UVSQ université PARIS-SACLAY | ISTY Institut des Sciences et Techniques des Yvelines CAMPUS DE MANTES EN YVELINES CAMPUS DE SAINT-QUENTIN-EN-YVELINES

LI-PARAD Laboratoire d'informatique parallélisme, réseaux, algorithmes, distribués | UVSQ université PARIS-SACLAY

Rapport de Stage

NDITAR Eliel Timon

10 août 2023

Encadrant académique :
Cédric Valensi

Encadrant académique :
William Jalby

REMERCIEMENTS

Je souhaite exprimer ma gratitude envers toute l'équipe MAQAO qui m'a encadré tout au long de ce stage. Je tiens à remercier chaleureusement Kevin Camus, Aurélien Delval et Max Hoffer pour leur soutien et leur accompagnement. Je voudrais également adresser mes remerciements spéciaux à Cedric Valensi, mon directeur d'équipe, ainsi qu'à William Jalby, mon directeur d'entreprise, pour leur précieuse guidance et leur confiance tout au long de ce stage.

RÉSUMÉ

Ce rapport présente le contenu du stage que j'effectue depuis plus d'un mois. Au cours de cette période, j'ai eu l'opportunité de travailler sur différentes architectures GPU et d'utiliser plusieurs outils d'analyse de performance, notamment RocProf, Nsight System et HPCToolkit. L'objectif principal est de profiler des codes issus de la bibliothèque de chimie quantique (QMCKL) qui s'exécutent sur des GPU et de comparer les résultats obtenus à l'aide des outils fournis par le constructeur (RocProf ou Nsight System) avec ceux obtenus grâce à HPCToolkit.

INTRODUCTION

Les GPU ont révolutionné le calcul intensif (HPC) en fournissant une puissance de calcul parallèle inégalée. Initialement destinés aux applications graphiques, ils sont devenus des composants essentiels dans le domaine du HPC, permettant d'accélérer diverses applications scientifiques et techniques. Dans ce contexte, les outils de profilage jouent un rôle crucial pour optimiser les performances des applications exécutées sur les GPU. Ils permettent d'analyser et d'évaluer le comportement des codes parallèles, en identifiant les goulots d'étranglement, les inefficacités et les opportunités d'optimisation. La comparaison des résultats obtenus par différents outils de profilage pour un même code exécuté sur les GPU revêt une importance particulière. Cette comparaison vise à valider la fiabilité des données collectées et à renforcer la confiance dans les informations fournies par les outils de profilage. De plus, elle permet de mettre en évidence les points forts et les faiblesses de chaque outil, facilitant ainsi le choix de l'outil le mieux adapté à nos besoins spécifiques.

I. CADRE DU STAGE

1. CONTEXTE DU STAGE

Le stage se déroule au sein du laboratoire Li-Parad, affilié à l'ISTY et à l'Université de Versailles Saint-Quentin-en-Yvelines (UVSQ). Au sein de ce laboratoire, je fais partie de l'équipe MAQAO, qui se consacre au développement d'un outil d'analyse et de profilage des performances d'applications destinées à s'exécuter sur des machines de calcul haute performance.
L'objectif principal de mon stage est de comparer les résultats du profilage d'applications pour GPU sur différentes architectures. Pour ce faire, j'utilise plusieurs outils de profilage, tels que RocProf, Nsight System et HPC Toolkit. Je compare les résultats obtenus avec l'outil fourni par le constructeur avec ceux d'un outil tiers. Ces comparaisons sont réalisées sur différentes architectures GPU, notamment NVIDIA et AMD.
Au sein de l'équipe MAQAO, je travaille spécifiquement dans l'entité dédiée aux GPU, en collaboration avec Aurélien Delval et Max Hoffer, qui sont les responsables de cette entité

2. LE LI-PARAD

Sur le site https://www.liparad.uvsq.fr/presentation-du-laboratoire on peut lire:
"Le *Li-PaRAD (Laboratoire d'informatique en Parallélisme Réseaux et Algorithmique Distribuée) rassemble des chercheurs travaillant sur les problématiques liées à l'utilisation coordonnée de plusieurs entités de calcul : parallélisme, réseau et algorithmique distribuée. Cette problématique, qui est fondamentale et traditionnelle en informatique, gagne en actualité et en importance avec les évolutions technologiques récentes. La grande majorité des processeurs commerciaux sont désormais multicœurs, et la communication de données numériques est omniprésente.*
Le Li-PaRAD est l'une des principales structures de recherche universitaire en informatique haute performance de l'Université Paris-Saclay, avec la particularité d'associer des compétences reconnues en réseaux, algorithmique distribuée, modélisation et optimisation. Cette combinaison permet d'appréhender les évolutions récentes des calculateurs haute performance : l'étude des protocoles réseau est essentielle dans la tendance actuelle à mêler calcul parallèle et distribué ; les concepts d'asynchronisme et de fautes, largement étudiés en algorithmique distribuée, permettent d'aborder de manière plus précise, de la théorie à la pratique, la manière optimale d'organiser les calculs. Enfin, les compétences en modélisation et en optimisation sont nécessaires à la fois dans l'étude des systèmes de calcul haute performance et dans la résolution pratique de problèmes académiques ou industriels à l'aide de cet outil ".

3. PRÉSENTATION DE L'ÉQUIPE ET DE MAQAO

- **Equipe :**

- **Cedric Valensi**: Directeur de l'équipe MAQAO, a développé le désassembleur de MAQAO (MADRAS)
- **Kevin Kamus**: Optimise et analyse les performances des codes de nos différents partenaires.
- **Emmanuel Oseret**: Travaille principalement sur la gestion du profilage d'applications OpenMP ainsi que sur l'analyse des différentes architectures de processeurs.
- **Mathieu Tribalat**: Travail sur l'interface OneView qui permet de visualiser les résultats de MAQAO.
- **Hugo Bolloré**: Travaille sur le portage des outils MAQAO sur les CPU ARM.
- **Jäsper Salah Ibnamar**: Travaille sur l'analyse des performances mémoire et assure l'administration/sécurité des serveurs et services du laboratoire.
- **Aurélien Delval**: Responsable GPU.
- **Max Hoffer**: Responsable GPU.

- **MAQAO :**

Sur le site web de MAQAO on peut lire:

"MAQAO (Modular Assembly Quality Analyzer and Optimizer) is a performance analysis and optimization framework operating at binary level with a focus on core performance. Its main goal of is to guide application developpers along the optimization process through synthetic reports and hints.

MAQAO mixes both dynamic and static analyses based on its ability to reconstruct high level structures such as functions and loops from an application binary. Since MAQAO operates at binary level, it is agnostic with regard to the language used in the source code and does not require recompiling the application to perform analyses. MAQAO has also been designed to concurrently support multiple architectures. Currently the Intel64 and Xeon Phi architectures are implemented."

Plus d'information sur http://www.maqao.org/

II. DOCUMENTATION

Différentes notions seront explorées, et il est essentiel de donner un aperçu rapide des technologies, des bibliothèques et des outils qui seront utilisés.

