ou recevoir des messages entres différentes tâches matérielles (MPI_Put, MPI_Get),
— de synchroniser les communications entre les tâches (MPI_Win_Wait, MPI_Win_complete),
— d'activer ou de désactiver des tâches matérielles (MPI_comm_spawn).

La couche application est constituée d'un ensemble de tâches matérielles ou d'IPs à paralléliser. Elle s'interface à la couche communication à partir d'un *template* VHDL appelé TIC (Task Integration Component). Le TIC est constitué de plusieurs modules et contient le code nécessaire pour l'interconnexion de chaque tâche matérielle avec un composant MPI-HCL de la plateforme. Le module qui instancie la tâche matérielle en VHDL est nommé HT_USER_FSM. La plateforme expérimentale est visible sur la ***Figure 3***. Elle utilise la version à quatre ports du crossbar configurable. Un template a été réalisé pour encapsulé la tâche matérielle (IP) à paralléliser. Chaque tâche matérielle reçoit un rang MPI unique durant l'exécution. Les primitives de communications sont utilisées pour décrire le flot de données entre les tâches matérielles. Avec MATIP, les tâches matérielles peuvent être hétérogènes et les communications entre elles sont compatibles au standard MPI-2 décrit dans [10]. Nous avons la contrainte de réduire les opérations nécessaires pour intégrer de nouvelles IP en utilisant MATIP.

2-2. Génération d'une tâche dans un environnement multitâche

Lors de l'exécution d'une application à traitement parallèles, il peut arriver qu'une fonctionnalité supplémentaire soit nécessaire pour le bon fonctionnement de l'application alors il faudra l'ajouter dynamiquement. Dans l'environnement MPI Cette opération est appelée *spawn* (qui signifie engendrer). Par exemple, une application de multitraitement des vidéos peut rencontrer un fichier vidéo ayant un type de

compression *(codec)* non standard. L'application doit charger et exécuter une fonction capable de prendre en charge ce type de compression du fichier vidéo. Dans notre implémentation de la fonction MPI_comm_spawn, nous avons reproduit cette fonctionnalité pour l'utiliser dans le cas des tâches matérielles dynamiques**.**

2-3. Etapes de l'implémentation de la fonction MPI_comm_spawn

Le processus d'initialisation des tâches matérielles obéi à la séquence ci-après :
1. Toutes les tâches matérielles (HT) sollicitent un rang en faisant un appel à MPI_init
2. Le composant MPI-HCL racine (c'est-à-dire celui qui est connecté au port 0 du NoC) fournit un rang à toutes les HTs ayant effectué l'appel à MPI_init.
3. Le composant racine MPI-HCL envoie un signal aux autres HTs lorsque ce processus d'initialisation est terminé.
4. Les HTs peuvent communiquer seulement après que ce processus d'initialisation soit terminé

Lors de l'appel à MPI_comm_spawn :
1. le composant MPI_HCL racine synchronise à l'aide d'une barrière toutes les tâches en cours d'exécution ;
2. Le composant racine MPI-HCL active les nouvelles tâches sur des ports libres du NoC ;
3. Les nouvelles tâches effectuent leur initialisation ;
4. Le composant racine MPI-HCL envoie un message collectif signalant la fin du processus *spawn* à toutes les tâches matérielles.
5. Les nouvelles tâches et les anciennes tâches peuvent à présent communiquer.

Le processus *spawn* peut être invoqué autant de fois, tant qu'il existe des ports libres sur le NoC, pour activer de nouvelles tâches. La fin de l'exécution d'une tâche libère automatiquement le port du NoC qu'elle occupait. Par rapport aux spécifications du standard MPI-2 [10] nous avons effectué quelques simplifications et quelques adaptations :
— une fenêtre mémoire est implicitement définie avant le début de chaque transfert ;
— quelques paramètres pour mémoriser l'état interne du processus sont ajoutés à chaque primitive à cause de la nature séquentielle synchrone de l'exécution de la tâche matérielle.

