Interactions entre calculs et communications au sein des systèmes HPC distribués

Philippe Swartvagher

Compas'2021 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, 5-9 juillet 2021

Interactions entre calculs et communications au sein des systèmes HPC distribués

Philippe SWARTVAGHER

Inria Bordeaux – Sud-Ouest,
200 Avenue de la Vieille Tour
33405 Talence - France
philippe.swartvagher@inria.fr

Résumé
Les supports d'exécution parallèles distribués permettent généralement d'exécuter simultanément les calculs et les communications pour atteindre de meilleures performances. Cet article étudie les interactions entre les calculs et les communications qui peuvent pénaliser leurs performances respectives. Nous présentons principalement les effets de la contention mémoire, cause principale de la dégradation des performances du calcul et des communications exécutés simultanément. Nous montrons que cette contention peut fortement pénaliser les performances du réseau, mais également perturber les calculs. Nous nous intéressons également aux différents facteurs qui font varier la contention mémoire et donc l'impact sur les performances.

Mots-clés : HPC, MPI, contention, mémoire

---

1. Introduction

Réaliser simultanément des calculs et des communications (par exemple avec des routines MPI non-bloquantes) est une pratique maintenant courante qui permet d'obtenir de meilleures performances. Cependant, il a été observé [5, 4] que dans certains cas, ce paradigme mène à des performances inférieures à ce qui aurait pu être obtenu si le calcul et les communications avaient été séquentialisés.

Puisque les interactions possibles entre le calcul et les communications ne sont pas beaucoup détaillées dans la littérature, mais seulement mentionnées, nous proposons d'étudier plus en détails les causes de ces interactions et de mesurer leurs impacts. Nous présentons principalement le cas de la contention mémoire, qui perturbe le plus les performances. Nous ciblons les systèmes HPC, ainsi seuls les réseaux rapides (INFINIBAND et OMNIPATH) ont été considérés.

En étudiant le cas de la contention mémoire provoquée par le calcul et les communications, nous montrons que cette contention peut pénaliser à la fois les calculs et les communications. Cet impact négatif peut varier selon le placement des données, le placement du *thread* réalisant les communications, la taille des messages échangés sur le réseau et l'intensité arithmétique des calculs. Nous présentons les impacts de chacun de ces paramètres.

L'article suit le plan suivant : la section 2 présente le contexte expérimental, la section 3 explore en détails les effets de la contention mémoire, la section 4 présente les travaux connexes dans la littérature et la dernière section conclut l'article.

2. Cadre expérimental

Pour mesurer les performances des communications et des calculs lorsqu'ils sont exécutés simultanément, nous avons développé un programme de *benchmark* reposant sur MPI et OpenMP. Un thread est dédié aux communications : il réalise des ping-pongs pour mesurer la latence et la bande-passante du réseau ; les autres *threads* font des tâches de calcul.

2.1. Protocole expérimental

Pour pouvoir comparer les performances des communications et des calculs lorsqu'ils sont exécutés simultanément ou séquentiellement, notre programme comprend plusieurs phases : (1) calculs seuls (sans communications en même temps), (2) communications seules (sans calculs en même temps), et (3) communications et calculs en même temps. Les calculs et les communications utilisent des données différentes et sont complètement indépendants les uns des autres. Chaque *thread* (ceux qui exécutent des calculs et celui qui communique) est *bindé* sur un cœur différent, ce qui a pour but de stabiliser les performances et assurer une meilleure reproductibilité. Une exécution de notre programme mesure soit la latence réseau, soit la bande-passante, mais pas les deux à la fois. Pour mesurer la latence, les ping-pongs échangent 4 octets à chaque fois (un flottant) et la bande-passante est mesurée en échangeant 64 Mo de données.
Tous les graphiques présentés par la suite comparent les performances des calculs et des communications exécutés séparément (courbes en traits pleins) et les performances lorsqu'ils sont exécutés ensemble (courbes en pointillés).