1. GPU

Sur Wikipédia (https://fr.wikipedia.org/wiki/Processeur_graphique) le GPU est présenté en ces termes:

" *Un **processeur graphique**, ou **GPU** (de l'anglais Graphics Processing Unit), également appelé **coprocesseur graphique** sur certains systèmes, est une unité de calcul assurant les fonctions de calcul d'image. Il peut être présent sous forme de circuit intégré (ou puce) indépendant, soit sur une carte graphique ou sur la carte mère, ou encore intégré au même circuit intégré que le microprocesseur général (on parle d'un SoC lorsqu'il comporte toutes*

les puces spécialisées). Dans la terminologie d'AMD un GPU intégré à la même puce que le processeur est appelé APU, et dans la terminologie d'Intel un IGP). La sortie peut être affichée à l'écran, ou retraité en RAM ou VRAM pour différents usages, telle que l'écriture sur mémoire de masse ou un nouveau traitement interne. Un processeur graphique a généralement une structure hautement parallèle (voir accélération matérielle) qui le rend efficace pour une large palette de tâches graphiques comme le rendu géométrique 2D ou 3D, s'opérant généralement dans une mémoire vidéo dédiée. Une unité de traitement du signal vidéo peut être couplée à ce processeur pour le décodage ou l'encodage de vidéo dans différents formats tels que Mpeg, etc.. Enfin, Ce parallélisme permet également le calcul intensif en général, lorsqu'il s'agit d'un ***GPGPU*** *(anglais : General Purpose GPU, signifiant GPU à vocation généraliste).* "

Deux principaux acteurs dominent le marché des GPU et c'est sur leurs architectures que le profilage sera effectué:

- **NVIDIA**: Leader, il est reconnu pour ses architectures GPU avancées, telles que la série NVIDIA Tesla. Ces GPU offrent des performances exceptionnelles pour les applications HPC et sont utilisés dans de nombreux supercalculateurs et centres de recherche. Leur matériel est programmable via du CUDA qui est un langage propriétaire.
- **AMD**: Il propose également des solutions GPU performantes, notamment avec sa série AMD Radeon Instinct. Ces GPU offrent une alternative compétitive pour le HPC, avec des performances et des fonctionnalités adaptées aux besoins des chercheurs et des scientifiques. Ils ont un langage propriétaire mais qui est open source : le HIP.

Afin de mener à bien les comparatifs, j'ai pu bénéficier d'accès à deux clusters de calcul : amdGPU et Turpan. Le cluster amdGPU est équipé de GPU AMD, tandis que le cluster Turpan est composé de GPU NVIDIA.

2. OpenMP ET OpenACC

- OpenMP:

  D'après la définition de Wikipédia: " ***OpenMP*** *(Open Multi-Processing) est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Cette API est prise en charge par de nombreuses plateformes, incluant GNU/Linux, OS X et Windows, pour les langages de programmation C, C++ et Fortran. Il se présente sous la forme d'un ensemble de directives, d'une bibliothèque logicielle et de variables d'environnement.* "
- OpenACC:

  En se renseignant sur Wikipedia on peut lire :Sur Wikipédia https://fr.wikipedia.org/wiki/OpenACC on peut lire: "***OpenACC*** *(pour* ***Open Acc****elerators) est un standard de programmation pour le calcul parallèle développé par Cray, CAPS, Nvidia et PGI (en). Ce standard est conçu pour permettre du calcul parallèle sur des systèmes hétérogènes CPU/GPU. Tout comme OpenMP, il est possible de rajouter des*

commandes dans du code source C, C++ et Fortran pour identifier des portions qui pourrait bénéficier d'une accélération, en utilisant des directives au compilateur. "
Plus d'informations sur https://www.openacc.org/

Les kernels utilisés pour les benchmarks ont été adaptés pour s'exécuter sur des GPU en utilisant les directives OpenMP et OpenACC. Cependant, pour nos comparaisons, seules les versions utilisant OpenMP ont été prises en compte.

3. OUTILS D'ANALYSE DE PERFORMANCE

Il existe différents outils de profiling sur GPU. Nous nous sommes principalement concentrés sur **RocProf**, **Nsight Systems** et **HPCToolkit**.

3.1. ROCPROF

Sur la documentation de RocProf https://rocm.docs.amd.com/projects/rocprofiler/en/latest/rocprof.html on peut lire : " *Le rocProf est un outil en ligne de commande qui utilise les API rocProfiler et rocTracer pour collecter des données de profilage sur les applications. Il est implémenté sous forme de script et permet d'attacher le profilage à une application spécifiée. Le rocProf utilise des plugins de profilage pour collecter des métriques, des traces matérielles et des traces d'activité de l'API d'exécution. Les données de profilage sont fournies en sortie dans différents formats et peuvent être visualisées à l'aide de Google Chrome tracing. Le rocProf facilite ainsi l'analyse et la compréhension des performances des applications.* "

3.2. NSIGHT SYSTEMS

Nsight Systems est un outil de profilage et d'optimisation de performances développé par NVIDIA. Il est spécialement conçu pour les développeurs travaillant sur des applications GPU, notamment dans les domaines du calcul intensif, de la réalité virtuelle et des jeux vidéo. Nsight Systems offre une visibilité approfondie sur le comportement de l'application et du GPU, en collectant des données sur les appels de fonctions, les transferts de données, les accès mémoire et d'autres activités clés.
Plus d'informations sur https://docs.nvidia.com/nsight-systems

3.3. HPCTOOLKIT

HPCToolkit est une suite d'outils intégrée qui permet de mesurer et d'analyser les performances des programmes, en particulier ceux qui sont accélérés par GPU. Il fournit des mesures précises du travail CPU, de la consommation de ressources et de l'inefficacité des programmes, en les associant au contexte d'appel complet. Il permet également de surveiller

les opérations GPU, de collecter des métriques au niveau de l'instruction à l'intérieur des kernels GPU et d'attribuer les coûts du travail GPU aux contextes d'appel hétérogènes. HPCToolkit est conçu pour une utilisation sur de grands systèmes parallèles et prend en charge une variété de types de codes, y compris les codes séquentiels, à thread, MPI et hybrides, ainsi que les codes accélérés par GPU. Ses outils de présentation permettent une analyse rapide des coûts d'exécution, de l'inefficacité et des caractéristiques de mise à l'échelle des programmes.
Plus d'informations sur http://hpctoolkit.org

4. TREX ET QMKL

TREX :

TREX est un projet européen qui vise à soutenir la recherche en chimie quantique dans le domaine du calcul haute performance (HPC). L'objectif du projet est de développer un code de simulation quantique haute précision qui peut rivaliser avec d'autres méthodes de simulation existantes. Le projet réunit de nombreuses petites et moyennes entreprises (PME) ainsi que des centres de recherche européens, dans le but de développer des solutions qui exploitent pleinement les capacités des ordinateurs exascale. TREX se concentre sur le développement de logiciels open source dans le domaine des simulations de chimie quantique stochastique. Parallèlement, TREX explore également des cas d'utilisation commerciale pour tirer parti de cette méthodologie, ainsi que pour développer et mettre en œuvre des composants logiciels et des services facilitant l'utilisation des ressources HPC par les opérateurs commerciaux et les communautés d'utilisateurs.
Pour plus d'informations, vous pouvez consulter le site web du projet : https://trex-coe.eu/

QMCKL :

QMCKL signifie : Quantum Monte Carlo Kernel Library, Plus d'informations sur https://trex-coe.eu/qmckl-library-software-developers-and-hpc-experts-0
Dans le cadre du projet **TREX**, des programmes de pointe pour les calculs Quantum Monte Carlo (QMC) sont analysés afin d'identifier les algorithmes clés à implémenter ensuite dans une bibliothèque logicielle QMCKL open source lisible par l'homme.
L'objectif avec QMCKL est de travailler avec des applications GPU réalisées dans le cadre d'un benchmark. Ces applications utilisent des kernels spécifiques tels que Jastrow, AOS et MOS qui ont été préalablement portés sur GPU en utilisant OpenMP ou OpenACC. Nous utilisons ensuite nos différents outils d'analyse de performance pour profiler ces applications puis comparons les résultats obtenus.