Les prototypes de nos fonctions VHDL sont :
procedure pMPI_Init(NextCtx : inout natural;signal Interf:inout Core_io;signal clkin:std_logic;signal SysRam :inout typ_dpram);

procedure pMPI_Comm_Spawn(NextCtx : inout natural; signal Interf:inout Core_io;signal clkin:std_logic; signal SysRam :inout typ_dpram; command : natural;argv : array of natural; maxprocs : natural; info : MPI_Info; root : natural;comm : MPI_Comm; intercomm : MPI_Comm; array_of_errcodes : array of natural);

Le composant MPI-HCL est présenté à la ***Figure 2*** la fonction de chaque module de ce composant est décrite dans [12].

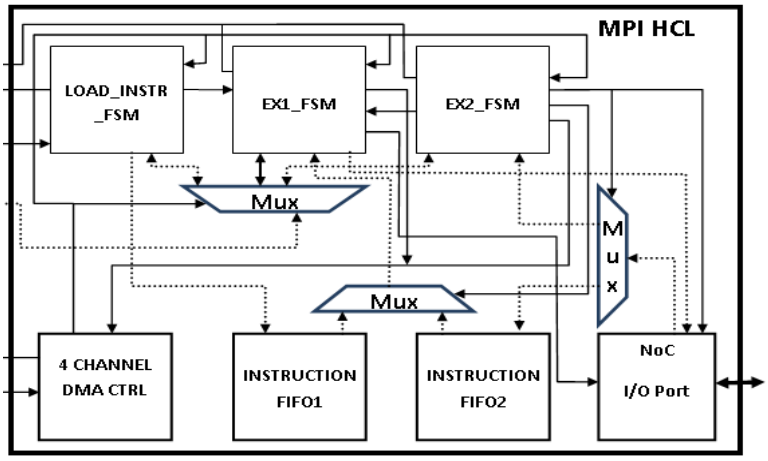

Figure 2 : *Architecture interne du middleware MPI-HCL*

2-4. Description du modèle d'intégration de la tâche matérielle (TIC)

Le TIC qui encapsule la tâche matérielle est visible sur la ***Figure 3***, il contient une RAM statique servant à stocker les messages reçus ou à envoyer. Nous avons réalisé et ajouté à ce TIC deux fonctions mem_rd() et mem_wr() qui permettent de lire et d'écrire dans cette mémoire RAM. Le module RAM ARBITER gère l'accès à la mémoire RAM en évitant des collisions entre les accès en lecture et écriture du composant MPI-HCL et ceux du module HT_User_FSM. Le module En_Fsm active/désactive le module Ht_User_Fsm lorsqu'une requête *spawn* est émise par le middleware MPI-HCL. Ce module sera amélioré dans les futurs développements pour supporter la reconfiguration dynamique partielle du MP-RSoC. L'interface I/O décrit l'ensemble des signaux nécessaires pour interconnecter HT_User_FSM et le composant MPI-HCL. Pour simplifier la manipulation de ces signaux dans le module HT_User_Fsm, des types personnalisés VHDL (structures) ont été créés et ajoutés au TIC. Le TIC inclus un package VHDL qui contient les déclarations de toutes les fonctions qui seront utilisées pour les communications et il contient aussi un fichier ayant deux constantes permettant de fixer la taille du NoC à générer et le nombre de tâches statiques à créer lors de la synthèse de la plateforme. Le transfert d'informations vers chaque tâche matérielle est réalisé par des appels de procédures VHDL. La taille de la RAM locale pour chaque tâche matérielle est paramétrable en fonction de chaque application. A partir du module HT_User_FSM, contenant le code VHDL de l'utilisateur les primitives de communications sont appelées en respectant la même convention que lors de l'appel de fonctions ou de procedures VHDL ordinaires. Le module HT_User_FSM est intégré au TIC, l'utilisateur de MATIP l'exploite, soit pour instancier une IP, soit pour décrire la fonctionnalité de sa tâche matérielle.

