Implémentation et évaluation de clusters de Raspberry Pi

STREIGNARD Rémi

Promoteur : Dr. Guillerme Duvillié

Master 2 en Architecture des Systèmes Informatiques

Année académique **2021-2022**

Implémentation et évaluation de clusters de Raspberry Pi

STREIGNARD Rémi

Promoteur : Dr. Guillerme Duvillié

Master 2 en Architecture des Systèmes Informatiques

Année académique 2021-2022

Pour l'entraide, le support et la bonne communication autour de la réalisation de ce travail de fin d'études, je tiens à remercier mon promoteur, le Dr. Guillerme Duvillié. Il a su, tout au long de mon travail, mais aussi de mon cursus de Master 2, apporter réponses à mes questions.

Je remercie aussi l'ensemble du corps professoral du Master en Architecture des Systèmes Informatiques - Hénallux pour leur enseignement au cours des deux dernières années et pour l'ensemble des conseils et critiques constructives qu'ils ont pu m'adresser.

Ensuite, je tiens à remercier mes amis et collègues avec qui j'ai pu partager ces deux années de projets et d'études.

Table des matières

Table des figures

Table des tableaux

Glossaire

Terme	Acronyme	Définition
Input/output	I/O	Également appelée I/O (prononcée comme eye-o), l'entrée/sortie est tout dispositif logiciel ou matériel conçu pour envoyer et recevoir des données vers et depuis un composant matériel de l'ordinateur.
Boot		Également appelé "boot up" ou parfois "start up", le démarrage est le processus qui consiste à mettre un ordinateur sous tension et à accéder au système d'exploitation.
Buffer		Lorsqu'on parle de mémoire, un buffer est un stockage temporaire dans la mémoire qui stocke des informations pendant le traitement d'autres informations.
Erasable programmable read-only memory	EEPROM	Abréviation de electrically erasable programmable read-only memory (mémoire morte programmable effaçable électriquement), l'EEPROM est une PROM qui peut être effacée et reprogrammée à l'aide d'une charge électrique. L'EEPROM a été développée par George Perlegos alors qu'il travaillait chez Intel en 1978 et, contrairement à la plupart des mémoires d'un ordinateur, elle mémorise ses données sans alimentation électrique.
kFLOPS (1000) MFLOPS (1 M) GFLOPS (1000 M)	FLOPS	Abréviation de floating-point operations per second, FLOPS est une mesure utilisée pour indiquer le nombre d'opérations à virgule flottante qu'un microprocesseur est capable d'effectuer par seconde.
High Performance Computer	HPC	Un ordinateur ou un ensemble d'ordinateurs qui agissent comme une machine collective capable de traiter d'énormes quantités de données. Les superordinateurs sont utilisés pour des tâches très complexes, telles que la recherche nucléaire ou les prévisions météorologique. La taille d'un superordinateur peut varier considérablement, en fonction du nombre d'ordinateurs qui le composent. Un superordinateur peut être composé de 10, 100, 1000 ou plus d'ordinateurs, travaillant tous ensemble.
Hyper-threading		HT (Hyper-Threading) est une technologie développée par Intel et introduite avec le processeur Xeon, puis incluse dans le processeur Intel Pentium 4. HT permet au processeur de travailler plus efficacement en traitant deux jeux d'instructions en même temps, ce qui le fait ressembler à deux processeurs logiques. En outre, les logiciels écrits pour les ordinateurs à deux ou plusieurs processeurs sont toujours compatibles avec HT.

Terme	Acronyme	Définition
Mainframes		Également appelé "big iron computer, le mainframe est un gros ordinateur central doté de plus de mémoire, d'espace de stockage et de puissance de traitement qu'un ordinateur standard. Un ordinateur central est utilisé par les gouvernements, les écoles et les entreprises pour renforcer la sécurité et traiter de grandes quantités de données, comme les statistiques sur les consommateurs, les données de recensement ou les transactions électroniques. Leur fiabilité et leur grande stabilité permettent à ces machines de fonctionner pendant très longtemps, voire des décennies.
Millions of Instructions per second	MIPS	Abréviation de millions d'instructions par seconde, MIPS est le nombre approximatif d'instructions qu'un processeur peut exécuter en une seconde.
Multithreading		Le multithreading est la capacité d'un logiciel ou d'un système d'exploitation à exécuter plusieurs threads du même programme en même temps, ce qui permet d'optimiser l'utilisation du temps CPU disponible. En utilisant le multithreading, un ordinateur peut exécuter et traiter plusieurs tâches en même temps.
Network Attached Storage	NAS	Un NAS est un périphérique de stockage qui est connecté à un réseau et fournit à ce dernier un stockage supplémentaire. Un NAS n'a pas de puissance de traitement propre, ce qui signifie qu'il ne peut pas être utilisé pour exécuter ou faire fonctionner des programmes partagés par le réseau.
Overclocking		Parfois abrégé en OC, l'overclock est une méthode permettant de configurer un ordinateur pour qu'il fonctionne plus rapidement que sa vitesse annoncée. L'overclocking est réalisé en réglant ou en modifiant des cavaliers, des interrupteurs à bascule, des paramètres CMOS, des mises à jour de micrologiciels ou en utilisant des utilitaires logiciels. L'overclocking permet aux utilisateurs d'obtenir une augmentation des performances et est le plus souvent effectué sur le processeur et la carte vidéo de l'ordinateur.
Overhead		En informatique, l'overhead est toute combinaison de temps de calcul, de mémoire, de bande passante ou d'autres ressources excédentaires ou indirectes nécessaires à l'exécution d'une tâche spécifique. Il s'agit d'un cas particulier de frais généraux d'ingénierie. L'overhead peut être un facteur décisif dans la conception d'un logiciel, en ce qui concerne la structure, la correction des erreurs et l'inclusion de fonctionnalités.
Parallel computing		En informatique, le terme parallèle décrit des tâches exécutées simultanément, sur des matériels distincts. Par exemple, dans le traitement numérique du signal (DSP), les échantillons d'un signal peuvent être divisés en unités de taille égale, chacune étant traitée en parallèle sur des processeurs ou des cœurs distincts. Les différents "morceaux" du signal peuvent être réunis dans une étape finale.

Terme	Acronyme	Définition
Parallel Virtual Machine	PVM	Une machine virtuelle parallèle (PVM) est un système informatique distribué créé par une série d'ordinateurs parallèles, qui sont tous fusionnés pour être affichés sous la forme d'une machine virtuelle unifiée. Ce cadre logiciel crée une architecture informatique distribuée à partir d'un système parallèle connecté qui fonctionne comme une seule unité pour traiter toute tâche informatique haut de gamme.
Password cracking		En cryptanalyse et en sécurité informatique, le craquage de mots de passe est le processus qui consiste à récupérer des mots de passe à partir de données qui ont été stockées ou transmises par un système informatique sous une forme brouillée.
Printed Circuit Board	PCB	Parfois abrégé en carte, un PCB est l'abréviation de circuit imprimé et a été inventé par Paul Eisler alors qu'il travaillait sur une radio en 1936. Un PCB est fabriqué en plastique ou en fibre de verre et contient des circuits intégrés et d'autres composants. La carte mère d'un ordinateur est un bon exemple de circuit imprimé que l'on trouve aujourd'hui dans tous les ordinateurs.
Rack		Un montage en rack est une description d'un dispositif matériel capable d'être monté dans un rack spécial ou dans le rack lui-même. Le montage en rack est couramment utilisé par les grandes entreprises pour maintenir leurs serveurs de réseau, routeurs, commutateurs ou autres dispositifs de réseau.
Shell scripts		Un shell script est un programme informatique conçu pour être exécuté par l'interpréteur de commandes Unix, un interpréteur de ligne de commande[1]. Les opérations typiques effectuées par les scripts shell comprennent la manipulation de fichiers, l'exécution de programmes et l'impression de texte.
System on a Chip	SOC	Un SoC (system on a chip) est un circuit intégré conçu pour incorporer tous, ou beaucoup, de composants informatiques nécessaires sur une seule puce. Un SoC peut avoir des processeurs, de la mémoire, des interfaces, des contrôleurs et des DSP sur une seule puce.
Threads		Un thread est un petit ensemble d'instructions conçu pour être programmé et exécuté par l'unité centrale indépendamment du processus parent. Par exemple, un programme peut avoir un thread ouvert qui attend qu'un événement spécifique se produise ou qui exécute un travail séparé, permettant au programme principal d'effectuer d'autres tâches. Un programme est capable d'avoir plusieurs threads ouverts en même temps et les termine ou les suspend une fois la tâche terminée ou le programme fermé.
	ARM	Le processeur ARM est un processeur RISC 32 bits, ce qui signifie qu'il est construit à l'aide du RISC (reduced instruction set computer) ISA (instruction set architecture). Les processeurs ARM sont des microprocesseurs et sont largement utilisés dans une grande partie des téléphones mobiles vendus chaque année, jusqu'à 98 % des téléphones mobiles.

1 Introduction

Le *Cluster computing* remonte aux années 1970, avec l'apparition des systèmes en temps partagé, conçus pour permettre à plusieurs utilisateurs de partager les ressources informatiques en simultané.

Auparavant, il y avait un seul ordinateur qui gérait toutes les tâches et tous les processus. Cet ordinateur unique était appelé "unité centrale de traitement" ou CPU. Cependant, avec cette conception, si le traitement d'une tâche prenait trop de temps, cela impliquait un temps d'attente pour le traitement des autres tâches.

Ainsi, le *Cluster computing* a changé la donne dans le monde de l'informatique, en offrant la capacité de créer des systèmes distribués dans lesquels la charge de travail est répartie entre plusieurs ordinateurs connectés les uns aux autres.

L'objectif de cette étude, en s'inspirant de précédents travaux de recherche traitant du *cluster computing* et du *benchmarking*, consiste à : d'une part, définir une procédure d'implémentation aisée d'un cluster ; d'autre part, évaluer les performances d'un tel système.

Le cluster mis en place lors de cette étude comporte 9 RPI qui s'articulent autour d'un réseau LAN doté d'un espace de stockage partagé. Plusieurs outils sont utilisés afin d'interconnecter et d'automatiser les configurations des 9 noeuds du cluster, et ce, pour le rendre flexible et évolutif.

Les performances du cluster sont ensuite étudiées lors d'une évaluation en 3 phases composée de : premièrement, une phase d'évaluation individuelle ; deuxièmement, une phase d'optimisation ; troisièmement, une phase d'évaluation en cluster.

Les étapes majeures de ces évaluations sont : 1) récolter 31 attributs sur les RPI qui permettent de dresser un état par défaut de leurs performances ; 2) évaluer les RPI après la modification de leur configuration afin d'offrir une visualisation des gains et/ou des pertes de performances, et ce, selon plusieurs métriques ; 3) évaluer les RPI dans un environnement en *parallel computing* où les 9 RPI s'articulent autour d'une tâche commune.

Les résultats de ces trois phases sont ensuite exposés en mettant l'emphase sur : les similitudes et/ou divergences de performances rencontrées au sein des 9 RPI ; les gains et/ou pertes octroyés par chaque optimisation et chaque modification apportées ; la puissance de traitement du cluster exprimée en GFLOPS ; les gains et/ou pertes en GFLOPS du cluster octroyés par les optimisations ; et la mise en relation des résultats avec de précédentes études.

Les travaux de recherche utilisés, les *scripts*, les configurations et optimisations ainsi que les résultats aux évaluations se retrouvent sur le répertoire GIT du projet : `https://gitlab.com/DTM-Henallux/MASI/etudiants/streignard-remi/t.f.e`

La suite de ce document est découpée comme suit : **2.** *Théorie* ; **3.** *Étude bibliographique* ; **4.** *Motivations* ; **5.** *Description du système* ; **6.** *Évaluation* ; **7.** *Implémentation* ; **8.** *Résultats expérimentaux* ; **9.** *Réalisations futures* ; **10.** *Conclusion*

2 Théorie

La section *Théorie* fournit une description et une base de connaissance aux thématiques principales dont ce travail fait l'objet. Ces thématiques sont les suivantes :

- premièrement, le *cluster computing* désigne la pratique visant à établir un système où un ensemble d'ordinateurs, appelés noeuds, est interconnecté au sein d'un réseau et perçu comme une entité unique ;
- deuxièmement, les *Beowulf clusters* désignent un type de cluster décrit comme étant un superordinateur virtuel à bas prix destiné à la parallélisation et au calcul de haute performance ;
- troisièmement, le *benchmarking* désigne la pratique visant à établir une évaluation des performances d'un produit, en vue de son optimisation et/ou de l'analyse de ses limites matérielles et logicielles.

2.1 Cluster computing [1, 2, 3, 4, 5, 6]

Le *cluster computing* désigne la pratique visant à établir un système où un ensemble d'ordinateurs, appelés noeuds, est interconnecté au sein d'un réseau et perçu comme une entité unique. Les noeuds du cluster sont utilisés conjointement pour résoudre des problèmes complexes. Leur mise en relation permet, en effet, d'obtenir une vitesse de calcul plus élevée ainsi qu'une intégrité des données accrue.

Quant à l'interconnexion entre les noeuds, elle est généralement réalisée au travers d'un LAN afin de rendre les communications rapides. Ainsi, du point de vue de l'utilisateur, le cluster se présente comme étant un système et une machine unique. Ce concept se nomme : la transparence du système.

En fonction de leur implémentation, les clusters peuvent être de plusieurs types tels que les clusters à haute disponibilité, les clusters à charge balancée et les clusters haute performance.

Tous trois fournissent des avantages qui leur sont propres, mais, plus globalement, les clusters améliorent la vitesse de calcul, l'intégrité des données et l'évolutivité.

2.1.1 Architecture et principes

Les clusters, outre leur définition, sont caractérisés par plusieurs éléments : premièrement, leur architecture ; deuxièmement, des principes de base ; troisièmement, leurs composants ; quatrièmement, leur accessibilité.

L'architecture d'un cluster est décrite comme étant un système, parallélisé ou distribué, composé de plusieurs ordinateurs articulés au sein d'un réseau local. Ce réseau local est généralement constitué d'un switch pour éviter que les communications internoeuds ne reposent sur la plus grande instabilité et le plus faible débit d'un réseau sans fil.

Les composants du cluster peuvent être visualisés sur la [Fig.1] : **1)** les noeuds, ordinateurs mono ou multiprocesseur, dotés de leur propre mémoire RAM, de leur propre système d'exploitation et de leurs propres I/O ;

2) le switch qui interconnecte l'ensemble des noeuds au sein du cluster dans le réseau local ; **3)** le potentiel serveur responsable du stockage partagé.

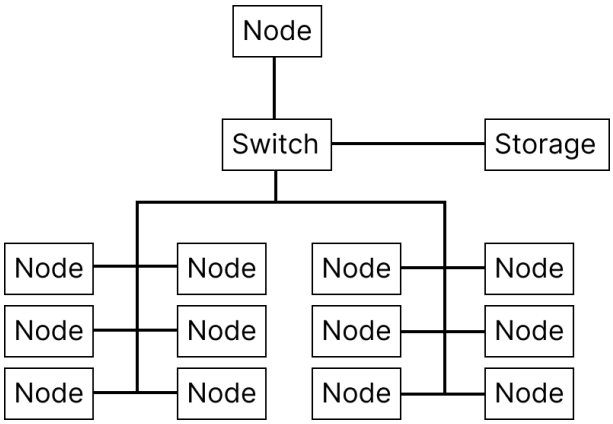


Figure 1: Architecture standard d'un cluster

Ensuite, différents principes permettent de caractériser un cluster :

- le premier est l'interconnexion des noeuds au sein d'un réseau dédié. L'objectif est de maximiser la bande passante, de réduire les communications internoeuds et de ne pas surcharger le réseau avec d'autres machines qui ne participent pas activement au cluster ;
- le second dicte que l'ensemble des noeuds doivent être du même type. Cela signifie que chaque noeud doit pouvoir participer sans que des règles spécifiques lui soient accordées, et ce, pour maximiser le concept de transparence dans le cluster ;
- le dernier est la présence d'un espace de stockage commun, qu'il soit partagé au travers d'un NAS ou distribué au sein même des noeuds. Cet espace de stockage a pour objectif de faciliter la gestion et l'accès des ressources dans le cluster.

Finalement, la notion d'accessibilité du cluster permet de définir si ce dernier est ouvert ou fermé. Ces deux états déterminent le moyen par lequel l'utilisateur peut utiliser le cluster ainsi que la présence ou non d'un noeud master.

Un cluster ouvert est caractérisé par un accès direct à l'ensemble de ses noeuds depuis internet. L'utilisateur peut ainsi utiliser des noeuds spécifiques du cluster. Cependant, cela implique de donner une plus grande attention à l'aspect sécurité du cluster.

Quant au cluster fermé, celui-ci n'est accessible qu'au travers d'un seul et même hôte passerelle : le master. C'est ensuite le master qui va communiquer avec le reste du cluster afin d'administrer et de déléguer les tâches vers les autres noeuds.

2.1.2 Types de clusters

Il existe plusieurs types de clusters, chacun étant destiné à une situation donnée et à la réalisation d'un type de tâche spécifique. Cela implique que leur architecture soit adaptée en fonction du contexte dans lequel le cluster doit être utilisé, et ce, afin d'obtenir les attributs désirés : vitesse, intégrité, redondance, etc.

Les principaux types de clusters sont les suivants :

- **Les clusters à charge balancée** - *load balancing clusters* sont utilisés pour distribuer et répartir les requêtes vers des services ou des applications entre les noeuds du cluster. L'objectif est d'équilibrer au mieux la charge de trafic entrant vers les noeuds afin de garantir que chacun de ceux-ci soit responsable d'une charge de travail optimisée au sein du cluster.

  Les avantages d'une telle architecture sont la diminution des surcharges réseau et l'amélioration des performances des noeuds. De plus, dans le cas où un noeud devient inapte à fonctionner, des algorithmes permettent de redistribuer les requêtes vers les noeuds disponibles.

- **Les clusters à haute disponibilité** - *High–Availability clusters* sont utilisés pour augmenter la redondance et garantir la haute disponibilité des services hébergés dans le cluster. Dans cette architecture : une partie des noeuds est responsable de l'hébergement et du bon fonctionnement des services ; une autre partie des noeuds est considérée comme système de secours. Ce système de secours est ensuite utilisé lors d'un dysfonctionnement matériel ou logiciel sur un ou plusieurs noeuds. L'objectif des clusters à haute disponibilité est donc d'offrir une disponibilité ininterrompue aux services.

  Les avantages d'une telle architecture sont la redondance, la robustesse et l'augmentation de la disponibilité des services et des applications. Contrairement aux clusters à charge balancée, dans le cas où un noeud devient inapte à fonctionner, les requêtes sont redistribuées vers les noeuds disponibles, tout en rajoutant des noeuds de secours au cluster afin d'équilibrer la perte.

- **Les clusters à haute performance** - *High-performance clusters* sont utilisés pour résoudre des calculs complexes par la mise en relation et l'agglomération des noeuds. Ces clusters sont couramment nommés HPC ou super calculateurs. L'objectif de ces derniers est de profiter de la puissance de traitement de chacun des noeuds pour résoudre des opérations complexes au travers d'algorithmes parallélisés.

  Les avantages d'une telle architecture sont la mobilisation d'une grande puissance de traitement, inatteignable par des ordinateurs seuls.

2.1.3 Avantages

L'utilisation des clusters introduit nombre d'avantages. Et bien que plusieurs d'entre d'eux soient dépendants du type de cluster choisi, d'autres avantages leurs sont communs. La liste ci-dessous décrit ces avantages :

- la **flexibilité et l'évolutivité** des clusters leurs permettent de facilement être mis à jour et étendus, que cela soit par leurs spécifications matérielles ou logicielles. Lorsque cela devient nécessaire, il est alors possible d'ajouter de nouveaux noeuds au cluster sans devoir reconsidérer tout le processus de déploiement ;

- le **rapport coût/performance** des clusters leur offre une position avantageuse sur les *mainframes*. Le coût des composants de base (noeuds) étant relativement abordable en comparaison aux *mainframes*, les clusters jouent alors sur le nombre plutôt que sur la puissance ;

- la **vitesse de traitement et les hautes performances** des clusters s'expliquent au travers des deux points précédents. En effet, le faible coût des leurs composants, joint à leur flexibilité, les clusters peuvent atteindre des performances similaires, voir supérieures, aux *mainframes* conventionnels ;

- la **haute disponibilité des ressources** des clusters est un avantage considérable. En effet, lors d'un dysfonctionnement dans le cluster entraînant la perte d'un des noeuds, l'architecture en cluster permet de maintenir un haut niveau de disponibilité, et ce, même si aucun mécanisme de redondance et de robustesse n'a été implémenté. De plus, la perte de performances due à la perte d'un noeud reste négligeable dans ce type d'architecture.

2.2 Beowulf cluster [7, 8, 9, 10, 11, 12]

Un *Beowulf cluster* est décrit comme étant un superordinateur virtuel à bas prix destiné à la parallélisation et au calcul de haute performance. Ces derniers sont décrits par un ensemble de principes et de caractéristiques. Dans l'ensemble, ils visent à maintenir une philosophie du rapport coût/performance par l'utilisation de matériel non propriétaire à bas prix et par l'utilisation de logiciels non propriétaires et gratuits, et ce, afin de concurrencer les superordinateurs traditionnels.

Ce chapitre décrit : premièrement, l'architecture des *Beowulf clusters* ; deuxièmement, leurs principes de base ; troisièmement, les avantages de ceux-ci.

Comme annoncé dans la section Motivations, le cluster implémenté dans ce travail s'inspire fortement des *Beowulf clusters*.

2.2.1 Architecture

Les Beowulfs sont des clusters fermés à haute performance. Dans l'ensemble, leur architecture [Fig.2] est fortement similaire à celle d'un cluster standard [Fig.1]. Ils sont constitués d'ordinateurs à bas prix et, habituellement, peu performants qui, une fois rassemblés, permettent d'atteindre des performances considérables.

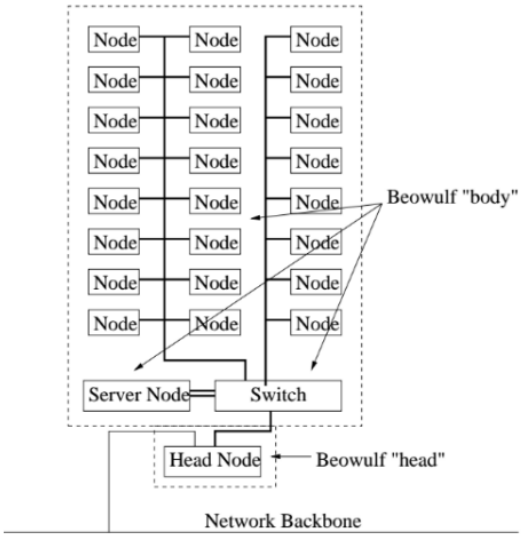


Figure 2: [12] Architecture d'un *Beowulf cluster*

Les composants des *Beowulf clusters*, visibles sur la figure [Fig.2], sont les suivants : premièrement, les noeuds, entités de calcul du cluster ; deuxièmement, le *Server node* responsable du stockage des fichiers et du partage des librairies ; troisièmement, le switch qui relie l'entièreté des noeuds du cluster ; quatrièmement, le *Head node*, ou master, qui dirige les opérations au sein du cluster ; cinquièmement, la *Network backbone* qui fournit les informations et les configurations réseau au cluster.

Dans l'architecture Beowulf, les noeuds et le *Server node* sont reliés autour d'un switch. Cela constitue le corps du *Beowulf cluster*. Quant au *Head node*, il est la seule interface dont dispose l'utilisateur pour interagir avec le cluster. Seul ce noeud master est capable de demander l'exécution d'opérations à l'ensemble des autres noeuds composant le cluster. C'est pourquoi les *Beowulf clusters* sont considérés comme fermés.

2.2.2 Principes

Les *Beowulf clusters* sont conçus pour concurrencer les superordinateurs traditionnels. Pour cela, ils reposent sur une philosophie qui vise à maximiser le rapport coût/performance et l'accessibilité à la création d'un cluster de ce type. Cette philosophie s'exprime au travers de plusieurs principes que voici :

- le cluster est formé sur base de matériel et de logiciels non propriétaires et, de préférence, libres et gratuits ;
- le système d'exploitation des noeuds, du *Head node* et du *Server node* doit être basé sous Unix et être non propriétaire ;
- le cluster est destiné à la parallélisation et dispose des librairies qui lui permettent d'accomplir cette tâche, comme MPI ou PVM ;
- les noeuds du cluster disposent tous de la même suite logicielle. Ils sont donc interchangeables ;
- le seul objectif des noeuds est de recevoir des opérations par le master, de résoudre ces opérations et de renvoyer les résultats vers le master ;
- le corps du *Beowulf cluster*, une fois configuré, n'est plus accessible par les utilisateurs, mais uniquement au travers du *Head node*, le master ;

Ainsi, l'aspect le plus fondamental des *Beowulf clusters* est la volonté de fournir un cluster à bas prix, dit *commodity cluster*. Cela implique la gratuité des logiciels et le refus des technologies propriétaires, mais aussi la flexibilité et l'évolutivité.

De plus, l'architecture n'étant définie que par ces quelques principes, cela permet d'adopter des stratégies matérielles et logicielles différentes lors de la réalisation d'un *Beowulf cluster*. Ce dernier peut ainsi être personnalisé au besoin et selon les objectifs fixés.

2.2.3 Avantages

Les *Beowulf clusters* viennent rajouter une surcouche aux avantages déjà attribués aux architectures clusterisées.

D'abord, la flexibilité, déjà apportée par la mise en cluster, est centrale aux *Beowulf clusters*. Le refus des technologies propriétaires et l'utilisation de technologies libres et gratuites leur permet de s'affranchir de toute dépendance envers un fournisseur. Ainsi, les architectures logicielles et matérielles du cluster peuvent facilement être adaptées pour s'aligner aux objectifs.

Ensuite, l'extensibilité. Par l'indépendance du matériel utilisé, toute machine disposant d'un noyau Unix et des librairies nécessaires peut être ajoutée au cluster afin d'augmenter la puissance de traitement de ce dernier.

Finalement, l'amélioration du rapport coût/performance qui découle des deux points précédents et qui représente le principal atout à l'utilisation d'une architecture Beowulf. Les *Beowulf clusters* sont décrits comme le moyen le plus efficace pour atteindre des capacités de calcul en GFLOPS pouvant rivaliser avec les clusters utilisant des technologies propriétaires, et ce, sans devoir en débourser le prix.

2.3 Benchmarking [13, 14, 15]

Le *benchmarking* désigne la pratique visant à établir une évaluation des performances d'un produit, en vue de son optimisation ou de l'analyse de ses limites matérielles et logicielles.

Ce chapitre fourni : premièrement, une description du *benchmarking* pour en comprendre l'essence ; deuxièmement, les points clés qui dictent la méthodologie de ceux-ci.

2.3.1 Description

Dans le cadre de l'analyse des performances des systèmes informatiques, le *benchmarking* est défini comme étant l'acte de mesurer et d'évaluer les performances informatiques, les protocoles de mise en réseau, les dispositifs et les réseaux, dans des conditions de référence, par rapport à une évaluation de référence. Le but de ce processus d'analyse comparative est de permettre une comparaison équitable entre différentes solutions, ou entre des développements ultérieurs d'un système en cours de test. Ces mesures comprennent des paramètres de performance primaires, recueillis directement auprès du système en cours de test (par exemple, le débit de l'application, la consommation d'énergie du nœud) et, dans le cas des réseaux sans fil, également des paramètres de performance secondaires, caractérisant l'environnement dans lequel le système en cours de test fonctionne (par exemple, les caractéristiques d'interférence, l'occupation des canaux). Les mesures de performance primaires et secondaires peuvent être complétées par des mesures technico-économiques, telles que le coût du dispositif ou la complexité opérationnelle.[15]

Ensuite, parmi les critères de performances, se retrouvent principalement deux termes que sont : efficacité et efficience ; termes qui caractérisent : d'une part, la capacité à atteindre l'objectif défini pour une entité ; d'autre part, la capacité à atteindre cet objectif de manière optimisée et en faisant preuve de bonnes performances.

Quant aux types de *benchmarks*, il en existe plusieurs, mais l'activité se divise principalement en deux types :

- premièrement, les *benchmarks* synthétiques qui visent à mesurer et étudier les performances d'un ou de plusieurs composants d'un appareil, de manière instantanée ou continue ;
- deuxièmement, les *benchmarks* applicatifs qui visent à mesurer et étudier les performances d'une application ou d'un service, habituellement sur des serveurs de production ou lors de la distribution d'une application.

2.3.2 Méthodologie

Les *benchmarks* ont pour objectif de fournir des résultats fiables et cohérents afin d'être optimaux et afin de remplir leur rôle de procédure d'évaluation. Pour cela, plusieurs points clés sont à considérer. Ils permettent de fixer une ligne directrice lors des évaluations afin qu'elles soient pertinentes.

Ces points clés sont les suivants :

- définir les critères d'efficience et d'efficacité qui seront étudiés au cours de l'évaluation, et ce, en veillant à déterminer ce qui apporte le maximum de valeur aux actifs ciblés ;
- définir un système d'indicateurs clés qui permettra de déterminer si les chiffres sont interprétables et pertinents lors d'une comparaison qui se doit d'être la plus juste possible afin de ne pas biaiser les résultats obtenus ;
- déterminer l'ensemble des actifs ainsi que les outils utilisés au cours de l'évaluation ;
- définir les mesures à entreprendre en vue des résultats obtenus, en veillant à vérifier l'ensemble desdits résultats avec les indicateurs clés dictés au second point ;
- établir le comparatif et les gains entre les actifs pré et post *benchmark*, qu'il s'agisse d'optimisations ou de remplacements ;
- sur base des comparatifs effectués au point précédent, si le besoin se fait ressentir, établir une liste des mesures nécessaires au rééquilibrage du *benchmark* et des mesures à prendre en réponse sur les actifs ;
- dresser un plan de réalisation des bonnes pratiques et des mesures d'optimisations à réaliser sur les actifs, tout en considérant que certaines de ces mesures devront être réalisées dans un ordre chronologique ou dépendre d'une autre mesure antérieure ;
- contrôler les résultats obtenus et réaliser une critique des points précédents et de leur déroulement.

Si ces points sont correctement implémentés et alignés à la situation étudiée, ils aideront à perfectionner le processus d'amélioration continue autour des actifs et de leurs performances. Cependant, ces évaluations sont difficiles à adapter à toutes les situations. Le *clustering* fait partie de ces situations.

Deux choses sont alors à considérer :

- d'abord, rares sont les appareils qui fournissent des résultats équivalents à 100% des performances annoncées par les constructeurs. En effet, nombreux sont les facteurs qui peuvent altérer de manière minime ou, au contraire, de manière non négligeable les résultats obtenus. Par exemple, les plus communs sont : la dégradation due à l'utilisation, des défauts de fabrication, un taux d'humidité variable, les mises à jours système, une mauvaise alimentation, ou encore un mauvais système de refroidissement. Il est donc préférable de dresser, s'ils sont connus, une liste des paramètres et des conditions de test rencontrées lors de la réalisation de l'évaluation ;
- ensuite, des évaluations telles que celles-ci peuvent avoir du mal à s'adapter à toute situation, et les architectures clusterisées en font partie. Car il est difficile de calculer les performances globales et moyennes d'un ensemble variables d'appareils, il est nécessaire d'établir des indicateurs clés fiables et alignés avec l'architecture étudiée.

3 Étude bibliographique

La section *Étude bibliographique* présente : premièrement, les principales observations réalisées sur base de travaux connexes aux thématiques abordées ; deuxièmement, un tableau récapitulatif des attributs principaux de chaque travail ; troisièmement, les éléments qui permettent de motiver les choix pris dans ce travail.

Les travaux étudiés dans cette section sont tous issus du milieu académique ou scientifique. Cela permet de fournir une fondation correcte au présent travail de recherche.

