Unification des mémoires réparties dans un système hétérogène avec accélérateur reconfigurable : exposé de principe

Erwan Lenormand, Loïc Cudennec, Henri-Pierre Charles

To cite this version:

Erwan Lenormand, Loïc Cudennec, Henri-Pierre Charles. Unification des mémoires réparties dans un système hétérogène avec accélérateur reconfigurable : exposé de principe. Conférence d'informatique en Parallélisme, Architecture et Système (Compas'2019), LIUPPA - Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (EA 3000), Jun 2019, Anglet, France. cea-02279337

HAL Id: cea-02279337
https://cea.hal.science/cea-02279337

Submitted on 5 Sep 2019

Unification des mémoires réparties dans un système hétérogène avec accélérateur reconfigurable : exposé de principe

Erwan Lenormand[1,2], Loïc Cudennec[2],Henri-Pierre Charles[3]

[1]Univ Paris-Saclay, erwan.lenormand@cea.fr
[2]CEA, List, F-91191, PC 172, Gif-sur-Yvette, France
[3]Univ Grenoble Alpes, CEA, List, F-38000 Grenoble, France

Résumé
Cet article fait l'exposé d'un principe d'architecture hybride logicielle/matérielle, basée sur la technologie OpenCAPI, pour intégrer un accélérateur de type FPGA dans une mémoire virtuellement partagée par le logiciel. Ce système a pour objectif d'offrir à l'utilisateur, sous forme d'un intergiciel, une vision unifiée des mémoires réparties d'un système pouvant être composé de plusieurs nœuds distants et hétérogènes. Cet intergiciel doit offrir une gestion des accès concurrents aux données entre des ressources physiquement séparées. Cet article présente le fonctionnement d'une mémoire virtuellement partagée logicielle ainsi qu'un état de l'art des nouvelles microarchitectures CPU-FPGA.

Mots-clés : Mémoire Virtuellement Partagée, Système Hétérogène, Système reconfigurable

---

1. Introduction

Les machines hétérogènes répondent à des besoins de performance (GFlops) et d'efficacité énergétique (GFlops/W) que les machines classiques composées de processeurs généralistes ne peuvent pas satisfaire. L'association GPGPU-CPU permet par exemple d'améliorer les performances calculatoires [4, 3], au prix d'un modèle de programmation devenu complexe.

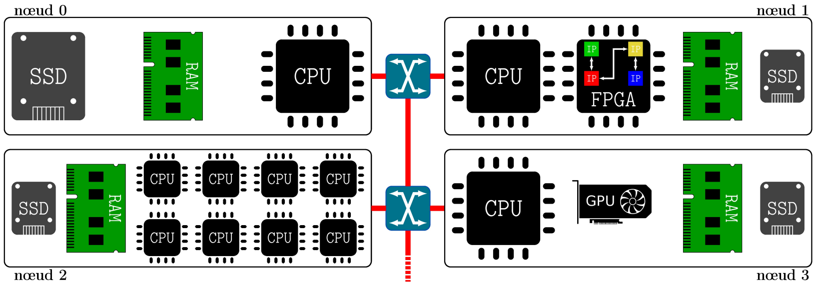

FIGURE 1 – Nœuds d'un micro-serveur hétérogène

Depuis une dizaine d'années, avec la généralisation des architectures embarquées grand public et les contraintes énergétiques associées, les processeurs basse consommation et les FPGA (Field Programmable Gate Arrays) sont intégrés dans les architectures en qualité de coprocesseurs ou nœuds à part entière. Un exemple est l'architecture micro-serveur (HPE Moonshot [2], Christmann RECS [20]) dans laquelle un fond de panier permet d'alimenter et de connecter des cartes d'extension. L'hétérogénéité provient du fait que ces cartes peuvent intégrer des unités de calcul et de stockage très différentes (figure 1), en fonction des besoins applicatifs et des contraintes opérationnelles : Intel Xeon avec GPU Nvidia, Arm Cortex avec GPU Mali, FPGA avec Arm Cortex intégré ou encore Nvidia Tegra. Des architectures similaires encore plus intégrées sont aussi conçues pour les systèmes embarqués dans l'automobile et pour les véhicules autonomes. Enfin, cette tendance se retrouve dans le domaine du HPC (High-Performance Computing), avec l'intégration de FPGA dans les routeurs réseaux hautes performances pour effectuer des traitements à la volée (Mellanox).