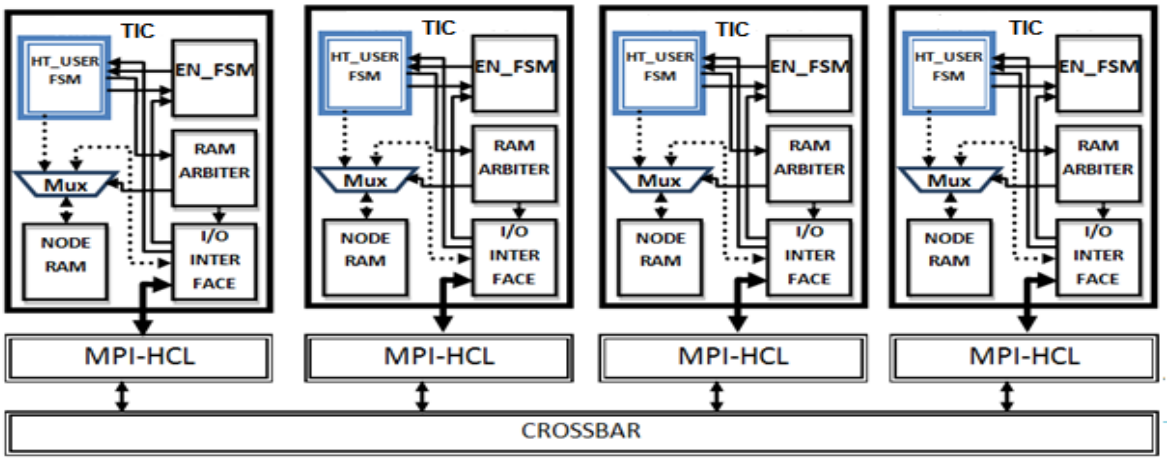

Figure 3 : *MP-RSoC MATIP instanciant 4 tâches matérielles (DEMO MATIP)*

3. Résultats

La plateforme MATIP s'exécute à une fréquence de 76 MHz sur un Xilinx Artix-7 xc7A100T FPGA et occupe un nombre raisonnable de ressources (registres, LUT, bloc RAM, ports d'E/S) au regard de sa complexité. La mise en œuvre du démonstrateur sur un FPGA Artix-7 xc7a100t donne un taux de consommation des ressources de 38 % ***(Tableau 1)***. Le surcoût qui quantifie les ressources utilisées par les composants MPI-HCL et TIC pour l'ajout d'une tâche matérielle est évalué à moins de 4 % des ressources totales du xc7a100t. Ainsi, MATIP ajoute un surcoût faible pour le déploiement dynamique des tâches matérielles à l'aide des primitives de la bibliothèque MPI. Le ***Tableau 1*** donne aussi les détails de l'implémentation de chaque module du démonstrateur MATIP. La couche MPI-HCL pour chaque tâche matérielle occupe entre 2428 et 2419 LUTs. Chaque TIC occupe entre 88 et 83 LUTs, consomme 2 Bloc RAM et 32 registres. La ***Figure 4*** présente les latences obtenues lors de l'exécution de chaque primitive de la couche de communication MPI-HCL. Nous avons calculé un débit de 12 cycles/octet pour le transfert d'un octet et 2,6 cycle/octet pour le transfert de 248 octets. Le ***Tableau 2*** montre les résultats comparatifs de MPI-HCL et d'autres plateformes. La primitive MPI_comm_spawn a une latence constante de 400 cycles quelle que soit la taille des données à transférer ; ce résultat signifie qu'elle ajoute à une application multiprocesseur qui s'exécute à 76 MHz un délai de 5,26 µs ; ceci est justifié par le fait que cette primitive effectue essentiellement des opérations de synchronisation sans transfert de données.