Le Raspberry Pi (RPI) est un nano-ordinateur monocarte (SoC) à processeur ARM de la taille d'une carte de crédit conçu par des professeurs du département informatique de l'université de Cambridge dans le cadre de la RPI Foundation.[16]

Outre les travaux utilisés comme références, un réel attrait se dessine autour des RPI depuis 2012, année de distribution du premier modèle. Cet attrait est issu de la notoriété grandissante, de l'architecture peu énergivore, mais surtout par le rapport coût/performances toujours plus ambitieux des SoC produites par la RPI Foundation.

Dans la littérature autour de l'utilisation de ces SoC, bien d'autres thématiques ont cependant été abordées. Ainsi, l'étude bibliographique se limite à celles qui se rapprochent assez étroitement de la tâche à accomplir. Bien que certains des travaux se voient quelque peu écartés en raison de leurs objectifs, la base commune de ces articles reste le *cluster computing*, l'utilisation de RPI et l'évaluation des performances de ces derniers.

Quant à la direction adoptée dans les prochains points, elle est la suivante :

- d'abord, présenter les dimensions principales rencontrées dans les articles ;
- ensuite, lier les concepts théoriques nécessaires à leur bonne compréhension ;
- finalement, citer les choix effectués par les auteurs.

3.1 Objectifs

Les documents sélectionnés ont tous pour objectif l'évaluation des performances dans un cluster de RPI. Cependant, chacun aborde ces évaluations de manière différente.

Dans [17] [18] [19] et [20] l'objectif est d'étudier les performances des clusters de RPI pour démontrer leur valeur face aux HPC conventionnels. Différents aspects sont considérés comme : les performances CPU en GFLOPS ; les performances en écriture et lecture sur les systèmes de fichiers ; les performances réseau.

Dans [21], en revanche, cette évaluation sera tournée vers le monde de la cybersécurité. Leur Bramble cluster a pour objectif d'établir les performances du cluster lors d'un exercice de *password cracking*.

Quant à [22], son objectif est d'effectuer les évaluations dans un cluster disposant d'Hadoop et de son système de fichiers distribué.

3.2 Modèles Soc

Réalisés sur plus d'une décennie, les travaux précédemment présentés ont chacun subi l'évolution des technologies. Cette évolution a donc fortement influencé le modèle des RPI choisi pour implémenter les clusters.

Ainsi, dans [22], travail mené entre 2012 et 2013, le modèle de RPI utilisé est le RPI première version. Dans [20], [17] et [18], respectivement réalisés en 2015, 2016 et 2018, les modèles choisis sont la RPI 2 et la RPI 2 model B.

Ensuite, dans [21] en 2019, c'est une nouvelle fois le premier modèle de RPI qui est employé. Cela marque une divergence dans la tendance, habituellement linéaire, du modèle utilisé en fonction de l'année de réalisation de l'étude. Cependant, rien ne justifie de ne pas porter une nouvelle fois de l'intérêt aux anciennes versions du RPI, bien que cela ne soit pas cité explicitement dans l'article.

Finalement, dans [19], qui date de 2020, c'est le RPI 3 model B qui est utilisé.

3.3 Architecture

De la même manière que les modèles de RPI utilisés, les architectures de cluster ont aussi évolué au fil du temps, des découvertes et des études réalisées.

Dans [17] [18] et [20], les clusters implémentés suivent les principes des *Beowulf clusters*, architecture détaillée au chapitre *Beowulf cluster*. Le premier dispose d'un cluster de 9 noeuds, dont 1 master ; le second dispose de 12 noeuds, dont 1 master. Le reste étant divisé en deux groupes inégaux ; le dernier dispose 10 noeuds, dont 1 master, formant un cluster impair.

Quant à l'architecture des clusters de [19], [21] et [22] , elle semble suivre ces mêmes principes sans que l'architecture Beowulf ne soit mentionnée. Les trois architectures disposent, en effet, d'un ensemble de noeuds parfaitement similaires disposés dans un réseau LAN qui leur est propre et destiné au *parallel computing*. De plus, elles disposent toutes d'un système de fichiers soit partagé, soit distribué et emploient des technologies comme MPI et Hadoop pour réaliser leurs tâches. Des attributs qui définissent donc l'architecture Beowulf. Leurs clusters disposent respectivement : de 10 noeuds, dont 1 master ; 5 noeuds, dont 1 master ; 64 noeuds, dont 1 master.

Bien que le nombre de noeuds soit entièrement dépendant du budget alloué et de la tâche à réaliser, le choix de former un cluster pair ou impair est, quant à lui, laissé au seul choix des auteurs. Le choix d'opter pour un cluster impair est pourtant plus facilement justifiable. Bien que n'étant pas une vérité absolue, les résultats des études sont majoritairement présentés par tranche de 2 noeuds. Ainsi, il est préférable de ne pas calculer les performances réalisées par le master. Celui-ci n'étant pas parfaitement identique aux autres noeuds, par les services qu'il héberge et par son rôle au sein du cluster. Dans le cas contraire, cela peut mener les performances étudiées à être biaisées.

Dans ce cas de figure, le master a pour seule tâche de commander les autres noeuds. Les évaluations de performances se voient être plus cohérentes, car elles sont menées sur des noeuds parfaitement similaires. De plus, cela permet de réduire la charge sur ce dernier lors des évaluations, de réduire les latences internoeuds et de toujours garder un point d'accès au cluster.

3.4 Résumé des références

Les tableaux ci-dessous représentent un résumé des dimensions étudiées [Tab.1] dans ce chapitre, afin de rendre leur comparaison plus aisée.

Table 1: Résumé des dimensions étudiées

Article	**Publication**	**Noeuds**	**Modèle RPI**
[22]	2012	64	RPI
[20]	2015	10	RPI 2
[17]	2016	9	RPI 2
[18]	2018	12	RPI 2
[21]	2019	5	RPI
[19]	2020	10	RPI 3

Article	**Architecture**	**Stockage**	**Technologies**
[22]	x	HDFS	HPL, HDFS, MPICH
[20]	Beowulf	local	HPL, MPI4py
[17]	Beowulf	local	MPI
[18]	Beowulf	NFS	HPL, MPI4py, MPICH
[21]	x	NFS	MPICH, JohnTheRipper
[19]	x	NFS	HPL, MPI

Article	**Communications**	**Système d'exploitation**	**Attribution des IP**
[22]	x	Raspbian	x
[20]	SSH	Raspbian	statique
[17]	x	Raspbian	statique
[18]	SSH	Raspbian	DHCP
[21]	SSH	Kali Linux	statique
[19]	x	Raspbian	DHCP

3.5 Attribution des IP

La conception réseau et l'attribution des adresses IP au sein du réseau sont opérées de deux méthodes.

La première méthode consiste à configurer les adresses IP statiquement, comme dans [17], [21] et [20]. C'est une configuration qui permet aux noeuds de toujours disposer de la même adresse et d'être facilement identifiés dans le cluster. Cependant, cela peut représenter, dépendamment de la taille du cluster, une tâche longue et répétitive.

Dans ce cas, il est plus facile d'opter pour la seconde méthode, comme dans [19]. Dans celle-ci, les équipements réseau hors noeuds, comme le switch, sont utilisés pour configurer un DHCP. Ce DHCP attribue automatiquement les adresses IP aux noeuds, rendant la tâche plus facile. Cependant, cette méthode dispose aussi d'une contrepartie. Si le DHCP est simplement configuré, rien ne garantit que les adresses seront toujours attribuées aux mêmes noeuds. Une question qui n'est à soulever que s'il est nécessaire d'identifier individuellement chacun des noeuds. Un cas qui reste relativement rare au sein d'un cluster, et qui n'est pas abordé dans l'article.

Dans de plus rares cas, comme dans [18], c'est le master qui fait office de passerelle et de routeur aux noeuds. Cela implique que l'entièreté du trafic passe par ce dernier. Ce qui peut représenter une charge assez lourde à maintenir lors des mises à jour du cluster.

3.6 Système d'exploitation

Le système d'exploitation majoritairement choisi dans les travaux est Raspbian. Celui-ci est le système d'exploitation par défaut des Socs proposés par la RPI Foundation.

Alors que certains se limitent à cette motivation, d'autres, comme [19], [17] et [18] motivent aussi leur choix par une recherche des performances optimales. En effet, Raspbian est une distribution Debian spécialement conçue pour l'architecture ARM de la RPI. Cela amène ses performances à être vantées, en comparaison aux autres systèmes d'exploitation disponibles.

Quant à [21], seul à ne pas avoir opté pour Raspbian, il utilise Kali Linux. Ce choix est justifié : d'une part, par le désir de fournir un environnement destiné à la cybersécurité. Kali Linux se désigne comme étant la distribution par excellence pour cette situation ; d'autre part, par la nécessité d'obtenir plus facilement les outils utilisés dans cette étude. Ces derniers sont directement fournis par la distribution.

3.7 Technologies

Les technologies mises en évidence au travers de la bibliographie sont les suivantes :

- HPL est : un logiciel qui résout un système linéaire dense (aléatoire) en arithmétique de double précision (64 bits) sur des ordinateurs à mémoire distribuée. Il peut donc être considéré comme une implémentation portable et librement disponible du *benchmark* Linpack pour le calcul haute performance.

Le progiciel HPL requiert la disponibilité sur votre système d'une implémentation de l'interface MPI (compatible avec la version 1.1). Une implémentation des sous-programmes d'algèbre linéaire de base BLAS ou de la bibliothèque de traitement d'images de signaux vectoriels VSIPL est également nécessaire. Des implémentations spécifiques aux machines ainsi que des implémentations génériques de MPI, de BLAS et de VSIPL sont disponibles pour une grande variété de systèmes.[23]

- L'interface MPI (Message Passing Interface) est : une norme de passage de messages standardisée et portable conçue pour fonctionner sur des architectures de calcul parallèles. La norme MPI définit la syntaxe et la sémantique des routines de bibliothèque qui sont utiles à un large éventail d'utilisateurs écrivant des programmes de passage de messages portables en C, C++ et Fortran.[24]

- MPICH est : une implémentation performante et largement portable de la norme MPI (Message Passing Interface). Les objectifs de MPICH sont : (1) de fournir une implémentation MPI qui supporte efficacement différentes plates-formes de calcul et de communication, y compris les clusters de commodité, les réseaux à haut débit et les systèmes de calcul haut de gamme propriétaires ; (2) de permettre une recherche de pointe sur MPI grâce à un cadre modulaire facile à étendre pour d'autres implémentations dérivées.[25]

- MPI4py est : une librairie qui fournit des liaisons Python pour le standard MPI, permettant aux applications Python d'exploiter plusieurs processeurs sur des stations de travail, des clusters et des superordinateurs.[26]

Dans [17], MPI est utilisé dans sa forme la plus simple au travers de scripts en *parallel computing* visant à calculer les performances du cluster. Dans cet article, les auteurs déconseillent HPL. Ils avancent que les performances sont fort dépendantes de l'algorithme utilisé lors des tests. L'algorithme peut alors devenir source de pertes de FLOPS et donc non pertinent. En contrepartie, les auteurs proposent de regarder le temps nécessaire à la résolution d'un problème donné. En l'occurrence, leur choix s'est porté sur un calcul des nombres premiers, car jugé linéaire lorsque l'algorithme est parallélisé.

Cependant, HPL reste le logiciel de *benchmarking* le plus utilisé et donc le plus propice à fournir des résultats facilement comparables. C'est pourquoi il est la solution envisagée par [22], [20], [19], [18].

Dans [21], en revanche, c'est l'utilisation de MPI qui est employée, mais cette fois groupée avec JohnTheRipper, un logiciel visant à craquer des mots de passe. Grâce à MPI, JohnTheRipper va pouvoir être parallélisé afin d'être exécuté sur l'ensemble des noeuds. Ainsi, les performances cumulées des noeuds permettent d'accélérer l'exercice de *password cracking*.

3.8 Stockage

Une autre considération fondamentale abordée lors de l'implémentation des clusters est celle du partage des données. En effet, l'attrait principal d'un cluster est de faire transparaître un ensemble de machines comme une seule et même entité. Pour cela, il est très courant de rencontrer des systèmes de fichiers : soit partagés, soit distribués.

Plusieurs méthodes permettent d'atteindre ce but : 1) disposer d'une machine dédiée à cet effet, auquel les noeuds du cluster ont tous accès ; 2) utiliser le stockage local de chacun des noeuds et le faire transparaître comme un seul et même stockage. Dans ce cas de figure, c'est un stockage distribué ; 3) au travers du protocole NFS, utiliser le stockage local d'une des machines du cluster comme stockage partagé et centralisé pour les noeuds.

Dans [21], [19] et [18], c'est la solution du stockage partagé grâce à NFS qui est utilisée. NFS : permet aux utilisateurs du réseau d'accéder à des fichiers partagés via TCP/IP. Grâce à des ordinateurs individuels jouant à la fois le rôle de clients et de serveurs, les utilisateurs de NFS peuvent manipuler les fichiers comme s'ils étaient stockés localement.[27] Dans les deux premiers articles, ce stockage est partagé par le master et opéré grâce à sa carte SD local. Quant au troisième, le stockage est partagé par son master, mais cette fois via un HDD connecté. Cette solution est motivée d'une part, par le désir de vouloir simplifier le partage des librairies ; d'autre part, pour simplifier la sauvegarde des résultats générée par les noeuds lors des évaluations.

Ensuite, dans [22], la solution est celle du stockage distribué. Celui-ci est implicite et amené par l'utilisation d'Hadoop. Le stockage des noeuds est alors perçu comme un stockage unique où les fichiers sont segmentés et ensuite liés sur le stockage local de chaque noeud. Bien que cela ne soit pas spécifié par les auteurs, cette solution vise habituellement à augmenter la vitesse en écriture et lecture des fichiers volumineux ainsi qu'à permettre une redondance des données, optimale lors de la perte d'un noeud.

Quant au cluster de [20], il ne dispose pas d'un stockage partagé et préfère l'utilisation de *Shell scripts* employés afin de copier les fichiers nécessaires sur les noeuds cibles.

3.9 Communications master-nodes

Un autre point d'intérêt soulevé dans les articles est la méthode employée afin de faciliter les communications dans le cluster. Plus spécifiquement, c'est la communication entre le master et les noeuds qui est abordée.

Dans [21], [18] et [20], la solution employée est celle la plus communément rencontrée : SSH. Elle est utilisée conjointement à l'utilisation des clés SSH.

Le protocole Secure Shell (SSH) est un protocole réseau cryptographique permettant d'exploiter des services réseau en toute sécurité sur un réseau non sécurisé. Ses applications les plus notables sont la connexion à distance et l'exécution de lignes de commande. Les applications SSH sont basées sur une architecture client-serveur, reliant une instance client SSH à un serveur SSH.[28]

Quant aux clés SSH, elles permettent d'éviter l'utilisation des noms d'utilisateur et des mots de passe lors de la connexion aux noeuds. Il est alors nécessaire de générer cette dernière sur les noeuds, et ensuite de la transmettre au master par une simple commande. De la même manière, la clé du master devra être partagée afin de pouvoir authentifier ce dernier lors d'une demande de connexion.

Cette solution permet, depuis le master, d'accéder à l'ensemble des noeuds sans devoir effectuer d'entrée au clavier. Cela vient considérablement renforcer l'efficacité des procédés automatisés au sein du cluster.

3.10 Résultats

Dans ce chapitre, les résultats relevés dans les articles et les réflexions amenées par les auteurs sont présentés.

- Dans [17], les résultats sont issus de l'utilisation de scripts MPI et l'architecture utilise des RPI 2.

  Afin de générer les résultats, le *benchmark* est divisé en 10 itérations où le nombre de processus sur chaque noeud varie entre 1 et 36. De plus, des pauses de 5 minutes sont effectuées entre chaque itération. Ces pauses sont motivées par la volonté de réduire au maximum les facteurs externes, tels que la température du cluster, dans les résultats obtenus.

Process Count	1	4	8	12	16	20	24	28	32	36
D 1	8046	3561	1605	1038	766	612	506	433	379	333
D 2	7985	3550	1605	1038	769	609	506	431	376	333
D 3	8009	3540	1604	1039	769	609	508	433	378	335
D 4	8028	3544	1598	1038	765	612	506	431	377	335
D 5	7983	3543	1597	1038	769	612	507	433	376	333
D 6	7980	3562	1597	1033	769	609	506	434	378	333
D 7	8026	3557	1601	1038	766	613	508	431	376	335
D 8	7983	3557	1605	1033	769	609	506	433	378	333
D 9	8035	3540	1604	1033	765	609	506	432	376	333
D 10	8026	3556	1597	1033	769	609	506	434	376	335
Average Time(s)	**8010**	**3551**	**1601**	**1036**	**767**	**610**	**506**	**432**	**377**	**333**

Figure 3: [17] Timing Results of Tests

La figure [Fig.3] représente les résultats en secondes nécessaires pour que l'ensemble du cluster termine le *benchmark* pour chaque itération en fonction du nombre de processus par noeud.

Les résultats tendent vers une certaine stabilité et une réduction des écarts à la moyenne en augmentant le nombre de noeuds. Il est alors avancé que l'algorithme utilisé est adapté pour être parfaitement linéaire. Cependant, des écarts se démarquent. Selon l'auteur, ces derniers sont causés par les communications internoeuds grandissantes avec le nombre de noeuds employé. Les résultats restent cependant linéaires.

Dans l'article, l'observation formulée par les auteurs est : "*Même les problèmes hautement parallélisables peuvent perdre l'immense avantage qu'ils ont en matière de rapidité lorsque la puissance de traitement nécessaire est atteinte*". [17]

En effet, au vu de la figure [Fig.4], en doublant le nombre de noeuds pour atteindre 72 coeurs (et donc 72 processus), un résultat théoriquement deux fois plus bas est obtenu, et ce, pour le double du coût. Ce gain de performances est alors considéré comme négligeable.

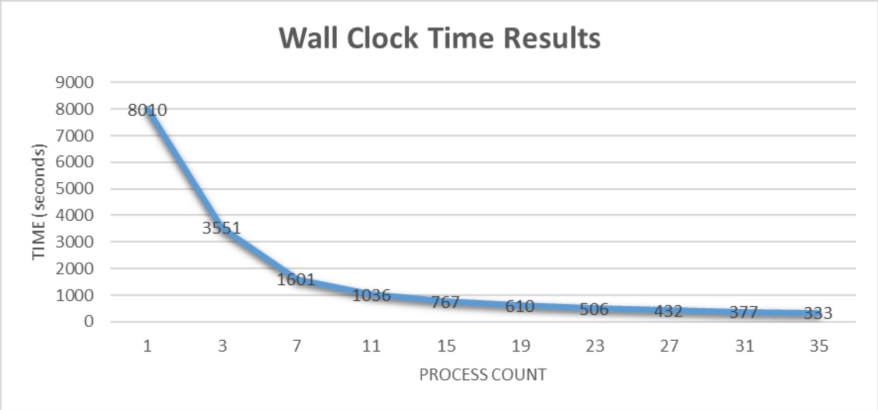


Figure 4: [17] Timing Graph of Tests (Root Process Excluded)

Deux conclusions sont alors tirées : d'abord, lorsque la barrière de la puissance de traitement nécessaire est atteinte, les performances du cluster perdent l'emprise qu'elles avaient sur les performances ; ensuite, il est important de considérer l'augmentation de l'*overhead* causée par les communications internoeuds afin de réduire celles-ci au mieux.

- Dans [19], les performances sont calculées sur base d'un algorithme de tri de transactions pair impair, et le cluster utilise des RPI 3. Les résultats sont générés en augmentant : d'une part, le nombre de processus ; d'autre part, le nombre de processeurs, et donc le nombre de noeuds.

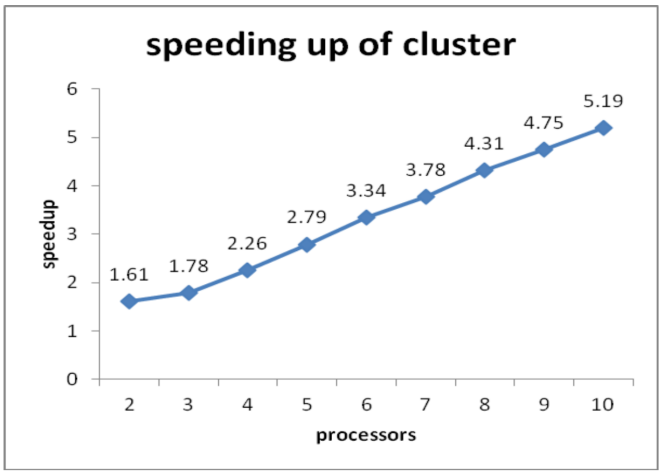


Figure 5: [19] Speedup of the odd-even transposition sort algorithm using Rpi

Quant aux résultats représentés sur la figure [Fig.5], ils tendent vers une augmentation presque linéaire du nombre d'opérations par seconde en fonction du nombre de noeuds. Ainsi, une évolution des performances similaires à celles rencontrées dans les articles précédents est retrouvée. Finalement, les performances observées sont jugées très prometteuses.

- Dans [20], le Pi Crust est constitué de 10 noeuds RPI 2 Quad-core cadencés à 900MHz.

  Quant aux résultats, ils sont segmentés en deux parties : d'abord, la matrice de multiplication en MPI qui montre des résultats linéaires de l'évolution du nombre d'opérations en fonction du nombre de noeuds ; ensuite, les résultats issus d'HPL qui montrent une évolution des MFLOPS en fonction nombre de noeuds.

  Dans la matrice, les résultats obtenus sont comparés au Texas AM [29] Supercomputer. Il est observé, par l'auteur, que ces derniers témoignent de performances bien plus faibles, mais qu'ils démontrent aussi une mise à l'échelle similaire avec le superordinateur.

  Quant aux résultats fournis par HPL, ils démontrent l'efficacité du cluster. En effet, l'auteur avance que ces résultats sont : *"Supérieurs à un Intel i7 3930K, processeur Hexa-core cadencé à 3.2GHz".* [20]

- Dans [22] les résultats sont segmentés en deux parties : d'abord, une partie en *single node* où HPL est exécuté sur une RPI *single core* ; ensuite, une partie sur les performances du cluster, de 1 à 64 noeuds.

  La première partie [Fig.6] met en évidence l'évolution des performances en kFLOPS en fonction de la taille du problème. Les résultats montrent alors une décroissance des performances allant de 65000 kFLOPS pour un problème de taille 100 vers approximativement 40000 kFLOPS pour un problème de taille 1000. Ces derniers démontrent alors l'importance de considérer différentes tailles de problème afin d'offrir une vue pertinente des performances dégagées par les RPI.

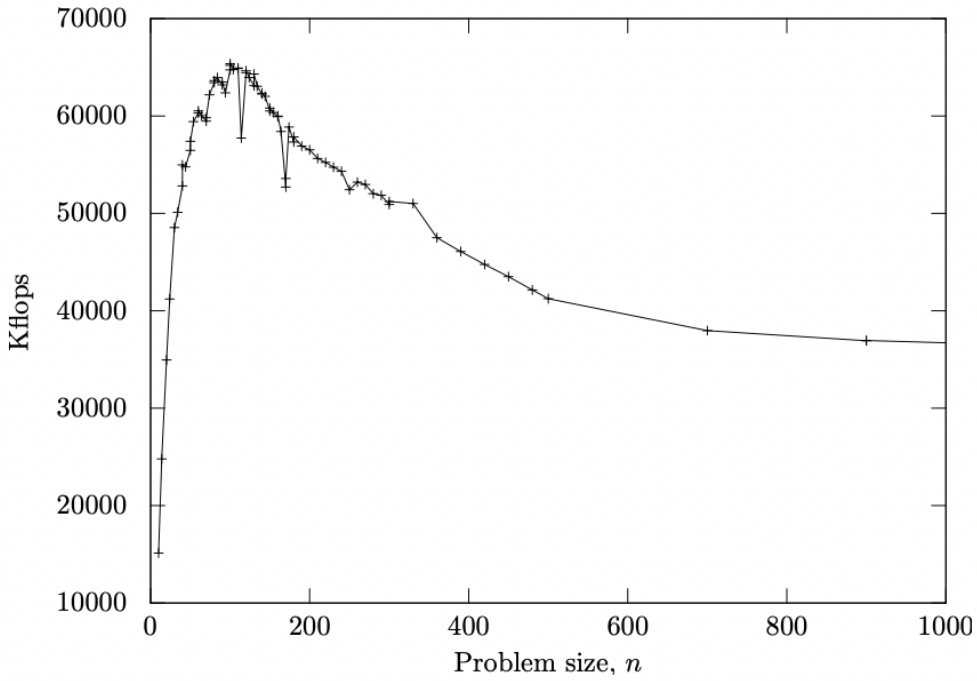


Figure 6: [22] Single precision performance of a single node on various problem sizes n using the LINPACK benchmark (tests carried out on Raspbian 16th Dec 2012 release with the Linux 3.6.11+ kernel)

Ensuite, dans la seconde partie [Fig.7], les mêmes résultats sont étudiés, mais en faisant varier : d'une part, la taille du problème ; d'autre part, le nombre de noeuds. Les observations principales sont : 1) plus la taille du problème est élevée, plus les performances dégagées tentent à être linéaires ; 2) de petites tailles de problèmes impliquent un accroissement des communications internoeuds. Cela amène le cluster à atteindre un seuil. Une fois ce seuil atteint, le gain en performances dû à l'ajout de nouveaux noeuds au cluster se voit relativement réduit, voir nullifié.

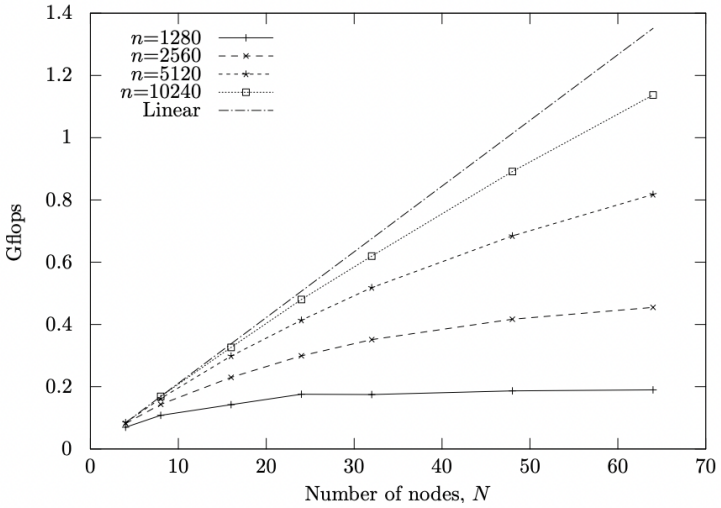


Figure 7: [22] Computational performance measured using HPL benchmark, as a function of number of nodes, N, for various orders n of the matrix A.

4 Motivations

La section *Motivations* décrit les choix réalisés et vient appuyer la direction prise dans le présent travail sur base de l'*Étude bibliographique* et les informations mises en évidence dans cette dernière.

Les choix effectués sont les suivants :

- Les études réalisées depuis plus de 10 ans sur les clusters de RPI témoignent d'un attrait toujours grandissant autour de ces dernières. Ainsi, le présent travail se concentre sur l'évaluation des performances du dernier modèle du RPI, le RPI 4 model B.
- L'architecture Beowulf tend à être la solution favorisée par le plus grand nombre, et par son faible coût ainsi que son rapport étroit avec le monde académique. La solution implémentée dans ce travail se base sur les principes des *Beowulf clusters* décrits au point *Principes* du chapitre *Beowulf cluster*.
- L'évolutivité du cluster est un atout majeur dans son implémentation, une infrastructure réseau et une configuration des RPI aisées et à long terme sont envisagées.
- L'évaluation du cluster repose sur l'exécution d'HPL. Il représente comme la solution à l'évaluation des performances des clusters RPI la plus pertinente, dû à sa linéarité, ainsi que celle qui favorise la comparaison des résultats.
- L'utilisation d'un stockage partagé via NFS représente la solution de stockage la plus évolutive et malléable au sein d'un cluster.
- SSH représente comme la solution de communication et de gestion des noeuds la plus évolutive et malléable
- Les résultats obtenus, considérés comme prometteurs, témoignent de l'intérêt des Socs de la RPI Foundation. Ainsi, le présent travail se concentre sur les attributs et les optimisations qui permettent d'obtenir le meilleur rapport performances par noeud au sein d'un cluster.

5 Description du système

La section *Description du système* décrit : premièrement, l'ensemble des composants utilisés dans ce travail ; deuxièmement, les technologies et les logiciels utilisés ainsi que leurs rôles respectifs ; troisièmement, l'architecture de la solution.

5.1 Matériel

Le matériel utilisé dans ce travail est composé des éléments suivants :

- 9x RPI 4 model B.

  Le Raspberry Pi 4 model B [Fig.8] est le dernier modèle en date proposé par la fondation du même nom. Disponible depuis juin 2019, il représente, en raison de ses performances, un réel tournant dans l'utilisation des appareils de la marque et dans la mise à disposition de nano-ordinateurs performants au grand public. [30]

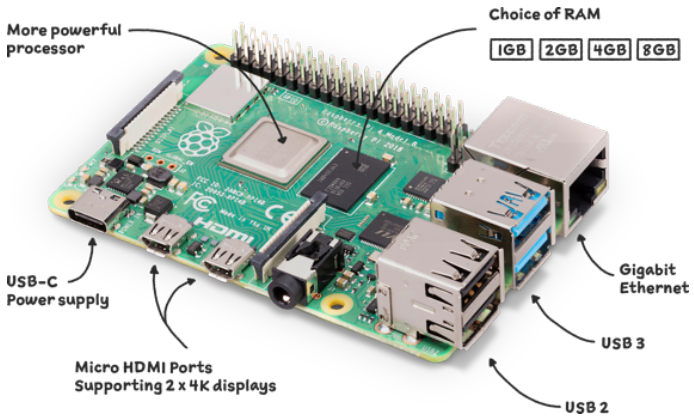


Figure 8: Raspberry Pi 4

Quant aux spécifications de la RPI, elles sont les suivantes :

- un processeur Quad-core cadencé à 1.5GHz qui dispose de l'architecture ARMv8 64Bits, tout en restant compatible ARMv7 32Bits ;
- une capacité en RAM LPDDR4-3200 SDRAM de 4Gb ;
- une puce WiFi certifiée IEEE 802.11ac avec des fréquences en dual-band de 2.4GHz ou 5GHz ;
- une connectivité Bluetooth 5.0 ;
- une alimentation 5V en USB C (précédemment micro usb) ;
- une compatibilité avec les modules de Power over Ethernet, permettant aux RPI d'être alimentés directement par port Ethernet, bien que nécessitant un PoE Hat et un switch disposant des fonctionnalités nécessaires ;
- pour le reste, il s'agit principalement des I/O de l'appareil tels que : deux ports USB 3.0, 2 ports USB 2.0, 2 ports micro-HDMI, des connectiques pour l'audio, la caméra et l'affichage.

- 5x kits Xute Raspberry Pi 4 [31] & 4x kits Labists Raspberry Pi 4 [32]

  Les kits Xute et Labists [Fig.9] sont constitués des composants suivants : un boitier PC/ABS de protection ; une alimentation USB-C 5V ; 2 câbles micro-HDMI (non utilisés dans ce travail) ; 3 dissipateurs thermiques pour les puces de la RPI ; un ventilateur pour le CPU ; une carte SD SanDisk Ultra de 64GB (Xute) ou 32 GB (Labists) (et son lecteur compatible USB-A et USB-C.

Figure 9: kits Xute & Labists Raspberry Pi 4

- Un NAS Synology RS815+ [Fig.10] qui intègre : 4 baies de stockage en 3.5" ou 2.5", pour un maximum de 24 TB ; 2GB de mémoire RAM interne ; des connecteurs en S-ATA III ; des fonctionnalités RAID, NFS, SSH, FTP, etc. ; 4 ports Ethernet Gigabit ; et un processeur Quad-core cadencé à 2400 MHz.