La programmation de ces systèmes hétérogènes est complexe car les modèles de programmation et les interfaces sont souvent spécifiques à chaque type de nœud. La programmation hybride est très répandue dans ces environnements : le code applicatif est décomposé en une partie répartie qui orchestre les transferts de données entre les nœuds (modèle de programmation répartie avec passage de message, MPI), et une partie métier implémentée selon différentes technologies pour s'exécuter localement sur des accélérateurs (modèle de programmation parallèle en mémoire partagée, Pthread, OpenMP, CUDA).

Le modèle de programmation hybride complexifie la conception d'applications numériques en faisant porter la responsabilité au développeur du découpage des données, de leur localisation et de leur transfert entre les tâches d'entrées-sorties et les tâches de traitement. De plus, une donnée doit être identifiée et prise en charge de différentes manières : sous forme d'un ou plusieurs messages à l'échelle inter-nœuds, sous forme d'un pointeur mémoire sur une machine NUMA (Non-Uniform Memory Access) ou encore manuellement redécoupée pour être transférée dans un accélérateur de type FPGA. À l'instar de la plateforme, le *traitement* de la donnée dans l'application est devenu hétérogène.

Des modèles de programmation comme le flot-de-données proposent de prendre en charge la gestion de données de manière transparente. Certains intergiciels tels que StarPu [8] permettent d'abstraire la plateforme répartie et par exemple de masquer les transferts vers des codes OpenCL pour GPU. L'application doit cependant être structurée en flot-de-données, ce qui n'offre pas la même granularité d'accès que dans le cas d'une mémoire partagée.

Dans ce travail nous proposons de conserver le modèle mémoire partagée à l'échelle de la plateforme hétérogène, tout en masquant à l'utilisateur les traductions d'adresses, les éventuelles copies mémoire et échanges de messages. Les données partagées doivent conserver un identifiant commun et des méthodes d'accès homogènes et cohérentes sur tous les nœuds de la plateforme. Pour cela, nous étudions la possibilité d'étendre une Mémoire virtuellement partagée (MVP) logicielle jusque dans les accélérateurs matériels. Nous proposons d'étudier spécifiquement le cas du FPGA, car il présente un bon compromis entre flexibilité et efficacité énergétique [18]. Cependant, leur utilisation reste limitée due à la complexité de programmation. Les outils HLS (High-Level Synthesis) permettent de réduire cette complexité et plusieurs architectures émergentes permettent aux FPGA de faire des accès directs et cohérents à la mémoire du processeur principal (Intel-Altera HARP, IBM CAPI, Convey HC-1) [12]. Nous présentons alors un exposé de principe basé sur la technologie Open Coherent Accelerator Processor Interface (OpenCAPI) [29].

La section 2 présente le fonctionnement d'une MVP logicielle. La section 3 présente un état de l'art des microarchitectures CPU-FPGA conçues pour permettre à l'accélérateur d'accéder directement à la mémoire du processeur. La quatrième section expose le fonctionnement de la MVP et de la technologie OpenCAPI, puis expose le principe de notre proposition. Dans la section 5, nous présentons des travaux similaires au notre. La section 6 conclut cet article.