Tableau 1 : *Utilisation des ressources après synthèse de plateforme de test dans un FPGA Artix-7 xc7A100T*

Nom du module	**Tranches *(Slices)***	**Registres**	**LUTs**	**LUT RAM**	**BRAM**
Total ressources du xc7a100t	**15 850**	**126 800**	**63 400**	**19 000**	**405**
DEMO_MATIP	**6 066 38 %**	**6 850 6 %**	**15 122 24 %**	**233 1 %**	**20 5 %**
Détails des ressources pour chaque couche et chaque module de MATIP					
Bstream loader	10	27	36	0	0
TIC1	327	274	805	0	2
TIC1.HT_task	280	242	717	0	0
TIC2	331	274	811	0	2
TIC2.HT_task	282	242	720	0	0
TIC3	268	222	687	0	2
TIC3.HT_task	225	190	604	0	0
TIC4	253	194	640	0	2
TIC4.HT_task	213	162	556	0	0
4 HT (ressources utilisées par les 4 tâches de l'application)	1000	836	2597	0	0
Surcoût TIC : 4 (TIC-HT) + Bstream loader	179	128	346	0	
MATIP L1& L2	4 464	5 070	11 351	96	8
MPI-HCLx4	3 798	4 348	9 696	96	
HCL1	972	1088	2428	24	0
HCL2	949	1088	2425	24	0
HCL3	963	1086	2424	24	0
HCL4	914	1086	2419	24	0
Interconnect crossbar 4x4	666	722	1655	0	8

Dans nos précédents travaux [12], nous avons décrit les primitives MPI_put and MPI_get, qui permettent d'envoyer et de recevoir des messages. Ce travail étend le précédent en ajoutant la primitive MPI_comm_spawn qui permet de connecter des tâches matérielles et de les activer lors de l'exécution. L'activation d'une tâche au seuil de son utilisation favorise la réduction de l'énergie électrique consommée dans le MP-RSoC.

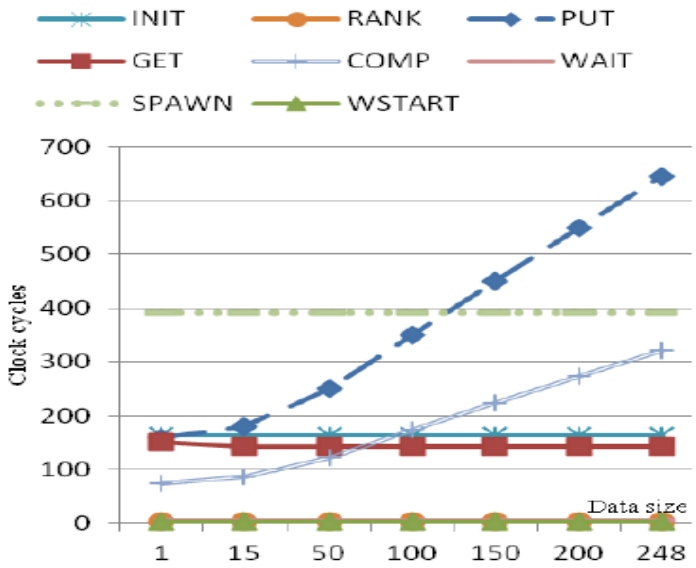

Figure 4 : *Latence des primitives du middleware MPI en cycles d'horloge*

4. Discussion

Nous avons calculé une latence de 3,43 µs pour le transfert d'un octet et une latence de 10,72 µs pour le transfert de 248 octets. Il est profitable d'échanger des données d'une taille maximum plutôt que de procéder à plusieurs échanges d'un petit nombre de données avec MATIP. Le ***Tableau 2*** récapitule les comparaisons entre MATIP et quelques plateformes MP-RSoC utilisant MPI. Nous comparons les latences mesurées sur chaque plateforme pour transférer la plus petite quantité de données. La mesure de la latence est exprimée en microsecondes ou en nombre de cycles d'horloge. La bonne performance du MATIP est justifiée par sa mise en œuvre langage de description du matériel VHDL comparé à SOC-MPI [13] et à la couche d'abstraction matérielle MPI HAL [14], tous deux codés en langage C et compilés pour les Softcores Nios II ou MicroBlaze. La plateforme TMD-MPI [15] affiche une latence de 8,5 µs pour le transfert de messages de longueur zéro entre les unités d'exécution instanciées dans le FPGA. MATIP donne de meilleurs résultats et montre qu'il est avantageux d'utiliser MPI-2 RMA pour assurer les communications dans le MP-RSoC.