2.2. Description des machines utilisées

Nous avons exécuté notre programme [1] sur plusieurs grappes de serveurs avec des caractéristiques différentes. Les résultats étant dans la majorité des cas similaires sur tous les serveurs utilisés, nous présenterons ici principalement les résultats obtenus sur les nœuds `henri`. Ces derniers sont composés de deux INTEL Xeon Gold 6140 cadencés à 2.3 GHz, avec 36 cœurs répartis sur 4 bancs NUMA totalisant 96 Go de mémoire RAM et sont équipés de carte réseau INFINIBAND ConnectX-4 EDR. Sur toutes les machines utilisées, l'*hyperthreading* est désactivé.
Les nœuds `henri` fonctionnent avec LINUX 5.7.7, DEBIAN 10 et GCC 9.3. Les résultats présentés utilisent MADMPI, l'interface MPI de NEWMADELEINE [1], mais des résultats similaires ont été observés avec d'autres bibliothèques MPI, comme OPENMPI 4.0.

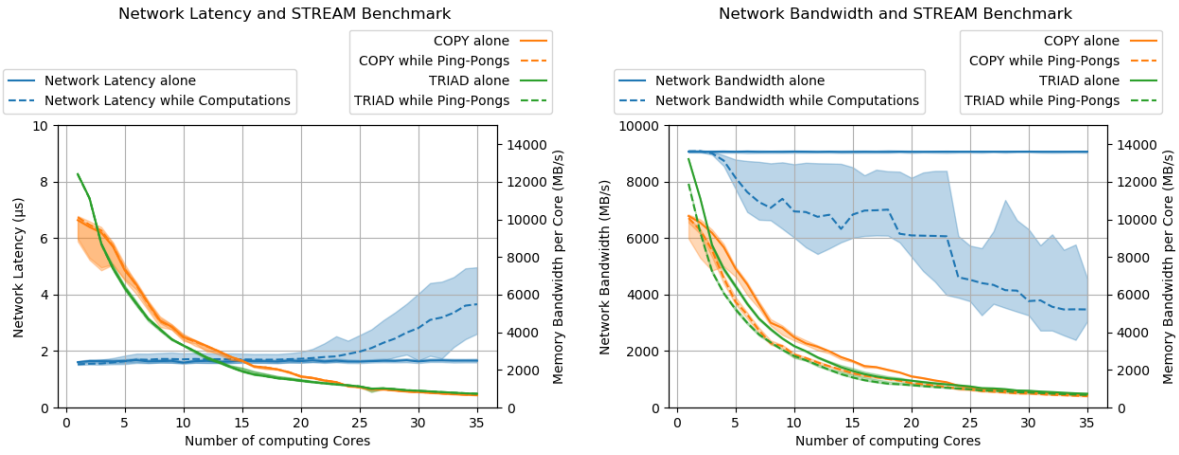
3. Contention mémoire

Le même bus mémoire est utilisé par les données circulant entre la mémoire vive et le processeur pour les calculs et par les données échangées sur le réseau. Nous montrons dans cette section qu'il peut y avoir de la contention entre les transferts des données servant aux communications et ceux des données servant aux calculs.

3.1. Provoquer de la contention mémoire

Nous générons de la contention mémoire à l'aide des routines du *benchmark* STREAM [6], qui font de simples opérations arithmétiques sur de grands tableaux : `COPY` ($b[i] \leftarrow a[i]$) et `TRIAD` ($c[i] \leftarrow a[i] + C \times b[i]$). Dans ces opérations, les accès mémoire sont le facteur limitant des performances, ce qui cause un trafic important sur le bus mémoire. Les boucles parcourant les tableaux sont parallélisées à l'aide d'OpenMP. Les performances sont mesurées par la bande-passante mémoire utilisée par cœur (on souhaite *maximiser* ces valeurs).

1. Disponible sous licence libre : `https://gitlab.inria.fr/pswartva/memory-contention`



(a) Mesure de la latence réseau

(b) Mesure de la bande-passante réseau

FIGURE 1 – Performances des calculs limités par la mémoire et des communications réseaux