2. Mémoire virtuellement partagée logicielle

Une MVP est un système qui permet de donner l'illusion qu'un ensemble de mémoires physiquement réparties n'en forment qu'une. Pour cela, un espace logique est construit, dans lequel il est possible d'effectuer des accès en lecture et écriture sur des données partagées, à partir de processus s'exécutant sur des nœuds potentiellement distants. Du point de vue applicatif, ces systèmes offrent une couche d'abstraction du matériel en masquant notamment la topologie physique de la machine en termes de hiérarchie mémoire et d'interconnexion de nœuds de calcul. Cette abstraction peut aussi apporter un effet d'agrégation des mémoires, bénéfique en termes de taille et de débit d'accès grâce à la coopération des nœuds voisins. Le modèle de programmation résultant est équivalent à un modèle de programmation classique sur une architecture centralisée de type NUMA.

La contrepartie d'une telle approche est que la complexité de gestion des données est laissée à la charge de la MVP, ce qui inclut la localisation, le transfert et la mise à disposition des données, ainsi que les problématiques de synchronisation de groupe inhérentes aux systèmes répartis. La MVP est implémentée sous forme d'un intergiciel ou d'une modification du système d'exploitation, ce qui peut introduire un surcoût pour les performances du code applicatif. Ce surcoût est principalement issu du nombre de messages échangés pour garantir la cohérence des données, qui est supérieur au nombre de messages échangés dans une implémentation de l'application directement conçue pour des systèmes répartis.

Les MVP ont été largement étudiées à partir de la fin des années 1980 avec des systèmes pour grappes de calculateurs [22, 9, 11, 5], puis pour les grilles de calculateurs avec des approches plus dynamiques [6, 7, 24, 23] et des systèmes intégrés parallèles à mémoires réparties [27]. Certaines MVP prennent en considération l'environnement hétérogène, mais principalement en ce qui concerne la représentation des données partagées [10, 32, 26, 25].

Dans le système A-DSM [19], un espace logique partagé est construit pour les systèmes hybrides combinant un processeur hôte et un accélérateur de type GPU. Cependant, cette MVP est asymétrique : le CPU peut accéder aux données stockées sur l'accélérateur mais l'inverse n'est pas possible. Dans ce travail, nous cherchons à offrir un espace d'adressage unique commun à toutes les unités de calcul et accessible à l'initiative de tâches déployées sur des processeurs classiques comme sur des accélérateurs reprogrammables tels que les FPGA.

3. Microarchitectures CPU-FPGA pour les systèmes de calcul haute performance

Parmi les différents types d'accélérateurs, les FPGA se présentent comme une des ressources les plus prometteuses pour les systèmes HPC grâce à leur fonctionnement basse consommation, leur haute efficacité énergétique et leur nature reprogrammable qui leur permet de s'adapter à différentes applications. Dans l'analyse [17], Escobar *et al.* identifient cinq types d'applications HPC pour lesquelles une exécution sur FPGA serait la plus efficace (e.g., cryptographie, reconnaissance de motifs, Jacobi, algorithmes de graphe et algèbre linéaire creuse). Cependant, pour devenir une ressource de calcul plus largement utilisée dans les systèmes HPC, les FPGA ont besoin d'avoir un modèle de programmation plus simple et d'être mieux intégrés dans les plateformes pour atteindre des bandes passantes plus élevées et des latences plus faibles. Ces motivations ont poussé le développement de nouveaux types de microarchitectures CPU-FPGA pour améliorer l'intégration de ces accélérateurs dans les serveurs de calcul.

3.1. Les différents type de microarchitectures CPU-FPGA

Les microarchitectures CPU-FPGA peuvent être classifiées en fonction de leurs topologies d'interconnexion avec la mémoire du processeur hôte et en fonction de leurs modèles mémoire [12].

Les topologies d'interconnexion se divisent en deux catégories. La première utilise un bus PCI express pour connecter l'ensemble des périphériques au processeur, tels que dans les systèmes Alpha Data Board, Microsoft Catapult, IBM Coherent Accelerator Processor Interface (CAPI). La seconde catégorie utilise de nouvelles technologies (OpenCAPI, QuickPath Interconnect (QPI), Front-Side Bus (FSB)) pour réaliser des connexions point-à-point entre un processeur hôte et un FPGA, tels que dans les systèmes OpenCAPI, Intel-Altera Hardware Accelerator Research Program (HARP) et Convey HC-1.