Tableau 2 : *Comparaison des latences de certaines plates-formes MPI SOC*

Auteurs	Plateformes	Latence		Type de transfert
		(µs)	(cycles)	
Saldana et al. [16]	TMD-MPI	8,5	340	Message de longueur zéro octets sur puce
Minhass et al. [15]	MPI HAL	23	1150	transfert MPI du paquet de taille minimum vers un voisin proche dans le même FPGA
Mahr et al. [14]	SOC-MPI	90	9000	Transfer de 128 Octets
GAMOM et al.	MATIP	3,43	93	Transfert d'un octet

Le ***Tableau 3*** présente la comparaison entre MATIP et FOSFOR en considérant le déploiement des tâches matérielles. Dans le tableau 3, nous montrons que les ressources supplémentaires requises pour déployer des tâches matérielles sur MATIP sont limitées par rapport à celles requises par FOSFOR [7] nous n'avons pas obtenu les données détaillées pour les plateformes SOC-MPI, TMD-MPI et MPI-HAL. Pour chaque plate-forme, nous avons comparé les ressources utilisées à la fois par le middleware et par le conteneur reconfigurable

(wrapper) nécessaire pour intégrer une tâche matérielle. Le middleware est la couche de communication MPI-HCL pour MATIP et pour FOSFOR il correspond à la fois aux couches HWOS (système d'exploitation physique) et CS (service de communication). La plate-forme MATIP occupe cinq fois moins de ressources que FOSFOR pour la tâche matérielle et occupe dix fois moins de ressources que FOSFOR pour la communication entre les tâches matérielles.

Tableau 3 : *Comparaison des ressources utilisées par les plateformes FOSFOR et MATIP*

FOSFOR DEMO[1]	**REGISTRES**	**LUTS**	**BRAMS**
Processeur MPSoC Leon	4463	7951	9
4 acteurs reconfigurables	8172	27256	8
Système d'exploitation (HW-OS)	11286	26292	0
Intergiciel matériel	2412	9118	0
NoC de type DRAFT 8 ports	920	2902	0
MATIP	REGISTRES	LUTS	BRAMS
Total des ressources	6 850	15 355	20
4 IP ou tâches matérielles	836	2597	0
Surcoût = [4 * (TIC-HT) + Bstream loader]	128	346	8
Intergiciel matériel MPI-HCL * 4	4 348	9 792	
NoC de type Crossbar 4x4	722	1655	8

5. Conclusion

Ce travail présente les résultats de l'implémentation du middleware MPI-HCL intégré à la plateforme pour MP-RSoC MATIP. MPI-HCL met en œuvre neuf primitives de communication du standard MPI-2. L'objet de MPI-HCL est de simplifier le déploiement d'applications parallèles comprenant des tâches matérielles en utilisant le standard populaire MPI. Nous avons particulièrement décrit la primitives MPI_comm_spawn qui permet d'instancier une tâche matérielle pendant l'exécution de l'application parallèle sur puce. L'utilisation de primitives VHDL, pour établir des communications entre tâches, simplifie la réutilisation d'IPs existantes lors de la conception du MP-RSoC. La structure en trois couches de MATIP facilite l'évolution de chaque couche et cet article présente une amélioration de la couche communication. Nous retenons que la couche de communication MPI-HCL occupe 5 fois moins de ressources que la couche qui joue le même rôle pour la plateforme FOSFOR et que la latence pour transférer un octet avec MPI-HCL est 2,5 fois plus petite que celle qui permet de transférer le plus petit nombre d'octet transférable avec TMD-MPI. Le nombre de tâches maximum que peut supporter MATIP est lié à la taille du NoC de la couche interconnexion, actuellement il est limité à 16 ports. Un aspect important de cette implémentation est qu'elle n'est pas dépendant d'une technologie particulière, nous pouvons déployer notre MP-RSoC sur différentes familles de FPGA de différents fabricants. Les améliorations futures de la plateforme MATIP seraient d'étendre les capacités en nombre de ports de la couche communication, et d'étendre la largeur du bus de données de la plateforme à 32 bits. L'introduction du mécanisme de la reconfiguration dynamique partielle pourrait améliorer la flexibilité du composant MPI-HCL de la couche application.