3.2. Impact de la contention mémoire

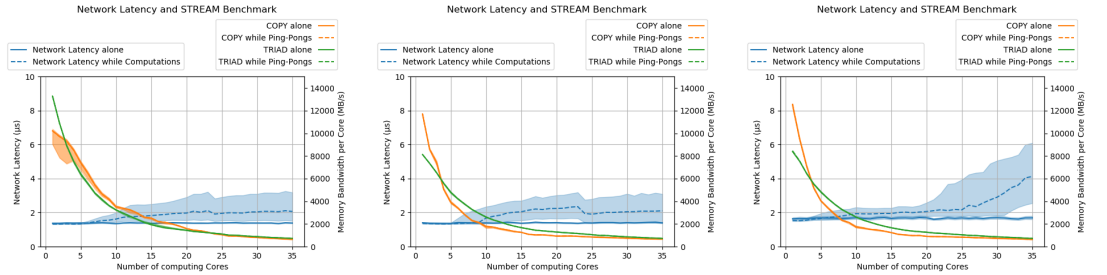
Pour l'exécution dont les performances sont représentées sur la FIGURE 1, la mémoire est allouée sur le banc NUMA auquel la carte réseau est connectée, le *thread* de communication est *bindé* sur le dernier cœur d'un autre banc NUMA, et les *threads* de calculs sont *bindés* sur les autres coeurs. La FIGURE 1a montre que la latence réseau est impactée lorsque au moins 22 cœurs exécutent les opérations de STREAM ; elle peut même doubler lorsque tous les cœurs disponibles font du calcul. En revanche, le calcul n'est pas impacté par les ping-pongs pour mesurer la latence. Le débit réseau est impacté beaucoup plus tôt : dès que 3 cœurs exécutent du calcul (FIGURE 1b). Lorsque tous les cœurs disponibles calculent, la bande-passante réseau n'atteint que le tiers de ses performances nominales. Lors des ping-pongs pour mesurer la bande-passante réseau, la bande-passante mémoire disponible pour le calcul est maintenant diminuée : ce qui est logique puisque les ping-pongs transfèrent alors plus de données (64 Mo).

3.3. Impact du placement des *threads* et des données

Nous étudions ici l'impact du placement des données manipulées et du *thread* de communication. Ce dernier est *bindé* soit sur un cœur du même banc NUMA auquel est branchée la carte réseau (*proche*), soit sur un coeur d'un autre banc NUMA (*loin*). De la même manière, nous allouons la mémoire utilisée par les calculs et les communications sur un banc NUMA proche ou loin de la carte réseau. Il est déjà établi que le placement [7] a un impact sur la performance des communications, nous cherchons à savoir si la contention mémoire aggrave ce phénomène.

La FIGURE 2 est complétée par la FIGURE 1, qui présente le cas « données *proches*, *thread* MPI *loin* ». Lorsque le *thread* de communication est proche de la carte réseau, la latence augmente dès que 6 cœurs sont utilisés pour du calcul, puis se stabilise autour de 2 µs. Lorsque ce thread est loin de la carte réseau, la latence augmente considérablement à partir de 25 *threads* qui calculent, et double même sa valeur nominale. Sans calcul simultanément, la latence est légèrement meilleure lorsque le *thread* de communication est près de la carte réseau (1.39 µs *vs* 1.67 µs) : les petits messages sont envoyés directement du CPU à la carte réseau, ainsi si le thread de communication est plus près de la carte réseau, la latence est meilleure.

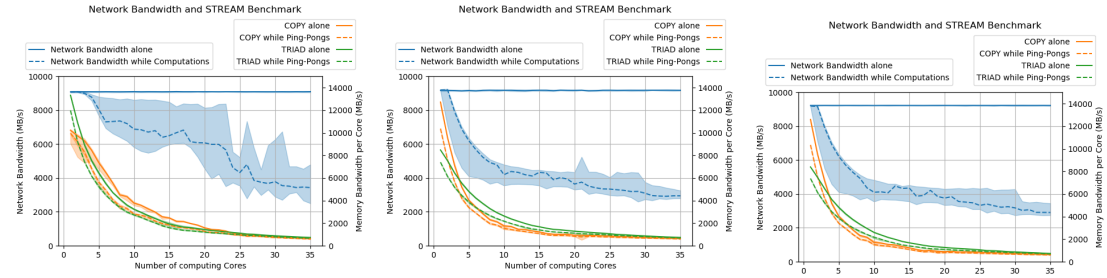
Globalement, le débit réseau ne dépend pas du placement du thread de communication, mais du placement des données : lorsqu'elles sont sur le même banc NUMA que celui auquel est



(a) Latence réseau, données : *proche*, *thread* MPI : *proche*

(b) Latence réseau, données : *loin*, *thread* MPI : *proche*

(c) Latence réseau, données : *loin*, *thread* MPI : *loin*



(d) Bande-passante réseau, données : *proche*, *thread* MPI : *proche*

(e) Bande-passante réseau, données : *loin*, *thread* MPI : *proche*

(f) Bande-passante réseau, données : *loin*, *thread* MPI : *loin*

FIGURE 2 – Impact du placement des données et du *thread* MPI

connectée la carte réseau, la bande-passante décroît proportionnellement au nombre de *threads* qui font du calcul. Lorsque les données sont loin de la carte réseau, la bande-passante réseau chute plus brutalement : les grands messages transmis sur le réseau étant envoyés avec un mécanisme de DMA (*Direct Memory Access*), le facteur important ici est le placement des données.