Figure 10: NAS Synology RS815

- Un switch Cisco Catalyst 3560 Series [Fig.11] disposant de : 48 ports Ethernet en base 10/100/1000 ; 4 Ports SFP Gigabit Ethernet ; 128 MB de RAM ; 32 MB de DRAM ; et des capacités POE, non utilisées dans ce projet mais valorisantes pour le futur du cluster.

Figure 11: Switch Cisco Catalyst 3560 Series

- 4x SSD Crucial CT500MX500SSD1 disposant de : une capacité de stockage annoncée de 500 Go ; une vitesse de lectures/écritures séquentielles jusqu'à 560/510 Mo par seconde ; une interface SATA 6Gb/s 2,5 pouces.
- 15x Câbles Ethernet Gb/s UTP Catégorie 6 pour relier les composants au Switch Cisco Catalyst 3560 ainsi que 3x multiprises pour l'alimentation du cluster.

5.2 Logiciels et technologies

La liste ci-dessous décrit les logiciels et les technologies utilisés lors de la réalisation de ce travail :

- **Ansible** est une plateforme logicielle libre pour la configuration et la gestion des ordinateurs. Elle combine le déploiement de logiciels multi-nœuds, l'exécution des tâches ad hoc, et la gestion de configuration. Elle gère les différents nœuds à travers SSH et ne nécessite l'installation d'aucun logiciel supplémentaire sur ceux-ci. Les modules communiquent via la sortie standard en notation JSON et peuvent être écrits dans n'importe quel langage de programmation. Le système utilise YAML pour exprimer des descriptions réutilisables de systèmes, appelées *playbook*.[33]

- **HPL** est un logiciel qui résout un système linéaire dense (aléatoire) en arithmétique de double précision (64 bits) sur des ordinateurs à mémoire distribuée. Il peut donc être considéré comme une implémentation portable et librement disponible du *benchmark* Linpack pour le calcul haute performance.

  Le paquet HPL fournit un programme de test et de chronométrage pour quantifier la précision de la solution obtenue ainsi que le temps qu'il a fallu pour la calculer. Les meilleures performances réalisables par ce logiciel sur votre système dépendent d'une grande variété de facteurs. Néanmoins, avec certaines hypothèses restrictives sur le réseau d'interconnexion, l'algorithme décrit ici et son implémentation jointe sont extensibles dans le sens où leur efficacité parallèle est maintenue constante par rapport à l'utilisation de la mémoire par processeur.[23]

- **MPI** (Message Passing Interface) est une norme de passage de messages standardisée et portable conçue pour fonctionner sur des architectures de calcul parallèles. Le standard MPI définit la syntaxe et la sémantique des routines de bibliothèque qui sont utiles à un large éventail d'utilisateurs écrivant des programmes de passage de messages portables en C, C++ et Fortran.[24]

- **NFS** (Network File System) permet à un utilisateur d'accéder, via son ordinateur (le client), à des fichiers stockés sur un serveur distant. Il est possible de consulter, mais aussi mettre à jour ces fichiers, comme s'ils étaient présents sur l'ordinateur client (c'est-à-dire comme des fichiers locaux classiques). Des ressources peuvent ainsi être stockées sur un serveur et accessibles via un réseau par une multitude d'ordinateurs connectés. Le NFS permet aussi un travail collaboratif sur un même document, ainsi que la sauvegarde et la centralisation de documents sur un même serveur.[34]

- **Sysbench, hardinfo, dd, hdparm, iperf et 7z** sont des logiciels utilisés afin de récolter des informations sur un système. Ils se concentrent sur les caractéristiques suivantes : les performances CPU, mémoire et écriture/lecture pour sysbench ; les performances globales du système ainsi que des *benchmarks* pour hardinfo ; les performances en écriture/lecture pour dd et hdparm ; les performances réseau pour iperf ; et la vitesse de compression et décompression pour 7z.

5.3 Architecture

L'architecture du cluster [Fig.12] se présente comme suit :

- Le cluster dispose de 9 noeuds RPI 4 model B, dont l'un est le master et les 8 autres sont des noeuds de calcul.
- Le cluster est divisé en trois zones de la manière suivante : **a)** la zone master qui sert d'interface et d'accès au cluster pour l'utilisateur ; **b)** la zone des noeuds qui contient les RPI qui exécuteront les tests ; **c)** la zone de stockage constituée du NAS Synology RS815+ qui sert de stockage partagé pour les noeuds.
- Les composants du cluster sont connectés au switch Cisco Catalyst 3560 afin de pouvoir disposer d'un réseau LAN.
- Le NAS Synology RS815+ est accessible au travers de NFS, et centralise les résultats (fichiers) produits par les noeuds.
- Le master communique avec l'ensemble des noeuds : d'une part, avec Ansible pour exécuter les *playbooks* de configuration et de *benchmark* ; d'autre part, par MPI afin d'exécuter HPL.
- Tous les composants connectés au switch Cisco Catalyst 3560 disposent d'une connexion internet fournie par l'établissement.

De par les points exprimés ci-dessus, le cluster réalisé valide en partie les *principes* de l'architecture Beowulf présentés au chapitre *Beowulf Cluster*. En effet, il respecte les principes des *Beowulf clusters* au travers de son architecture, son type de cluster et son objectif, mais utilise du matériel propriétaire : switch Cisco Catalyst 3560 et NAS Synology RS815+. Le cluster ne peut donc pas être considéré comme tel, bien que s'en rapprochant étroitement.

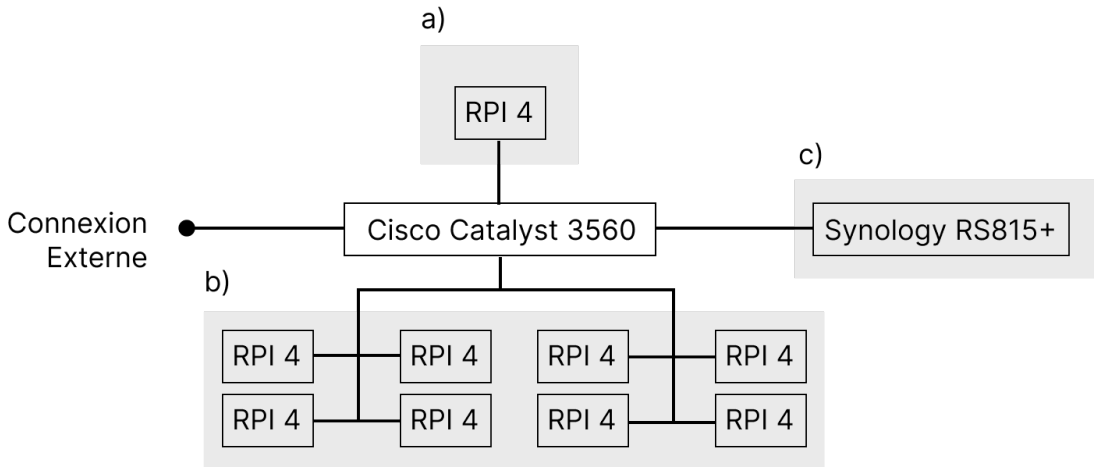


Figure 12: Architecture du cluster

6 Évaluation

La section *Évaluation* décrit : premièrement, la méthodologie adoptée lors des évaluations de performances ; deuxièmement, l'environnement dans lequel ces évaluations sont réalisées ; troisièmement, l'ensemble des attributs observés ; quatrièmement, le système de score utilisé pour quantifier les performances ; cinquièmement, la création de profils de performances.

6.1 Méthodologie

Le processus d'évaluation est divisé en trois phases : d'abord, une phase de validation où les noeuds sont testés individuellement au travers d'outils de *benchmarks* et comparés les uns aux autres afin d'établir les performances moyennes d'un RPI et pour de déceler de potentielles anomalies ; ensuite, une phase d'optimisation où les configurations par défaut des noeuds sont altérées afin d'améliorer les performances de ces derniers, et ce, sous plusieurs métriques ; finalement, une phase d'évaluation en cluster où les performances sont calculées en GFLOPS au travers d'HPL afin de déterminer les capacités du cluster ainsi que les profils les plus prometteurs.

Les trois phases citées se rapportent aux chapitres *Phase 1 : évaluation individuelle*, *Phase 2 : optimisation* et *Phase 3 : évaluation en cluster* de la section *implémentation*.

6.2 Validation

Cette première évaluation vise, avant tout, à la validation des noeuds utilisés dans le cluster. Cette validation est subdivisée en trois objectifs :

- premièrement, récolter les valeurs sur les performances des noeuds sous plusieurs dimensions et plusieurs métriques comme les performances CPU, les performances mémoires, les performances en lecture et écriture sur les stockages et les performances réseau. L'ensemble de ces valeurs sert ensuite à attribuer un score global moyen aux RPI dans leur état par défaut.

- deuxièmement, déceler les anomalies au sein des RPI. Ces anomalies peuvent être de plusieurs natures, telles qu'un dysfonctionnement matériel ou des performances dégradées dues à un problème de fabrication, une corruption du stockage lors de la réplication de la carte, de mauvaises conditions de test induisant une hausse de la température, etc. L'étude individuelle des performances des RPI permet ainsi de déterminer si les noeuds présentent tous des performances relativement similaires qui ne biaiseront pas les résultats obtenus lors de l'évaluation en cluster. Dans le cas d'une anomalie, le RPI est réinstallé et vérifié à nouveau afin de juger si oui, ou non, il a sa place dans le cluster.

- troisièmement, quantifier et donner une représentation graphique aux métriques récoltées. Ainsi, un score moyen est attribué aux performances des RPI pour les métriques suivantes : CPU, mémoire, stockage et réseau. Ces scores sont ensuite utilisés, dans le chapitre *Optimisation*, comme comparatif afin de déterminer quelles sont les configurations qui avantagent l'une ou l'autre dimension.

6.3 Optimisation

Dans le processus de *benchmarking* du cluster, l'étape d'optimisation a pour but d'améliorer, plus ou moins considérablement, les performances dégagées par les RPI. Cette optimisation est principalement réalisée au travers de configurations logicielles, et non matérielles.

Sur les RPI, c'est dans le fichier */boot/firmware/config.txt* que se trouve la majeure partie des optimisations pouvant être apportées. Les variables et valeurs considérées par ce fichier sont listées dans la documentation officielle sur le site web Raspberryp[35].

S'y retrouve : les options d'affichages ; les options au *boot* (démarrage) ; les options de contrôle des I/O ; les options d'*overclocking* ; les options de gestion de la mémoire ; et d'autres options liées au hardware des RPI.

Quant au reste des optimisations, il s'agit de l'installation et de la dés-installation de logiciels, de la dés-activation de services ou encore de modifications apportées au système d'exploitation.

Une fois ces configurations et optimisations effectuées, un score global moyen ainsi qu'un score moyen par métrique sont attribués aux RPI afin de constituer les différents profils d'optimisation. Ces profils sont ensuite utilisés dans le chapitre *Evaluation en cluster* afin de déterminer quels sont les profils qui fournissent les meilleures performances à l'exécution d'HPL.

6.4 Évaluation en cluster

La phase d'évaluation en cluster en prise en charge par le logiciel HPL. Elle a pour objectif de capturer les performances des noeuds en situation de cluster et de déterminer quels sont les profils qui offrent les meilleures performances dans une situation où le CPU est mis à l'épreuve. HPL utilise le GFLOPS comme métrique, et c'est sur base de cette dernière que les différents profils sont comparés.

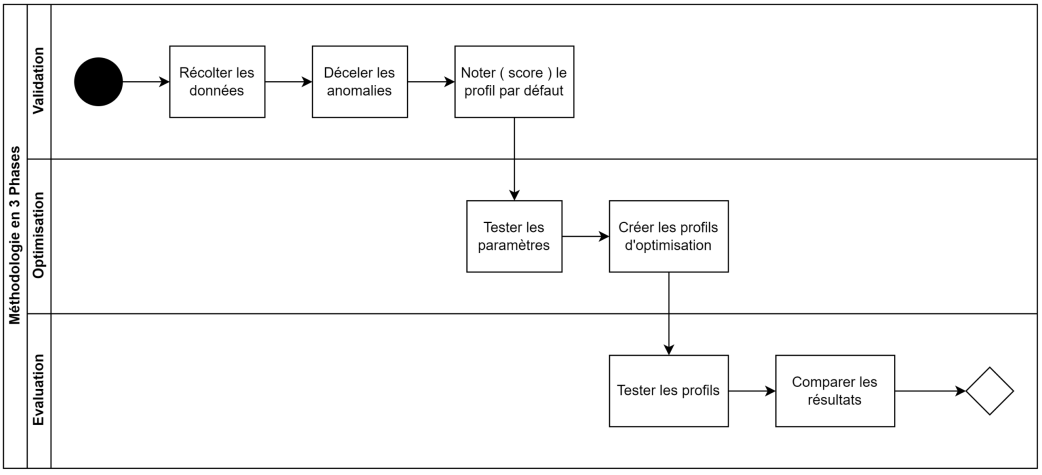


Figure 13: Méthodologie en 3 Phases

6.5 Conditions de test

Les conditions de test se divisent en deux catégories : d'abord, les conditions liées à l'environnement dans lequel le cluster est implémenté ; ensuite, les conditions dans lesquelles sont réalisées les évaluations.

Pour les conditions liées à l'environnement, elles sont relativement simples. Les RPI disposent tous du boîtier issu des kits Xute ou Labists présentés dans le chapitre *Matériel*. Ce boîtier dispose d'un ventilateur intégré et branché sur les pins 5V, le profil de refroidissement le plus puissant.

Outre cela, le cluster est implémenté dans une pièce dédiée, à température ambiante et sans système de refroidissement spécifique. Les RPI sont disposés de manière à être alignés horizontalement, et non pas verticalement comme dans un rack, afin de rendre l'utilisation du ventilateur intégré la plus efficiente possible.

Quant aux conditions des évaluations, elles sont basées sur un seul principe : effectuer une pause de 10 minutes entre chaque exécution d'un *benchmark* (procédure complète) ainsi qu'une pause de 1 minute entre chaque outil du *benchmark*. Ces pauses ont pour objectif de laisser les noeuds retrouver leur état initial entre chaque exécution d'outil. Cela doit permettre d'obtenir des valeurs pertinentes et non biaisées pour les attributs étudiés.

6.6 Attributs étudiés

Ce chapitre décrit les attributs récoltés lors de la *Validation* et étudiés lors de la *Phase 1 : évaluation individuelle*. Ces valeurs sont générées par l'utilisation de 6 outils de *benchmark*. Elles sont utilisées afin d'attribuer un score aux noeuds du cluster : d'abord, dans leur état initial ; ensuite, après la phase d'optimisation.

6.6.1 7Z

L'évaluation exécutée par 7Zip permet de récupérer des valeurs en MIPS lors d'une phase de compression puis de décompression. Elle est segmentée en trois itérations : une première en *single-thread* ; une deuxième en *multi-thread* sans *hyper-threading* ; une troisième en *multi-thread* avec *hyper-threading*. Cela amène l'évaluation à être exécutée en variant : d'une part, le nombre de *threads* alloués au *benchmark*, par tranche de 1, 2 et 4 ; d'autre part, la mémoire RAM allouée au *benchmark*, allant d'approximativement 200MB à 1GB.

Plus précisément, lors de chaque itération, les valeurs en MIPS sont calculées pour les algorithmes LZMA, Deflate, BZip2, PPMD, Delta, BCJ, AES256CBC, CRC32, CRC64, SHA256, SHA1 et BLAKE2sp. Cela permet d'obtenir, au terme du *benchmark*, une moyenne fiable sur l'ensemble des algorithmes.

Les valeurs retenues pour l'étude des attributs sont : le R/U Rating (score normalisé pour 100% d'utilisation du CPU) moyen en MIPS pour la compression ; le R/U Rating moyen en MIPS pour la décompression. Cela pour chacune des trois itérations, pour un total de 6 valeurs par noeud.

Compression R/U Rate single-thread	Decompression R/U Rate single-thread
Compression R/U Rate multi-thread	Decompression R/U Rate multi-thread
Compression R/U Rate hyper-threading	Decompression R/U Rate hyper-threading

6.6.2 dd

L'outil dd est utilisé pour la copie simplifiée de données à bas niveau et permet de récupérer des mesures simples des performances d'I/O séquentielles. L'évaluation par dd est segmentée en trois itérations : une première avec une taille de blocs de 256MB ; une deuxième avec une taille de blocs de 512MB ; une troisième avec une taille de blocs de 1GB.

Lors de chacune des itérations, un seul fichier est écrit. De plus, l'option *conv= fdatasync* (synchronisation des I/O) est utilisée pour désactiver la mise en cache afin de fournir des résultats plus précis. L'opération est réalisée en local sur la carte SD des RPI.

La valeur retenue pour l'étude des attributs est la vitesse d'écriture du fichier en MB/s. Cela pour chacune des trois itérations, pour un total de 3 valeurs par noeud.

Local write speed 256	Local write speed 512	Local write speed 1024

6.6.3 hdparm

Hdparm permet l'évaluation de la vitesse en lecture. Il écrit directement en brute sur le stockage, sans tenir compte du système de fichiers afin d'évaluer au mieux les performances de ce dernier. L'évaluation est segmentée en deux itérations : une première où Hdparm est utilisé avec les paramètres standards ; une deuxième où le paramètre *–direct* permet de contourner la mémoire cache du stockage.

Les valeurs retenues pour l'étude des attributs sont : la vitesse de lecture en mémoire cache en MB/s ; la vitesse de lecture bufferisée sur le stockage en MB/s ; la vitesse de lecture O_DIRECT en mémoire cache en MB/s ; la vitesse de lecture O_DIRECT sur le stockage en MB/s. Cela représente un total de 4 valeurs par noeud.

Cached reads speed	Buffered disk reads speed
O_DIRECT cached reads speed	O_DIRECT disk reads

6.6.4 iperf

Iperf permet une mesure active de la bande passante maximale atteignable en réseaux. Il nécessite d'établir une connexion entre une machine serveur et une machine client afin de calculer la bande passante en Mb/s. L'évaluation par iperf est segmentée en cinq itérations en faisant varier la taille des blocs comme suit : 8KB, 64KB, 256KB, 512KB, 1024KB.

La valeur retenue pour l'étude des attributs est la bande passante lors d'un transfert Mb/s. Cela pour chacune des cinq itérations, pour un total de 5 valeurs par noeud.

bandwith 8	bandwith 64	bandwith 256
bandwith 512	bandwith 1024	

6.6.5 hardinfo

Hardinfo est un profileur système et un outil de *benchmark* pour les systèmes Linux, qui rassemble des informations sur le matériel et le système d'exploitation. Il effectue des tests de référence et génère des rapports. Ce dernier est utilisé afin d'automatiser l'exécution de 6 *benchmarks* destinés à évaluer les performances du CPU sous divers aspects.[36]

Ces six *benchmarks* [36] sont les suivants :

- **CPU Blowfish** est un algorithme de chiffrement par blocs de 64 bits à clé symétrique.
- **CPU CryptoHash** est une fonction de signature cryptographique qui associe des données de taille arbitraire à un tableau de bits de taille fixe. Il s'agit d'une fonction à sens unique, qu'il est pratiquement impossible d'inverser, et qui est utilisée dans les signatures numériques, l'authentification des messages et les fonctions de hachage pour indexer les données dans les tables de hachage.
- **CPU Fibonacci** est une série de nombres dans laquelle chaque nombre est la somme des deux nombres précédents, comme 1, 1, 2, 3, 5, 8, etc. Il teste la capacité de traitement des nombres entiers d'un CPU.
- **CPU N-Queens** trouve un moyen de placer un nombre variable de reines sur un échiquier de façon à ce que deux reines ne se menacent pas mutuellement en partageant la même ligne, colonne ou diagonale.
- **CPU Zlib** est une librairie utilisée pour la compression des données, qui est utilisée par le programme de compression de fichiers gzip. Ce *benchmark* est exigeant en mémoire, ses résultats reflètent donc la vitesse de la RAM.
- **FPU FFT** "transformée de Fourier rapide" convertit un signal en fréquences et vice-versa. Elle est utilisée dans le traitement numérique du signal audio et le traitement du signal d'image, et est une indication de la vitesse à laquelle un processeur peut traiter la vidéo dans un logiciel.
- **FPU Raytracing** est une technique de rendu permettant de générer une image en traçant le chemin de la lumière sous forme de pixels dans un plan d'image et en simulant les effets de ses rencontres avec des objets virtuels. Il teste la capacité du processeur à traiter les nombres à virgule flottante.

Les valeurs retenues pour l'étude des attributs sont les résultats générés par chacun des *benchmarks* ci-dessus. Plus un score est élevé, plus il témoigne de bonnes performances.

Blowfish Score	CryptoHash Score	Fibonacci Score	Zlib Score
N-Queens Score	FFT Score	Raytracing Score	

6.6.6 sysbench

Sysbench permet l'évaluation des performances CPU, de la mémoire et des opérations en lecture et écriture sur le stockage. L'évaluation est segmentée en trois itérations : un première où Sysbench est utilisé pour évaluer les performances CPU au travers d'opérations sur les nombres premiers ; une deuxième où Sysbench évalue les capacités de la mémoire en allouant un *buffer* et en lisant ou en écrivant à partir de celui-ci, chaque fois pour la taille d'un pointeur (32bit ou 64bit), et à chaque exécution jusqu'à ce que la taille totale du *buffer* ait été lue ou écrite ; une troisième où Sysbench évalue les I/O en réalisant des opérations en lectures et écritures séquentielles, en lectures et écritures aléatoires, ou une combinaison des deux.[37]

Quant aux paramètres utilisés lors des évaluations, ils sont les suivants : pour la partie CPU, le nombre premier maximal "cpu-max-prime" à atteindre est de 20000, le nombre de *threads* est défini à 4, et ce, sur une durée de 10 secondes ; pour la partie mémoire, le nombre de *threads* est défini à 4 et l'évaluation est réalisée sur une durée de 10 secondes ; pour la partie I/O, les opérations sont effectuées en lecture et écriture aléatoire "rndrw" sur un fichier de 6GB et le nombre de *threads* est défini à 4.

Les valeurs retenues pour l'étude des attributs sont : le nombre total d'évènements pour la partie CPU ; le nombre d'opérations effectuées, la vitesse moyenne de transfert en MB/s pour la partie mémoire ; le nombre d'opérations effectuées, la vitesse de transfert moyenne en MB/s et le nombre total d'évènements pour la partie I/O. Cela représente un total de 6 valeurs par noeud.

Total events CPU	Total operations Memory	Mean transfer speed Memory
Total operations IO	Mean transfer speed IO	Total events IO

6.7 Système de score

Au cours des évaluations, deux scores sont attribués : premièrement, un score global moyen ; deuxièmement, un score moyen par métrique.

Le score global moyen est obtenu en effectuant la somme de l'ensemble des valeurs obtenues remises sous forme de pourcentage. Ainsi, pour le profil par défaut des RPI, chaque valeur est égale à 100 et le score global moyen est égal à la somme des 31 attributs, soit 3100.

Le score moyen par métrique, lui, est calculé en ne gardant que les attributs qui sont liés à cette métrique. Comme le score global moyen, leurs valeurs sont remises sous forme de pourcentages.

Les scores moyens par métrique du profil par défaut sont les suivants :

- **CPU** : équivaut 1400. Soit la somme des attributs Compression R/U Rate Single-thread, Compression R/U Rate Multi-thread, Compression R/U Rate Hyper-Threading, Decompression R/U Rate Single-thread, Decompression R/U Rate Multi-thread, Decompression R/U Rate Hyper-Threading, Blowfish Score, CryptoHash Score, Fibonacci Score, N-Queens Score, Zlib Score, FFT Score, Raytracing Score et Total events CPU.

- **Mémoire** : équivaut 200. Soit la somme des attributs Total operations Memory, Mean transfer speed Memory.
- **Stockage** : équivaut 1000. Soit la somme des attributs Local write speed 256, Local write speed 512, Local write speed 1024, Cached reads speed, Buffered disk reads speed, O_DIRECT cached reads speed, O_DIRECT disk reads, Total operations IO, Mean transfer speed IO et Total events IO.
- **Réseau** : équivaut 500. Soit la somme des attributs bandwith 8, bandwith 64, bandwith 256, bandwith 512, bandwith 1024.

Quant aux scores des autres profils, ils sont calculés en suivant la même opération. Ils sont ensuite utilisés comme comparatif afin de déterminer quel est le gain en performances global et par métrique des différents profils.

6.8 Profils de performances

Les profils de performances sont créés afin d'étudier les gains et les pertes de performances du cluster lors de l'exécution d'HPL, détaillée dans le chapitre *Phase 3 : évaluation en cluster* de la section *implémentation*. C'est pourquoi la métrique CPU et le taux d'opérations à la seconde effectué par les RPI sont prédominants sur les autres métriques.

Parmi les profils de base, se trouvent : premièrement, un profil standard qui représente les RPI dans leur configuration d'origine et sans aucune altération, excepté la mise à jour complète du système ; deuxièmement, un profil stable où les performances CPU des RPI sont boostées, sans pour autant atteindre les performances maximales, qui peuvent, à long terme, endommager le matériel ; troisièmement, un profil haute performance où les performances CPU des RPI sont poussées à leur limite afin de déceler les performances maximales pouvant être fournies par les RPI.

À ces trois profils de base se joignent d'autres profils, créés sur base des résultats présentés dans les chapitres *Phase 1 : évaluation individuelle* et *Phase 2 : optimisation* de la section *Résultats expérimentaux*, où des optimisations sont réalisées afin de maximiser les performances des noeuds et de concentrer ces dernières autour de la métrique CPU des RPI.

Finalement, ces profils de performances sont comparés et critiqués au travers des résultats fournis par l'exécution d'HPL dans le chapitre *Phase 3 : évaluation en cluster* de la section *Résultats expérimentaux*, et ce, afin d'évaluer la pertinence de ces derniers dans le cadre d'une architecture clusterisée.

7 Implémentation

7.1 Implémentation du cluster

Le chapitre *Implémentation du cluster* décrit les étapes préliminaires à l'évaluation des performances du cluster : premièrement, le montage des RPI ; deuxièmement, la configuration réseau pour connecter les composants du cluster, à savoir : le NAS Synology RS815+, le switch Cisco Catalyst 3560 Series et les 9 RPI issues des kits Xute et Labists.

7.1.1 Montage des RPI

Le montage des RPI est réalisé de la manière suivante :

- les 3 dissipateurs sont placés à leurs emplacements respectifs (l'emplacement exact est défini dans le guide d'installation fourni dans le kit) ;
- le Soc est placé et vissé dans la partie inférieure du boîtier ;
- le ventilateur est placé et vissé sur la partie supérieure du boîtier ;
- le ventilateur est branché sur les pins 5V du RPI.

La présence des dissipateurs est motivée par la prise en puissance (notion électrique), et donc par la température accrue, du RPI 4. Cette hausse de température est mesurée à hauteur de plus de 50% par rapport au précédent modèle de RPI.[38] La [Fig.14] présente la différence de température entre plusieurs modèles de RPI :

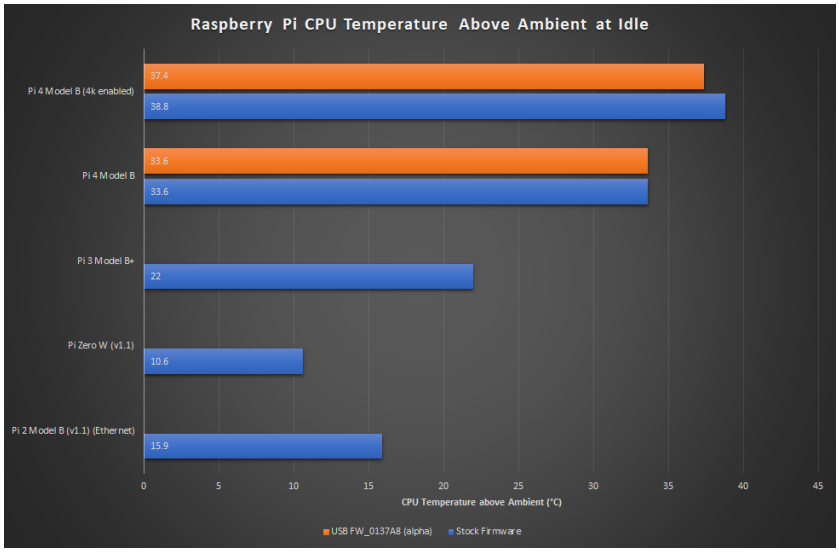


Figure 14: [38] Raspberry Pi CPU Temperature Above Ambient at Idle

Pour la même raison, le ventilateur est branché sur le profil 5V du RPI. En effet, il est possible d'utiliser le profil 3V du RPI afin de réduire la puissance du ventilateur. Bien que cela permette de réduire le bruit généré par le ventilateur, cela résulte aussi en une baisse de son efficacité et donc une baisse du refroidissement du boîtier.

7.1.2 Configuration réseau

La configuration réseau du cluster est mise en place à travers les étapes suivantes.

D'abord, le switch Cisco Catalyst 3560 est alimenté et branché en utilisant sa configuration par défaut. Le switch est utilisé comme un *hub* qui reçoit ses informations réseaux par le vlan dans lequel il se trouve.

Ensuite, le vlan dans lequel le cluster est monté appartient au réseau 192.113.50.0/24 et a comme valeur de DNS 8.8.8.8. De plus, il dispose d'un service DHCP externe au cluster qui permet de rendre le cluster plus évolutif et d'améliorer la rapidité de sa configuration.

Finalement, après leur branchement au switch avec les câbles Ethernet Cat 6, les noeuds et le NAS Synology RS815+ disposent d'une IP avec laquelle il peuvent communiquer dans ce réseau fermé.

7.2 Phase 1 : évaluation individuelle

Le chapitre *Phase 1 : évaluation individuelle* décrit la procédure d'installation nécessaire au fonctionnement d'Ansible et à la configuration des RPI afin d'analyser individuellement leurs performances.

7.2.1 Installation des RPI

La configuration du master, qui sert de référent pour les autres noeuds, se divise en 5 étapes :

1. L'installation du système d'exploitation se fait via le logiciel Raspberry Pi Imager [Fig.15]. Sur celui-ci, le premier bouton permet de sélectionner le système d'exploitation désiré. Le choix de ce dernier est motivé par la procédure d'installation automatisée d'HPL issue du répertoire GIT nécessaire au point *Installation d'HPL* du chapitre *Phase 3 : Évaluation du cluster*. L'image Ubuntu Server 20.04.4 LST 64-bit pour arm64 se trouve sous la section Ubuntu dans Other general-purpose OS ;

2. Le second bouton permet de sélectionner le stockage, la carte SD d'origine, sous le nom Generic Mass-Storage USB Device de 32GB. Quant aux RPI qui disposent d'une carte SD de 64GB, elles n'utiliseront que la moitié du stockage (32GB) ;

3. Depuis ses dernières versions, le logiciel Raspberry Pi Imager permet de directement opérer plusieurs configurations comme : définir le nom d'hôte, activer SSH, définir le nom d'utilisateur et le mot de passe de la RPI. Auparavant, ces configurations devaient être réalisées après l'écriture de la carte SD au travers de plusieurs fichiers. Pour une configuration rapide des noeuds, SSH est activé, le nom d'utilisateur est défini sur *pi* et le mot de passe est défini sur *root* ;

4. Le troisième bouton permet l'écriture de la carte et l'installation du système d'exploitation ;

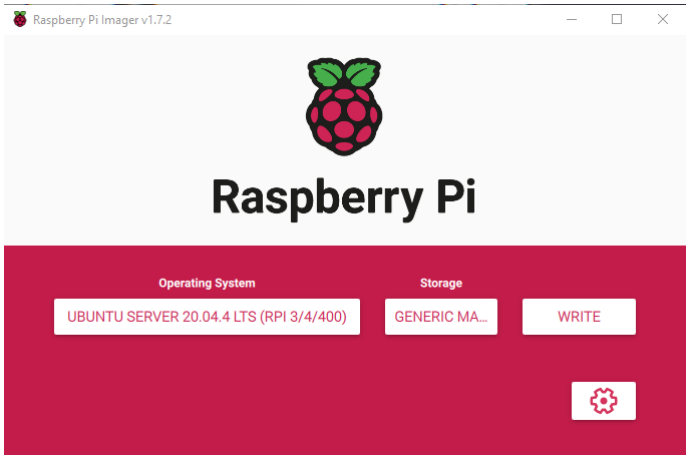


Figure 15: Installation du système d'exploitation : Raspberry Pi Imager

5. Après avoir inséré la carte SD et démarré le RPI, il est possible de s'y connecter via SSH. Pour cela, plusieurs possibilités existent. La plus simple, celle choisie, est de passer par le terminal d'un système disposant du service SSH. Une fois connecté avec les identifiants définis précédemment, la dernière consiste à mettre à jour le système. Pour cela, il suffit de rentrer les commandes :

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

7.2.2 Réplication des cartes SD

L'étape de réplication des cartes SD sert à considérablement réduire le temps d'installation des RPI. Elle est prise en charge par le logiciel Balena Etcher [Fig.16] et elle est répétée pour chaque noeud du cluster.