Les modèles mémoire se divisent également en deux catégories. La première utilise une mémoire séparée et privée sur l'accélérateur, tels que dans les systèmes Alpha Data Board et Microsoft Catapult. Avec ce modèle, l'application hôte doit allouer deux espaces mémoire (un sur le FPGA et un dans la mémoire du processeur) et doit gérer la copie des données. La seconde catégorie utilise une mémoire partagée et cohérente entre le FPGA et le processeur, tels que les systèmes IBM CAPI, OpenCAPI, Intel-Altera HARP et Convey HC-1. Ce modèle permet au FPGA de réaliser des accès directs à la mémoire sans supervision de l'application hôte. Avec ce modèle, l'espace d'adressage utilisé par le FPGA peut être virtuel, les adresses sont traduites par le dernier niveau de cache du processeur, ou l'espace d'adressage peut être physique et les adresses sont traduites par une MMU (Memory Management Unit) implémentée sur le FPGA.

3.2. Influence du modèle de mémoire sur les performances de communications

Il existe un écart important entre les bandes passantes annoncées par les fabricants des technologies d'interconnexion et les bandes passantes mesurées entre les FPGA et la mémoire du processeur. L'étude [13] met en lumière ces écarts sur plusieurs microarchitectures. Par exemple, pour une plateforme intégrant une Alpha Data Board reliée à la mémoire par un bus PCI express (8 Go/s de bande passante annoncée) la bande passante mesurée est de seulement 1.6 Go/s. Les protocoles d'accès à la mémoire liés aux modèles mémoire ont un impact déterminant sur ces performances. Ainsi, pour les systèmes à mémoires séparées la gestion des communications par l'application hôte (allocation de tampons mémoires et copie des données) provoque une chute de la bande passante [12].

Pour les systèmes à mémoire partagée, où le surcoût de gestion logicielle des communications est supprimé grâce aux accès directs à la mémoire, la traduction des adresses virtuelles par le FPGA peut entrainer une dégradation des performances d'accès mémoire. La faible fréquence d'horloge des FPGA entraine un fonctionnement beaucoup plus lent de la MMU que sur le processeur. Ainsi, la traduction des adresses fait doubler la latence d'accès sur l'Intel-Altera HARP [12]. Le temps de chargement de la table des pages sur l'accélérateur rend les miss TLB extrêmement pénalisant faisant chuter la bande passante.

4. Cas d'étude : intégrer un FPGA dans une MVP à l'aide d'OpenCAPI

Les nouvelles microarchitectures CPU-FPGA permettent de partager la mémoire du processeur avec un FPGA. Cependant, le manque de primitives de synchronisations entre les deux ressources limite le modèle de programmation. En implémentant un système mémoire cohérent avec une bande passante élevée et une latence faible entre le processeur et le FPGA, OpenCAPI ouvre la voie à une mémoire virtuellement partagée entre plusieurs nœuds de calcul intégrant des ressources hétérogènes.

4.1. Une MVP pour machines hétérogènes

Dans ces travaux nous considérons une MVP logicielle développée pour les systèmes hétérogènes [14, 15]. Ce système offre un espace logique partagé dans lequel chaque adresse correspond à un *chunk*. Un *chunk* est une unité indivisible de la mémoire dont la taille est définie par l'utilisateur. Cette taille peut varier de l'octet jusqu'à la taille maximale allouable par un *malloc* (e.g., pour une application de traitement d'images la taille d'une ligne d'image). Une allocation dans la MVP peut s'effectuer sur une séquence de *chunks* dont les adresses ne sont pas nécessairement contigües. En revanche, l'espace physique alloué en mémoire locale lors d'un accès à une séquence de *chunks* est contigu afin de permettre l'arithmétique de pointeur.