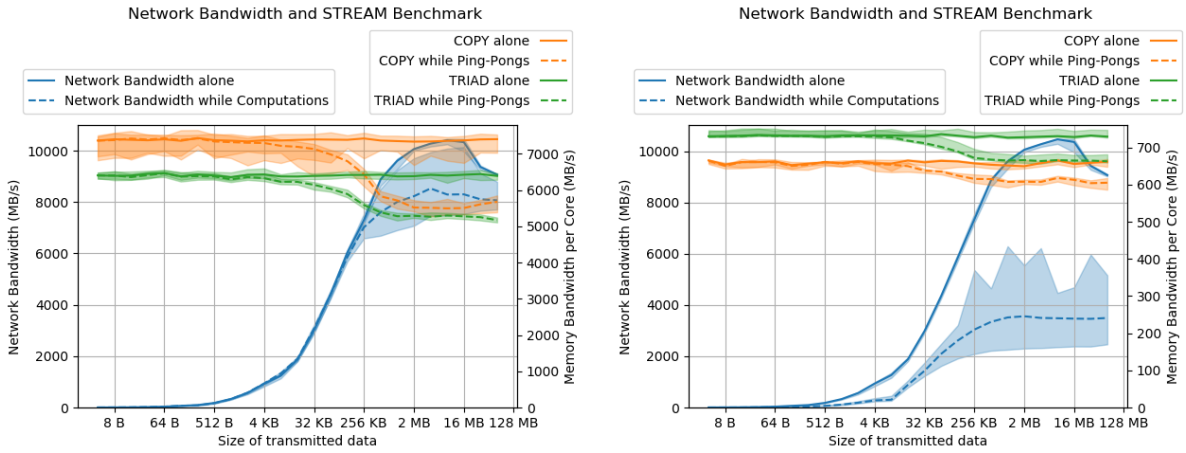
Dans toutes les configurations, les ping-pongs mesurant la latence réseau ne perturbent pas les opérations de STREAM, mais les mesures de bande-passante réseau font perdre jusqu'à 25 % (avec 5 *threads* qui calculent) de bande-passante mémoire disponible pour STREAM.

Pour résumer, quand le *thread* de communication est loin de la carte réseau, la latence réseau est plus sensible à la contention sur le bus mémoire. La bande-passante réseau est plus impactée lorsque les données sont loin de de la carte réseau. Dans tous les cas, transmettre de grands messages sur le réseau pénalise plus les calculs.

3.4. Impact de la taille des messages transmis sur le réseau

Nous nous intéressons maintenant à l'impact de la taille des messages communiqués. Les précédentes expériences nous ont montré que les performances des calculs et du réseau varient différemment si nous mesurons la latence ou le débit (la taille des messages étant différente). Nous mesurons donc maintenant les performances obtenues par STREAM et par le réseau selon la taille des messages échangés. L'expérience est faite dans le cas où STREAM est le plus impacté par les communications (avec 5 *threads* qui calculent) dans le cas où la bande-passante réseau est le plus impactée (avec 35 *threads* qui calculent), comme montré par la FIGURE 1b.

Avec 5 cœurs qui exécutent les calculs de STREAM (FIGURE 3a), les communications commencent à se dégrader pour des messages de 64 Ko, mais le calcul est impacté plus tôt : à partir de 4 Ko transmis. Lorsque 35 cœurs calculent (FIGURE 3b), le réseau est impacté plus tôt qu'avec 5 cœurs : à partir de 128 octets et les calculs sont toujours gênés à partir de 4 Ko par message.



(a) 5 cœurs qui calculent

(b) 35 cœurs qui calculent

FIGURE 3 – Impact de la taille des messages transmis

D'une part, plus il y a de cœurs qui calculent, plus la pression sur le bus mémoire est élevée, impactant le réseau. D'autre part, transmettre de grands messages sur le réseau peut causer suffisamment de trafic mémoire pour perturber les calculs, même réalisés par seulement 5 cœurs.