III. ENVIRONNEMENT EXPÉRIMENTAL

1. CLUSTER AmdGPU

Modèle de processeur	AMD Ryzen 5 5500
Virtualisation	AMD-V
DVFS (gouverneur, réglages)	ondemand, fmin=1.4 GHz
Linux	Ubuntu 20.04.1
Version noyau	5.15.0-71-generic
Nombre CPU(s)	12
GPU	AMD Radeon PRO W6800
Archi	gfx1030
GCC Version	9.4.0
OpenMP	4.5

2. CLUSTER TURPAN

Modèle de processeur	Neoverse-N1
DVFS (gouverneur, réglages)	ondemand, fmin=1 GHz
Linux	Red Hat Enterprise Linux 8.6 (Ootpa)
Version noyau	4.18.0-372.9.1.el8.aarch64
Nombre CPU(s)	80
GPU	NVIDIA A100
NVC version	22.9-0
GCC	11.2.0
OpenMP Version	4.5

IV. RÉALISATIONS

1. GPU AMD

Afin de mener les benchmarks sur les GPU AMD, un accès au cluster AmdGPU m'a été donné. Une bonne nouvelle était que le cluster était déjà équipé d'un module rocm qui incluait les outils RocProf. La présence préinstallée de RocProf sur le cluster a considérablement simplifié le processus d'analyse des performances. J'ai donc pu utiliser ses outils directement, sans avoir à effectuer une installation supplémentaire.

1.1. Problèmes Rencontrés

1.1.1. QMCKL

Des problèmes sont survenus lors de l'installation de la bibliothèque QMCKL, notamment lors de la tentative de liaison dynamique avec la bibliothèque qmckl_gpu. La configuration n'a pas fonctionné correctement, "*autoconf*" ne parviens pas à trouver QMCkl GPU par lui même, ce qui a nécessité de lier statiquement la bibliothèque dans la variable *CFLAGS* pour résoudre le problème. Mais ce problème a été réglé grâce à l'ajout de "*--with-qmckl_gpu*" dans les options de configurations, avec cette option il est plus facile de spécifier le chemin du répertoire contenant les sous-répertoires "*lib/*" et "*include/*". Il peut aussi être nécessaire d'indiquer le chemin de la bibliothèque "*Trexio*"

1.1.2. HPCTOOLKIT

Avec HPCToolkit les problèmes sont survenus lors de l'installation, en effet l'installation manuelle ne pose aucun problème (la procédure de l'installation manuelle se trouve en **Annexe).**

Deux problèmes sont survenus:

1. Un problème lié à l'installation automatique avec **spack,** l'installation ne semble pas prendre en compte rocm même si il est installé sur la machine, ce qui a dû nécessité des échanges avec les responsables de chez HPCToolkit pour déterminer quel était le problème. Suite à ces échanges, il en est ressorti comme solution :
   - Avoir une version plus récente de Spack
   - Ajouter le compilateur GCC voulu dans "*compilers.yaml*" pour suggérer à Spack d'utiliser ce compilateur spécifique pour construire l'ensemble de la pile logicielle. Par exemple, avec GCC 12.2 on obtient :

     ```
     packages:
       all:
         compiler: [gcc@12.2.0]
     ```
   - Pour que Spack détecte les packages ROCm, il faut les ajouter en tant que packages externes dans le fichier packages.yaml. Pour empêcher Spack de les construire à partir des sources, il est possible d'ajouter "*buildable: false*" à ces entrées. HPCToolkit

dépend de 4 composants ROCm. En supposant qu'ils soient installés aux emplacements habituels cela devrait donner :

```
packages:
    hip:
        buildable: false
        externals:
        - spec: hip@5.2.5
          prefix: /opt/rocm-5.2.5
      hsa-rocr-dev:
        buildable: false
        externals:
        - spec: hsa-rocr-dev@5.2.5
          prefix: /opt/rocm-5.2.5
      rocprofiler-dev:
        buildable: false
        externals:
        - spec: rocprofiler-dev@5.2.5
          prefix: /opt/rocm-5.2.5
      roctracer-dev:
        buildable: false
        externals:
        - spec: roctracer-dev@5.2.5
          prefix: /opt/rocm-5.2.5
```

- Pour activer la prise en charge des GPU AMD dans HPCToolkit, l'activation de la variante +rocm pour hpctoolkit est nécessaire. Si plusieurs versions de ROCm sont présent, il faut spécifier les 4 versions en tant que dépendances afin que la version soit cohérente, exemple :

```
$ spack install hpctoolkit +rocm ^hip@5.2.5 ^hsa-rocr-dev@5.2.5
^rocprofiler-dev@5.2.5 ^roctracer-dev@5.2.5.
```

Une version de ROCm >= à 5.3 est recommandé

2. Le deuxième problème est cet échec à la fin de l'exécution de l'application :

```
out_ao: gpu/gpu-activity-process.c:661: gpu_memory_process:
Assertion 'host_op_node != NULL' failed.
```

Aucune information sur l'exécution n'est fournie et l'application ne se termine pas (il est nécessaire d'utiliser Ctrl+C pour l'arrêter).

Le problème est dû à une copie de données en dehors d'une région cible. Le problème a été récemment corrigé dans notre branche de développement.

Ces commandes permettent de régler le problème:

```
spack uninstall hpctoolkit
spack install hpctoolkit@develop +rocm ^hip@5.4.3 ^hsa-rocr-
dev@5.4.3 ^rocprofiler-dev@5.4.3 ^roctracer-dev@5.4.3
```

1.2. RÉSULTATS

Pour obtenir les résultats, le profilage a été effectué sur les kernels **AOS** et **MOS** en prenant pour données entrantes le fichier *"Alz_small.h5" situé dans le répertoire "data", car vue les faibles performances du cluster AMD GPU, avec le* fichier *"Alz_large.h5" les kernels prendront énormément de temps pour s'exécuter.*

1.2.1. RocProf VS HPCToolkit

Pour chaque application, RocProf et HPCToolkit fond urnit une liste de métriques associées à chaque kernel. Parmi les métriques remontées par RocProf on peut citer: **temps d'exécution**, le **nombre d'appels au kernel** , la **moyenne**, le **pourcentage** etc. Mais ici seul le pourcentage et le temps d'exécution des kernels nous intéresse. Les résultats sont fournis dans un tableau.

- **Résultats pour AOS**

Name	RocProf				HPCToolkit			
	Duration (s)		Percentage (%)		Duration (s)		Percentage (%)	
	Small	Large	Small	Large	Small	Large	Small	Large
qmckl_compute_ao_vgl_gaussian_device	0,027	0,103	46,68	51,704	0,027	0,102	46,30	0,516
qmckl_finalize_ao_basis_hpc_device	0,011	0,038	19,35	19,083	0,011	0,037	19,50	0,188
qmckl_compute_ao_vgl_gaussian_device	0,010	0,023	17,41	11,746	0,011	0,024	18,00	0,119
qmckl_compute_ao_basis_shell_gaussian_vgl_device	0,005	0,021	8,04	10,443	0,005	0,021	7,90	0,106
qmckl_set_point_device	0,002	0,006	3,81	3,007	0,002	0,006	3,70	0,030
qmckl_tensor_set_device	0,002	0,004	2,84	2,226	0,002	0,004	2,80	0,023

Rappel des caractéristiques du cluster amdGPU

Virtualisation: AMD-V**DVFS (gouverneur, réglages):** ondemand, fmin=1.4 GHz.

Linux: Ubuntu 20.04.1. **Version noyau:** 5.15.0-71-generic.