La MVP est organisée comme un réseau semi-structuré dans lequel des clients exécutent le code utilisateur et effectuent des accès aux données partagées. Chaque client est connecté à un serveur en particulier. Les serveurs sont organisés dans un réseau pair-à-pair et ont la responsabilité de la gestion des données et des métadonnées.

Les accès aux *chunks* s'effectuent sous le contrôle d'un protocole de cohérence. Il est possible de choisir des protocoles de cohérence différents pour chaque *chunk*. Le protocole de cohérence implémenté par défaut est un protocole à 4 états MESI [16] (modifié, exclusif, partagé et invalide). Dans ce protocole, chaque *chunk* est attribué à un nœud référent qui est responsable des informations sur l'état du *chunk*, la version et d'un vecteur de présence indiquant les copies sur les clients et les serveurs.

L'utilisation d'un accélérateur dans le modèle de tâche de la MVP implique de concevoir l'application en suivant un modèle de programmation hybride dans lequel le processeur responsable de l'accélérateur est utilisé comme mandataire pour accéder à la MVP d'une part, et aux primitives de copie des données vers l'accélérateur d'autre part. L'objectif de ce travail est de rendre l'ensemble des services de la MVP accessible à partir des noyaux de calcul déployés sur les accélérateurs, et ainsi permettre de réaliser des accès cohérents et synchronisés, au moyen de primitives d'exclusion mutuelle, à des données pouvant être réparties sur des nœuds distants.

4.2. La microarchitecture CPU-FPGA OpenCAPI

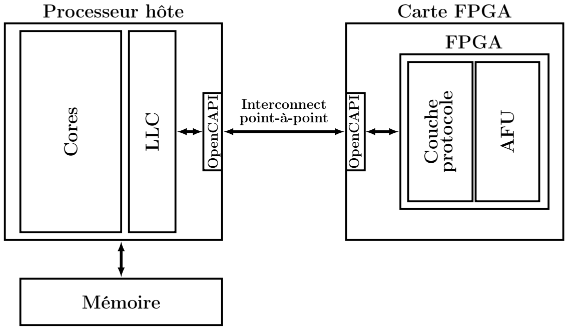

FIGURE 2 – Vue d'ensemble d'une microarchitecture CPU-FPGA avec OpenCAPI

OpenCAPI est un protocole d'interface périphérique qui permet à une tâche exécutée sur un FPGA d'utiliser l'espace d'adressage virtuel de l'application accélérée pour accéder directement à la mémoire du processeur hôte [29]. La figure 2 illustre la topologie d'un tel système. L'accélérateur et le processeur hôte sont reliés par un interconnect point-à-point. La tâche exécutée sur le FPGA (Accelerator Functional Unit (AFU)) utilise des adresses virtuelles pour faire des accès mémoire. Une couche matérielle implémentée sur le FPGA convertit les requêtes d'accès mémoire en requêtes OpenCAPI. Le dernier niveau de cache du processeur (LLC) sert de cache pour le FPGA, il réalise la traduction des adresses virtuelles en adresses physiques, réalise les accès à la mémoire et conserve les données accédées par le FPGA. Ainsi la taille des données partagées avec le FPGA est limitée par la capacité du dernier niveau de cache du processeur. Actuellement, seuls les processeurs Power9 d'IBM intègrent une interface OpenCAPI. Le consortium OpenCAPI annonce avoir mesuré une bande passante de 22 Go/s et une latence de 378 ns entre un Power9 et un FPGA VU3P de Xilinx [28]. Parallèlement, la fondation OpenPOWER a conçu le framework SNAP [1] pour faciliter le développement d'application sur de tels systèmes. Le projet propose notamment un flot de conception intégrant l'outil VivadoHLS et un environnement de simulation permettant de valider le comportement d'une application.