Références

[1] - HAN, SONG et al, "Reconfigurable Processor for Deep Learning in Autonomous Vehicles." *Itu Journal-Ict Discoveries*, (2017)

[2] - NVIDIA CUDA, "NVIDIA CUDA C programming guide," Nvidia Corporation, 20 (18) (2011) 8

[3] - JASON CONG, ZHENMAN FANG, MICHAEL LO, HANRUI WANG, JINGXIAN XU and SHAOCHONG ZHANG, Understanding Performance Differences of FPGAs and GPUs : In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18). ACM, New York, NY, USA, (2018) 288 - 288, https://doi.org/10.1145/3174243.3174970

[4] - R. U. A. N. YUE, A Novel Methodology on Optimizing the Performance of Multi-core Processor Using FPGA. DEStech Transactions on Computer Science and Engineering cece, (2017)

[5] - NANE, RAZVAN et al., A survey and evaluation of fpga high-level synthesis tools, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35, 10 (2016) 1591 - 1604

[6] - B. D. T. Inc., An independent evaluation of high-level synthesis tools for xilinx fpgas, (2010), http ://www.xilinx.com/technology/dsp/BDTI techpaper.pdf

[7] - L. GANTEL, A. KHIAR, B. MIRAMOND, M. E. A. BENKHELIFA, L. KESSAL, F. LEMONNIER and J. LE RHUN, Enhancing reconfigurable platforms programmability for synchronous data-flow applications, ACM Trans. *Reconfigurable Technol. Syst.*, 5 (3) (2012) 1 - 14, http ://doi.acm.org/10.1145/2362374.2362378

[8] - SO, HAYDEN KWOK-HAY and ROBERT BRODERSEN, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH." ACM Transactions on Embedded Computing Systems (TECS), 7.2 (2008) 14

[9] - L. MATILAINEN, E. SALMINEN, T. D. HÄMÄLÄINEN & M. HÄNNIKÄINEN, Multicore Communications API (MCAPI) implementation on an FPGA multiprocessor. Embedded Computer Systems, SAMOS, International Conference on. IEEE, (2011)

[10] - GROPP, WILLIAM, HOEFLER, TORSTEN, THAKUR, RAJEEV et al., *Using advanced MPI : Modern features of the message-passing interface*. MIT Press, (2014)

[11] - C. BECKHOFF, D. KOCH and T. JIM, *GOAHEAD : A Partial Reconfiguration Framework,* in 20th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, (2012) 37 - 44

[12] - R. GAMOM NGOUNOU EWO, E. KIEGAING, M. MBOUENDA, H. FOTSIN and B. GRANADO, *Hardware mpi-2 functions for multi-processing reconfigurable system on chip, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW)*, (2013) 273 - 280

[13] - I. L. TIM MATTSON, *Scc a tale of two processors,* (2010), http://i2pc.cs.illinois.edu/ presentations/2010 05 06 Mattson Slides.pdf

[14] - P. MAHR, C. LORCHNER, H. ISHEBABI and C. BOBDA, *Soc-mpi: A flexible message passing library for multiprocessor systems-on-chips*, in Reconfigurable Computing and FPGAs, 2008. ReConFig '08. International Conference on, (Dec 2008) 187 - 192

[15] - W. H. MINHASS, J. OBERG and I. SANDER, *Design and implementation of a plesiochronous multi-core 4x4 network-on-chip fpga platform with mpi hal support,* in Proceedings of the 6th FPGAworld Conference. ACM, (2009) 52 - 57

[16] - M. SALDANA, D. NUNES, E. RAMALHO and P. CHOW, *Configuration and programming of heterogeneous multiprocessors on a multi-fpga system using TMD-MPI,* in Reconfigurable Computing and FPGA's, 2006, IEEE International Conference on, ReConFig 2006, Sept (2006) 1 - 10