Nombre CPU(s): 12. **GPU:** AMD Radeon PRO W6800. **Archi:** gfx1030

GCC Version: 9.4.0 **OpenMP:**4.5

Résultats du profilage effectué avec RocProf et HPCToolkit sur le cluster AMD GPU pour AOS (les kernels représentant moins de 1% du temps d'exécution ne sont pas présentés)

- **Résultats pour MOS**

Name	RocProf				HPCToolkit			
	Duration (s)		Percentage (%)		Duration (s)		Percentage (%)	
	Small	Large	Small	Large	Small	Large	Small	Large
qmckl_compute_mo_basis_mo_vgl_device	10,75	258,21	99,2	99,9	10,8	260,00	99,2 %	99,9 %

Rappel des caractéristiques du cluster amdGPU
Virtualisation: AMD-V
DVFS (gouverneur, réglages): ondemand, fmin=1.4 GHz
Linux: Ubuntu 20.04.1 **Version noyau:** 5.15.0-71-generic
Nombre CPU(s): 12. **GPU:** AMD Radeon PRO W6800 (gfx1030). **Archi:** gfx1030
GCC Version: 9.4.0 **OpenMP:** 4.5

Résultats du profilage effectué avec RocProf et HPCToolkit sur le cluster AMD GPU pour les kernels MOS (les kernels représentant moins de 1% du temps d'exécution ne sont pas présentés)

1.2.3. CONCLUSION

Bien que, pour AOS et MOS, sur certains kernels les chiffres diffèrent à quelque chose près après la virgule, dans l'ensemble les résultats sont similaires. Je peux donc conclure en disant que les deux outils donnent les mêmes résultats (bien que toute fois ce ne soit pas une égalité stricte). Les graphiques ci dessous montrent les résultats des deux outils sur les kernels avec un coût important (ayant le temps d'exécution les plus haut et un pourcentage > 1 %). Les suffixes et préfixes non nécessaires ont été enlevés du nom des kernels.

- **Comparaison des résultats des deux outils sur le cluster AMD GPU en ne tenant compte que des principaux kernels de AOS**

Rappel des caractéristiques du cluster amdGPU
Virtualisation: AMD-V. **DVFS (gouverneur, réglages):** ondemand, fmin=1.4 GHz
Linux: Ubuntu 20.04.1. **Version noyau:** 5.15.0-71-generic
Nombre CPU(s): 12. **GPU:** AMD Radeon PRO W6800 (gfx1030) **Archi:** gfx1030
GCC Version: 9.4.0. **OpenMP:** 4.5

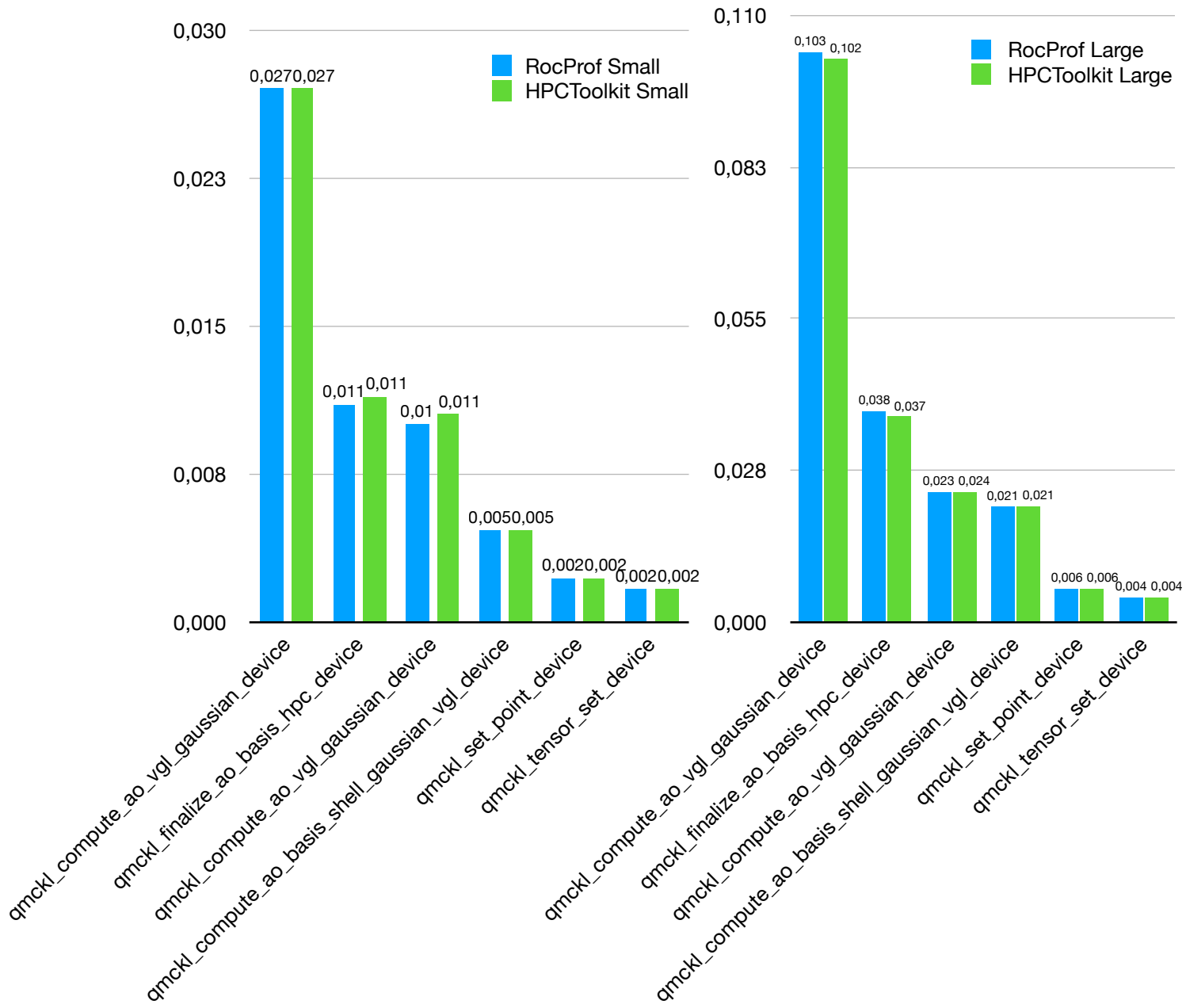


Diagramme comparant les temps d'exécution des principaux kernels AOS (avec les temps les plus importants) de RocProf et de HPCToolkit sur le cluster AMD GPU

- **Comparaison des résultats des deux outils sur le cluster AMD GPU en ne tenant compte que des principaux kernels de MOS**

Rappel des caractéristiques du cluster amdGPU

Virtualisation: AMD-V. **DVFS (gouverneur, réglages):** ondemand, fmin=1.4 GHz
Linux: Ubuntu 20.04.1. **Version noyau:** 5.15.0-71-generic
Nombre CPU(s): 12. **GPU:** AMD Radeon PRO W6800 **Archi:** gfx1030
GCC Version: 9.4.0. **OpenMP:** 4.5