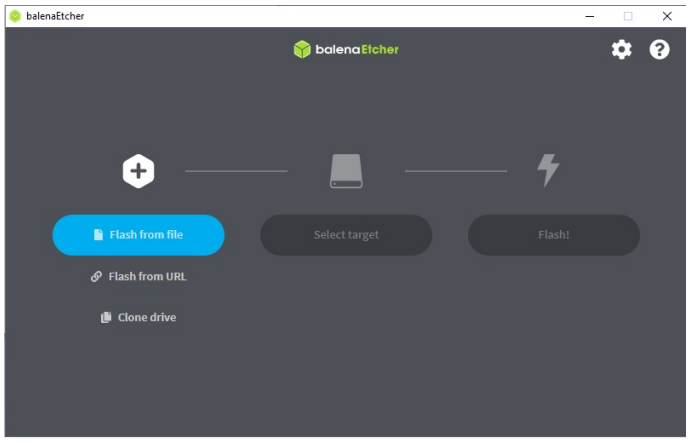


Figure 16: Réplication des cartes SD : Balena Etcher

Sur Balena Etcher : le premier bouton permet de sélectionner la carte SD précédemment écrite, utilisée comme carte SD de référence ; le second bouton permet de sélectionner la nouvelle carte SD à écrire ; le dernier bouton permet de flasher la carte et de cloner la première dans la deuxième.

7.2.3 Master : configuration d'Ansible

Sur le noeud master, l'installation d'Ansible est effectuée par le biais des commandes suivantes :

```
$ sudo apt update
$ sudo apt install software-properties-common -y
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible -y
```

Une fois Ansible installé, il est nécessaire de spécifier, au travers du fichier */etc/ansible/hosts*, les adresses IP des noeuds du cluster. Le fichier est divisé en plusieurs sections déterminées par les balises : [master], [cluster] et [all:children].

Par défaut, les communications établies par Ansible utilisent le système de clés SSH afin d'identifier les noeuds. Cependant, pour faciliter le processus d'installation et de configuration des noeuds, l'authentification est effectuée par l'utilisation des noms d'hôte et des mots de passe.

D'abord, la vérification par clés SSH est désactivée dans le fichier */etc/ansible/ansible.cfg* grâce aux lignes suivantes :

```
[defaults]
host_key_checking = False
```

Ensuite, pour valider les connexions sans clés SSH, les variables suivantes sont passées à l'ensemble des noeuds par la balise [all:vars] dans le fichier */etc/ansible/hosts*.

- **ansible_connection=ssh** : permet de définir le moyen de communication utilisé par Ansible.
- **ansible_user=pi** : permet de définir le nom d'hôte utilisé lors de la connexion ssh.
- **ansible_ssh_pass=root** : permet de définir le mot de passe utilisé lors de la connexion ssh.

Finalement, la configuration d'Ansible est vérifiée grâce à la commande suivante :

```
$ ansible cluster -m ping
192.168.0.34 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

7.2.4 Noeuds : installation des outils

L'installation des outils de *benchmarking* sur les noeuds et leur préparation au cluster sont prises en charge par Ansible au travers de plusieurs *playbooks*, fichiers yaml (.yml) utilisés par Ansible.

Le *playbook install.yml* sert à installer toute la suite de outils utilisés au point *Exécution des tests individuels*. Ces outils sont : iperf, sysbench, hardinfo, p7zip-full, hdparm, dd et uuid-runtime.

Parmi ces derniers, seul uuid-runtime ne sert pas directement à évaluer les performances des noeuds. Il est utilisé afin de générer des identifiants pour le nom des dossiers où sont enregistrés les résultats des évaluations.

Quant aux autres outils installés, ils se concentrent sur la récupération d'informations sur les noeuds telles que la charge CPU, le taux de compression et décompression, la vitesse d'écriture et lecture sur la carte SD, etc.

7.2.5 Configuration du stockage partagé

Le stockage partagé du cluster est implémenté à travers le NAS Synology RS815+ via une interface web accessible sur le port 5000 à l'IP du NAS.

Afin d'installer correctement le NAS, et ce, sans y rajouter des configurations non nécessaires, ce dernier est réinitialisé grâce au bouton RESET se trouvant sous le port Ethernet LAN 1. Une fois le NAS redémarré, il est uniquement nécessaire d'introduire le nom d'utilisateur et le nouveau mot de passe.

D'abord, comme annoncé dans le chapitre *Matériel*, le NAS dispose de 4 SSD Crucial MX500 dont la capacité utilisable est de 465.8 Go [Fig.17]. Au travers de l'interface de configuration de Synology, ces 4 disques sont configurés comme un seul et même volume. Ce volume est installé avec un RAID 6 afin d'ajouter une tolérance aux pannes et aux pertes de données.

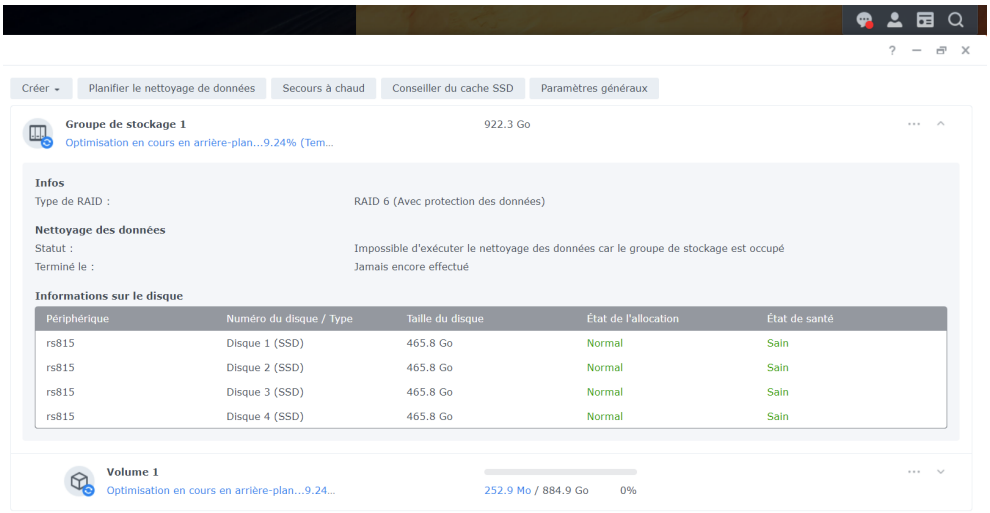


Figure 17: Configuration du volume et RAID6 sur le NAS Synology RS815+

Ensuite, le volume est rendu accessible pour les autres composants du cluster. À cette fin, NFS est configuré [Fig.18] sur ledit volume par les manipulations suivantes :

- premièrement, activer l'utilisation d'NFS version 4 dans les paramètres de protocoles utilisables par le Synology ;
- deuxièmement, donner les autorisations en lecture et écriture aux utilisateurs *Guest* du NAS ;
- troisièmement, accorder les accès NFS aux noeuds au travers des permissions NFS et lier les utilisateurs du cluster. Pour cela, une règle est créée par noeud, sur base de leur IP.

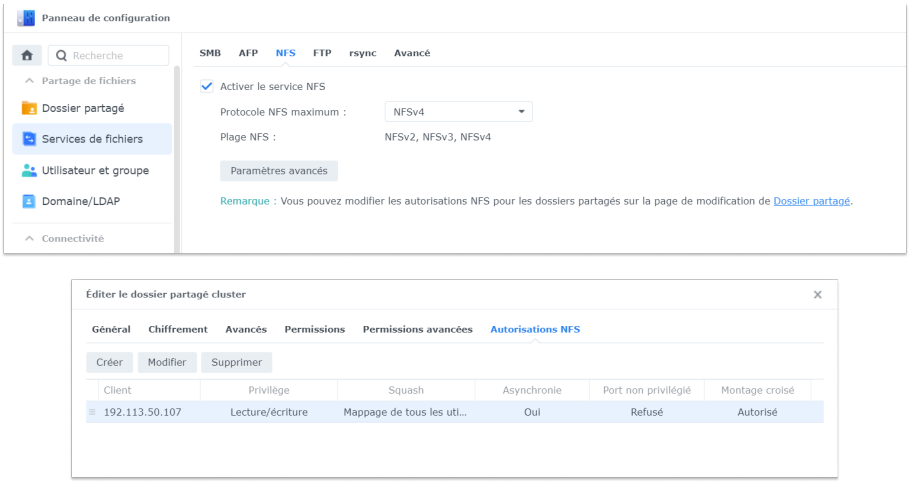


Figure 18: Configuration de NFS et du stockage partagé sur le NAS Synology RS815+

Après cela, la liaison des noeuds au répertoire partagé du Synology est pris en charge par le *playbook nfs.yml*. Il permet, à chaque ajout de noeuds dans le cluster ou lors du redémarrage de ce dernier, de réinitialiser l'accès au stockage partagé. Cela permet de garder le cluster sans état et de rendre le cluster moins dépendant de la configuration du stockage partagé.

Outre cette vérification, le *playbook nfs.yml* permet d'automatiser l'installation de service nfs-common ainsi que l'exécution de la commande suivante, pour monter le répertoire sur l'ensemble du cluster.

```
$ mount -t nfs 192.113.50.105:/volume1/cluster/ /home/pi/cluster_shared
```

Ainsi, tout fichier écrit dans le dossier partagé est accessible en lecture et en écriture par les noeuds du cluster, et ce, directement sur le NAS Synology RS815+.

7.2.6 Master : installation de Jupyter Notebook

Un dernier *playbook, jupyter.yml*, sert à installer le service Jupyter Notebook sur le master. Celui-ci a pour objectif : le traitement et la visualisation des résultats générés par les noeuds au cours des évaluations.

Après avoir exécuté le *playbook*, la commande suivante permet de démarrer le service Jupyter et de le rendre accessible sans authentification afin de rendre l'accès plus aisé :

```
$ cd /home/pi/notebook && jupyter notebook --ip 0.0.0.0
    --NotebookApp.token='' --NotebookApp.password=''
```

L'interface web de Jupyter est ensuite accessible sur l'IP du master au port 8888.

7.2.7 Exécution des tests individuels

Une fois les composants et les noeuds connectés entre eux, le *playbook bench.yml* sert à exécuter les tests d'évaluation sur le cluster. Ce dernier exécute les outils iperf, sysbench, hardinfo, p7zip-full, hdparm et dd installés à l'étape *Noeuds : installation des outils*.

Chaque commande du *playbook* est exécutée avec un temps d'intervalle de 1 minute pour laisser les RPI retrouver leur état initial et ne pas biaiser les performances de ces derniers.

Ensuite, les résultats générés par le *playbook* sont enregistrés au format .txt : d'une part, dans un dossier propre à l'évaluation sous le nom *benchmark-uuid* ; d'autre part, dans des sous-dossiers propres à chaque noeud, nommés en fonction de l'IP des RPI. Ces derniers sont accessibles sur le stockage partagé *Cluster* présent sur le NAS.

Finalement, au travers de *codes sous Jupyter notebook*, les résultats sont traités afin de ne récupérer que les informations nécessaires à l'analyse des attributs étudiés.

7.3 Phase 2 : optimisation

Le chapitre *Phase 2 : optimisation* décrit les configurations apportées aux noeuds afin de les optimiser. Ces configurations sont segmentées en plusieurs métriques : premièrement, les optimisations CPU ; deuxièmement, les optimisations GPU ; troisièmement, les optimisations des I/O de RPI ; quatrièmement, les optimisations du système d'exploitation et du système de fichiers; cinquièmement, les optimisations de la mémoire RAM.

La majorité de ces optimisations sont directement appliquées dans le fichier */boot/-firmware/config.txt* sur chacun des RPI grâce au *playbook oc.yml*, où les valeurs des paramètres sont présentées dans la documentation dudit fichier sur le site web Raspberrypi.[35]

Parce que certaines de ces configurations peuvent endommager les RPI, un système de protection est mis en place directement par le système d'exploitation. Cette protection est présentée dans la documentation de la manière suivante :

L'*overclocking* et la surtension seront désactivés au moment de l'exécution lorsque le SoC atteint temp_limit, qui est par défaut de 85°C, afin de refroidir le SoC. Vous ne devriez pas atteindre cette limite avec les Raspberry Pi 1 et Raspberry Pi 2, mais vous êtes plus susceptible de le faire avec les Raspberry Pi 3 et Raspberry Pi 4. L'*overclocking* et la surtension sont également désactivés lorsqu'une situation de sous-tension est détectée.[35]

Dans le cas d'une désactivation de l'*overclocking*, les RPI deviennent alors incapables de *booter* et le fichier *config.txt* doit être modifié manuellement ou momentanément réinitialisé par une combinaison de touches avant le *boot*.[39]

7.3.1 Optimisations CPU

L'optimisation du CPU des RPI est effectuée au travers des paramètres *arm_freq* et *arm_freq_min*, *arm_boost* et *over_voltage*. Ces optimisations sont réalisées dans le fichier */boot/firmware/config.txt*

Les premiers, *arm_freq* et *arm_freq_min*, permettent de définir la fréquence adoptée par le CPU. Cette fréquence peut varier entre 600MHz et 2100MHz. Cependant, les fréquences les plus souvent rencontrées lors des évaluations de performances sont les suivantes : 600MHz (fréquence minimale), 1500MHz (fréquence par défaut), 1800MHz (fréquence overclockée recommandée) et 2100MHz (fréquence overclockée maximale conseillée).

Le second, *arm_boost*, permet de définir la fréquence adoptée par le CPU à sa valeur overclockée recommandée (1800MHz) sans devoir passer par le paramètre *arm_freq*. Le paramètre *arm_freq* reste cependant prioritaire s'il est configuré.

Le dernier, *over_voltage*, permet de modifier le voltage attribué au CPU et au GPU de la RPI. Sa valeur varie entre -16 et 8. La valeur par défaut est 0 et la valeur maximale conseillée est 6. Si celle-ci est outrepassée, le RPI peut devenir instable et la garantie est alors désactivée.[35]

La figure [Fig.19] présente la relation entre ces paramètres et leurs répercutions sur la tension, la température, la puissance et l'augmentation approximative des performances du CPU :

Clock (MHz)	Overvoltage	Vcore	Max temp. (°C | °F)	Power (Watt)	Preformance increase	Remarks
0	0	0.8625		1.5		RPi 4 shut down
200	0	0.8625		1.75		RPi 4 min working clock
600	0	0.8625		2.8		RPi 4 running idle
1500	0	0.8625	82 | 180	7		Factory settings
1600	1	0.8875	80 | 176	7.6	6.6 %	
1700	2	0.9125	78 | 172	8.3	13.3 %	
1800	3	0.9375	77 | 170	8.9	20 %	
1900	4	0.9625	75 | 167	9.5	26.6 %	
2000	6	1.0125	72 | 162	11	33.3 %	
2100	6	1.0125	72 | 162	11	40 %	
	7	1.0375	56 | 132	11.7		no improvement
	8	1.0625	50 | 122	12.3		no improvement

Figure 19: [40] Tableau des valeurs d'overclocking CPU de la RPI 4

Les profils CPU évalués sont les suivants :

arm_freq 1500MHz	arm_freq 1800MHz	arm_freq 2100MHz
over_voltage 0	over_voltage 3	over_voltage 6

7.3.2 Optimisations GPU

L'optimisation du GPU des RPI est effectuée au travers des paramètres *gpu_freq* et *over_voltage* ainsi que les paramètres *core_freq* , *h264_freq*, *isp_freq*, *v3d_freq* et *hevc_freq*, tous commandés par le paramètre *gpu_freq*. Ces optimisations sont réalisées dans le fichier */boot/firmware/config.txt*

Le premier, *gpu_freq*, permet de définir la fréquence adoptée par le GPU incorporé au CPU. Cette fréquence peut varier entre 200MHz et 750MHz. Cependant, les fréquences les plus souvent rencontrées lors des évaluations de performances sont les suivantes : 500MHz (fréquence par défaut) et 750MHz (fréquence overclockée maximale conseillée).

Le second, *over_voltage*, comme précisé dans le point précédent, permet de modifier le voltage attribué au CPU et au GPU des RPI.

Quant aux autres paramètres listés ci-dessus, ils permettent respectivement de commander la fréquence du processeur GPU en MHz, la fréquence du bloc vidéo matériel en MHz, la fréquence du bloc pipeline du capteur d'images en MHz, la fréquence du bloc 3D en MHz et la fréquence du bloc High Efficiency Video Codec en MHz.

Cependant, comme il est spécifié dans la documentation de la RPI, les paramètres commandant la fréquence GPU influencent les performances du CPU, car ils gèrent le cache L2 et le bus mémoire. C'est pourquoi, au travers des évaluations, différentes valeurs sont testées afin de mesurer cette influence sur les performances du CPU.

Les profils GPU évalués sont les suivants :

gpu_freq 200MHz	gpu_freq 500MHz	gpu_freq 750MHz

7.3.3 Optimisations des I/O

L'optimisation des I/O des RPI est effectuée au travers des étapes suivantes : premièrement, la désactivation du Bluetooth et du WiFi ; deuxièmement, la désactivation des LEDs des RPI. Ces optimisations sont réalisées dans le fichier */boot/firmware/config.txt*

Comme présenté dans le chapitre *Configuration réseau*, les noeuds sont tous connectés au cluster au travers du switch Cisco Catalyst 3560 par des câbles Ethernet. Les services Bluetooth et WiFi ne sont donc pas nécessaires au bon fonctionnement du cluster et ne nécessitent pas d'être fonctionnels.

La désactivation du Bluetooth et du WiFi sont effectuées par l'ajout des lignes suivantes au fichier */boot/firmware/config.txt* :

```
dtoverlay=disable-WiFi
dtoverlay=disable-bt
```

Quant à la désactivation des LEDs, elle est effectuée par l'ajout des lignes suivantes dans le même fichier :

```
# Disable the PWR LED
dtparam=pwr_led_trigger=none
dtparam=pwr_led_activelow=off
# Disable the Activity LED
dtparam=act_led_trigger=none
dtparam=act_led_activelow=off
# Disable ethernet port LEDs
dtparam=eth_led0=4
dtparam=eth_led1=4
```

Ces étapes permettent de réduire la consommation des RPI d'approximativement 50mA et de concentrer l'essentiel de la puissance des RPI sur leur CPU.[41]

7.3.4 Optimisations du système d'exploitation

L'optimisation du système d'exploitation des RPI est effectuée au travers des étapes suivantes : premièrement, la mise à jour complète du système d'exploitation, de son noyau et de son *firmware* ; deuxièmement, la mise à jour d'EEPROM ; troisièmement, la désinstallation de snapd, un logiciel de gestion de paquets similaire à apt ; quatrièmement, le nettoyage du cache et des fichiers obsolètes.

La mise à jour complète du système permet de s'assurer que les dernières versions stables de chacun des logiciels et l'ensemble de leurs dépendances sont installées sur les RPI. Cela a pour effet, la majeure partie du temps, d'améliorer les performances de ces dernières.

La mise à jour du système est réalisée grâce aux commandes suivantes :

```
$ sudo apt update -y
$ sudo apt full-upgrade -y
$ sudo reboot
```

EEPROM signifie "electrically erasable programmable read-only memory" (mémoire morte programmable effaçable électriquement). Il s'agit d'un type de mémoire non volatile utilisé dans les ordinateurs, intégré dans les micro-contrôleurs ... pour stocker des quantités relativement faibles de données en permettant d'effacer et de reprogrammer des octets individuels.[42]

Le Raspberry Pi 4 est partiellement démarré à partir de deux EEPROMs. Ces EEPROMs sont programmées après l'assemblage du PCB en usine. La mise à jour du micro-logiciel dans les deux EEPROMs, avec les améliorations apportées récemment par l'équipe Raspberry, n'augmentera pas immédiatement la vitesse d'horloge, mais diminuera la chaleur en raison de la réduction de la consommation d'énergie.[43]

La mise à jour du firmware EEPROM le plus récent est réalisée grâce aux commandes suivantes :

```
$ sudo apt-get install rpi-eeprom -y
$ sudo rpi-eeprom-update -a
$ sudo reboot
```

Le gestionnaire de paquet Snap permet aux développeurs de distribuer plus facilement des logiciels aux distributions Linux. Cependant : 1) dans l'implémentation du présent cluster, seul apt est utilisé pour installer les programmes, Snap n'est donc d'aucune utilité ; 2) ce dernier est connu pour impacter les performances des RPI au *boot* ; 3) le retirer permet de libérer de l'espace de stockage.[44]

La désinstallation de Snap et la suspension de son service sont effectuées grâce aux commandes suivantes :

```
$ sudo apt-get autoremove --purge -y snapd gnome-software-plugin-snap
$ sudo apt-mark hold snapd
```

Finalement, après avoir réalisé ces étapes, le système d'exploitation est nettoyé afin de retirer tout fichier et toute dépendance non utilisé ou devenu inutile sur le système grâce aux commandes suivantes :

```
$ sudo apt autoremove
$ sudo apt clean
```

Ces optimisations sont réalisées au travers du *playbook system.yml*.

7.3.5 Optimisations de la mémoire RAM

L'optimisation de la mémoire RAM de la RPI est évaluée : premièrement, par la désactivation des fichiers swap ; deuxièmement par l'activation de la compression ZRAM.

Les fichiers swap sont un type de mémoire virtuelle, puisqu'ils ne sont pas stockés dans la RAM physique. Ils étendent la quantité de mémoire disponible à laquelle un ordinateur peut accéder en transférant la mémoire utilisée par les processus inactifs de la RAM vers un fichier d'échange.[45]

La désactivation des fichiers swap est réalisée par le script suivant [44] et s'effectue en trois étapes : 1) désactivation des fichiers swap ; 2) suppression des entrées de type *swapfile* dans le fichier */etc/fstab* ; 3) suppression des fichiers swap actifs.

```
$ wget -O - https://teejeetech.com/scripts/jammy/disable_swapfile | bash
```

La compression zRAM permet de remédier l'utilisation des fichiers swap en compressant la RAM la moins utilisée pour faire de la place. Les programmes les moins sollicités dans la RAM sont compressés afin d'offrir davantage de ressources aux processus actifs. La RAM, une fois compressée, est certes un tout petit peu plus lente, mais ceci évite de monopoliser du temps pour les échanges vers le disque, par essence moins rapides qu'une compression de fichiers presque instantanée. L'impact sur le processeur restant très négligeable.[46]

L'activation de la compression ZRAM est réalisée grâce aux commandes suivantes :

```
$ sudo apt install -y zram-config
$ sudo reboot
```

Ces optimisations sont réalisées au travers du *playbook zram.yml*.

7.3.6 Informations sur les optimisations

L'ensemble des optimisations présentées dans les points précédents ne représente pas forcément en gain en performances pour les RPI. Au contraire, certaines peuvent même affecter négativement le système ou rendre les RPI instables.

Chacune de ces optimisations est donc testée afin de mettre en évidence son utilité au sein du cluster. Cela permet aussi de définir, au travers d'un pourcentage, le gain **(+105%)** et/ou la perte **(-95%)** en performances associés à chacune de ces configurations.

À la suite de cette *Phase 2 : optimisation*, l'objectif est donc d'analyser les résultats obtenus afin de créer les profils de performances qui sont utilisés lors de la *Phase 3 : évaluation en cluster* au travers de l'exécution d'HPL.

7.4 Phase 3 : évaluation en cluster

Le chapitre *Phase 3 : évaluation en cluster* décrit la procédure d'installation nécessaire au fonctionnement d'HPL et à l'évaluation des performances du cluster : d'abord, en *single node* sur le master ; ensuite, sur l'ensemble des noeuds du cluster.

7.4.1 Installation d'HPL

La compilation complète d'HPL et de ses dépendances ainsi que leur installation est une étape risquée. Comme avancé dans [20], la documentation autour de cette procédure est relativement lacunaire et dépend de beaucoup de paramètres. Alors, pour faciliter l'installation d'HPL sur le cluster, une procédure automatisée est utilisée. Cette procédure de arif-ali [47] date de juin 2020 et implique, comme seule contrainte, l'utilisation du système d'exploitation Ubuntu dans sa version 20.04.4. De plus, elle est destinée à l'utilisation d'HPL sur les RPI model 4 B, ce qui convient parfaitement à ce travail.

La procédure d'installation se divise en plusieurs parties au travers de scripts :

- La première partie, dans le fichier CONFIG, permet de définir les *flags* utilisés afin de compiler HPL et ses dépendances. Ces *flags* sont des paramètres [48] passés au compilateur afin de définir le comportement à adopter lors de la création de l'exécutable, HPL dans le cas présent. Par exemple, le *flag -mtune=cortex-a72* permet de définir que l'architecture visée par le compilateur est une Cortex A72 ARM.
- La deuxième partie, dans le fichier *install_deps.sh*, permet d'installer le compilateur Fortran nécessaire à la compilation d'HPL.
- La troisième partie, dans le fichier *make_mpich.sh*, permet de compiler la librairie MPICH. MPICH est nécessaire à l'exécution d'HPL et, plus précisément, aux communications internoeuds lors des évaluations.
- La quatrième partie, dans le fichier *make_openblas.sh*, permet de compiler la librairie OpenBLAS "Basic Linear Algebra Subprograms". OpenBLAS est nécessaire à l'exécution d'HPL et, plus précisément, à l'exécution des calculs effectués lors des évaluations.
- La cinquième partie, dans le fichier *make_hpl.sh*, permet la compilation d'HPL.

Ces 5 étapes sont automatisées par l'exécution du script *compile_all.sh*. Après quoi, HPL est exécuté depuis le fichier *xhpl* dans le dossier */home/pi/rpi-hpl-workdir/hpl-2.3/bin/rpi4-mpich*.

7.4.2 Vérification en single node

Pour l'étape de vérification et d'exécution d'HPL, le site web advancedclustering [49] permet de rentrer les champs nombre de noeuds, nombre de coeurs par noeud, mémoire RAM par noeud et la taille des blocs afin d'obtenir des valeurs de départ pertinentes [Fig.20] pour l'exécution de HPL.

Ces valeurs peuvent ensuite être modifiées en fonction du cluster par essais - erreurs afin d'obtenir les valeurs qui fournissent les meilleures performances en GFLOPS, alors jugées comme optimales.

Output

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
20352         Ns
1            # of NBs
192           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
```

Figure 20: Valeurs de départ fournies par advancedclustering.

Les détails et la documentation de chacun des attributs modifiables dans le fichier *hpl.dat* se trouvent sur le site web netlib.[50]

Elles sont ensuite copiées dans le fichier *HPL.dat* dans le dossier *configs*. L'IP de chaque noeud est ensuite référencée dans le fichier *nodes-rpi*. HPL considère chaque coeur du RPI individuellement, c'est pourquoi chaque noeud doit disposer de 4 lignes.

Finalement, HPL est exécuté en *single node* grâce aux commandes suivantes :

```
$ cd rpi-hpl-workdir/hpl-2.3/bin/rpi4-mpich/
$ mpiexec -f nodes-rpi ./xhpl
```

Le résultat [Fig.21] présenté par HPL prend la forme suivante :

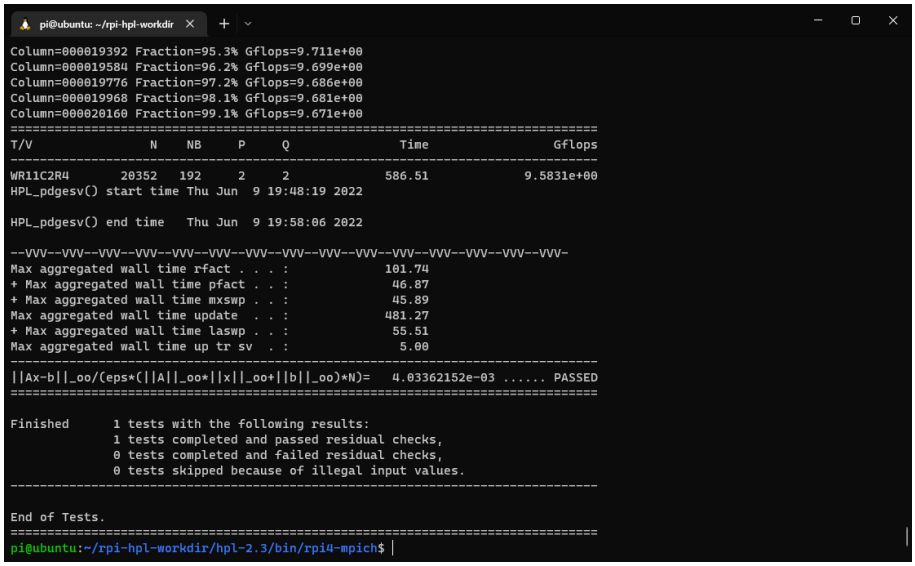


Figure 21: Résultat présenté par HPL en *single node*

7.4.3 Préliminaires à l'exécution en cluster

À ce stade, HPL est fonctionnel en *single node* sur le master. Le RPI dispose de l'ensemble des logiciels nécessaires à l'exécution en cluster. Ainsi, de la même manière que lors de l'évaluation individuelle, la carte du master va être répliquée. La procédure de réplication est identique à celle présente au point *Réplication des cartes SD*. Cela permet de gagner un temps considérable en évitant de recompiler HPL sur chacun des noeuds et de certifier que tous les noeuds sont parfaitement similaires.

7.4.4 Exécution en cluster

Pour l'exécution d'HPL en cluster, le programme nécessite l'utilisation des clés SSH pour permettre la communication avec l'ensemble des noeuds.

Sur chacun des noeuds, les commandes suivantes sont exécutées afin de générer et de s'échanger les clés SSH :

```
$ ssh-keygen
$ ssh-copy-id -i ~/.ssh/id_rsa MASTER_IP_ADDRESS
```

Pour qu'HPL fonctionne en cluster au travers de mpiexec, le PATH de la librairie MPICH doit être ajouté au PATH du master, grâce à la commande ci-dessous. Mpiexec est l'exécutable qui permet de lancer les scripts MPI.

```
$ export PATH=/opt/mpich/3.3.2/bin/:$PATH
```

Ensuite, les paramètres de configuration d'HPL issus du site web advancedclustering sont adaptés au cluster dans le fichier *HPL.dat*, sans oublier de rajouter les 4 IP par noeuds dans le fichier *nodes-pi*.

Le fichier *HPL.dat* nécessite d'être copié sur l'ensemble des noeuds. Pour ce faire, le *playbook* hpl.yml est utilisé.