3.5. Des applications plus ou moins limitées par le débit mémoire

Tous les codes de calculs ne sont pas sensibles de la même manière à la bande-passante mémoire qui leur est mise à disposition. Les calculs exécutés par STREAM sont un cas extrême où le débit mémoire est le facteur limitant. À un autre extrême, certains programmes peuvent être limités par la vitesse de calcul du processeur. La pression exercée sur la mémoire par un calcul est définie par son *intensité arithmétique* (comme utilisée dans le *roofline model* [8]). Elle est définie par le nombre de FLOPs par octet de données manipulé. Plus cette valeur est faible, plus la vitesse du calcul sera limitée par le débit mémoire.

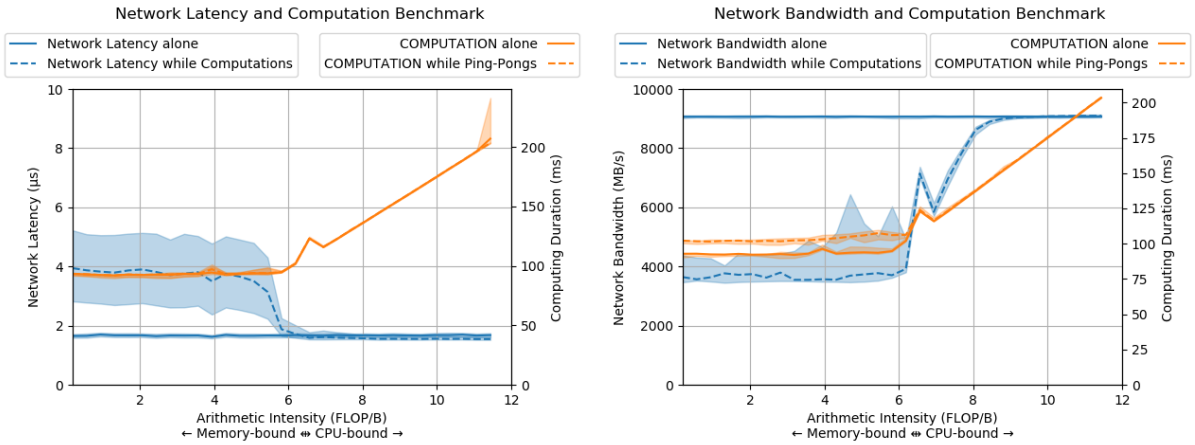
Pour voir l'impact de l'intensité arithmétique des calculs exécutés en parallèle des communications, nous avons adapté l'opération `TRIAD` de STREAM pour pouvoir faire varier son intensité arithmétique. En pratique, nous avons ajouté une boucle qui répète un certain nombre de fois les opérations sur un même élément du tableau, avant de passer à l'élément suivant.

Les résultats de cette expérience, présentés par la FIGURE 4, montrent que des calculs avec une très faible intensité arithmétique ont un fort impact sur les communications. Avec une intensité arithmétique inférieure à 6 flop/octet, la pression sur le bus mémoire gêne les communications : la latence double alors que la durée du calcul reste constante (le calcul est bien limité par les accès mémoire et les petits messages échangés ne gênent pas le calcul). Le débit réseau baisse de 60 % et le calcul est ralenti de 10 % (les messages plus grands perturbent plus le calcul).

Lorsque l'intensité arithmétique dépasse 6 flop/octet, les calculs deviennent limités par la vitesse des opérations arithmétiques et les besoins en débit mémoire diminuent : les calcul et les communications ne se gênent plus mutuellement puisqu'il n'y a plus de contention mémoire.