4.3. Principe de la proposition

Le principe de la proposition est de répartir les services d'un serveur de la MVP entre le FPGA et le processeur hôte. La figure 3 est un exemple de topologie de ce système. Deux nœuds de calcul sont connectés par un routeur. Sur chaque nœud un serveur (S) de la MVP gère les copies locales des *chunks* et les requêtes d'acquisition (*acquire*) et de libération (*release*). Sur le nœud 0 un FPGA communique avec le CPU via la mémoire partagée mise en place grâce à la technologie OpenCAPI. Un micro-serveur (µS) implémenté sur le FPGA centralise les requêtes d'acquisition et de libération générées par les noyaux de calcul (µC). Ce micro-serveur mémorise localement l'état des *chunks* accédés par le FPGA (valide après une acquisition et invalide après une libération). Pour conserver l'homogénéité des serveurs de la MVP, une tâche cliente (S-FPGA) est utilisée comme interface entre le micro-serveur du FPGA et un serveur de la MVP. Les nouvelles requêtes sont transmises par le micro-serveur au client S-FPGA qui va réaliser les demandes d'accès ou de libération auprès du serveur auquel il est rattaché. Lorsqu'une requête est satisfaite, le client S-FPGA indique au micro-serveur ses nouveaux droits d'accès. Ces *métadonnées* échangées devraient correspondre à des messages de petites tailles (64 octets, taille d'un flit OpenCAPI) et donc ne devraient pas réduire le débit entre le CPU hôte et le FPGA.

5. Travaux similaires

Les premiers travaux sur les mémoires virtuellement partagées logicielles ont eu pour objectif d'unifier les mémoires de systèmes composés de plusieurs processeurs homogènes disposés en grappes [5, 9, 11, 22] ou en grilles [7, 6, 24]. Plus récemment des approches ont été proposées pour partager virtuellement la mémoire d'un processeur hôte avec celles de ses accélérateurs. Les systèmes A-DSM [19] et RSVM [21] mettent en place des MVP logicielles entre un processeur et un accélérateur de type GPU. Dans ces deux systèmes le processeur est en charge de l'allocation des espaces mémoire, du transfert et de la cohérence des données entre les deux mémoires physiques. Dans le système RSVM une tâche exécutée sur le processeur permet à l'accélérateur de faire des requêtes d'accès mémoire cohérents pendant l'exécution d'un noyau de calcul. Les travaux [30] proposent une solution mixte logicielle/matérielle permettant le partage virtuel de la mémoire d'un processeur avec un accélérateur de type processeur massivement multi-cœurs. Dans notre exposé de principe le processeur n'est pas en charge de l'allocation des espaces dans la mémoire de l'accélérateur, ni du transfert des données. Il se distingue également par le type d'accélérateur visé et par la possibilité de les intégrer dans des systèmes composés de plusieurs nœuds. Dans les travaux [31], une solution mixte logicielle/matérielle a été proposée pour donner à un FPGA des accès cohérents à la mémoire d'un processeur hôte. Cependant, ces travaux ne présentent pas de mécanismes permettant la gestion d'accès concurrents aux données.