Finalement, xhpl est exécuté avec la même commande que celle utilisée dans l'étape précédent :

```
$ cd rpi-hpl-workdir/hpl-2.3/bin/rpi4-mpich/
$ mpiexec -f nodes-rpi ./xhpl
```

Le résultat [Fig.22] présenté par HPL prend la forme suivante :

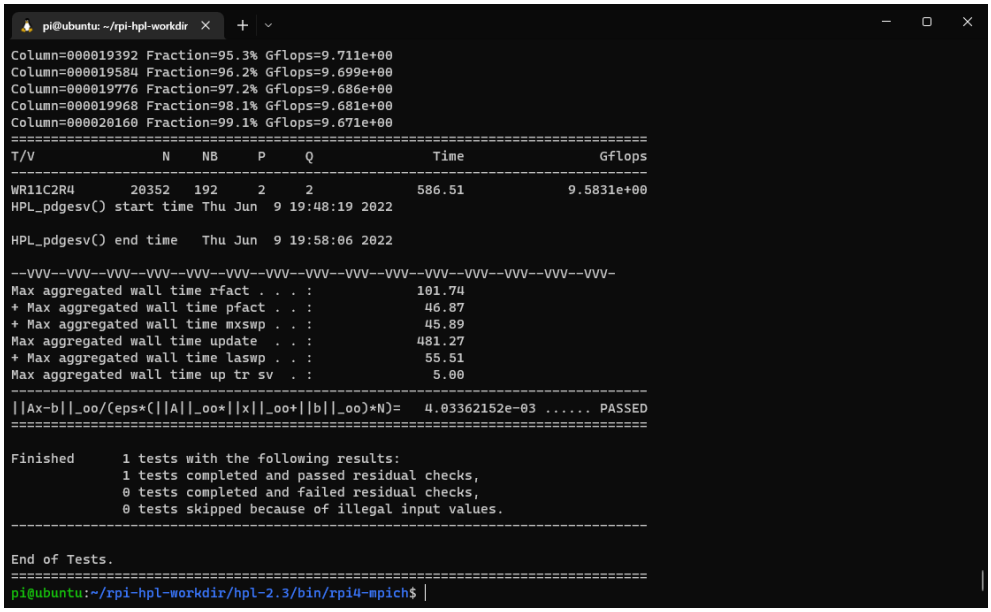


Figure 22: Résultat présenté par HPL en cluster

8 Résultats expérimentaux

La section *Résultats expérimentaux* sert d'analyse et de critique aux données récoltées aux cours des trois phases suivantes : *Phase 1 : évaluation individuelle*, *Phase 2 : optimisation*, *Phase 3 : évaluation en cluster*, et ce, en suivant la *méthodologie* exprimée dans la section *Évaluation*.

8.1 Phase 1 : évaluation individuelle

Les objectifs de la *Phase 1 : évaluation individuelle* sont : premièrement, récolter les valeurs sur les performances des noeuds sous plusieurs dimensions et plusieurs métriques ; deuxièmement, déceler les anomalies au sein des RPI au travers des valeurs récoltées.

L'évaluation est réalisée sur 8 Raspberry Pi 4 model B (plus 1 noeud master) dont 5 RPI sont issus du kit Xute et 3 RPI sont issus kit Labists, présentés dans le chapitre Matériel. De plus, parmi les noeuds, 5 RPI disposent d'une carte SD de 32GB et 3 RPI disposent d'une carte SD de 64GB. Tous les autres paramètres sont similaires à l'ensemble des noeuds et ont été présentés dans les chapitres *Implémentation du cluster* et *Phase 1 : évaluation individuelle*.

Les informations sur les noeuds du cluster sont représentées dans le tableau [Tab.2] :

Table 2: Informations sur les noeuds du cluster

Noeud	kit	Stockage
0	Xute	32GB
1	Xute	64GB
2	Xute	64GB
3	Xute	64GB
4	Xute	64GB
5	Labists	32GB
6	Labists	64GB
7	Labists	32GB

Grâce à cette disposition des noeuds, les quatre profils possibles sont étudiés : Xute 32GB, Xute 64GB, Labists 32GB et Labists 64GB.

Pour rappel, lors de la *Phase 1 : évaluation individuelle*, les 8 noeuds sont dans une configuration par défaut, dite *idle*, sans qu'aucune optimisation ne soit apportée. Seule la mise à jour complète du système d'exploitation a été effectuée avant d'exécuter l'évaluation.

8.1.1 Valeurs récoltées

Les valeurs récoltées à la suite de l'*Exécution des tests individuels* sont traitées et formatées au travers de *codes sous Jupyter notebook*. Ces valeurs sont synthétisées sous la forme d'un fichier Excel contenant l'entièreté des attributs observés sur chaque noeud, et ce, par *benchmark*.

Les valeurs suivantes représentent les valeurs par défaut des 8 noeuds [Tab.4], sans aucune optimisation, pour chacun des 31 attributs observés [Tab.3]. Le détail et l'origine des attributs se trouvent au chapitre *Attributs étudiés*.

Table 3: 31 attributs étudiés

MIPS	**0.** Compression R/U Rate single-thread	**1.** Decompression R/U Rate single-thread
	2. Compression R/U Rate multi-thread	**3.** Decompression R/U Rate multi-thread
	4. Compression R/U Rate hyper-threading	**5.** Decompression R/U Rate hyper-threading

MB/s	**6.** Local write speed 256	**7.** Local write speed 512
	8. Local write speed 1024	
	9. Cached reads speed	**10.** Buffered disk reads speed
	11. O_DIRECT cached reads speed	**12.** O_DIRECT disk reads

Mb/s	**13.** bandwith 8	**14.** bandwith 64
	15. bandwith 256	**16.** bandwith 512
	17. bandwith 1024	

Score	**18.** Blowfish Score	**19.** CryptoHash Score
	20. Fibonacci Score	**21.** Zlib Score
	22. N-Queens Score	**23.** FFT Score
	24. Raytracing Score	

25. Total events CPU	**26.** Total operations Memory (op/s)
27. Mean transfer speed Memory (MB/s)	
28. Total operations IO per second (op/s)	**29.** Mean transfer speed IO (MB/s)
30. Total events IO	

Table 4: Valeurs récoltées pour les 8 noeuds lors de la *Phase 1 : évaluation individuelle*

Noeud	0	1	2	3	4	5	6	7
0.	1879	1896	1901	1887	1892	1635	1626	1632
1.	2075	2093	2098	2075	2078	1804	1795	1801
2.	1713	1712	1717	1716	1715	1501	1490	1499
3.	3713	3709	3721	3716	3721	3269	3243	3265
4.	1497	1508	1502	1504	1499	1339	1328	1331
5.	5932	5977	5967	5971	5949	5305	5271	5280
6.	15.2	26.5	25.0	27.5	25.0	16.5	27.2	15.5
7.	15.7	26.6	28.1	27.0	27.0	16.4	27.9	18.9
8.	15.7	28.9	29.6	29.1	27.4	17.0	28.9	19.5
9.	857.25	853.16	845.67	862.27	840.74	839.73	810.98	837.82
10.	43.03	43.16	43.07	43.21	43.10	42.63	42.91	43.36
11.	37.54	37.23	37.00	39.03	36.49	38.80	36.13	38.26
12.	42.24	42.58	42.26	42.41	42.33	40.58	41.06	41.59
13.	2.39	2.26	2.16	2.29	2.46	3.78	3.81	3.78
14.	2.51	2.38	1.81	2.36	2.50	3.84	3.78	3.82
15.	2.44	2.38	1.97	3.01	2.97	4.02	3.77	3.69
16.	2.22	2.18	2.25	2.74	2.19	3.97	3.56	3.93
17.	2.20	2.27	2.12	2.42	2.56	3.98	3.81	3.56
18.	5.553889	5.546159	5.530751	5.548200	5.538038	6.654699	6.647025	6.651872
19.	324.476106	323.503795	323.909505	323.536671	324.435617	270.406129	270.472938	270.413629
20.	1.717110	1.719604	1.717050	1.717313	1.717826	2.061311	2.061242	2.063111
21.	8.164499	8.226475	8.337231	8.229580	8.264991	9.899169	9.902961	9.909504
22.	0.429085	0.427084	0.431654	0.421877	0.432087	0.361979	0.354999	0.360415
23.	4.686533	4.528495	4.577917	4.765554	4.934964	4.377966	4.944774	4.981734
24.	2.228595	2.154190	2.195322	2.210964	2.198376	2.591440	2.668257	2.628044
25.	28183	28188	28193	28173	28181	23487	23477	23485
26.	90385040	90430473	89830604	88287664	90140194	75205049	75531068	73311199
27.	8823.59	8827.95	8769.38	8618.68	8799.63	7341.20	7372.99	7156.28
28.	1054.93	1287.94	1330.67	1275.96	1324.36	1267.7	1292.1	1371.32
29.	3.61	4.41	4.55	4.365	4.535	4.765	4.425	4.69
30.	315997	385970	398750	382330	397304	399975	387136	411014

8.1.2 Analyse des données

L'analyse des données se découpe en trois parties : premièrement, les attributs affectés par les différents kits ; deuxièmement, les attributs affectés par la carte SD ; troisièmement, les attributs peu ou pas affectés entre les différents profils.

Premièrement : Les attributs affectés par le kit sont les 20 attributs suivants : Compression R/U Rate single-thread, Decompression R/U Rate single-thread, Compression R/U Rate multi-thread, Decompression R/U Rate multi-thread, Compression R/U Rate hyper-threading, Decompression R/U Rate hyper-threading, bandwith 8, bandwith 64, bandwith 256, bandwith 512, bandwith 1024, Blowfish Score, CryptoHash Score, Fibonacci Score, Zlib Score, N-Queens Score, Raytracing Score, Total events CPU, Total operations Memory et Mean transfer speed Memory.

Si on considère le kit Xute comme kit de référence, certains attributs sont affectés positivement, en présentant des performances accrues. Ces 9 attributs sont représentés dans les deux tableaux [Tab.5] :

Table 5: Attributs affectés positivement

Réseau	Attributs	Xute	Labists	Taux
	bandwith 8	2.312	3.79	163.92%
	bandwith 64	2.312	3.81	164.93%
	bandwith 256	2.554	3.826	149.83%
	bandwith 512	2.316	3.82	164.93%
	bandwith 1024	2.314	3.783	163.49%

CPU/GPU	Attributs	Xute	Labists	Taux
	Blowfish Score	5.54	6.65	119.98%
	Fibonacci Score	1.71	2.06	120.03%
	Zlib Score	8.24	9.90	120.12%
	Raytracing Score	2.19	2.62	119.64%

Ces attributs sont : d'une part, représentatifs d'une interface réseau (port Ethernet) plus performante ; d'autre part, représentatifs d'une meilleure optimisation du GPU pour les RPI du kit Labists.

Quant aux attributs affectés négativement, ce sont les 11 attributs représentés dans les deux tableaux [Tab.6] :

Table 6: Attributs affectés négativement

CPU	Attributs	Xute	Labists	Taux
	Compression R/U Rate single-thread	1891	1631	86.25%
	Decompression R/U Rate single-thread	2083.8	1800	86.38%
	Compression R/U Rate multi-thread	1714.6	1496.66	87.28%
	Decompression R/U Rate multi-thread	3716	3259	87.70%
	Compression R/U Rate hyper-threading	1502	1332.66	88.72%
	Decompression R/U Rate hyper-threading	5959.2	5285.33	88.69%
	CryptoHash Score	323.97	270.43	83.47%
	N-Queens Score	0.42	0.35	83.83%
	Total events CPU	28183.6	23483	83.32%

RAM	Attributs	Xute	Labists	Taux
	Total operations Memory	89814795	74682438	83.15%
	Mean transfer speed Memory	8767.84	7290.15	83.14%

Ces attributs sont : d'une part, représentatifs d'une meilleure optimisation CPU pour les attributs donnés ; d'autre part, représentatifs d'une meilleure optimisation de la RAM pour les RPI du kit Xute.

Dans l'ensemble, le kit Labists offre : **1)** un gain approximatif de **60%** sur les performances réseau ; **2)** un gain approximatif de **20%** sur les performances CPU/GPU pour les attributs affectés positivement [Tab.5].

Quant au kit Xute, il offre : **1)** un gain approximatif de **16%** sur les performances CPU pour les attributs affectés négativement [Tab.6] ; **2)** un gain approximatif de **17%** sur les performances de la mémoire vive (RAM).

Deuxièmement : Les attributs affectés par les différences de carte SD [Tab.7] sont les 3 attributs suivants : Local write speed 256, Local write speed 512, Local write speed 1024.

Table 7: Attributs affectés par la carte SD

Stockage	Attributs	32GB	64GB	Taux
	Local write speed 256	15.73	26.24	166.77%
	Local write speed 512	17	27.32	160.70%
	Local write speed 1024	17.4	28.78	165.40%

En comparaison aux RPI disposant d'une carte SD de 32GB, celles qui disposent d'une carte SD de 64GB voient leur performances en écriture augmenter d'approximativement **64%.**

Pour rappel, les RPI qui disposent d'une carte SD de 64GB n'ont que 32GB de disponible. En effet, tous les noeuds ont été installés par le processus de *Réplication des cartes SD*, ce qui a pour effet de brider le système à 32GB.

Troisièmement : Les attributs peu ou pas affectés [Tab.8] entre les différents profils de RPI sont les 8 attributs suivants : Cached reads speed, Buffered disk reads speed, O_DIRECT cached reads speed, O_DIRECT disk reads, FFT Score, Total operations IO per second, Mean transfer speed IO, Total events IO.

Table 8: Attributs peu ou pas affectés

Unité	Attributs	Min	Max	Moyenne
MB/s	Cached reads speed	810.98	862.27	843.45
MB/s	Buffered disk reads speed	42.63	43.36	43.05
MB/s	O_DIRECT cached reads speed	36.13	39.03	37.56
MB/s	O_DIRECT disk reads	40.58	42.58	41.88
	FFT Score	4.37	4.98	4.72
op/s	Total operations IO per second	1054.93	1371.32	1275.62
MB/s	Mean transfer speed IO	3.61	4.76	4.41
	Total events IO	315997	411014	384809.5

Ces attributs sont majoritairement liés au stockage des RPI et semblent rester relativement stables, malgré les différences de kit et/ou de carte SD. Le FFT Score, quant à lui, se rapporte à du traitement vidéo, comme exprimé au travers du point *hardinfo*. Il est donc lié au CPU des RPI. Cependant, malgré les divergences observées dans les tableaux [Tab.5] et [Tab.6], il reste relativement stable.

8.1.3 Comparaison des résultats

La comparaison des résultats vise à : premièrement, mettre en évidence les différences de performances entre les 4 profils de RPI cités dans le [Tab.2] ; deuxièmement, établir un score global moyen et un score moyen par métrique des performances par défaut, tous deux décrits dans le chapitre *Système de score*.

Premièrement : Les différences de performances entre les 4 profils sont calculées en utilisant le profil RPI Xute 32GB comme profil de référence. Chacun des attributs obtient une valeur équivalente à 100, pour un total de 3100. Quant aux autres profils, la valeur de leurs attributs représente le pourcentage de gain par rapport au profil de référence. Ces scores représentent un score moyen par profil. Les scores moyens par profil sont représentés dans le tableau [Tab.9] :

Table 9: Tableaux des scores moyens par profil

Profil	Score	Taux
Xute 32GB	3100	100%
Xute 64GB	3401.892	109.73%
Labists 32GB	3455.058	111.45%
Labists 64GB	**3621.845**	**116.83%**

Ainsi, un gain approximatif de **11%** est observé entre les profils qui disposent d'une carte SD de 32GB ainsi qu'un gain approximatif de **7%** entre les profils qui disposent d'une carte SD de 64GB. Quant au gain de performances entre le profil au score le plus bas et celui au score le plus haut, il équivaut à approximativement **17%**.

Cependant, ces résultats sont à contraster. En effet, les valeurs présentées dans les attributs affectés positivement [Tab.5] et les attributs affectés négativement [Tab.6] démontrent que la métrique réseau est prédominante au travers de 5 attributs. Cette dernière peut fausser l'interprétation des résultats. Les scores moyens par profil obtenus, sans la métrique réseau, sont représentés dans le tableau [Tab.10] :

Table 10: Tableaux des scores moyens par profil

Profil	Score	Taux
Xute 32GB	2600	100%
Xute 64GB	**2898.11**	**111.46%**
Labists 32GB	2637.03	101.42%
Labists 64GB	2823.78	108.60%

Cette fois, le gain entre les profils Xute et Labists à 32GB n'est plus que de **1.42%**. Quant aux profils à 64GB, le profil Xute l'emporte avec un gain approximatif de **3%** par rapport au kit Labists et un gain total de **11%** par rapport au profil de référence. Dans le cas d'un cluster de type HPC, comme celui implémenté au cours de ce travail, les RPI au profil Xute 64GB sont celles qui sont les plus favorables de fournir des résultats concluant.

En conclusion, bien que les noeuds présentent des performances non similaires, aucun ne peut être jugé comme non fonctionnel ou présentant des anomalies. Les 8 noeuds sont donc validés pour la *Phase 2 : optimisation*.

Deuxièmement : Le score global moyen et le score moyen par métrique des performances par défaut sont calculés en effectuant la moyenne des valeurs récoltées pour les 8 noeuds, et ce, pour chacun des attributs. Ces valeurs sont représentées dans le tableau [Tab.11] :

Table 11: Valeurs moyennes du profil par défaut

Métrique	Attribut	Valeur	Unité
CPU	Compression R/U Rate single-thread	1793.5	MIPS
	Decompression R/U Rate single-thread	1977.375	MIPS
	Compression R/U Rate multi-thread	1632.875	MIPS
	Decompression R/U Rate multi-thread	3544.625	MIPS
	Compression R/U Rate hyper-threading	1438.5	MIPS
	Decompression R/U Rate hyper-threading	5706.5	MIPS
Stockage	Local write speed 256	22.3	MB/s
	Local write speed 512	23.45	MB/s
	Local write speed 1024	24.5125	MB/s
	Cached reads speed	843.4525	MB/s
	Buffered disk reads speed	43.05875	MB/s
	O_DIRECT cached reads speed	37.56	MB/s
	O_DIRECT disk reads	41.88125	MB/s
Réseau	bandwith 8	2.86625	Mb/s
	bandwith 64	2.875	Mb/s
	bandwith 256	3.03125	Mb/s
	bandwith 512	2.88	Mb/s
	bandwith 1024	2.865	Mb/s
CPU/GPU	Blowfish Score	5.958829125	
	CryptoHash Score	303.8942988	
	Fibonacci Score	1.846820875	
	Zlib Score	8.86680125	
	N-Queens Score	0.4023975	
	FFT Score	4.724742125	
	Raytracing Score	2.3593985	
	Total events CPU	26420.875	
RAM	Total operations Memory	84140161.38	
	Mean transfer speed Memory	8213.7125	MB/s
Stockage	Total operations IO per second	1275.6225	op/s
	Mean transfer speed IO	4.41875	MB/s
	Total events IO	384809.5	

Ces valeurs constituent les valeurs de référence pour la *Phase 2 : optimisation* et sont utilisées afin de calculer les gains de performances subis lors de chaque optimisation.

Le score global moyen est obtenu en effectuant la somme de l'ensemble des valeurs obtenues remises sous forme de pourcentage. Ainsi, pour le profil par défaut des RPI, chaque valeur est égale à 100 et le score global moyen est égal à la somme des 31 attributs, soit 3100.

Quant aux scores moyens par métrique, ils sont calculés en ne gardant que les attributs qui sont liés à chaque métrique. Comme le score global moyen, leurs valeurs sont remises sous forme de pourcentages. Le détail de ces derniers est donné au chapitre *Système de score*.

8.2 Phase 2 : optimisation

Les objectifs de la *Phase 2 : optimisation* sont : premièrement, réaliser les optimisations annoncées dans la section *Implémentation* ; deuxièmement, calculer le score global moyen et le score moyen par métriques du cluster, et ce, après chaque optimisation ; troisièmement, calculer les gains de performances et établir une comparaison avec le profil par défaut ; quatrièmement, créer les profils qui sont utilisés lors de la *Phase 3 : évaluation en cluster*.

Pour rappel, l'objectif de la *Phase 3 : évaluation en cluster* est d'exécuter une évaluation de type HPC grâce au logiciel HPL. Au travers de cette section, l'objectif est donc de trouver les optimisations qui viennent maximiser la métrique CPU des noeuds. Les détails de chaque optimisation se trouvent au chapitre *Phase 2 : optimisation*.

8.2.1 Optimisations CPU

L'optimisation CPU est réalisée en deux temps : premièrement, grâce aux paramètres *arm_freq* et *arm_freq_min* ; deuxièmement, grâce au paramètre *over_voltage*.

1) Les résultats générés par les paramètres *arm_freq=1800* et *arm_freq_min=1800* (CPU 1800MHz) sont [Tab.12] :

Table 12: Valeurs moyennes pour *arm_freq=1800*

Attributs	Val. défaut	Val. 1800MHz	Taux
Compression R/U Rate single-thread (MIPS)	1793.5	1889.125	105.33%
Decompression R/U Rate single-thread (MIPS)	1977.375	2081.875	105.28%
Compression R/U Rate multi-thread (MIPS)	1632.875	1704.75	104.4%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3687.125	104.02%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1481.375	102.98%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	5872.75	102.91%
Local write speed 256 (MB/s)	22.3	21.025	94.28%
Local write speed 512 (MB/s)	23.45	22.325	95.2%
Local write speed 1024 (MB/s)	24.5125	24.45	99.75%
Cached reads speed (MB/s)	843.4525	835.0975	99.01%
Buffered disk reads speed (MB/s)	43.05875	43.28875	100.53%
O_DIRECT cached reads speed (MB/s)	37.56	38.56875	102.69%
O_DIRECT disk reads (MB/s)	41.88125	42.51125	101.5%
bandwith 8 (Mb/s)	2.86625	2.865	99.96%
bandwith 64 (Mb/s)	2.875	2.86	99.48%
bandwith 256 (Mb/s)	3.03125	2.88625	95.22%
bandwith 512 (Mb/s)	2.88	2.8775	99.91%
bandwith 1024 (Mb/s)	2.865	2.875	100.35%
Blowfish Score	5.958829125	5.561257125	93.33%
CryptoHash Score	303.8942988	324.9679764	106.93%
Fibonacci Score	1.846820875	1.717105125	92.98%
Zlib Score	8.86680125	8.236897875	92.9%
N-Queens Score	0.4023975	0.401508375	99.78%
FFT Score	**4.724742125**	**5.300535125**	**112.19%**
Raytracing Score	2.3593985	2.16969125	91.96%
Total events CPU	26420.875	28171.375	106.63%
Total operations Memory	84140161.38	88620550.25	105.32%
Mean transfer speed Memory (MB/s)	8213.7125	8651.24	105.33%
Total operations IO per second (op/s)	1275.6225	1226.15	96.12%
Mean transfer speed IO (MB/s)	4.41875	4.24875	96.15%
Total events IO	382309.5	367532.75	96.13%

Les attributs affectés positivement **(+ 105%)** sont les attributs Compression R/U Rate single-thread, Decompression R/U Rate single-thread, CryptoHash Score, FFT Score, Total events CPU, Total operations Memory et Mean transfer speed Memory. Ces attributs sont similaires à ceux représentés dans [Tab.6] et sont représentatifs d'une forte relation avec la configuration CPU des RPI.

Quant aux attributs affectés négativement **(- 95%)**, ils sont les suivants : Local write speed 256, Blowfish Score, Fibonacci Score, Zlib Score, Raytracing Score. Ces attributs sont similaires à ceux représentés dans [Tab.5] et sont représentatifs d'une forte relation avec la puissance du CPU attribuée à la partie GPU des RPI.

Ainsi, le gain relatif aux attributs affectés positivement est de **6.71%** et la perte relative aux attributs affectés négativement est de **6.92%**.

Les performances sur les scores métriques sont les suivantes : gain de **1.54%** pour la métrique CPU ; gain de **5.32%** pour la métrique RAM ; perte de **1.87%** pour la métrique stockage ; perte de **1.02%** pour la métrique réseau.

Le gain sur le score global moyen est de **0.27%** et le gain sur la métrique CPU hors GPU est de **5.04%**.

2) Les résultats générés par les paramètres *arm_freq=2100* et *arm_freq_min=2100* (CPU 2100MHz) sont [Tab.13] :

Table 13: Valeurs moyennes pour *arm_freq=2100*

Attributs	**Val. défaut**	**Val. 2100MHz**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	1590.625	88.69%
Decompression R/U Rate single-thread (MIPS)	1977.375	1751.75	88.59%
Compression R/U Rate multi-thread (MIPS)	1632.875	1418.625	86.88%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3057	86.24%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1216.125	84.54%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	4823.75	84.53%
Local write speed 256 (MB/s)	22.3	21.025	94.28%
Local write speed 512 (MB/s)	23.45	22.9375	97.81%
Local write speed 1024 (MB/s)	24.5125	24.5625	100.2%
Cached reads speed (MB/s)	843.4525	856.28375	101.52%
Buffered disk reads speed (MB/s)	43.05875	43.26125	100.47%
O_DIRECT cached reads speed (MB/s)	37.56	38.40625	102.25%
O_DIRECT disk reads (MB/s)	41.88125	42.41	101.26%
bandwith 8 (Mb/s)	2.86625	2.85625	99.65%
bandwith 64 (Mb/s)	2.875	2.9775	103.57%
bandwith 256 (Mb/s)	3.03125	3.07	101.28%
bandwith 512 (Mb/s)	2.88	2.89875	100.65%
bandwith 1024 (Mb/s)	2.865	2.9	101.22%
Blowfish Score	5.958829125	4.839371	81.21%
CryptoHash Score	**303.8942988**	**374.0990819**	**123.1%**
Fibonacci Score	1.846820875	1.490678875	80.72%
Zlib Score	8.86680125	7.12975325	80.41%
N-Queens Score	0.4023975	0.436219875	108.41%
FFT Score	4.724742125	5.250317625	111.12%
Raytracing Score	2.3593985	1.901458	80.59%
Total events CPU	26420.875	32485.25	122.95%
Total operations Memory	84140161.38	101674147.8	120.84%
Mean transfer speed Memory (MB/s)	8213.7125	9929.48625	120.89%
Total operations IO per second (op/s)	1275.6225	1234.19375	96.75%
Mean transfer speed IO (MB/s)	4.41875	4.2775	96.8%
Total events IO	382309.5	369883.5	96.75%

Les attributs affectés positivement sont les attributs CryptoHash Score, N-Queens Score, FFT Score, Total events CPU, Total operations Memory et Mean transfer speed Memory.

Les attributs affectés négativement sont les attributs Compression R/U Rate single-thread, Decompression R/U Rate single-thread, Compression R/U Rate multi-thread, Decompression R/U Rate multi-thread, Compression R/U Rate hyper-threading, Decompression R/U Rate hyper-threading, Local write speed 256, Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score.

La forte diminution des performances pour les attributs du *benchmark 7Zip* s'explique principalement par l'incapacité (crash) des noeuds n°5 et n°7 [Tab.2] à exécuter le *benchmark*. Les valeurs pour les attributs de compression et de décompression sont donc égales à 0 pour ces noeuds. De plus, le noeud n°6 a été incapable de dépasser 1900MHz de fréquence.

Quant aux autres attributs affectés positivement et négativement, ce sont ceux déjà présentés dans le point *1)* où l'écart continue de se creuser.

Ainsi, le gain relatif aux attributs affectés positivement est de **17.88%** et la perte relative aux attributs affectés négativement est de **14.86%**.

Les performances sur les scores métriques sont les suivantes : perte de **6.58%** pour la métrique CPU ; gain de **20.86%** pour la métrique RAM ; perte de **1.19%** pour la métrique stockage ; gain de **1.27%** pour la métrique réseau.

La perte sur le score global moyen est de **1.81%** et la perte sur la métrique CPU hors GPU est de **1.50%**.

3) Lors de l'étape d'optimisation avec les paramètres *arm_freq=2100*, *arm_freq_min=2100* (CPU 2100MHz) et *over_voltage=3*, les 8 noeuds ont été incapables de démarrer. Cela s'explique par le fait que les RPI soient en sous-tension comme exprimé au travers de [Fig.19], bien que dans le point *2)*, ils aient fonctionnés sans cette tension supplémentaire.

4) Les résultats générés par les paramètres *arm_freq=2100* et *arm_freq_min=2100* (CPU 2100MHz) et *over_voltage=6* sont [Tab.14]:

Table 14: Valeurs moyennes pour *arm_freq=2100* et *over_voltage=6*

Attributs	**Val. défaut**	**Val. 2100MHz 6v**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	1587.625	88.52%
Decompression R/U Rate single-thread (MIPS)	1977.375	1748.625	88.43%
Compression R/U Rate multi-thread (MIPS)	1632.875	1419.125	86.91%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3056.5	86.23%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1221.75	84.93%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	4842.875	84.87%
Local write speed 256 (MB/s)	22.3	19.175	85.99%
Local write speed 512 (MB/s)	23.45	21.95	93.6%
Local write speed 1024 (MB/s)	24.5125	23.5875	96.23%
Cached reads speed (MB/s)	843.4525	857.02875	101.61%
Buffered disk reads speed (MB/s)	43.05875	43.2525	100.45%
O_DIRECT cached reads speed (MB/s)	37.56	38.24375	101.82%
O_DIRECT disk reads (MB/s)	41.88125	42.47875	101.43%

Attributs	Val. défaut	Val. 6v	Taux
bandwith 8 (Mb/s)	2.86625	2.97125	103.66%
bandwith 64 (Mb/s)	2.875	3.04375	105.87%
bandwith 256 (Mb/s)	3.03125	2.90375	95.79%
bandwith 512 (Mb/s)	2.88	3	104.17%
bandwith 1024 (Mb/s)	2.865	2.92	101.92%
Blowfish Score	5.958829125	4.83806425	81.19%
CryptoHash Score	**303.8942988**	**374.2515189**	**123.15%**
Fibonacci Score	1.846820875	1.490681375	80.72%
Zlib Score	8.86680125	7.114624625	80.24%
N-Queens Score	0.4023975	0.464619375	115.46%
FFT Score	4.724742125	5.255875125	111.24%
Raytracing Score	2.3593985	1.92900975	81.76%
Total events CPU	26420.875	32478.625	122.93%
Total operations Memory	84140161.38	102198350.8	121.46%
Mean transfer speed Memory (MB/s)	8213.7125	9977.15625	121.47%
Total operations IO per second (op/s)	1275.6225	1159.385	90.89%
Mean transfer speed IO (MB/s)	4.41875	3.895625	88.16%
Total events IO	382309.5	347446.625	90.88%

Les attributs affectés positivement sont les attributs bandwith 64, CryptoHash Score, N-Queens Score, FFT Score, Total events CPU, Total operations Memory et Mean transfer speed Memory.

Les attributs affectés négativement sont les attributs Compression R/U Rate single-thread, Decompression R/U Rate single-thread, Compression R/U Rate multi-thread, Decompression R/U Rate multi-thread, Compression R/U Rate hyper-threading, Decompression R/U Rate hyper-threading, Local write speed 256, Local write speed 512, Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score, Total operations IO per second, Mean transfer speed IO et Total events IO.

L'observation réalisée dans le point *2)* se poursuit et les noeuds n°5 et n°7 restent incapables d'exécuter le *benchmark 7Zip*. Le noeud n°6 est toujours bloqué à 1900MHz. De plus, d'autres attributs se rajoutent à la liste des attributs affectés négativement témoignant d'une forte instabilité lorsque le CPU est soumis à une haute fréquence CPU à une haute tension.

Ainsi, le gain relatif aux attributs affectés positivement est de **17.36%** et la perte relative aux attributs affectés négativement est de **13.79%**.

Les performances sur les scores métriques sont les suivantes : perte de **5.96%** pour la métrique CPU ; gain de **21.46%** pour la métrique RAM ; perte de **4.90%** pour la métrique stockage ; gain de **2.28%** pour la métrique réseau.

La perte sur le score global moyen est de **2.52%** et la perte sur la métrique CPU hors GPU est de **0.74%**.

conclusion) Pour la suite des *benchmarks*, la fréquence CPU des noeuds n°5, n°6 et n°7 [Tab.2], les RPI Labists, a été fixée à 1900MHz afin d'éviter les instabilités et les erreurs lors des tests. Les autres noeuds restent à 2100MHz.

8.2.2 Optimisations GPU

L'optimisation GPU est réalisée grâce au paramètre *gpu_freq* qui varie entre 200MHz et 750MHz.