4. Travaux connexes

La contention mémoire causée par les communications et le calcul est observée par CHAI *et al.* [3], mais sans mesurer les impacts de cette contention. BALAJI *et al.* ont mesuré [2] le tra-



(a) Latence réseau

(b) Bande-passante réseau

FIGURE 4 – Impact de l'intensité arithmétique des calculs sur les communications

fic mémoire causé par les communications utilisant RDMA sur des réseaux INFINIBAND à 10 Gbps, mais sans s'intéresser aux interactions avec des calculs faits simultanément. Un modèle théorique du partage du débit mémoire entre calculs et communications a été proposé par LANGGUTH *et al.* [5]. Ce modèle nécessite qu'une fois que le calcul ou la communication est terminée, le calcul ou la communication exécuté simultanément reprenne alors sa performance nominale. Cependant, nous visons des cas où il y a en permanence des calculs et des communications, ils ne peuvent donc jamais obtenir tout le débit mémoire offert par le système. GROVES *et al.* ont présenté le phénomène NiMC (*Network-induced Memory Contention*) [4] : ils ont étudié la contention mémoire produite par les communications avec et sans RDMA pour une large gamme d'applications. Cependant, ils ne se sont intéressés qu'aux performances des calculs et nous appliquons déjà les solutions qu'ils proposent pour réduire la contention mémoire.
Notre travail se distingue de ce qui est déjà présent dans la littérature puisqu'il mesure à la fois l'impact sur le calcul et sur les communications, s'intéresse à l'impact de la taille des messages échangés et évalue également l'impact de l'intensité arithmétique des calculs.

5. Conclusion

Exécuter en parallèle du calcul et des communications est une pratique qui permet d'obtenir globalement de meilleures performances. Cependant, un tel paradigme peut mener à des performances sous-optimales du calcul et des communications. L'objectif de ce papier était d'étudier et de mesurer les différentes interactions possibles entre les calculs et les communications qui mènent à ces dégradations de performances. Nos expériences montrent que la contention mémoire produite par les mouvements des données nécessaires aux calculs et aux communications peut avoir un fort impact sur les performances des calculs et des communications. Cette contention mémoire (et donc son impact) varie selon le placement des données utilisées et du *thread* de communication, la taille des messages échangés et l'intensité arithmétique des calculs.
Ces expériences doivent encore être complétées en mesurant ces interactions sur d'autres processeurs (AMD et ARM), mais également en mesurant l'impact sur de « véritables » applications numériques qui souffriraient de ces interactions. Un objectif plus avancé est de prédire ces interactions pour prendre des décisions qui minimiseraient ces impacts.

Remerciements

This work is supported by the Agence Nationale de la Recherche, under grant ANR-19-CE46-0009.
This work is supported by the Région Nouvelle-Aquitaine, under grant 2018-1R50119 *HPC scalable ecosystem*.
Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/`).
This work was granted access to the HPC resources of CINES under the allocation 2019-A0060601567 attributed by GENCI (Grand Equipement National de Calcul Intensif).
L'auteur tient à remercier Alexandre DENIS, Emmanuel JEANNOT, Brice GOGLIN, Amina GUERMOUCHE et Samuel THIBAULT pour leur aide et conseils précieux apportés pour la réalisation de ce travail.

Bibliographie

1. Aumage (O.), Brunet (E.), Furmento (N.) et Namyst (R.). – NewMadeleine : a Fast Communication Scheduling Engine for High Performance Networks. – In *Workshop on Communication Architecture for Clusters (CAC 2007)*, Long Beach, California, United States, mars 2007.
2. Balaji (P.), Shah (H.) et Panda (D.). – Sockets vs RDMA Interface over 10Gigabit Networks : An In-depth analysis of the Memory Traffic Bottleneck. 01 2004.
3. Chai (L.), Gao (Q.) et Panda (D. K.). – Understanding the Impact of Multi-Core Architecture in Cluster Computing : A Case Study with Intel Dual-Core System. – In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pp. 471–478, 2007.
4. Groves (T.), Grant (R. E.) et Arnold (D.). – NiMC : Characterizing and Eliminating Network-Induced Memory Contention. – In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 253–262, 2016.
5. Langguth (J.), Cai (X.) et Sourouri (M.). – Memory Bandwidth Contention : Communication vs Computation Tradeoffs in Supercomputers with Multicore Architectures. – In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 497–506, 2018.
6. McCalpin (J.). – Memory bandwidth and machine balance in high performance computers. *IEEE Technical Committee on Computer Architecture Newsletter*, 12 1995, pp. 19–25.
7. Moreaud (S.) et Goglin (B.). – Impact of numa effects on high-speed networking with multi-opteron machines. 11 2007.
8. Williams (S.), Waterman (A.) et Patterson (D.). – Roofline : an insightful visual performance model for multicore architectures. *Communications of the ACM*, vol. 52, n4, 2009, pp. 65–76.