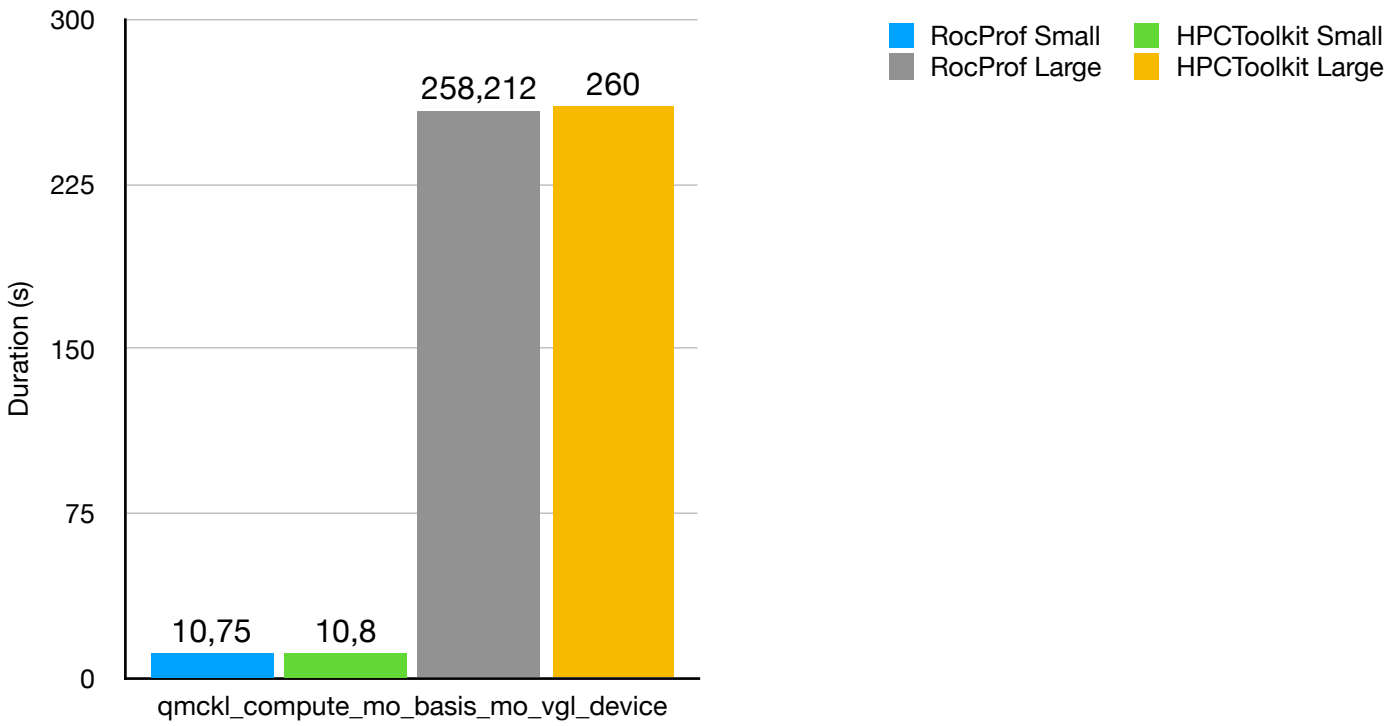


Diagramme comparant les temps d'exécution des principaux kernels MOS (avec les temps les plus importants) de RocProf et de HPCToolkit sur le cluster AMD GPU

2. GPU NVIDIA

Pour obtenir les résultats, le profilage a été effectué sur les kernels **AOS** et **MOS** en prenant pour données entrantes le fichier *"Alz_large.h5" situé dans le répertoire "data"*.
J'ai également utilisé un code issu du repository git **cea-hpc** qui s'exécute sur GPU: **patter4gpu**

2.1. Problèmes Rencontrés

Du coté du cluster Turpan les seuls problèmes rencontrés était ceux liés à la bibliothèque QMCKL notamment lors de la tentative de liaison dynamique avec la bibliothèque qmckl_gpu, mais et du fait que *"autoconf"* ne parviens pas à trouver QMCkl GPU par lui même, mais grâce à l'ajout de *"--with-qmckl_gpu"* dans les options de configurations permettant ainsi de spécifier le chemin du répertoire contenant les sous-répertoires *"lib/"* et *"include/"* le problème a pu être résolu
Il peut également avoir un problème avec la bibliothèque *Trexio* du fait également que *"autoconf"* ne parviens pas à le trouver, pour cela il suffit d'ajouter le chemin d'installation de la bibliothèque dans la variable *"LD_LIBRARY_PATH"*

Du côté de HPCToolkit aucun problème n'est à signaler, l'utilisation de Spack étant la manière à privilégier.

2.2. RÉSULTATS

2.2.1. Nsight Systems VS HPCToolkit

Pour chaque application, Nsight Systems ainsi que HPCToolkit fournit une liste de métriques associées à chaque kernel telles que le **temps d'exécution**, le **nombre d'appels au kernel** , la **moyenne**, etc.

- **Résultats pour AOS**

Name	Nsight Systems				HPCToolkit			
	Duration (s)		Percentage (%)		Duration (s)		Percentage (%)	
	Small	Large	Small	Large	Small	Large	Small	Large
qmckl_compute_ao_vgl_gaussian_device_18	0,0432	1,71	73,5	96.8	0,0432	1,71	74,1	96,8
qmckl_compute_ao_vgl_gaussian_device_16	0,0037	0,0240	6,3	1.4	0,0037	0,0243	6,3	1,4
qmckl_finalize_ao_basis_hpc_device_42	0,0042		7,2		0,0040		6,9	
qmckl_set_point_device_4	0,0041		6,9		0,0038		6,6	
qmckl_compute_ao_basis_shell_gaussian_vgl_device_4	0,0021		3,6		0,0021		3,6	
qmckl_tensor_set_device_8	0,0007		1,2		0,0007		1,2	

Rappel des caractéristiques du cluster Turpan

DVFS (gouverneur, réglages): ondemand, fmin=1 GHz
Linux: Red Hat Enterprise Linux 8.6 (Ootpa)
Version noyau: 4.18.0-372.9.1.el8.aarch64
Nombre CPU(s): 80
GPU: NVIDIA A100, **NVC version:** 22.9-0
GCC: 11.2.0, **OpenMP Version:** 4.5

Résultats du profilage effectué avec Nsight Systems et HPCToolkit sur le cluster Turpan pour les kernels de AOS (les kernels représentant moins de 1% du temps d'exécution ne sont pas présentés)

Rappel des caractéristiques du cluster Turpan

DVFS (gouverneur, réglages): ondemand, fmin=1 GHz

Linux: Red Hat Enterprise Linux 8.6 (Ootpa). **Version noyau:** 4.18.0-372.9.1.el8.aarch64

Nombre CPU(s): 80. **GPU:** NVIDIA A100

NVC version: 22.9-0. **GCC:** 11.2.0 **OpenMP Version:** 4.5

- **Résultats pour MOS**

Name	Nsight Systems				HPCToolkit			
	Duration (s)		Percentage (%)		Duration (s)		Percentage (%)	
	Small	Large	Small	Large	Small	Large	Small	Large
qmckl_compute_mo_basis_mo_vgl_device_2	2,94	146	96,7	97,9	2,94	146	96,7	97,9
qmckl_compute_ao_vgl_gaussian_device_18	0,08	3,13	2,6	2,1	0,08	3,14	2,6	2,1

Résultats du profilage effectué avec Nsight Systems et HPCToolkit sur le cluster Turpan pour les kernels de MOS (les kernels avec un pourcentage inférieur à 1% ne sont pas représentés dans le tableau)

- **Résultats pour patter4gpu**

Name	Total time (ms)	Time (%)
kernel_1	0,06828	38,9
kernel_2	0,053	30,2
kernel_3	0,029	16,5
kernel_4	0,01404	8,0
kernel_5	0,00428	2,4
kernel_6	0,00368	2,1
kernel_7	0,00344	2,0

NB:

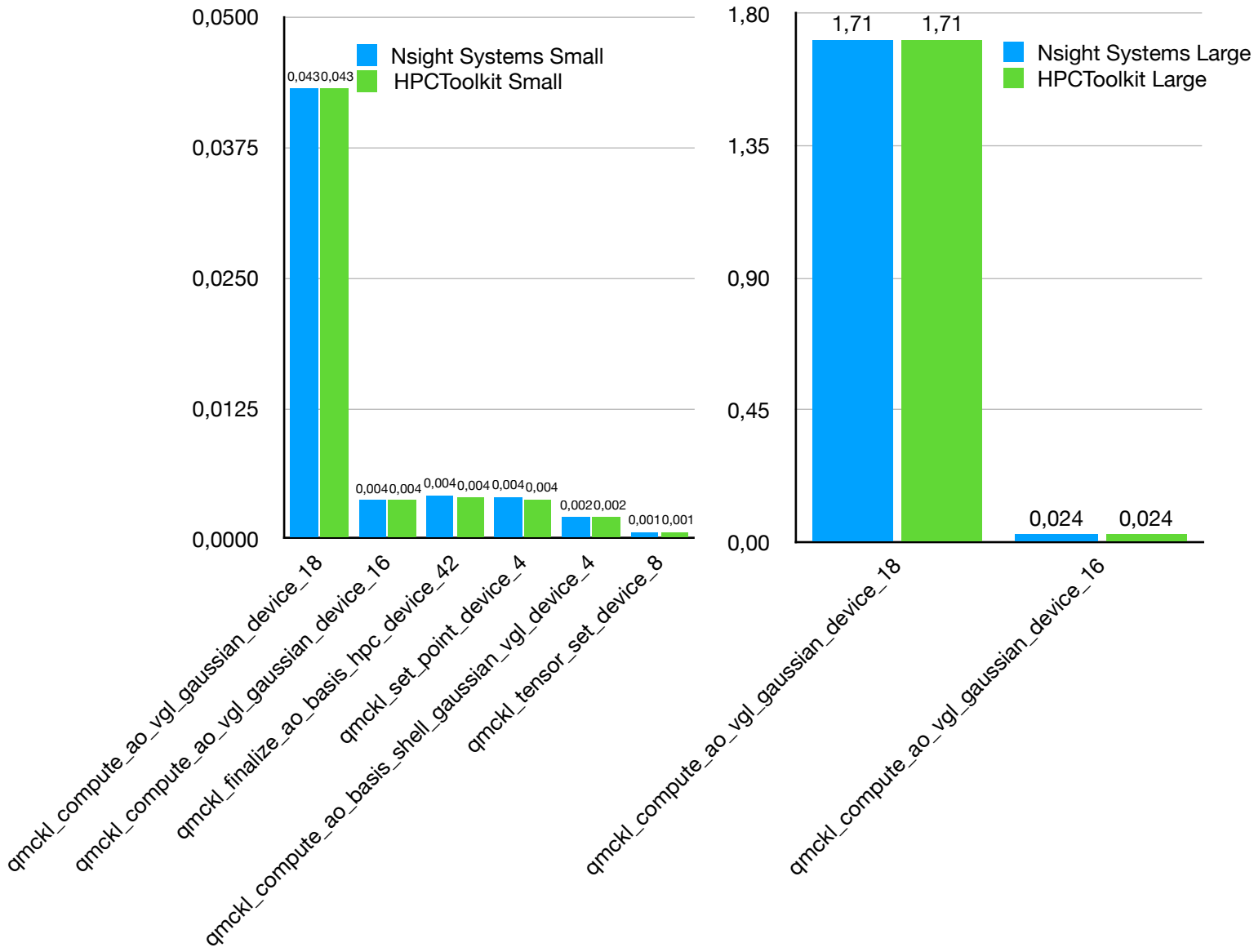
Résultats du profilage effectué avec Nsight Systems sur le cluster Turpan pour les kernels de patter4gpu

Le profilage avec HPCToolkit n'a pas été réalisé, en effet lorsque j'essai d'exécuter le code avec les commandes de profilage de HPCToolkit (hpcrun), le code tourne a l'infini.

2.2.3. CONCLUSION

Là encore les résultats concordent, les deux outils donnent a peu près les mêmes résultats. Les diagrammes ci dessous nous donne un aperçu de cette similarité.

- **Comparaison des résultats des deux outils sur le cluster Turpan en ne tenant compte que des principaux kernels de AOS**



Rappel des caractéristiques du cluster Turpan

DVFS (gouverneur, réglages): ondemand, fmin=1 GHz

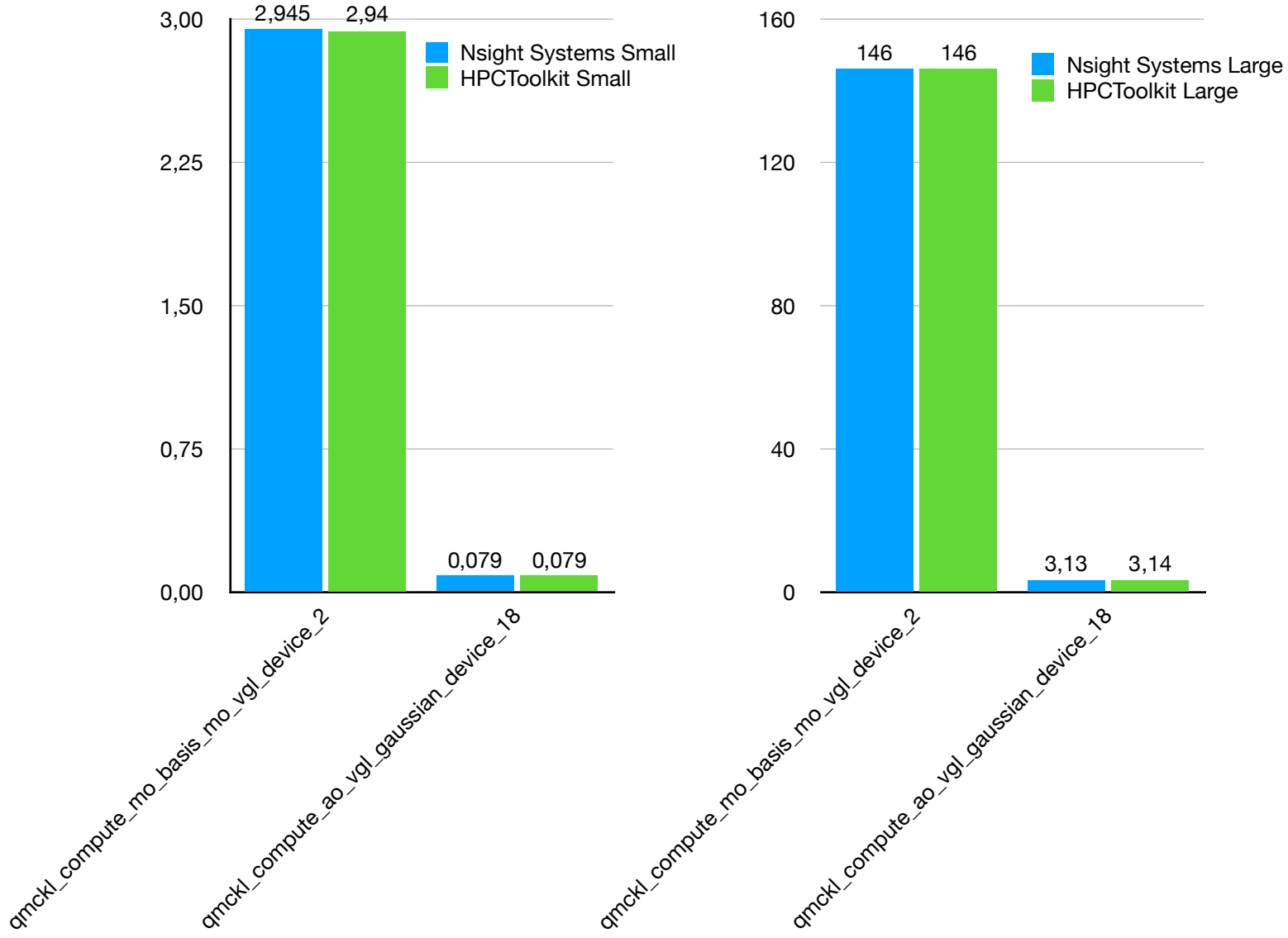
Linux: Red Hat Enterprise Linux 8.6 (Ootpa)**Version noyau:** 4.18.0-372.9.1.el8.aarch64

Nombre CPU(s): 80**GPU:** NVIDIA A100

NVC version: 22.9-0. **GCC:** 11.2.0. **OpenMP Version:** 4.5

Diagramme comparant les temps d'exécution des principaux kernels de AOS (avec les temps les plus importants) de Nsight Systems et de HPCToolkit sur le cluster Turpan

- **Comparaison des résultats des deux outils sur le cluster Turpan en ne tenant compte que des principaux kernels de MOS**



Rappel des caractéristiques du cluster Turpan

DVFS (gouverneur, réglages): ondemand, fmin=1 GHz

Linux: Red Hat Enterprise Linux 8.6 (Ootpa). **Version noyau:** 4.18.0-372.9.1.el8.aarch64

Nombre CPU(s): 80. **GPU:** NVIDIA A100

NVC version: 22.9-0. **GCC:** 11.2.0. **OpenMP Version:** 4.5

Diagramme comparant les temps d'exécution des principaux kernels de MOS (avec les temps les plus importants) de Nsight Systems et de HPCToolkit sur le cluster Turpan

3. GPUBLAS

3.1. Différence entre avec transfert et sans transfert pour m= 500 et m=3000

Dans cette section, j'ai utilisé HPCToolkit pour analyser une disparité de facteur 100 entre des mesures de multiplication de matrices en simple précision (SP) et en double précision (DP). Ces mesures ont été réalisées pour des matrices présentant des configurations très spécifiques. Les paramètres de variation incluent une taille de matrice (M) variant de 500 à 3000 par incréments de 100, ainsi que des valeurs de k variant de 1 à 32 par incréments de 1. Les analyses ont été effectuées avec transfert de données et sans transfert pour les valeurs M de 500 et 3000, et dans les deux cas avec k fixé à 32.