1) Les résultats générés par les paramètres *gpu_freq=200* (GPU 200MHz), *arm_freq =2100*, *arm_freq_min=2100* (CPU 2100MHz) et *over_voltage=6* sont [Tab.15]:

Table 15: Valeurs moyennes pour *gpu_freq=200*, *arm_freq=2100* et *over_voltage=6*

Attributs	**Val. défaut**	**Val. 200MHz**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	2045.875	114.07%
Decompression R/U Rate single-thread (MIPS)	1977.375	2252.75	113.93%
Compression R/U Rate multi-thread (MIPS)	1632.875	1815.75	111.2%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3903.75	110.13%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1561.875	108.58%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6201.25	108.67%
Local write speed 256 (MB/s)	22.3	20.225	90.7%
Local write speed 512 (MB/s)	23.45	22.1625	94.51%
Local write speed 1024 (MB/s)	24.5125	23.7875	97.04%
Cached reads speed (MB/s)	843.4525	753.5475	89.34%
Buffered disk reads speed (MB/s)	43.05875	43.24375	100.43%
O_DIRECT cached reads speed (MB/s)	37.56	38.38125	102.19%
O_DIRECT disk reads (MB/s)	41.88125	42.25625	100.9%
bandwith 8 (Mb/s)	2.86625	2.9275	102.14%
bandwith 64 (Mb/s)	2.875	2.95	102.61%
bandwith 256 (Mb/s)	3.03125	3.1	102.27%
bandwith 512 (Mb/s)	2.88	2.9625	102.86%
bandwith 1024 (Mb/s)	2.865	2.96875	103.62%
Blowfish Score	5.958829125	4.97192675	83.44%
CryptoHash Score	**303.8942988**	**365.395512**	**120.24%**
Fibonacci Score	1.846820875	1.52952025	82.82%
Zlib Score	8.86680125	7.3122685	82.47%
N-Queens Score	0.4023975	0.4242435	105.43%
FFT Score	4.724742125	5.27979275	111.75%
Raytracing Score	2.3593985	1.975901125	83.75%
Total events CPU	26420.875	31696.375	119.97%
Total operations Memory	84140161.38	98855986.13	117.49%
Mean transfer speed Memory (MB/s)	8213.7125	9650.725	117.5%
Total operations IO per second (op/s)	1275.6225	1187.42	93.09%
Mean transfer speed IO (MB/s)	4.41875	4.11625	93.15%
Total events IO	382309.5	355891.25	93.09%

Les attributs affectés positivement sont les attributs Compression R/U Rate single-thread, Decompression R/U Rate single-thread, Compression R/U Rate multi-thread, Decompression R/U Rate multi-thread, Compression R/U Rate hyper-threading, Decompression R/U Rate hyper-threading, CryptoHash Score, N-Queens Score, FFT Score, Total events CPU, Total operations Memory et Mean transfer speed Memory.

Les attributs affectés négativement sont les attributs Local write speed 256, Local write speed 512, Cached reads speed, Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score, Total operations IO per second, Mean transfer speed IO et Total events IO.

La décision de fixer la fréquence CPU à 1900MHz pour les noeuds n°5, n°6 et n°7, prise à la *conclusion)* du point *Optimisation CPU*, permet de ré-obtenir des valeurs favorables lors de l'exécution du *benchmark* 7Zip.

Quant aux autres attributs, ils sont similaires à ceux des précédents points et présentent sensiblement les mêmes valeurs. Bien que la fréquence du GPU soit diminuée, les valeurs récoltées pour les attributs Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score sont supérieures à celles présentes dans [Tab.13] et [Tab.14]. Cela démontre une plus grande stabilité des noeuds.

Ainsi, le gain relatif aux attributs affectés positivement est de **13.24%** et la perte relative aux attributs affectés négativement est de **11.38%**.

Les performances sur les scores métriques sont les suivantes : gain de**4.03%** pour la métrique CPU ; gain de **17.49%** pour la métrique RAM ; perte de **4.56%** pour la métrique stockage ; gain de **2.69%** pour la métrique réseau.

Le gain sur le score global moyen est de **1.91%** et le gain sur la métrique CPU hors GPU est de **12.39%**.

2) Les résultats générés par les paramètres *gpu_freq=500* (GPU 500MHz), *arm_freq =2100*, *arm_freq_min=2100* (CPU 2100MHz) et *over_voltage=6* sont [Tab.16]:

Table 16: Valeurs moyennes pour *gpu_freq=500*, *arm_freq=2100* et *over_voltage=6*

Attributs	**Val. défaut**	**Val. 500MHz**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	2087.5	116.39%
Decompression R/U Rate single-thread (MIPS)	1977.375	2297.125	116.17%
Compression R/U Rate multi-thread (MIPS)	1632.875	1864	114.15%
Decompression R/U Rate multi-thread (MIPS)	3544.625	4017.125	113.33%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1607.625	111.76%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6379.625	111.8%
Local write speed 256 (MB/s)	22.3	20.7875	93.22%
Local write speed 512 (MB/s)	23.45	22.375	95.42%
Local write speed 1024 (MB/s)	24.5125	24.075	98.22%
Cached reads speed (MB/s)	843.4525	853.69875	101.21%
Buffered disk reads speed (MB/s)	43.05875	43.26	100.47%
O_DIRECT cached reads speed (MB/s)	37.56	38.3175	102.02%
O_DIRECT disk reads (MB/s)	41.88125	42.5625	101.63%
bandwith 8 (Mb/s)	2.86625	2.965	103.45%
bandwith 64 (Mb/s)	2.875	2.94	102.26%
bandwith 256 (Mb/s)	3.03125	2.93875	96.95%
bandwith 512 (Mb/s)	2.88	2.9525	102.52%
bandwith 1024 (Mb/s)	2.865	2.9625	103.4%
Blowfish Score	5.958829125	4.947430875	83.03%
CryptoHash Score	**303.8942988**	**366.0371946**	**120.45%**
Fibonacci Score	1.846820875	1.529502875	82.82%
Zlib Score	8.86680125	7.316504625	82.52%
N-Queens Score	0.4023975	0.42949825	106.73%
FFT Score	4.724742125	5.255233875	111.23%
Raytracing Score	2.3593985	1.958106375	82.99%
Total events CPU	26420.875	31698.375	119.97%
Total operations Memory	84140161.38	99236451.75	117.94%
Mean transfer speed Memory (MB/s)	8213.7125	9687.92	117.95%
Total operations IO per second (op/s)	1275.6225	1161.68625	91.07%
Mean transfer speed IO (MB/s)	4.41875	4.02875	91.17%
Total events IO	382309.5	348209.875	91.08%

Les attributs affectés positivement sont similaires au point *1)*.

À l'exception de l'attribut Local write speed 512, les attributs affectés négativement sont similaires au point *1)*.

Malgré l'augmentation de la fréquence GPU à 500MHz, les valeurs récoltées pour les attributs Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score sont inférieures à celles présentes dans le point *1)*. Cependant, la différence entre celles-ci est de moins d'**1%**. Le passage de la fréquence de 200MHz à 500MHz n'a donc que très peu d'intérêt pour les attributs GPU.

Ainsi, le gain relatif aux attributs affectés positivement est de **14.82%** et la perte relative aux attributs affectés négativement est de **12.77%**.

Les performances sur les scores métriques sont les suivants : gain de **5.23%** pour la métrique CPU ; gain de **17.94%** pour la métrique RAM ; perte de **3.46%** pour la métrique stockage ; gain de **1.71%** pour la métrique réseau.

Le gain sur le score global moyen est de **2.68%** et le gain sur la métrique CPU hors GPU est de **14.19%**.

3) Les résultats générés par les paramètres *gpu_freq=750* (GPU 750MHz), *arm_freq =2100*, *arm_freq_min=2100* (CPU 2100MHz) et *over_voltage=6* sont [Tab.17] :

Table 17: Valeurs moyennes pour *gpu_freq=750*, *arm_freq=2100* et *over_voltage=6*

Attributs	**Val. défaut**	**Val. 750MHz**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	2100.375	117.11%
Decompression R/U Rate single-thread (MIPS)	1977.375	2312.75	116.96%
Compression R/U Rate multi-thread (MIPS)	1632.875	1881.25	115.21%
Decompression R/U Rate multi-thread (MIPS)	3544.625	4062.125	114.6%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1622.125	112.77%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6434.625	112.76%
Local write speed 256 (MB/s)	22.3	21.175	94.96%
Local write speed 512 (MB/s)	23.45	22.6	96.38%
Local write speed 1024 (MB/s)	24.5125	24.4	99.54%
Cached reads speed (MB/s)	843.4525	892.33875	105.8%
Buffered disk reads speed (MB/s)	43.05875	43.25375	100.45%
O_DIRECT cached reads speed (MB/s)	37.56	38.35	102.1%
O_DIRECT disk reads (MB/s)	41.88125	42.49375	101.46%
bandwith 8 (Mb/s)	2.86625	2.865	99.96%
bandwith 64 (Mb/s)	2.875	2.8625	99.57%
bandwith 256 (Mb/s)	3.03125	2.86875	94.64%
bandwith 512 (Mb/s)	2.88	3.0125	104.6%
bandwith 1024 (Mb/s)	2.865	2.875	100.35%
Blowfish Score	5.958829125	4.935165	82.82%
CryptoHash Score	**303.8942988**	**365.6791913**	**120.33%**
Fibonacci Score	1.846820875	1.529605375	82.82%
Zlib Score	8.86680125	7.298538625	82.31%
N-Queens Score	0.4023975	0.45145825	112.19%
FFT Score	4.724742125	5.229380375	110.68%
Raytracing Score	2.3593985	1.9561075	82.91%
Total events CPU	26420.875	31700.375	119.98%
Total operations Memory	84140161.38	99806306.75	118.62%
Mean transfer speed Memory (MB/s)	8213.7125	9748.59875	118.69%
Total operations IO per second (op/s)	1275.6225	1184.455	92.85%
Mean transfer speed IO (MB/s)	4.41875	4.00625	90.66%
Total events IO	382309.5	354947.5	92.84%

Les attributs affectés positivement sont similaires au point *1)* et *2)* et y ajoutant Cached reads speed.

Les attributs affectés négativement sont similaires au point *2)* en y ajoutant bandwith 256.

L'observation réalisée au point *2)* se poursuit car l'augmentation de la fréquence de 500MHz à 750MHz n'affecte les performances GPU des RPI que de moins de **1%**. Cependant, les performances CPU ont aussi augmentées.

Ainsi, le gain relatif aux attributs affectés positivement est de **15.05%** et la perte relative aux attributs affectés négativement est de **11.47%**.

Les performances sur les scores métriques sont les suivantes : gain de **5.96%** pour la métrique CPU ; gain de **18.65%** pour la métrique RAM ; perte de **2.30%** pour la métrique stockage ; perte de **0.18%** pour la métrique réseau.

Le gain sur le score global moyen est de **3.12%** et le gain sur la métrique CPU hors GPU est de **15.25%**.

8.2.3 Optimisations I/O

L'optimisation des I/O est réalisée grâce aux paramètres *dtoverlay* et *dtparam* qui permettent la désactivation du Bluetooth, du WiFi et des LEDs sur les RPI.

Les résultats générés par les paramètres *gpu_freq=750*, *arm_freq=2100*, *arm_freq_min =2100* et *over_voltage=6* avec désactivation des I/O sont [Tab.18] :

Table 18: Valeurs moyennes pour *gpu_freq=750*, *arm_freq=2100* et *over_voltage=6* avec désactivation des I/O

Attributs	**Val. défaut**	**Val. I/O**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	2045.75	114.06%
Decompression R/U Rate single-thread (MIPS)	1977.375	2249.25	113.75%
Compression R/U Rate multi-thread (MIPS)	1632.875	1812.5	111%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3895.125	109.89%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1557.625	108.28%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6166.875	108.07%
Local write speed 256 (MB/s)	22.3	21.875	98.09%
Local write speed 512 (MB/s)	23.45	23.225	99.04%
Local write speed 1024 (MB/s)	24.5125	24.625	100.46%
Cached reads speed (MB/s)	843.4525	747.53125	88.63%
Buffered disk reads speed (MB/s)	43.05875	43.27375	100.5%
O_DIRECT cached reads speed (MB/s)	37.56	38.66	102.93%
O_DIRECT disk reads (MB/s)	41.88125	42.30625	101.01%
bandwith 8 (Mb/s)	2.86625	2.88875	100.78%
bandwith 64 (Mb/s)	2.875	2.8725	99.91%
bandwith 256 (Mb/s)	3.03125	2.87875	94.97%
bandwith 512 (Mb/s)	2.88	2.8775	99.91%
bandwith 1024 (Mb/s)	2.865	3.26	113.79%
Blowfish Score	5.958829125	4.9557225	83.17%
CryptoHash Score	**303.8942988**	**365.7678973**	**120.36%**
Fibonacci Score	1.846820875	1.529505	82.82%
Zlib Score	8.86680125	7.329751875	82.67%
N-Queens Score	0.4023975	0.42163525	104.78%
FFT Score	4.724742125	5.283256125	111.82%
Raytracing Score	2.3593985	1.945939125	82.48%
Total events CPU	26420.875	31686.25	119.93%
Total operations Memory	84140161.38	98946910.25	117.6%
Mean transfer speed Memory (MB/s)	8213.7125	9659.52625	117.6%
Total operations IO per second (op/s)	1275.6225	1293.9825	101.44%
Mean transfer speed IO (MB/s)	4.41875	4.3575	98.61%
Total events IO	382309.5	387846.75	101.45%

Les attributs affectés positivement sont les attributs Compression R/U Rate single-thread, Decompression R/U Rate single-thread, Compression R/U Rate multi-thread, Decompression R/U Rate multi-thread, Compression R/U Rate hyper-threading, Decompression R/U Rate hyper-threading, bandwith 1024, CryptoHash Score, FFT Score, Total events CPU, Total operations Memory et Mean transfer speed Memory.

Les attributs affectés négativement sont les attributs Cached reads speed, bandwith 256, Blowfish Score, Fibonacci Score, Zlib Score et Raytracing Score.

Ainsi, le gain relatif aux attributs affectés positivement est de **13.84%** et la perte relative aux attributs affectés négativement est de **14.22%**.

Les performances sur les scores métriques sont les suivantes : gain de **3.79%** pour la métrique CPU ; gain de **17.60%** pour la métrique RAM ; perte de **0.79%** pour la métrique stockage ; gain de **1.87%** pour la métrique réseau.

Le gain sur le score global moyen est de **2.89%** et le gain sur la métrique CPU hors GPU est de **12.19%**.

8.2.4 Optimisations système

L'optimisation du système d'exploitation est réalisée grâce à la mise à jour du système, à l'installation et la mise à jour du firmware EEPROM, à la suppression du service Snap et au nettoyage du système.

Les résultats générés par les paramètres *gpu_freq=750*, *arm_freq=2100*, *arm_freq_min =2100* et *over_voltage=6* avec désactivation des I/O et les optimisations système sont [Tab.19] :

Table 19: Valeurs moyennes pour *gpu_freq=750*, *arm_freq=2100* et *over_voltage=6* avec désactivation des I/O et optimisation du système d'exploitation

Attributs	**Val. défaut**	**Val. Sys**	**Taux**
Compression R/U Rate single-thread (MIPS)	1793.5	2037.25	113.59%
Decompression R/U Rate single-thread (MIPS)	1977.375	2241.125	113.34%
Compression R/U Rate multi-thread (MIPS)	1632.875	1816.125	111.22%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3909.125	110.28%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1562	108.59%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6189.875	108.47%
Local write speed 256 (MB/s)	22.3	22.2875	99.94%
Local write speed 512 (MB/s)	23.45	23.4	99.79%
Local write speed 1024 (MB/s)	24.5125	24.525	100.05%
Cached reads speed (MB/s)	843.4525	753.80625	89.37%
Buffered disk reads speed (MB/s)	43.05875	43.0925	100.08%
O_DIRECT cached reads speed (MB/s)	37.56	37.875	100.84%
O_DIRECT disk reads (MB/s)	41.88125	42.01625	100.32%
bandwith 8 (Mb/s)	2.86625	2.8575	99.69%
bandwith 64 (Mb/s)	2.875	2.8525	99.22%
bandwith 256 (Mb/s)	3.03125	2.88375	95.13%
bandwith 512 (Mb/s)	2.88	2.8575	99.22%
bandwith 1024 (Mb/s)	2.865	2.85875	99.78%
Blowfish Score	5.958829125	4.941391875	82.93%
CryptoHash Score	**303.8942988**	**365.2547673**	**120.19%**
Fibonacci Score	1.846820875	1.529741625	82.83%
Zlib Score	8.86680125	7.313273125	82.48%

Attributs	Val. défaut	Val. Sys	Taux
N-Queens Score	0.4023975	0.45367925	112.74%
FFT Score	4.724742125	5.30611375	112.3%
Raytracing Score	2.3593985	1.961683375	83.14%
Total events CPU	26420.875	31687.5	119.93%
Total operations Memory	84140161.38	99140489.5	117.83%
Mean transfer speed Memory (MB/s)	8213.7125	9678.47125	117.83%
Total operations IO per second (op/s)	1275.6225	1255.4475	98.42%
Mean transfer speed IO (MB/s)	4.41875	4.225	95.62%
Total events IO	382309.5	376223.625	98.41%

Les attributs affectés positivement sont similaires au point *Optimisations I/O*, à l'exception de bandwith1024 et en y ajoutant N-Queens Score

Les attributs affectés négativement sont similaires au point *Optimisations I/O*, à l'exception de bandwith 256.

Ainsi, le gain relatif aux attributs affectés positivement est de **13.86%** et la perte relative aux attributs affectés négativement est de **15.85%**.

Les performances sur les scores métriques sont les suivantes : gain de **4.43%** pour la métrique CPU ; gain de **17.83%** pour la métrique RAM ; perte de **1.72%** pour la métrique stockage ; perte de **1.40%** pour la métrique réseau.

Le gain sur le score global moyen est de **2.37%** et le gain sur la métrique CPU hors GPU est de **13.06%**.

8.2.5 Optimisations RAM

L'optimisation de la mémoire RAM est réalisée grâce à la désactivation des fichiers swap et à l'activation de la compression ZRAM.

Les résultats générés par les paramètres *gpu_freq=750*, *arm_freq=2100*, *arm_freq_min =2100* et *over_voltage=6* avec désactivation des I/O, les optimisations système et les optimisation RAM sont [Tab.20] :

Table 20: Valeurs moyennes pour *gpu_freq=750*, *arm_freq=2100* et *over_voltage=6* avec désactivation des I/O, optimisation du système et optimisation RAM

Attributs	Val. défaut	Val. RAM	Taux
Compression R/U Rate single-thread (MIPS)	1793.5	2041.5	113.83%
Decompression R/U Rate single-thread (MIPS)	1977.375	2249.25	113.75%
Compression R/U Rate multi-thread (MIPS)	1632.875	1819.125	111.41%
Decompression R/U Rate multi-thread (MIPS)	3544.625	3916.25	110.48%
Compression R/U Rate hyper-threading (MIPS)	1438.5	1563.625	108.7%
Decompression R/U Rate hyper-threading (MIPS)	5706.5	6197	108.6%
Local write speed 256 (MB/s)	22.3	22.0625	98.93%
Local write speed 512 (MB/s)	23.45	23.1625	98.77%
Local write speed 1024 (MB/s)	24.5125	23.9375	97.65%
Cached reads speed (MB/s)	843.4525	748.38875	88.73%
Buffered disk reads speed (MB/s)	43.05875	43.07375	100.03%
O_DIRECT cached reads speed (MB/s)	37.56	37.93125	100.99%
O_DIRECT disk reads (MB/s)	41.88125	42.0825	100.48%
bandwith 8 (Mb/s)	2.86625	2.8975	101.09%
bandwith 64 (Mb/s)	2.875	2.85	99.13%

Attributs	Val. défaut	Val. RAM	Taux
bandwith 256 (Mb/s)	3.03125	2.8675	94.6%
bandwith 512 (Mb/s)	2.88	2.88625	100.22%
bandwith 1024 (Mb/s)	2.865	3.04	106.11%
Blowfish Score	5.958829125	4.9519985	83.1%
CryptoHash Score	**303.8942988**	**365.5791228**	**120.3%**
Fibonacci Score	1.846820875	1.52946825	82.82%
Zlib Score	8.86680125	7.297535625	82.3%
N-Queens Score	0.4023975	0.458314	113.9%
FFT Score	4.724742125	5.312168	112.43%
Raytracing Score	2.3593985	1.943435625	82.37%
Raytracing Score	2.3593985	1.943435625	82.37%
Total events CPU	26420.875	31692.75	119.95%
Total operations Memory	84140161.38	99209491.5	117.91%
Mean transfer speed Memory (MB/s)	8213.7125	9685.2175	117.92%
Total operations IO per second (op/s)	1275.6225	1256.835	98.53%
Mean transfer speed IO (MB/s)	4.41875	4.229375	95.71%
Total events IO	382309.5	376677	98.53%

Les attributs affectés positivement sont similaires au point *Optimisations système*, en y ajoutant bandwith 1024.

Les attributs affectés négativement sont similaires au point *Optimisations système*, en y ajoutant bandwith 256.

Ainsi, le gain relatif aux attributs affectés positivement est de **13.48%** et la perte relative aux attributs affectés négativement est de **14.35%**.

Les performances sur les scores métriques sont les suivants : gain de **4.56%** pour la métrique CPU ; gain de **17.91%** pour la métrique RAM ; perte de **2.17%** pour la métrique stockage ; gain de **0.22%** pour la métrique réseau.

Le gain sur le score global moyen est de **2.55%** et le gain sur la métrique CPU hors GPU est de **13.33%**.

8.2.6 Observations et conclusions

Dans le point *Optimisations CPU*, les résultats démontrent que :

1. parmi les attributs liés aux CPUs, Blowfish Score, Fibonacci Score, Zlib Score, Raytracing Score sont étroitement liés avec les performances de la partie GPU. Il est observé que les RPI répartissent la puissance qui leur est attribuée entre ces deux métriques, CPU et/ou GPU. Augmenter l'une des deux implique alors la diminution de la seconde ;

2. à partir d'une fréquence CPU de 2100MHz, les RPI deviennent instables. Certaines (noeud n°6) deviennent incapables de démarrer avec cette configuration. D'autres (noeuds n°5 et n°7) échouent à réaliser des tâches complexes basées sur l'utilisation du CPU ;

3. une mauvaise combinaison entre la fréquence attendue et la tension accordée aux RPI résulte en une incapacité à démarrer ;

4. une meilleure combinaison entre la fréquence attendue et la tension accordée aux RPI, comme indiqué dans [Fig.19], permet aux RPI de démarrer. Cependant, malgré la tension supplémentaire attribuée, certaines RPI (noeud n°5, n°6 et n°7) restent incapables de démarrer avec une fréquence supérieure à 1900MHz. De plus, cette configuration, trop instable pour les RPI ne fait que creuser l'écart avec les performances du profil par défaut, le score global moyen subissant une perte de **2.52%** ;

5. la configuration qui offre les meilleurs performances (score global moyen et métrique CPU hors GPU) est celle où la fréquence CPU est poussée à 1800MHz, soit la fréquence maximale recommandée par le constructeur.

Dans le point *Optimisations GPU*, les résultats démontrent que :

1. la décision de fixer la fréquence CPU à 1900MHz pour les noeuds n°5, n°6 et n°7 permet de ré-obtenir des valeurs favorables lors de l'exécution du *benchmark* 7Zip ainsi qu'un gain de **4.03%** pour la métrique CPU. Auparavant, la métrique CPU subissait une perte de **5.96%** au point *4)* ;

2. l'augmentation de la fréquence GPU de 200MHz à 750MHz n'impacte que très légèrement les performances des attributs liés au GPU. La variation de performance est approximativement de **1%** ;

3. l'augmentation de la fréquence GPU de 200MHz à 750MHz implique un gain de **1.93%** pour la métrique CPU, **1.16%** pour la métrique RAM, **2.26%** pour la métrique stockage ainsi qu'une perte de **2.87%** pour la métrique réseau ;

4. l'augmentation de la fréquence GPU de 200MHz à 750MHz implique un gain de **2.86%** pour la métrique CPU sans les attributs GPU ;

5. bien que le gain soit faible, plus aucune instabilité n'est à déplorer après les optimisations.

Dans les points *Optimisations I/O*, *Optimisations système* et *Optimisations RAM* les résultats démontrent que :

1. en comparaison au point *3)* des *Optimisations GPU*, les performances obtenues sur le score global moyen par les 3 catégories d'optimisations sont diminuées de **0.40%**.

2. en comparaison au point *3)* des *Optimisations GPU*, les performances de la métrique CPU hors GPU résultant des 3 catégories d'optimisations sont diminuées de **2%**.

3. bien que les optimisations réalisées puissent être intéressantes à long terme et pour la longévité du RPI, elles ne représentent pas un gain de performances.

Quant aux résultats moyens générés par l'ensemble des profils d'optimisation, ils sont les suivants [Tab.21]:

Table 21: Résultats moyens en pourcentage pour l'ensemble des profils d'optimisation

MIPS	**0.** Compression R/U Rate single-thread	**1.** Decompression R/U Rate single-thread
	2. Compression R/U Rate multi-thread	**3.** Decompression R/U Rate multi-thread
	4. Compression R/U Rate hyper-threading	**5.** Decompression R/U Rate hyper-threading

MB/s	**6.** Local write speed 256	**7.** Local write speed 512
	8. Local write speed 1024	
	9. Cached reads speed	**10.** Buffered disk reads speed
	11. O_DIRECT cached reads speed	**12.** O_DIRECT disk reads

Mb/s	**13.** bandwith 8	**14.** bandwith 64
	15. bandwith 256	**16.** bandwith 512
	17. bandwith 1024	

Score	**18.** Blowfish Score	**19.** CryptoHash Score
	20. Fibonacci Score	**21.** Zlib Score
	22. N-Queens Score	**23.** FFT Score
	24. Raytracing Score	

25. Tot. events CPU	**26.** Tot. ops. Memory (op/s)	**27.** Mean trans. speed Mem. (MB/s)
28. Tot. ops. IO per sec. (op/s)	**29.** Mean trans. speed IO (MB/s)	**30.** Tot. events IO

Att.	**1800**	**2100**	**6v**	**200**	**500**	**750**	**IO**	**Sys**	**RAM**
0.	105.33	88.69	88.52	114.07	116.39	**117.11**	114.06	113.59	113.83
1.	105.28	88.59	88.43	113.93	116.17	**116.96**	113.75	113.34	113.75
2.	104.4	86.88	86.91	111.2	114.15	**115.21**	111	111.22	111.41
3.	104.02	86.24	86.23	110.13	113.33	**114.6**	109.89	110.28	110.48
4.	102.98	84.54	84.93	108.58	111.76	**112.77**	108.28	108.59	108.7
5.	102.91	84.53	84.87	108.67	111.8	**112.76**	108.07	108.47	108.6
6.	94.28	94.28	85.99	90.7	93.22	94.96	98.09	**99.94**	98.93
7.	95.2	97.81	93.6	94.51	95.42	96.38	99.04	**99.79**	98.77
8.	99.75	100.2	96.23	97.04	98.22	99.54	**100.46**	100.05	97.65
9.	99.01	101.52	101.61	89.34	101.21	**105.8**	88.63	89.37	88.73
10.	**100.53**	100.47	100.45	100.43	100.47	100.45	100.5	100.08	100.03
11.	102.69	102.25	101.82	102.19	102.02	102.1	**102.93**	100.84	100.99
12.	101.5	101.26	101.43	100.9	**101.63**	101.46	101.01	100.32	100.48
13.	99.96	99.65	103.66	102.14	**103.45**	99.96	100.78	99.69	101.09
14.	99.48	103.57	**105.87**	102.61	102.26	99.57	99.91	99.22	99.13
15.	95.22	101.28	95.79	**102.27**	96.95	94.64	94.97	95.13	94.6
16.	99.91	100.65	**104.17**	102.86	102.52	104.6	99.91	99.22	100.22
17.	100.35	101.22	101.92	103.62	103.4	100.35	**113.79**	99.78	106.11
18.	**93.33**	81.21	81.19	83.44	83.03	82.82	83.17	82.93	83.1
19.	106.93	123.1	**123.15**	120.24	120.45	120.33	120.36	120.19	120.3
20.	**92.98**	80.72	80.72	82.82	82.82	82.82	82.82	82.83	82.82
21.	**92.9**	80.41	80.24	82.47	82.52	82.31	82.67	82.48	82.3
22.	99.78	108.41	**115.46**	105.43	106.73	112.19	104.78	112.74	113.9
23.	112.19	111.12	111.24	111.75	111.23	110.68	111.82	112.3	**112.43**
24.	**91.96**	80.59	81.76	83.75	82.99	82.91	82.48	83.14	82.37
25.	106.63	**122.95**	122.93	119.97	119.97	119.98	119.93	119.93	119.95
26.	105.32	120.84	**121.46**	117.49	117.94	118.62	117.6	117.83	117.91
27.	105.33	120.89	**121.47**	117.5	117.95	118.69	117.6	117.83	117.92
28.	96.12	96.75	90.89	93.09	91.07	92.85	**101.44**	98.42	98.53
29.	96.15	96.8	88.16	93.15	91.17	90.66	**98.61**	95.62	95.71
30.	96.13	96.75	90.88	93.09	91.08	92.84	**101.45**	98.41	98.53
Tot.	100.28	98.2	97.48	101.92	102.69	**103.13**	102.9	102.37	102.56

En conclusion, les *profils de performances* retenus pour la *Phase 3 : évaluation en cluster* sont : premièrement, le profil par défaut [Tab.11] des noeuds, sans qu'un aucune optimisation ne soit apportée ; deuxièmement, le profil optimisé résultant du point *3)* des *Optimisations GPU*, car il est celui ayant présenté les meilleures performances ; troisièmement, un profil intermédiaire, plus stable, qui vise un équilibre entre les performances et la longévité des noeuds.

Les paramètres de chacun de ces trois profils sont les suivants [Tab.22]:

Table 22: Paramètres des profils de la Phase 3 : évaluation en cluster

Paramètre	Profil par défaut	Profil intermédiaire	Profil optimisé
arm_freq	1500	1800	2100*
arm_freq_min	600	1800	2100*
gpu_freq	500	500	750
over_voltage	0	3	6

***** : la fréquence des noeuds n°5, n°6 et n°7 [Tab.2] reste bridée à 1900MHz pour les raisons exprimées à la *conclusion)* du point *Optimisations CPU*.

8.3 Phase 3 : évaluation en cluster

Les objectifs de la *Phase 3 : évaluation en cluster* sont : premièrement, l'exécution d'HPL en cluster sur les 8 noeuds ; deuxièmement, l'évaluation des trois profils créés à la suite de la Phase 2 : optimisations ; troisièmement, la comparaison entre les résultats acquis et ceux présentés dans la section *Étude bibliographique*.

8.3.1 Fonctionnement HPL

Annoncé dans le chapitre *Technologies*, HPL est un logiciel qui résout un système linéaire dense (aléatoire) en arithmé-tique de double précision (64 bits) sur des ordinateurs à mémoire distribuée. Il peut donc être considéré comme une implémentation portable et librement disponible du *benchmark* Linpack pour le calcul haute performance.[23]

Lors de son utilisation, HPL nécessite d'être configuré au travers de plusieurs paramètres tels que N, NB, P et Q.

- N définit la taille du problème (sous forme d'une matrice) à résoudre. Dans [18], il est dit que la taille du problème doit être la plus large possible, tout en s'alignant avec la mémoire RAM disponible.

  N se calcule grâce à la formule suivante [18] :

$$N_{max} = \sqrt{\frac{Mémoire\_en\_GB * 1024^3 * nb\_noeuds}{8}} * Z$$

  où Z représente le coefficient de réduction : 80% dans le cas présent.