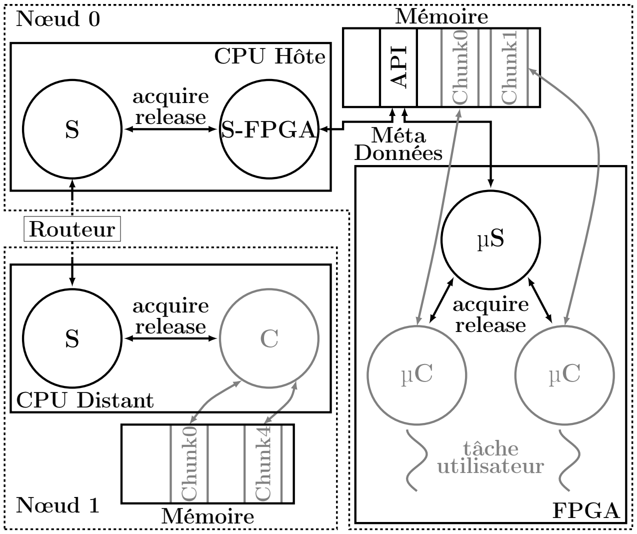

FIGURE 3 – Topologie du système proposé

6. Conclusion

L'hétérogénéité des ressources de calcul dans les systèmes HPC permettent d'atteindre une meilleure efficacité énergétique et d'obtenir plus de puissance de calcul. Cependant, ces performances se payent au prix d'une complexité de programmation plus élevée, notamment due au modèle mémoire répartie. Dans cet article nous avons exposé un principe pour intégrer un FPGA dans une MVP à l'aide de la technologie OpenCAPI et exportable vers d'autres technologies offrant le même modèle mémoire entre le CPU et l'accélérateur. Ce système doit permettre à l'utilisateur de développer un noyau de calcul, exécuté sur un accélérateur reconfigurable, en utilisant un modèle de programmation reposant sur des primitives d'exclusion mutuelle pour réaliser des accès concurrents aux données sans connaître leurs emplacements mémoire.

Bibliographie

1. Capi snap framework hardware and software. – https ://github.com/open-power/snap.
2. Hpe serveur moonshot. – Hewlet Packard Entreprise. https ://www.hpe.com/us/en/servers/moonshot.html.
3. Sierra's system details. – https ://hpc.llnl.gov/hardware/platforms/sierra/.
4. Summit's specifications and features. – https ://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.
5. Amza (C.), Cox (A. L.), Dwarkadas (S.), Keleher (P.), Lu (H.), Rajamony (R.), Yu (W.) et Zwaenepoel (W.). – TreadMarks : Shared memory computing on networks of workstations. *IEEE Computer*, vol. 29, n2, 1996, pp. 18–28.
6. Antoniu (G.) et Bougé (L.). – Dsm-pm2 : A portable implementation platform for multithreaded dsm consistency protocols. – In *Proceedings of the 6th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pp. 55–70. Springer-Verlag, 2001.
7. Antoniu (G.), Bougé (L.) et Jan (M.). – JuxMem : an adaptive supportive platform for data-sharing on the grid. *Scalable Computing : Practice and Experience*, vol. 6, n3, 2005, pp. 45–55.
8. Augonnet (C.), Thibault (S.), Namyst (R.) et Wacrenier (P.-A.). – StarPU : A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation : Practice and Experience, Special Issue : Euro-Par 2009*, vol. 23, 2011, pp. 187–198.
9. Bershad (B. N.), Zekauskas (M. J.) et Sawdon (W. A.). – The Midway distributed shared memory system. – In *Proceedings of the 38th IEEE International Computer Conference*, pp. 528–537, 1993.
10. Bisiani (R.) et Forin (A.). – Multilanguage parallel programming of heterogeneous machines. *IEEE Trans. Comput.*, vol. 37, n8, 1988, pp. 930–945.
11. Carter (J. B.), Bennett (J. K.) et Zwaenepoel (W.). – Implementation and performance of Munin. – In *13th ACM Symposium on Operating Systems Principles*, pp. 152–164, 1991.
12. Choi (Y.-K.), Cong (J.), Fang (Z.), Hao (Y.), Reinman (G.) et Wei (P.). – A quantitative analysis on microarchitectures of modern cpu-fpga platforms. – In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.
13. Choi (Y.-K.), Cong (J.), Fang (Z.), Hao (Y.), Reinman (G.) et Wei (P.). – In-depth analysis on microarchitectures of modern heterogeneous cpu-fpga platforms. *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, n1, 2019.
14. Cudennec (L.). – Software-Distributed Shared Memory over heterogeneous micro-server architecture. – In *Euro-Par 2017 : Parallel Processing Workshops*, 2017.
15. Cudennec (L.). – Merging the Publish-Subscribe Pattern with the Shared Memory Paradigm. – In *Euro-Par 2018 : Parallel Processing Workshops*, 2018.
16. Culler (D.), Singh (J.) et Gupta (A.). – *Parallel Computer Architecture : A Hardware/Software Approach*. – Morgan Kaufmann, 1998, 1st édition. The Morgan Kaufmann Series in Computer Architecture and Design.
17. Escobar (F. A.), Chang (X.) et Valderrama (C.). – Suitability analysis of fpgas for heterogeneous platforms in hpc. *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, n2, 2016, pp. 600–612.
18. Fowers (J.), Brown (G.), Cooke (P.) et Stitt (G.). – A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. – In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 47–56, 2012.
19. Gelado (I.), Stone (J. E.), Cabezas (J.), Patel (S.), Navarro (N.) et Hwu (W.-m. W.). – An asymmetric distributed shared memory model for heterogeneous parallel systems. *SIG-*