Rappel des caractéristiques du cluster Turpan

DVFS (gouverneur, réglages): ondemand, fmin=1 GHz

Linux: Red Hat Enterprise Linux 8.6 (Ootpa). **Version noyau:** 4.18.0-372.9.1.el8.aarch64

Nombre CPU(s): 80. **GPU:** NVIDIA A100

NVC version: 22.9-0. **GCC:** 11.2.0. **OpenMP Version:** 4.5

- **m=500**

Name	SP				DP			
	Time (ms)		Time(%)		Time (s)		Time(%)	
	WO_DT	DT	WO_DT	DT	WO_DT	DT	WO_DT	DT
gpu copyout	1,22E-03	3,65E-02	13,6	68,1	2,45E-03	7,49E-02	24,4	78,3
gpu copyin	1,02E-05	9,15E-03	0,1	17,1	1,48E-05	1,30E-02	0,1	13,6
cutlass_80_tensorop					7,57E-03	7,72E-03	75,5	8,1
ampere_sgemm	7,78E-03	7,97E-03	86,3	14,9				

Résultats du profilage effectué avec HPCToolkit sur le cluster Turpan afin d'expliquer la différence entre avec transfert et sans transfert pour m= 500

- **m=3000**

Name	SP				DP			
	Time (ms)		Time(%)		Time (s)		Time(%)	
	WO_DT	DT	WO_DT	DT	WO_DT	DT	WO_DT	DT
gpu copyout	1,31E-01	2,84E+00	75,8	97,5	2,65E-01	5,69+00	76,7	97,6
gpu copyin	3,40E-05	3,12E-02	0	1	6,29E-05	5,90E-02	0	1
ampere_sgemm	4,18E-02	4,25E-02	24,2	1,5	8,01E-02	8,12E-02	23,2	1,4

Résultats du profilage effectué avec HPCToolkit sur le cluster Turpan afin d'expliquer la différence entre avec transfert et sans transfert pour m= 3000

3.1.1. CONCLUSION

Les résultats confirment les conclusions que nous avions déjà déduites précédemment concernant cet écart de facteur 100. Cet écart est principalement attribuable au fait que nous avons pris en considération le temps nécessaire pour les allocations, les copies et les libérations de mémoire. Les résultats obtenus grâce à HPCToolkit révèlent que, dans le cas des données obtenues sans transfert, les opérations liées à la mémoire comptent pour 25 % de l'ensemble des opérations effectuées sur le GPU. En soustrayant ces opérations du calcul des GFLOPS, nous pouvons nous attendre à obtenir un résultat supérieur. De plus, étant donné ce pourcentage significatif, le résultat final sera considérablement amélioré.

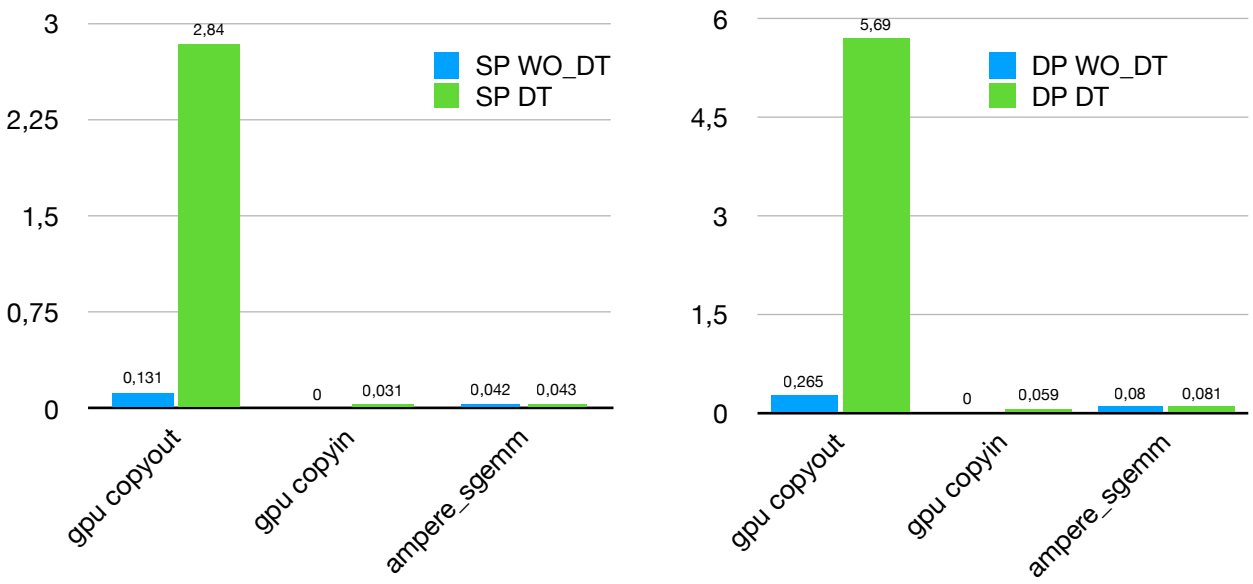


Diagramme illustrant les mesures de cublas_wo_dt et cublas_dt pour m=3000 et k=32, résultant d'une analyse effectuée à l'aide de HPCToolkit.

CONCLUSION

En définitive, ce stage a été une expérience extrêmement enrichissante à la fois sur le plan personnel et professionnel. Il m'a permis de plonger dans le domaine passionnant du HPC (High-Performance Computing) qui, bien que peu connu du grand public, joue un rôle crucial dans de nombreux domaines et a un impact significatif sur notre vie quotidienne.

Au cours de ce stage, j'ai eu l'opportunité de me familiariser avec des technologies avancées telles que les GPU AMD et NVIDIA, ainsi que les outils de profilage et d'analyse de performances tels que RocProf, Nsight Systems et HPCToolkit. J'ai également eu la chance de travailler sur le cluster AmdGPU et d'accéder aux ressources nécessaires pour réaliser des benchmarks et des analyses approfondies.

L'une des principales découvertes de ce stage a été la différence de performances entre les kernels de type AOS sur les GPU AMD et NVIDIA. Cette observation a suscité mon intérêt à approfondir l'étude de ces différences et à explorer les raisons pour lesquelles l'exécution est plus rapide sur les GPU NVIDIA. Cette analyse future me permettra de mieux comprendre les optimisations possibles et d'optimiser davantage les performances des applications sur différents types de GPU.

Par ailleurs, j'ai également été confronté à des défis techniques, tels que le problème rencontré avec HPCToolkit sur les GPU AMD et l'installation de cet outil sur le cluster Turpan. Bien que ces obstacles aient représenté des défis, ils m'ont également permis d'apprendre à résoudre des problèmes complexes et de développer mes compétences en matière de débogage et de gestion des configurations système.

En conclusion, ce stage m'a apporté une compréhension approfondie du domaine du HPC et m'a permis d'acquérir des compétences pratiques dans l'utilisation des GPU et des outils de profilage. J'ai également appris à relever des défis techniques et à travailler de manière autonome. Je suis reconnaissant d'avoir eu cette opportunité et je suis convaincu que les connaissances et l'expérience acquises au cours de ce stage seront précieuses pour ma carrière.