- NB définit la taille de blocs utilisée pour la distribution des données. Dans [18], il est dit que les valeurs de NB qui fournissent de bons résultats se trouvent dans la tranche [96 - 256] par incrément de 8. Pour NB : une valeur trop faible réduit les performances ; une valeur trop forte déséquilibre la distribution des données.
- P représente le nombre de lignes de processus dans la matrice PxQ.
- Q représente le nombre de colonnes de processus dans la matrice PxQ.

Quant aux résultats fournis par HPL, ils sont annoncés en GFLOPS, en milliards d'opérations à virgule flottante par seconde.

Finalement, pour faciliter la configuration d'HPL, le site web advancedclustering [49] a été utilisé. La configuration générée par le site se montre fiable et cohérente avec la formule de N présentée ci-dessus.

8.3.2 Évaluation des profils d'optimisation

Les trois profils cités dans [Tab.22] ont été évalués tour à tour en utilisant une même configuration pour HPL, et ce, afin que les résultats ne soient affectés que par les profils.

La configuration du fichier *HPL.dat* est obtenue en entrant les paramètres suivants sur advancedclustering : nombre de noeuds = 8 ; nombre de coeurs = 4 ; mémoire par noeud = 3280MB (80%) ; taille de blocs (NB) = 192.

1) Le résultat généré par l'exécution d'HPL sur le cluster avec les paramètres *arm_freq=1500*, *arm_freq_min=600*, *gpu_freq=500* et *over_voltage=0* (profil par défaut [Tab.22]), est le suivant [Tab.23]:

Table 23: Résultat HPL pour le profil par défaut

N	NB	P	Q	Temps (sec)	GFLOPS
52224	192	4	8	2180.7	43.54

2) Lors de l'exécution d'HPL avec les paramètres *arm_freq=1800*, *arm_freq_min=1800*, *gpu_freq=500* et *over_voltage=3* (profil intermédiaire [Tab.22]), plusieurs noeuds se sont éteints : noeud n°5 lors de la première itération ; noeud n°7 lors de la deuxième itération ; noeuds n°5, n°6 et n°7 lors de la troisième itération. Le résultat est donc nul.

3) L'exécution d'HPL avec les paramètres *arm_freq=2100*, *arm_freq_min=2100*, *gpu_freq=750* et *over_voltage=6* (profil optimisé [Tab.22]) fournit le même résultat qu'au point *2)*. Les noeuds n°5, n°6 et n°7 s'éteignent. Le résultat est donc nul.

Suite à ces résultats, les noeuds n°5, n°6 et n°7 (kit Labists) sont retirés du cluster afin de tester l'exécution d'HPL uniquement sur les noeuds du kit Xute. Une nouvelle configuration du fichier *HPL.dat* est générée avec les paramètres suivants : nombre de noeuds = 5 ; nombre de coeurs = 4 ; mémoire par noeud = 3280MB (80%) ; taille de blocs (NB) = 192.

1) Le résultat généré par l'exécution d'HPL avec les paramètres *arm_freq=1500*, *arm_freq_min=600*, *gpu_freq=500* et *over_voltage=0* est le suivant [Tab.24] :

Table 24: Résultat HPL pour le profil par défaut avec 5 noeuds

N	NB	P	Q	Temps (sec)	GFLOPS
41088	192	4	5	1873.17	24.68

2) Le résultat généré par l'exécution d'HPL avec les paramètres *arm_freq=1800*, *arm_freq_min=1800*, *gpu_freq=500* et *over_voltage=3* est le suivant [Tab.25] :

Table 25: Résultat HPL pour le profil intermédiaire avec 5 noeuds

N	NB	P	Q	Temps (sec)	GFLOPS
41088	192	4	5	1845.88	25.05

3) Lors de l'exécution d'HPL avec les paramètres *arm_freq=2100*, *arm_freq_min=2100*, *gpu_freq=750* et *over_voltage=6*, le noeud n°3 s'est éteint. Le résultat est donc nul.

8.3.3 Étude des paramètres

Afin de compléter l'évaluation en cluster, et pour récolter plus de donnée sur les capacités du cluster, des tests supplémentaires sont exécutés en analysant : premièrement, l'évolution des performances en GFLOPS pour un NB et un taux d'utilisation de la RAM statiques, en faisant évoluer le nombre de noeuds ; deuxièmement, l'évolution des performances en GFLOPS sur les 8 noeuds, mais en faisant varier le taux d'utilisation de la RAM.

1) Les résultats générés pour un NB de 128 et une utilisation de la RAM de 80% en faisant varier la taille du cluster sont les suivants [Tab.26] :

Table 26: Résultats HPL pour NB=128 et RAM=80% en variant le nombre de noeuds

nb. noeud	N	Temps	RAM	GFLOPS	Taux
1	18432	426.8	80%	9.8	100%
2	26112	756.6	80%	15.6	159.18%
4	36864	1337	80%	25	255.10%
6	45312	2056	80%	30	306.12%
8	52224	2253	80%	42	428.57%

Dans le [Tab.26], les valeurs fournies par advancedclustering doivent maximiser la taille du problème N en fonction de la quantité de mémoire RAM disponible, du taux d'utilisation de RAM alloué et du nombre de noeuds.

Dans le chapitre *Résultats*, [22] informait sur l'importance d'avoir une taille de problème élevée afin d'obtenir des résultats linéaires et cohérents, car une taille de problème basse implique un accroissement des communications internoeuds et une réduction des performances obtenues.

Ainsi, en appliquant strictement la formule de *N max* donnée au point *Fonctionnement HPL*, les valeurs de N doivent correspondre aux valeurs suivantes.

nb. noeud	N advancedclustering	N formule	Taux
1	18432	18536.38	100.57%
2	26112	26214.4	100.39%
4	36864	37072.76	100.57%
6	45312	45404.6727	100.2%
8	52224	52428.8	100.39%

Les valeurs de N obtenues par l'application de la formule sont légèrement supérieures à celles utilisées. Cependant, bien qu'annoncées avec 4GB, les RPI disposent souvent de moins de mémoire RAM. Réaliser les tests avec un N légèrement plus petit permet alors d'éviter les risques de saturation de la RAM ainsi que les erreurs lors de l'exécution d'HPL.

Ensuite, les résultats du [Tab.26] mettent en évidence la linéarité des clusters avancée dans [18], [19], [20] et [22]. Cette linéarité s'exprime au travers de : l'évolution des performances en GFLOPS en fonction de la taille du problème N [Fig.23] ; l'évolution du temps de résolution en fonction de la taille du problème N [Fig.24] ; l'évolution des performances en GFLOPS en fonction du nombre de noeuds [Fig.25].

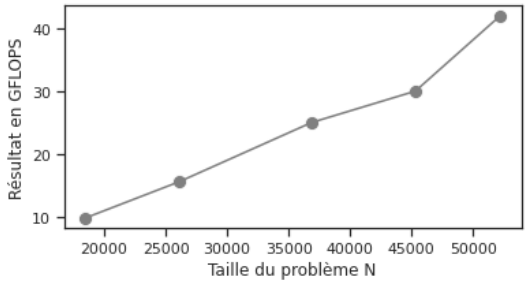


Figure 23: Évolution des performances en fonction de la taille du problème N

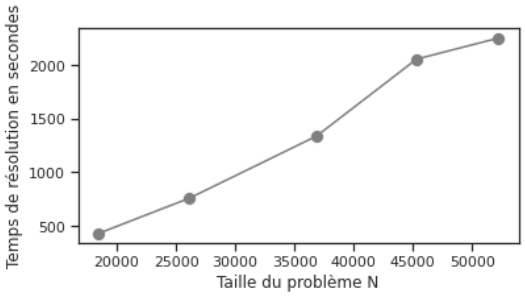


Figure 24: Évolution du temps de résolution en fonction de la taille du problème N

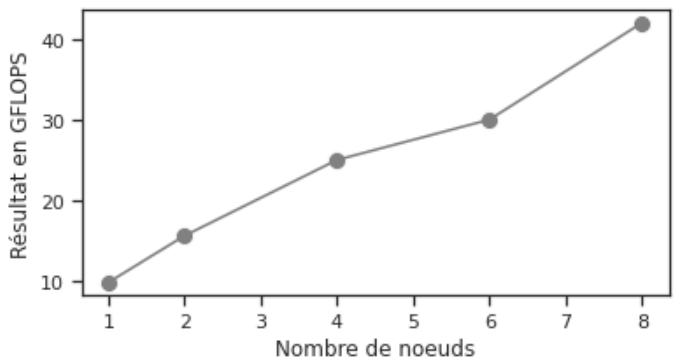


Figure 25: Évolution des performances fonction du nombre de noeuds

Ces données, superposées dans un même graphe [Fig.26], peuvent d'ailleurs être comparées aux résultats [Fig.27] présentés dans [18].

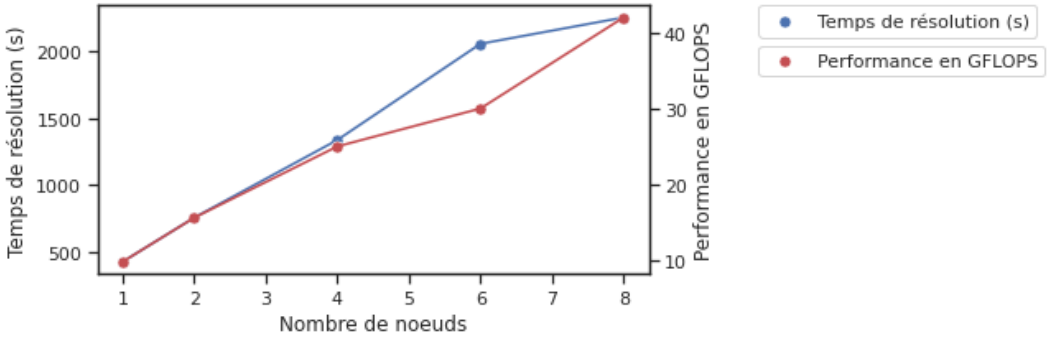


Figure 26: Évolution du temps de résolution et des performances en fonction du nombre de noeuds

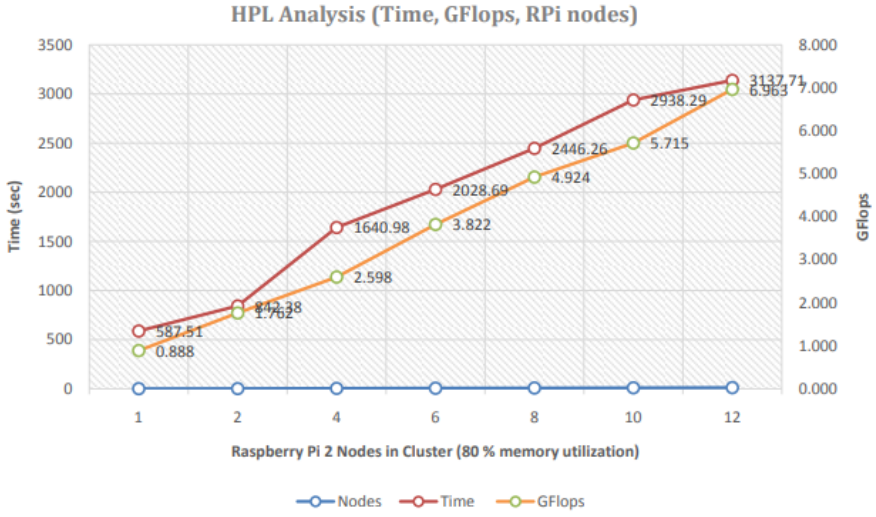


Figure 27: [18] HPL cluster performances results in GFLOPS with 80% system memory utilization (4 threads per node)

2) Les résultats générés pour un NB de 128 et pour 8 noeuds en faisant varier le taux d'utilisation de la RAM sont les suivants [Tab.27] :

Table 27: Résultats HPL pour NB=128 en variant le taux d'utilisation de la RAM

nb. noeud	N	Temps	RAM	GFLOPS	Taux
8	52224	2253	80%	42	100%
8	54016	2423	85%	43.34	103.19%
8	55552	2572	90%	44.43	105.78%

Augmenter le taux d'utilisation de la RAM permet d'allouer plus de ressources à HPL faisant, de facto, augmenter la taille du problème N. Cette opération permet de maximiser les performances du cluster, tout en prenant le risque d'atteindre le seuil limite de mémoire RAM supportable par les RPI.

Cependant, les noeuds du cluster ne disposant d'aucun autre logiciels, seul la RAM nécessaire au fonctionnement du système d'exploitation est utilisée. Cela correspond à approximativement 150MB sur les 3.8GB disponibles. Ainsi, le cluster a parfaitement supporté une augmentation de 10% supplémentaire au taux d'utilisation de la RAM.

Cette optimisation de la taille du problème N permet d'obtenir un gain de performances supplémentaire de **5.8%** lors de l'exécution d'HPL avec les 8 noeuds.

8.3.4 Analyse supplémentaire

Les résultats exposés lors de l'*Étude des paramètres* peuvent être comparés à ceux présentés dans [18]. Pour rappel, l'étude réalisée dans [18] évaluait les performances d'un *Beowulf cluster* composé de RPI 2 model B, et ce, grâce à l'exécution d'HPL.

Les résultats peuvent donc être directement comparés afin de calculer le gain de performances entre le RPI 2 model B et le RPI 4 model B. Ces résultats sont les suivants [Tab.28] :

Table 28: Tableaux de comparaison des résultats fournis par HPL avec [18] entre les RPI 2 model B et RPI 4 model B

nb. noeuds	perf. RPI2 (GFLOPS)	perf. RPI4 (GFLOPS)	Taux
1	0.888	9.8	1103.6%
2	1.762	15.6	885.36%
4	2.598	25	962.28%
6	3.822	30	784.93%
8	4.924	42	852.97%

Pour un RPI seul, il est très prometteur de voir qu'entre 2015 (RPI 2) et 2019 (RPI 4) un gain de performances d'approximativement **1000%** est observé. Quant aux résultats pour 8 noeuds, ils sont tout aussi impressionnants avec un gain de performances de **750%**.

Ainsi, un RPI 4 model B (ligne rouge) seul permet d'obtenir des performances plus de deux fois plus élevées qu'un cluster de 8 RPI 2 model B (ligne bleue). Ces performances sont visibles sur la [Fig.28].

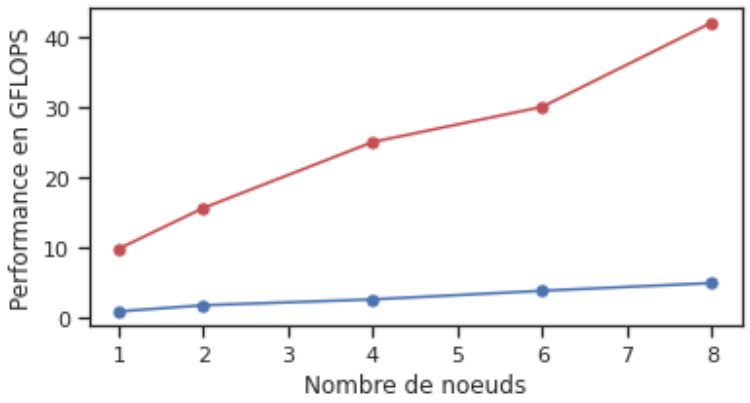


Figure 28: Comparaison des performances en fonction du nombre de noeuds [18]

8.3.5 Observations et conclusions

Dans le point *Évaluation des profils d'optimisation*, les résultats démontrent que :

1. les divergences de matériel présentes au sein du cluster empêchent HPL de s'exécuter correctement sans rencontrer d'erreurs ;
2. en restreignant le cluster aux 5 noeuds du kit Xute, le profil intermédiaire s'exécute correctement, bien que fournissant des résultats peu différents du profil par défaut. Pour rappel, les résultats présentés au point 1) des *Optimisations CPU* affichaient un gain de **1.54%** pour la métrique CPU et un gain de **5.04%** pour la métrique CPU sans GPU. Quant au gain calculé dans ce point, il est de **1.49%**, ce qui correspond, bien qu'un peu à la baisse ;
3. même en retirant les noeuds n°5, n°6 et n°7, plus favorables à subir des erreurs et à s'éteindre, une fréquence trop élevée (2100MHz) entraîne un échec de l'exécution d'HPL. Ainsi, bien que les noeuds aient réussi à exécuter la procédure d'évaluation lors de la *Phase 1 : évaluation individuelle*, ces échecs témoignent de l'instabilité subie par les RPI soumis à de trop hautes fréquences.

Dans le point *Étude des paramètres*, les résultats démontrent que :

1. l'évolution des performances d'un cluster de RPI visible dans [Fig.23], [Fig.24], [Fig.25], [Fig.26] et calculée grâce à HPL tend à être linéaire, appuyant les observations réalisées dans [18], [19], [20] et [22].
2. l'ajout d'un noeud supplémentaire au cluster permet d'augmenter ses performances en GFLOPS d'approximativement **8 GFLOPS**.
3. augmenter le taux d'utilisation de la RAM jusqu'à 90%, maximisant ainsi la taille du problème, permet d'obtenir un gain de **5.8%** des performances sur 8 noeuds, et ce, sans subir d'erreurs. Cela équivaut à un gain supplémentaire de **24.8%** des performances d'un RPI et permet d'atteindre **453.4%** des performances calculées en *single node*.

Dans le point *Analyse supplémentaire*, les résultats démontrent que :

1. en seulement 4 ans, l'évolution des RPI leur permet de passer de 0.8 GFLOPS à presque 10 GFLOPS, pour un gain de performances de près de **1000%**.
2. l'augmentation des performances résultant de l'ajout de nouveaux noeuds au cluster se montre de plus en plus conséquente, tout en conservant la linéarité déjà démontrée par les précédents modèles.

9 Réalisations futures

La section *Réalisations futures* met en évidence des pistes qui n'ont pas pu être abordées dans le présent travail, mais qui peuvent venir le compléter ou l'étendre au travers de futurs travaux. Il en existe, cependant, beaucoup d'autres.

Ces pistes sont les suivantes :

- Dans les chapitres *Analyse des données* et *Comparaison des résultats*, les résultats prouvent que des variations assez conséquentes dans les performances des RPI sont causées par le choix de la carte SD et par les différents kits. Un comparatif de ces derniers peut alors apporter des réponses quant aux gains de performances envisageables entre du matériel de "bonne" et de "mauvaise" facture, tout en considérant la répercussion sur le coût du kit.
- Dans le chapitre *Évaluation des profils d'optimisation*, les résultats ne sont pas concluants quant aux profils intermédiaire et optimisé. Trouver la cause exacte l'incapacité à exécuter HPL et trouver le seuil auquel les erreurs commencent à apparaître peut permettre de maximiser les performances du cluster et de mieux comprendre les conséquences introduites par des divergences de matériel.
- Dans les chapitres *Optimisations système* et *Optimisations RAM*, l'optimisation du système d'exploitation est abordée. Cependant, très peu de performances sont retirées des optimisations réalisées dans ce chapitre. Étudier les divergences de performances entre les différents systèmes d'exploitation disponibles sur RPI ainsi que les optimisations pouvant y être apportées représente une piste d'intérêt.
- Dans le chapitre *Étude des paramètres*, la variation du nombre de noeuds et celle du taux d'utilisation de la RAM sont exposées. D'autres paramètres d'HPL peuvent cependant être modifiés et analysés afin d'obtenir une vue complète des gains et/ou pertes de performances occasionnés par ces derniers.
- Dans le chapitre *Analyse supplémentaire*, il est très prometteur de voir que les performances des RPI ont drastiquement évolué au cours des dernières années. Établir un comparatif complet entre les récents modèles et les plus anciens peut alors être envisagé afin de représenter l'évolution des performances des processeurs ARM employés.
- Dans le chapitre *Conditions de test*, le refroidissement des RPI est abordé. Celui-ci est, dans le cas présent, pris en charge par les boîtiers fournis par les kits Xute et Labists. Cependant, dans le chapitre *Montage des RPI*, [38] met en garde contre la hausse des températures mesurées entre les RPI 4 et les précédents modèles. Ainsi, étudier les gains et/ou pertes de performances sous différentes conditions peut amener des réponses quant aux conditions optimales pour un cluster donné.

10 Conclusion

1) L'*Étude bibliographique* a permis de mettre en évidence les points d'intérêt courant dans la création d'un cluster : l'agencement du réseau au travers de l' *attribution des IP*, des *communications entre les noeuds* et de l'accès au *stockage partagé* ; mais aussi, les méthodes d'évaluation des clusters au travers des chapitres *technologies* et *Résultats*.

Ces résultats s'accordaient sur la forte linéarité des performances des clusters de RPI et les performances prometteuses de ces dernières, bien que les tests, le matériel et le nombre de noeuds soient différents dans chaque étude. Les conclusions tirées par les auteurs étaient les suivantes : même les problèmes hautement parallélisables peuvent perdre l'immense avantage qu'ils ont en matière de rapidité lorsque la puissance de traitement nécessaire est atteinte. [17] ; les résultats témoignent de performances bien plus faibles que d'autres superordinateurs, mais ils démontrent aussi une mise à l'échelle similaire à ces derniers. [20] ; (HPL) plus la taille du problème est élevée, plus les performances dégagées tentent à être linéaires. [22]

2) La section *Évaluation* a permis de définir, en accord avec la *méthodologie* présentée dans le chapitre *Benchmarking*, les différents points clés des évaluations tels que : les critères d'efficience et d'efficacité (*la métrique CPU*) ; le système d'indicateurs clés (*les 31 attributs*) ; les actifs évalués (*les 9 RPI*) ; les mesures à entreprendre (*la méthodologie en 3 phases*) ; la méthode de comparaison des actifs (*les profils de performances*, *le système de scores*) ; les mesures de rééquilibrage (*la phase de validation*) ; les mesures d'optimisations (*la phase d'optimisation*) ; le contrôle des résultats (*la phase d'évaluation en cluster*).

3) La section *Implémentation* a porté l'emphase sur la création de la procédure d'implémentation du cluster, en ayant comme objectif l'automatisation des tâches ainsi que la simplicité à redéployer et à étendre ce dernier.

Pour ce faire, la *configuration réseau* a été grandement simplifiée par l'attribution automatique des IP par DHCP et l'étape de *réplication des cartes SD* a permis de gagner un temps considérable dans l'installation des noeuds.

Les actions prises sur le cluster, donc sur l'ensemble des noeuds, ont ensuite été exclusivement réalisées par l'utilisation d'Ansible et d'HPL. Au travers de quelques *playbooks*, Ansible automatise complètement, sur les 9 noeuds en simultané : l'*installation des outils* ; la *configuration du stockage partagé* ; l'*exécution des tests individuels* ; le changement des profils d'optimisation ; la modification des paramètres HPL ; l'*installation de Jupyter notebook*.

4) La phase d'*évaluation individuelle* a démontré que bien que les RPI ne diffèrent que sur peu de points (Kit et carte SD), leurs quelques différences entraînent des écarts de performances considérables (approximativement **16%**) entre les noeuds. Cela témoigne aussi de l'importance de trouver des fournisseurs de confiance. Bien que le matériel semble similaire sur le papier, il a été prouvé que, dépendamment de la métrique, des écarts de performances atteignant jusqu'à **20%** peuvent être observés.

La phase d'*optimisation* a démontré que les différences de matériel ont eu un impact considérable sur les performances. Certains noeuds n'ont pas réussi à atteindre les optimisations désirées et ont même cessé de fonctionner. Cela amène à considérer les fortes instabilités rencontrées par les RPI lorsque les paramètres d'optimisations dépassent ceux recommandés par le constructeur.

Les gains de performances, eux, sont de l'ordre de seulement quelques pourcents (**3%**) pour les performances globales, mais, dépendamment de la métrique étudiée, les performances peuvent atteindre des gains d'approximativement **15%** à **20%**. Des gains qui sont, alors, non négligeables si le cluster doit être utilisé pour une tâche bien spécifique alignée avec la bonne métrique.

La phase d'*évaluation en cluster* a démontré que malgré les tentatives d'instaurer une stabilité dans le cluster, les RPI disposant d'un profil d'optimisation n'ont pas réussi à supporter les tests d'HPL, et des erreurs ont mis fin à l'ensemble de ces tests. Avec les profils par défaut, par contre, les résultats obtenus par HPL ont appuyé les observations des auteurs cités dans l'étude bibliographique et une linéarité comparable à celles présentées a été observée.

Finalement, les *Réalisations futures* mettent un terme à cette étude et permettent de faire le point sur les principales pistes qui restent à explorer, bien qu'il y en existe beaucoup d'autres.

A Annexes

A.1 Codes Jupyter

```
import warnings
import pandas as pd

warnings.filterwarnings('ignore')

# Init des noeuds
benchmark = "benchmark_c91f6c5a-0129-11ed-aca3-f9fed7356354_ZRAM"

!ls
    /home/pi/cluster_shared/benchmark_c91f6c5a-0129-11ed-aca3-f9fed7356354_ZRAM
    > nodes.txt

nodes = []
with open("nodes.txt", encoding = 'utf-8') as f:
    nodes = f.read().splitlines()
nodes

# Cr ation du dataframe contenant l'ensemble des donn es.
dataframe = pd.DataFrame(columns = ['Host',
                                'Compression R/U Rate single-thread
                                    (MIPS)', 'Decompression R/U Rate
                                    single-thread (MIPS)',
                                'Compression R/U Rate multi-thread
                                    (MIPS)', 'Decompression R/U Rate
                                    multi-thread (MIPS)',
                                'Compression R/U Rate hyper-threading
                                    (MIPS)', 'Decompression R/U Rate
                                    hyper-threading (MIPS)',
                                'Local write speed 256 (MB/s)', 'Local
                                    write speed 512 (MB/s)', 'Local write
                                    speed 1024 (MB/s)',
                                'Cached reads speed (MB/s)', 'Buffered
                                    disk reads speed (MB/s)',
                                'O_DIRECT cached reads speed (MB/s)',
                                    'O_DIRECT disk reads (MB/s)',
                                'bandwith 8 (Mb/s)', 'bandwith 64 (Mb/s)',
                                    'bandwith 256 (Mb/s)', 'bandwith 512
                                    (Mb/s)', 'bandwith 1024 (Mb/s)',
                                'Blowfish Score', 'CryptoHash Score',
                                    'Fibonacci Score', 'Zlib Score',
                                'N-Queens Score', 'FFT Score', 'Raytracing
                                    Score',
                                'Total events CPU', 'Total operations
                                    Memory', 'Mean transfer speed Memory
                                    (MB/s)',
                                'Total operations IO per second (op/s)',
                                    'Mean transfer speed IO (MB/s)',
                                    'Total events IO'
                                 ])

for node in nodes:

    folder = f"/home/pi/cluster_shared/{benchmark}/{node}/"

    # Cr ation du profil complet d'un noeud

    # Partie Sysbench CPU
    infos = []
```

```
    with open(f"{folder}sysbench_cpu.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    total_events = infos[19].split(" ")[-1]

    sysbench_cpu_profile = [total_events]
#     print(sysbench_cpu_profile)

    # Partie Sysbench M moire
    infos = []
    with open(f"{folder}sysbench_memory.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    total_operations = infos[18].split(" ")[-4]
    transfer_speed = infos[20].split(" ")[-2].strip('(')

    sysbench_memory_profile = [total_operations, transfer_speed]
#     print(sysbench_memory_profile)

    # Partie Sysbench FileIO
    infos = []
    with open(f"{folder}sysbench_fileio.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    total_operations = float(infos[25].split(" ")[-1]) +
        float(infos[26].split(" ")[-1]) + float(infos[27].split("
        ")[-1])
    mean_transfer_speed = (float(infos[30].split(" ")[-1]) +
        float(infos[31].split(" ")[-1])) / 2
    total_events = infos[35].split(" ")[-1]

    sysbench_IO_profile = [total_operations, mean_transfer_speed,
        total_events]
#     print(sysbench_IO_profile)

    # Partie 7Zip compression
    infos = []
    with open(f"{folder}7z.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    compressiong_rate_st = [ele for ele in [x.strip(' ') for x in
        infos[62].split(" ")] if ele.strip()][2]
    decompression_rate_st = [ele for ele in [x.strip(' ') for x in
        infos[62].split(" ")] if ele.strip()][3]
    compressiong_rate_mt = [ele for ele in [x.strip(' ') for x in
        infos[119].split(" ")] if ele.strip()][2]
    decompression_rate_mt = [ele for ele in [x.strip(' ') for x in
        infos[119].split(" ")] if ele.strip()][3]
    compressiong_rate_ht = [ele for ele in [x.strip(' ') for x in
        infos[176].split(" ")] if ele.strip()][2]
    decompression_rate_ht = [ele for ele in [x.strip(' ') for x in
        infos[176].split(" ")] if ele.strip()][3]

    seven_profile = [compressiong_rate_st, decompression_rate_st,
                     compressiong_rate_mt, decompression_rate_mt,
                     compressiong_rate_ht, decompression_rate_ht]
    # print(seven_profile)

    # Partie hardinfo
    infos = []
    with open(f"{folder}hardinfo.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()
```

```
    blowfish = infos[1].split(";")[2][8:]
    cryptohash = infos[3].split(";")[2][8:]
    fibonacci = infos[5].split(";")[2][8:]
    nqueens = infos[7].split(";")[2][8:]
    zlib = infos[9].split(";")[2][8:]
    fft = infos[11].split(";")[2][8:]
    raytracing = infos[13].split(";")[2][8:]

    hardinfo_profile = [blowfish, cryptohash, fibonacci, nqueens,
                        zlib, fft, raytracing]
#     print(hardinfo_profile)

    # Partie DD
    infos = []
    with open(f"{folder}dd.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    local_256_speed = infos[2].split(",")[-1][1:-5]
    local_512_speed = infos[5].split(",")[-1][1:-5]
    local_1024_speed = infos[8].split(",")[-1][1:-5]

    dd_profile = [local_256_speed, local_512_speed, local_1024_speed]
#     print(dd_profile)

    # Partie hdparm
    infos = []
    with open(f"{folder}hdparm.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    cached_read_speed = infos[2].split(" ")[-2]
    buffered_read_speed = infos[3].split(" ")[-2]
    O_DIRECT_cached_read_speed = infos[6].split(" ")[-2]
    O_DIRECT_disk_read_speed = infos[7].split(" ")[-2]

    hdparm_profile = [cached_read_speed, buffered_read_speed,
                      O_DIRECT_cached_read_speed,
                         O_DIRECT_disk_read_speed]
#     print(hdparm_profile)

    # Partie Iperf
    infos = []
    with open(f"{folder}iperf.txt", encoding = 'utf-8') as f:
        infos = f.read().splitlines()

    bandwith_8 = infos[6].split(" ")[-2]
    bandwith_64 = infos[13].split(" ")[-2]
    bandwith_256 = infos[20].split(" ")[-2]
    bandwith_512 = infos[27].split(" ")[-2]
    bandwith_1024 = infos[34].split(" ")[-2]

    iperf_profile = [bandwith_8, bandwith_64, bandwith_256,
        bandwith_512, bandwith_1024]
#     print(iperf_profile)

#     # Partie finale

    complete_profile = [node] + seven_profile + dd_profile +
        hdparm_profile + iperf_profile + hardinfo_profile +
        sysbench_cpu_profile + sysbench_memory_profile +
        sysbench_IO_profile

    df_temp = pd.DataFrame([complete_profile],  columns = ['Host',
                               'Compression R/U Rate single-thread
```

```
                                (MIPS)', 'Decompression R/U Rate
                                single-thread (MIPS)',
                            'Compression R/U Rate multi-thread
                                (MIPS)', 'Decompression R/U Rate
                                multi-thread (MIPS)',
                            'Compression R/U Rate hyper-threading
                                (MIPS)', 'Decompression R/U Rate
                                hyper-threading (MIPS)',
                            'Local write speed 256 (MB/s)', 'Local
                                write speed 512 (MB/s)', 'Local write
                                speed 1024 (MB/s)',
                            'Cached reads speed (MB/s)', 'Buffered
                                disk reads speed (MB/s)',
                            'O_DIRECT cached reads speed (MB/s)',
                                'O_DIRECT disk reads (MB/s)',
                            'bandwith 8 (Mb/s)', 'bandwith 64 (Mb/s)',
                                'bandwith 256 (Mb/s)', 'bandwith 512
                                (Mb/s)', 'bandwith 1024 (Mb/s)',
                            'Blowfish Score', 'CryptoHash Score',
                                'Fibonacci Score', 'Zlib Score',
                            'N-Queens Score', 'FFT Score', 'Raytracing
                                Score',
                            'Total events CPU', 'Total operations
                                Memory', 'Mean transfer speed Memory
                                (MB/s)',
                            'Total operations IO per second (op/s)',
                                'Mean transfer speed IO (MB/s)',
                                'Total events IO'
                             ])
    dataframe = pd.concat([dataframe, df_temp], ignore_index=True)

# Export des donn es vers un fichier Excel

filename = "benchmark_stats.xlsx"

dataframe.to_excel(filename, index=False)
df_excel = pd.read_excel(filename)
df_excel

# Calcul de la moyenne par attribut
df_mean = pd.DataFrame([df_excel.mean()])
df_mean

# Graphe des attributs par profil
import matplotlib.pyplot as plt
import seaborn as sns

for i, column in enumerate(df_excel.columns[1:]):
    fig = plt.figure(i)
    sns.barplot(x=df_excel.index, y=df_excel[column], palette="crest")
    fig.savefig(f"pics/all_{i}.png")

# Remise en pourcentage
df_perc = df_excel.copy(deep=True)

if 'Host' in df_perc.columns:
     df_perc.pop('Host')

for i in df_perc.index:
    if i != 0:
        for x, column in enumerate(df_perc.columns):
            if df_perc.at[i,column] != df_perc.at[0,column] :
                df_perc.at[i,column] = df_perc.at[i,column] * 100 /
```

```
                    df_perc.at[0,column]
            else :
                df_perc.at[i,column] = 100.0

for column in df_perc.columns:
    df_perc.at[0,column] = 100

df_perc

# Graphe sous forme de benchmark
target_id = 1

pal = []
for item in df_perc.iloc[target_id]:
    if item >= 105:
        pal.append('lightgreen')
    elif item <= 95:
        pal.append('lightcoral')
    else:
        pal.append('lavender')
customPalette = sns.set_palette(pal)


graph_size = df_perc.iloc[target_id].count()
f, ax = plt.subplots(figsize=(10, graph_size/2))

# Plot the total crashes
sns.set_color_codes("pastel")
sns.barplot(x=df_perc.iloc[0], y=df_perc.columns, label="idle",
    color="lavender")

# Plot the crashes where alcohol was involved
sns.set_color_codes("muted")
sns.barplot(x=df_perc.iloc[target_id], y=df_perc.columns,
    label="optimised", palette=customPalette)
ax.bar_label(ax.containers[1], size=12, color='black',
    label_type='center')
ax.set(xlabel='Percentage')

# Calcul du score par m trique
df_cpu = df_perc.copy(deep=True)
df_gpu = df_perc.copy(deep=True)
df_memory = df_perc.copy(deep=True)
df_storage = df_perc.copy(deep=True)
df_network = df_perc.copy(deep=True)

df_cpu = df_cpu[['Compression R/U Rate single-thread (MIPS)',
    'Decompression R/U Rate single-thread (MIPS)',
                     'Compression R/U Rate multi-thread (MIPS)',
                         'Decompression R/U Rate multi-thread (MIPS)',
                     'Compression R/U Rate hyper-threading (MIPS)',
                         'Decompression R/U Rate hyper-threading
                         (MIPS)',
                     'Blowfish Score', 'CryptoHash Score', 'Fibonacci
                         Score', 'Zlib Score','N-Queens Score',
                     'FFT Score', 'Raytracing Score', 'Total events
                         CPU']]
df_memory = df_memory[['Total operations Memory', 'Mean transfer speed
    Memory (MB/s)']]
df_storage = df_storage[['Local write speed 256 (MB/s)', 'Local write
    speed 512 (MB/s)', 'Local write speed 1024 (MB/s)',
                             'Cached reads speed (MB/s)', 'Buffered disk
                                 reads speed (MB/s)',
```

```
                            'O_DIRECT cached reads speed (MB/s)',
                                'O_DIRECT disk reads (MB/s)',
                            'Total operations IO per second (op/s)',
                                'Mean transfer speed IO (MB/s)', 'Total
                                events IO']]
df_network = df_network[['bandwith 8 (Mb/s)', 'bandwith 64 (Mb/s)',
    'bandwith 256 (Mb/s)',
                             'bandwith 512 (Mb/s)', 'bandwith 1024 (Mb/s)']]
df_gpu = df_gpu[['Compression R/U Rate single-thread (MIPS)',
    'Decompression R/U Rate single-thread (MIPS)',
                        'Compression R/U Rate multi-thread (MIPS)',
                            'Decompression R/U Rate multi-thread (MIPS)',
                        'Compression R/U Rate hyper-threading (MIPS)',
                            'Decompression R/U Rate hyper-threading
                            (MIPS)',
                        'CryptoHash Score','N-Queens Score',
                        'FFT Score', 'Total events CPU']]

sum_cpu = df_cpu.sum(axis = 1)
print(f"score : {sum_cpu[target_id]}")
sum_cpu_perc = sum_cpu / 1400 * 100
print(f"percentage : {sum_cpu_perc[target_id]}")

sum_memory = df_memory.sum(axis = 1)
print(f"score : {sum_memory[target_id]}")
sum_memory_perc = sum_memory / 200 * 100
print(f"percentage : {sum_memory_perc[target_id]}")

sum_storage = df_storage.sum(axis = 1)
print(f"score : {sum_storage[target_id]}")
sum_storage_perc = sum_storage / 1000 * 100
print(f"percentage : {sum_storage_perc[target_id]}")

sum_network = df_network.sum(axis = 1)
print(f"score : {sum_network[target_id]}")
sum_network_perc = sum_network / 500 * 100
print(f"percentage : {sum_network_perc[target_id]}")

# Graphe des scores moyens par m trique
mean_score_metrics = pd.DataFrame([[sum_cpu_perc[target_id],
    sum_memory_perc[target_id], sum_storage_perc[target_id],
    sum_network_perc[target_id]]],
                                  columns = ['cpu score', 'memory
                                      score', 'storage score', 'network
                                      score'])

mean_score_metrics

f, ax = plt.subplots(figsize=(8, 6))
sns.set_theme(style="ticks")
sns.barplot(x=mean_score_metrics.columns, y=mean_score_metrics.iloc[0],
    color='lightgrey')
sns.lineplot(x=mean_score_metrics.columns, y=100, color='black')
ax.set(ylabel='Mean score of all attributs')
ax.set(ylim=(0, 150))
```