PLAN Notices, vol. 45, n3, 2010, pp. 347–358.

20. Griessl (R.), Peykanu (M.), Hagemeyer (J.), Porrmann (M.), Krupop (S.), v. d. Berge (M.), Kiesel (T.) et Christmann (W.). – A scalable server architecture for next-generation heterogeneous compute clusters. – In *2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, pp. 146–153, 2014.
21. Ji (F.), Lin (H.) et Ma (X.). – RSVM : A region-based software virtual memory for GPU. – In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pp. 269–278, 2013.
22. Li (K.). – IVY : a shared virtual memory system for parallel computing. – In *Proc. 1988 Intl. Conf. on Parallel Processing*, pp. 94–101, 1988.
23. Nelson (J.), Holt (B.), Myers (B.), Briggs (P.), Ceze (L.), Kahan (S.) et Oskin (M.). – Latency-tolerant software distributed shared memory. – In *2015 USENIX Annual Technical Conference*, pp. 291–305. USENIX Association, 2015.
24. Nicolae (B.), Antoniu (G.), Bougé (L.), Moise (D.) et Carpen-Amarie (A.). – Blobseer : Next-generation data management for large scale infrastructures. *J. Parallel Distrib. Comput.*, vol. 71, 2011, p. 169–184.
25. Pinheiro (E.), Chen (D.), Dwarkadas (H.), Parthasarathy (S.) et Scott (M.). – S-dsm for heterogeneous machine architectures. 2000.
26. Rinard (M. C.), Scales (D. J.) et Lam (M. S.). – Heterogeneous parallel programming in jade. – In *Proceedings Supercomputing '92*, pp. 245–256, 1992.
27. Ross (J. A.) et Richie (D. A.). – Implementing openshmem for the adapteva epiphany risc array processor. *Procedia Computer Science*, vol. 80, 2016, pp. 2353 – 2356. – International Conference on Computational Science.
28. Slota (M.). – Opencapi technology. – https ://openpowerfoundation.org/wp-content/uploads/2018/04/Myron-Slota.pdf, 2018.
29. Stuecheli (J.), Starke (W. J.), Irish (J. D.), Arimilli (L. B.), Dreps (D.), Blaner (B.), Wollbrink (C.) et Allison (B.). – Ibm power9 opens up a new era of acceleration enablement : Opencapi. *IBM Journal of Research and Development*, vol. 62, n4/5, 2018, pp. 8 :1–8 :8.
30. Vogel (P.), Marongiu (M.) et Benini (L.). – Lightweight virtual memory support for zero-copy sharing of pointer-rich data structures in heterogeneous embedded socs. *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, n7, 2017, pp. 1947–1959.
31. Vogel (P.), Marongiu (M.) et Benini (L.). – Exploring shared virtual memory for fpga accelerators with a configurable iommu. *IEEE Transactions on Computers*, vol. 68, n4, 2019, pp. 510–525.
32. Zhou (S.), Stumm (M.), Li (K.) et Wortman (D.). – Heterogeneous distributed shared memory. *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, n5, 1992, pp. 540–554.