BIBLIOGRAPHIE

https://rocm.docs.amd.com/projects/rocprofiler/en/latest/rocprof.html

https://github.com/TREX-CoE/

https://www.calmip.univ-toulouse.fr/

https://docs.nvidia.com/nsight-systems/

http://hpctoolkit.org/

ANNEXE

A. INSTALLATIONS

La procédure d'installation des trois outils est assez bien documentée sur leur site web officiel respectif, donc je conseillerai de se référer directement sur ces sites pour une potentielle installation.

- **HPCToolkit:** http://hpctoolkit.org/software-instructions.html
- **RocProf:** https://rocm.docs.amd.com/en/latest/deploy/linux/
- **Nsight Systems:** https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html

B. UTILISATIONS

Le but ici est de montrer les différentes commandes utiles pour réaliser une analyse de performance à l'aide de nos trois différents outils. Pour ce faire, à titre d'exemple, notre application cible aura pour nom "*bench_os*" .

a. HPCToolkit

L'analyse avec HPCToolkit se découpe en plusieurs étapes qu'il est nécessaire de suivre dans l'ordre pour réaliser un bon profilage. Trois commandes seront utilisés à travers ces étapes: **hpcrun, hpcstruct, hpcprof.**

1. **Measure GPU and CPU execution unobtrusively with hpcrun:** les quatre premiers exemples ci dessous présentent les différentes utilisations de la commande hpcrun, il n'est pas nécessaire de tous les utilisés, un seul suffit. La dernière permet de spécifier le nom du répertoire de sortie :

   - GPU profiling (-e gpu=[nvidia,amd,opencl,level0])
     Exemple: `hpcrun -e gpu=nvidia ./bench_os`
   - GPU tracing (-t)
     Exemple: `hpcrun -e gpu=nvidia -t ./bench_os`
   - GPU PC sampling (NVIDIA GPU only)
     Exemple: `hpcrun -e gpu=nvidia,pc -t ./bench_os`
   - CPU and GPU profiling
     Exemple: `hpcrun -e REALTIME -e gpu=nvidia -t ./bench_os`
   - Specify output directory
     Exemple: `hpcrun -o <measurements-dir>`

En utilisation hpcrun sauf si le répertoire de sortie est précisé, un répertoire du style hpctoolkit-<app>-measurements sera crée. Par exemple pour notre application on aura: **hpctoolkit-bench_os-measurements**

2. **Recover program structure with hpcstruct:** les commandes ci dessous doivent tous les deux êtres utilisés afin de réaliser à la fois des analyses sur le CPU et le GPU
   - Analyze CPU binaries

     Exemple: `hpcstruct bench_os`

     Cela va créer un fichier avec l'extension *"hpcstruct"*, par exemple pour notre application on aura: **bench_os.hpcstruct**
   - Analyze all GPU binaries in <measurements-dir>

     Exemple: `hpcstruct hpctoolkit-bench_os-measurements`

3. **Correlate performance data with program structure using hpcprof**
   - Use a single process to combine performance data

     Exemple: `hpcprof -S bench_os.hpcstruct hpctoolkit-bench_os-measurements`
   - Specify output directory

     Exemple: `hpcprof -o <database-dir> -S bench_os.hpcstruct hpctoolkit-bench_os-measurements`

Un répertoire hpctoolkit-bench_os-database sera créé et c'est ce répertoire qui sera utilisé pour la visualisation dans hpcviewer

4. **hpcviewer presents calling context sensitive GPU and CPU metrics and behaviors over time**

b. RocProf

"Rocprof" peut être utilisé pour le profilage GPU en utilisant des **compteurs matériels (HW counters)** et une **trace d'application**.

Le profilage GPU est contrôlé par un fichier d'entrée qui définit une liste de métriques/ compteurs et une portée de profilage. Un fichier d'entrée est fourni en utilisant l'option -i <fichier d'entrée>. Un fichier CSV de sortie est généré avec une ligne par kernel soumis. Chaque ligne contient le nom du kernel, les paramètres du kernel et les valeurs des compteurs. En utilisant l'option "--stats", les statistiques d'exécution du kernel peuvent être générées au format CSV. Actuellement, le profilage présente une limitation de sérialisation des kernels soumis. Voici un exemple de fichier d'entrée qui a pour nom "*input.txt*":

```
# Perf counters group 1
pmc : Wavefronts VALUInsts SALUInsts SFetchInsts
# Perf counters group 2
pmc : TCC_HIT[0], TCC_MISS[0]
# Filter by dispatches range, GPU index and kernel names
# supported range formats: "3:9", "3:", "3"
range: 1 : 4
```

```
gpu: 0 1 2 3
kernel: simple Pass1 simpleConvolutionPass2
```

Exemple: `rocprof --stats -i input.txt ./bench_os`

Les compteurs (counters) et les métriques dérivées (derived metrics) disponibles peuvent être consultés à l'aide des options '--list-basic' pour les compteurs et '--list-derived' pour les métriques dérivées. La sortie pour les compteurs indique le nombre d'instances de blocs (block instances) et le nombre de registres de compteurs de blocs (block counter registers). La sortie pour les métriques dérivées affiche les expressions des métriques.

Exemple:

- `rocprof —list-basic`
- `rocprof —list-derived`

Le suivi d'application pris en charge inclut le suivi de l'API d'exécution et du suivi de l'activité GPU. Les exécutions supportées sont : ROCr (API HSA) et HIP. Les activités GPU prises en charge sont : l'exécution de noyaux (kernel), la copie asynchrone de mémoire et les paquets de barrières. La trace est générée au format JSON, compatible avec le suivi Chrome ou sur le site https://perfetto.dev/. La trace se compose de plusieurs sections avec des chronologies pour le suivi de l'API par thread et l'activité GPU. Les événements des chronologies affichent le nom de l'événement et ses paramètres. Les options prises en charge sont : '--hsa-trace', '--hip-trace' et '--sys-trace', où 'sys trace' est une trace combinée pour HIP et HSA.

Exemple: rocprof --hsa-trace --hip-trace -i input.txt ./bench_os

Après avoir exécuté cette commande, 'rocprof' générera une trace au format JSON compatible avec le suivi Chrome ou sur https://perfetto.dev/. La trace contiendra des chronologies pour l'API HSA et l'activité GPU HIP, avec des événements correspondant aux opérations effectuées par l'application 'bench_os'. Ces événements incluront des détails tels que les paramètres des noyaux (kernels), les copies asynchrones de mémoire, les paquets de barrières, etc.

c. Nsight Systems

Les lignes de commande de Nsight Systems peuvent prendre l'une des deux formes suivantes :

- nsys [global_option]

  Exemple: `nsys [-v|-h]`
- nsys [command_switch][optional command_switch_options][application] [optional application_options]

  Exemple: `nsys profile -t openmp,openacc,cuda -o report-os-gpu bench_os`

On peut lancer l'application et commencer l'analyse en spécifiant les options à la commande **nsys profile** comme dans l'exemple ci-dessus. Plusieurs fichiers avec pour préfixe le nom de spécifié dans la commande (-o report-os-gpu) seront générés et notamment avec les extensions *"nsys-rep"* et *"sqlite"*. A titre d'exemple pour notre application on aura : **report-**

aos-gpu.nsys-rep ou **report-aos-gpu.sqlite .** Avec ces fichiers on peut effectuez un post-traitement des résultats existants de Nsight Systems pour générer des informations statistiques.

Exemple: `nsys stats report-aos-gpu.nsys-rep`

Alternativement, on peut contrôler le lancement d'une application et la collecte de données en utilisant des commandes interactives dans l'interface de ligne de commande (CLI) à savoir: **analyze, cancel, export, launch, nvprof, recipe, sessions, stats, etc.**

Plus d'information sur https://docs.nvidia.com/nsight-systems/UserGuide/index.html#abstract

Exemple: `nsys launch <application> [application-arguments]`