A.2 Codes Ansible

A.2.1 install.yml

```
- name: tools installation
  hosts: master
  become: yes
  tasks:
    - name: iperf installation
      apt:
        name: iperf
        state: latest

- name: tools installation
  hosts: cluster
  become: yes
  tasks:
    - name: sysbench installation
      apt:
        name: sysbench
        state: latest

    - name: hardinfo installation
      apt:
        name: hardinfo
        state: latest

    - name: p7zip-full installation
      apt:
        name: p7zip-full
        state: latest

    - name: iperf installation
      apt:
        name: iperf
        state: latest

    - name: hdparm installation
      apt:
        name: hdparm
        state: latest

    - name: uuidgen installation
      apt:
        name: uuid-runtime
        state: latest
```

A.2.2 bench.yml

```
# Cette partie Master met en place la structure de fichier n cessaire
# et d fini les variables    fournir aux noeuds.
- name: Setting up benchmark by Master
  hosts: master
  tasks:

    - name: Set directory
      shell: echo "benchmark_$(uuidgen -t)"
      register: command_output

    - name: Set directory as fact ( share )
      set_fact:
```

```
          directory: "{{ command_output.stdout }}"

      - name: Create the benchmark directory
        ansible.builtin.file:
          path: /home/$USER/cluster_shared/{{ directory }}
          state: directory
          mode: '0755'

# Cette partie lance les benchmarks sur l'ensemble des noeuds
  - name: Benchmarking tools execution on Nodes
    hosts: cluster
    #hosts: master
    vars:
      benchdir: "{{ hostvars[groups['master'][0]]['directory'] }}"
      time: 60
    tasks:
      - name: Get node infos
        shell: echo "$(ip -4 addr show eth0 | grep -oP
            '(?<=inet\s)\d+(\.\d+){3}')"
        register: nodeid

      - name: Create the node directory (if it does not exist)
        ansible.builtin.file:
          path: /home/$USER/cluster_shared/{{ benchdir }}/{{
              nodeid.stdout }}
          state: directory
          mode: '0755'

# Benchmarks sysbench
      - name: Run sysbench - cpu benchmark
        shell: sysbench cpu --cpu-max-prime=20000 --num-threads=4 run >
            /home/$USER/cluster_shared/{{ benchdir }}/{{ nodeid.stdout
            }}/sysbench_cpu.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: Run sysbench - memory benchmark
        shell: sysbench memory --num-threads=4 run >
            /home/$USER/cluster_shared/{{ benchdir }}/{{ nodeid.stdout
            }}/sysbench_memory.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: Run sysbench - fileio benchmark
        shell: sysbench fileio --file-total-size=6G prepare >
            /home/$USER/cluster_shared/{{ benchdir }}/{{ nodeid.stdout
            }}/sysbench_fileio.txt

      - name: Run sysbench - fileio benchmark
        shell: sysbench fileio --file-total-size=6G --num-threads=4
            --file-test-mode=rndrw --max-time=300 --max-requests=0 run
            > /home/$USER/cluster_shared/{{ benchdir }}/{{
            nodeid.stdout }}/sysbench_fileio.txt

      - name: Run sysbench - fileio benchmark
        shell: sysbench fileio --file-total-size=6G cleanup

      - name: Sleep
        pause:
```

```
          seconds: "{{ time | int }}"

# Benchmark hardinfo
      - name: Run hardinfo
        shell: hardinfo -f text > /home/$USER/cluster_shared/{{
            benchdir }}/{{ nodeid.stdout }}/hardinfo.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

# Benchmark 7Z
      - name: Run 7z - compression benchmark
        shell: 7z b -mm=* -mmt=* > /home/$USER/cluster_shared/{{
            benchdir }}/{{ nodeid.stdout }}/7z.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

# Benchmark iperf
      - name: launch iperf 8k
        shell: iperf -fM -c 192.113.50.101 -w 8k > iperf.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch iperf 64k
        shell: iperf -fM -c 192.113.50.101 -w 64k >> iperf.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch iperf 256k
        shell: iperf -fM -c 192.113.50.101 -w 256k >> iperf.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch iperf 512k
        shell: iperf -fM -c 192.113.50.101 -w 512k >> iperf.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch iperf 1024k
        shell: iperf -fM -c 192.113.50.101 -w 1024k >> iperf.txt

      - name: save iperf file
        shell: mv iperf.txt /home/$USER/cluster_shared/{{ benchdir
            }}/{{ nodeid.stdout }}/iperf.txt

      - name: Set directory
        shell: echo "dd_$(uuidgen -t)"
        register: command_output


# Benchmark dd
      - name: launch dd
```

```
        shell: dd if=/dev/zero of=/home/pi/{{ command_output.stdout
            }}.img bs=256M count=1 conv=fdatasync 2>> dd.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch dd
        shell: dd if=/dev/zero of=/home/pi/{{ command_output.stdout
            }}.img bs=512M count=1 conv=fdatasync 2>> dd.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

      - name: launch dd
        shell: dd if=/dev/zero of=/home/pi/{{ command_output.stdout
            }}.img bs=1G count=1 conv=fdatasync 2>> dd.txt

  # Sauvegarde
      - name: save dd file
        shell: mv dd.txt /home/$USER/cluster_shared/{{ benchdir }}/{{
            nodeid.stdout }}/dd.txt

      - name: Sleep
        pause:
          seconds: "{{ time | int }}"

# N cessite les permissions SUDO pour lancer hdparm
  - name: hdparm
    hosts: cluster
    become: yes
    tasks:
      - name: Get node infos
        shell: echo "$(ip -4 addr show eth0 | grep -oP
            '(?<=inet\s)\d+(\.\d+){3}')"
        register: nodeid

      - name: Launch hdparm
        shell: sudo hdparm -tT /dev/mmcblk0p2 > hdparm.txt

      - name: Launch hdparm
        shell: sudo hdparm -tT --direct /dev/mmcblk0p2 >> hdparm.txt

# Sauvegarde des fichiers
  - name: hdparm
    hosts: cluster
    vars:
      benchdir: "{{ hostvars[groups['master'][0]]['directory'] }}"
    tasks:
      - name: Get node infos
        shell: echo "$(ip -4 addr show eth0 | grep -oP
            '(?<=inet\s)\d+(\.\d+){3}')"
        register: nodeid

      - name: save hdparm file
        shell: mv hdparm.txt /home/$USER/cluster_shared/{{ benchdir
            }}/{{ nodeid.stdout }}/hdparm.txt

# Cleaning des fichiers par le master
  - name: Cleaning up  by Master
    hosts: master
    tasks:
```

```
    - name: Clean dd images
      shell: rm -rf /home/pi/cluster_shared/dd_*

    - name: Clean dd images
      shell: rm -rf /home/pi/test*
```

A.2.3 nfs.yml

```
- name: nfs folder cluster setup
  hosts: cluster
  become: yes
  tasks:
    - name: nfs-common installation
      apt:
        name: nfs-common
        state: latest

    - name: create the nfs directory
      ansible.builtin.file:
        path: /home/pi/cluster_shared
        owner: pi
        group: pi
        state: directory
        mode: '0766'

    - name: set share folder
      shell: mount -t nfs 192.113.50.105:/volume1/cluster/
          /home/pi/cluster_shared
      register: command_output
```

A.2.4 jupyter.yml

```
- name: Jupyter installation
  hosts: master
  become: yes
  tasks:
    - name: python3-pip installation
      apt:
        name: python3-pip
        state: latest

    - name: python3-dev installation
      apt:
        name: python3-dev
        state: latest

    - name: upgrade pip
      shell: pip3 install --upgrade pip

    - name: create the node directory (if it does not exist)
      ansible.builtin.file:
        path: /home/pi/notebook
        state: directory
        mode: '0755'

    - name: install jupyter
      shell: cd /home/pi/notebook && pip install jupyter

# You can then launch jupyter with the following command :
```

```
# cd /home/pi/notebook && jupyter notebook --ip 0.0.0.0
    --NotebookApp.token='' --NotebookApp.password=''
```

A.2.5 oc.yml

```
- name: Change config
  hosts: cluster
  become: yes
  tasks:

    - name: Copy config file from NAS
      shell: cp
          /home/pi/cluster_shared/config_2100_6_750_bt_wifi_led.txt
          /boot/firmware/config.txt
```

A.2.6 zram.yml

```
- name: ZRAM optimisations
  hosts: cluster
  become: yes
  tasks:

    - name: disable swap files
      shell: wget -O -
          https://teejeetech.com/scripts/jammy/disable_swapfile | bash

    - name: ZRAM compression installation
      apt:
        name: zram-config
        state: latest
```

A.2.7 system.yml

```
  - name: System optimisations
    hosts: cluster
    become: yes
    tasks:

      - name: apt-get update
        apt:
          update_cache: yes

      - name: full upgrade
        apt:
          upgrade: full

      - name: EEPROM installation
        apt:
          name: rpi-eeprom
          state: latest

      - name: EEPROM update
        shell: rpi-eeprom-update -a

      - name: autoremove
        shell: apt-get autoremove --purge -y snapd
            gnome-software-plugin-snap
```

```
    - name: stop snapd
      shell: apt-mark hold snapd

    - name: remove deps
      apt:
        autoremove: yes

    - name: clean deps
      apt:
        autoclean: yes
```

A.2.8 hpl.yml

```
- name: change HPL.dat config
  hosts: cluster
  become: yes
  tasks:

    - name: Copy HPL.dat file
      shell: cp /home/pi/cluster_shared/HPL.dat
          /home/pi/rpi-hpl-workdir/hpl-2.3/bin/rpi4-mpich/HPL.dat
```

A.2.9 reboot.yml

```
- name: reboot
  hosts: cluster
  become: yes
  tasks:

    - name: reboot
      shell: reboot
```

A.3 Configurations

A.3.1 Profil 1800MHz

```
arm_freq=1800
arm_freq_min=1800
```

A.3.2 Profil 2100MHz

```
arm_freq=2100
arm_freq_min=2100
```

A.3.3 Profil 2100MHz 3v

```
arm_freq=2100
arm_freq_min=2100

over_voltage=3
```

A.3.4 Profil 2100MHz 6v

```
arm_freq=2100
arm_freq_min=2100

over_voltage=6
```

A.3.5 Profil 2100MHz 6v 200MHz

```
arm_freq=2100
arm_freq_min=2100
over_voltage=6

gpu_freq=200
```

A.3.6 Profil 2100MHz 6v 500MHz

```
arm_freq=2100
arm_freq_min=2100
over_voltage=6

gpu_freq=500
```

A.3.7 Profil 2100MHz 6v 750MHz

```
arm_freq=2100
arm_freq_min=2100
over_voltage=6

gpu_freq=750
```

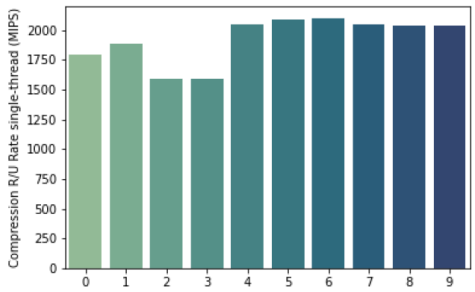
A.3.8 Profil 2100MHz 6v 750MHz I/O

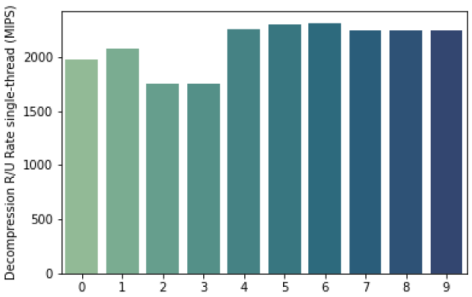
```
arm_freq=2100
arm_freq_min=2100
over_voltage=6

gpu_freq=750

dtoverlay=disable-wifi
dtoverlay=disable-bt

# Disable the PWR LED
dtparam=pwr_led_trigger=none
dtparam=pwr_led_activelow=off
# Disable the Activity LED
dtparam=act_led_trigger=none
dtparam=act_led_activelow=off
# Disable ethernet port LEDs
dtparam=eth_led0=4
dtparam=eth_led1=4
```

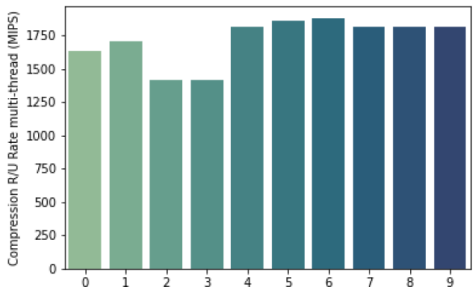
A.4 Résultats

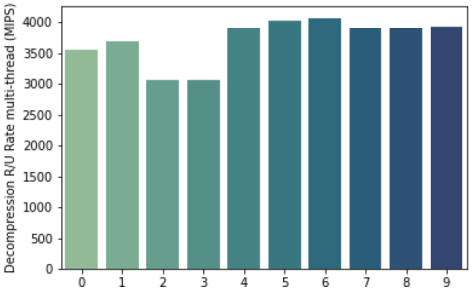
A.4.1 Graphes des scores moyens par attribut et par profil

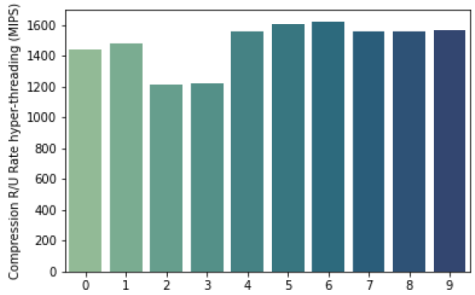
* Les 10 profils sont dans l'ordre où ils ont été présentés dans le document.

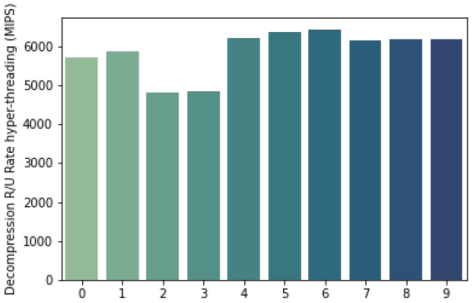


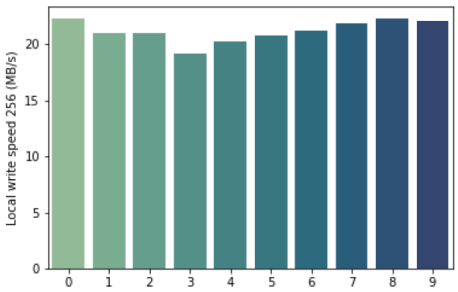


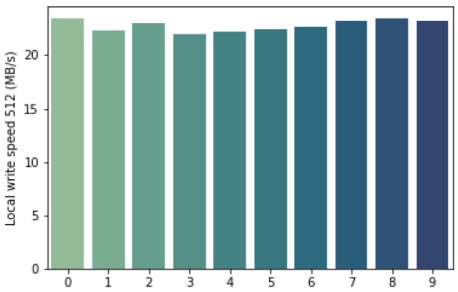


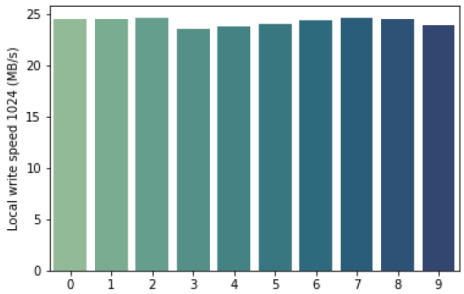


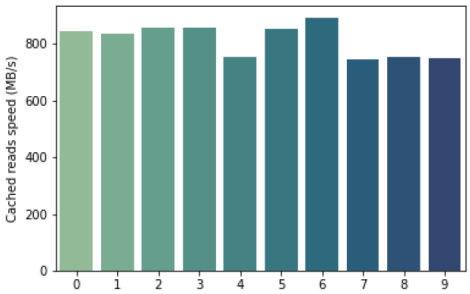


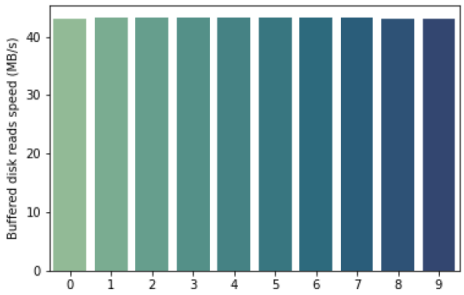


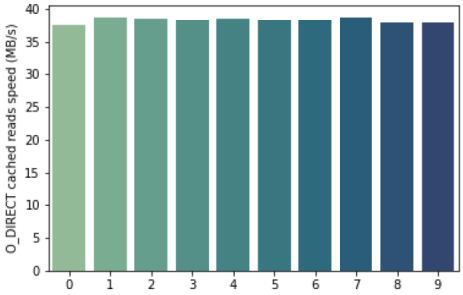


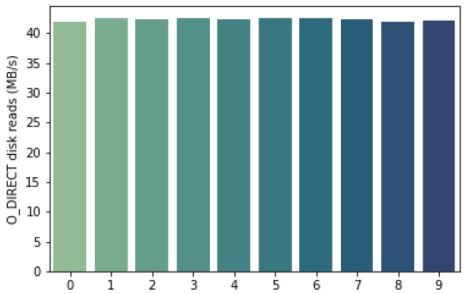


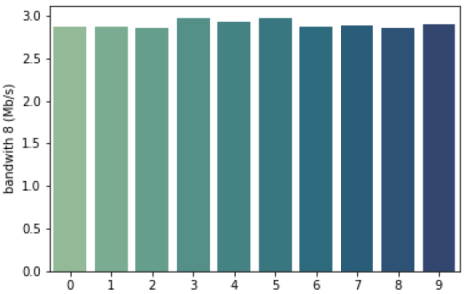

Local write speed 1024 (MB/s)
0 5 10 15 20 25
0 1 2 3 4 5 6 7 8 9

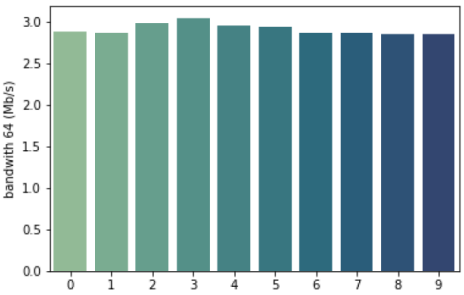

Cached reads speed (MB/s)
0 200 400 600 800
0 1 2 3 4 5 6 7 8 9

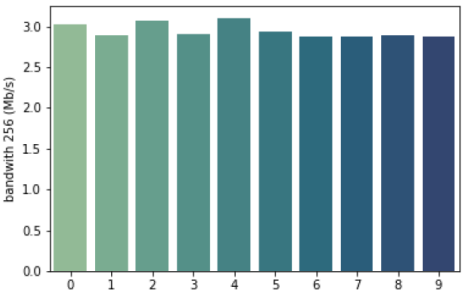

Buffered disk reads speed (MB/s)
0 10 20 30 40
0 1 2 3 4 5 6 7 8 9

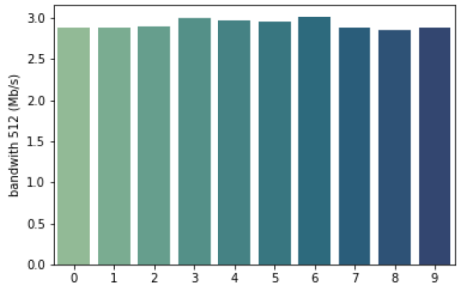

O_DIRECT cached reads speed (MB/s)
0 5 10 15 20 25 30 35 40
0 1 2 3 4 5 6 7 8 9

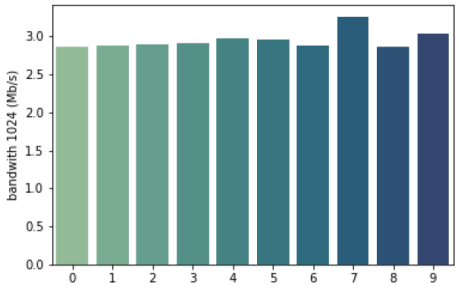

O_DIRECT disk reads (MB/s)
0 5 10 15 20 25 30 35 40
0 1 2 3 4 5 6 7 8 9

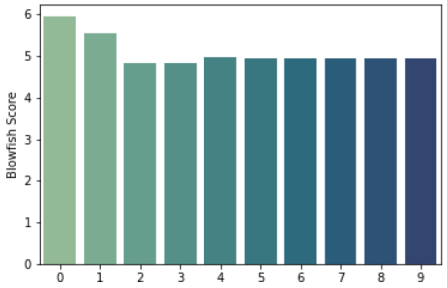

bandwith 8 (Mb/s)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 1 2 3 4 5 6 7 8 9

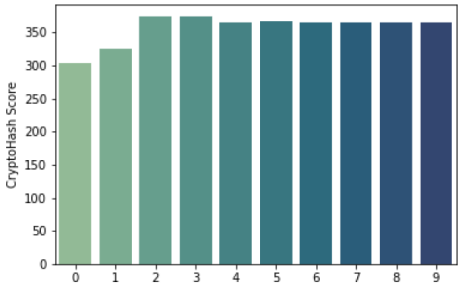

bandwith 64 (Mb/s)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 1 2 3 4 5 6 7 8 9

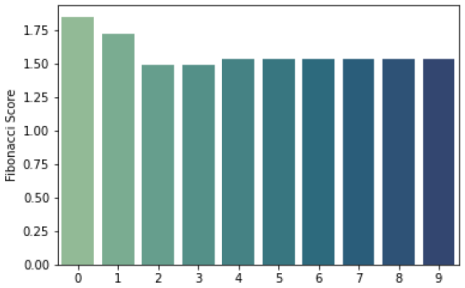

bandwith 256 (Mb/s)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 1 2 3 4 5 6 7 8 9

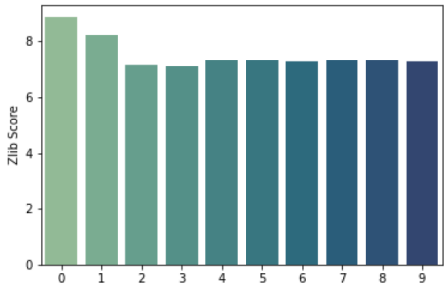

bandwith 512 (Mb/s)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 1 2 3 4 5 6 7 8 9

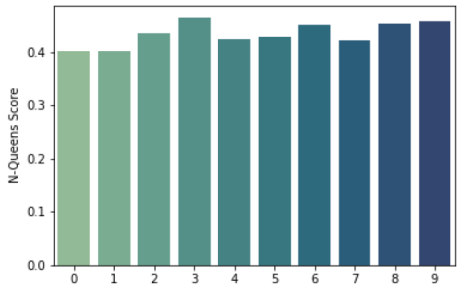

bandwith 1024 (Mb/s)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 1 2 3 4 5 6 7 8 9

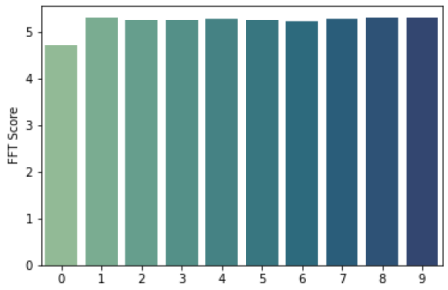

Blowfish Score
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7 8 9

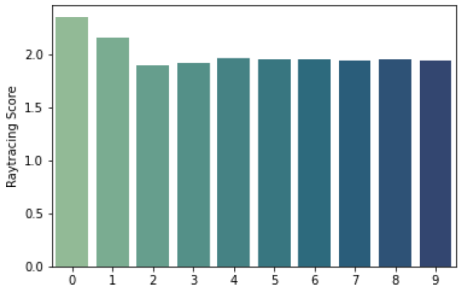

CryptoHash Score
0 50 100 150 200 250 300 350
0 1 2 3 4 5 6 7 8 9

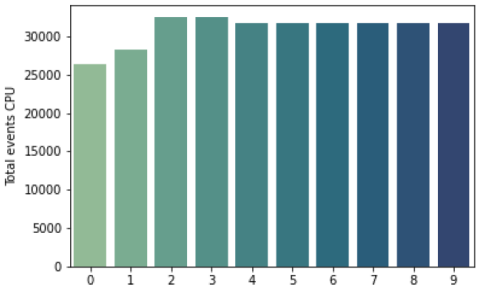

Fibonacci Score
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
0 1 2 3 4 5 6 7 8 9

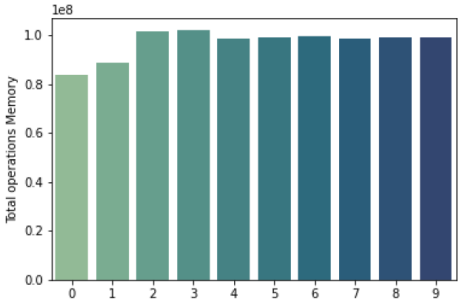

Zlib Score
0 2 4 6 8
0 1 2 3 4 5 6 7 8 9

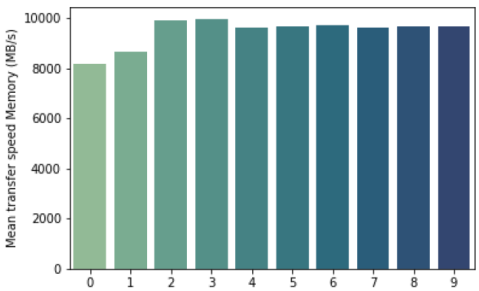

N-Queens Score
0.0 0.1 0.2 0.3 0.4
0 1 2 3 4 5 6 7 8 9

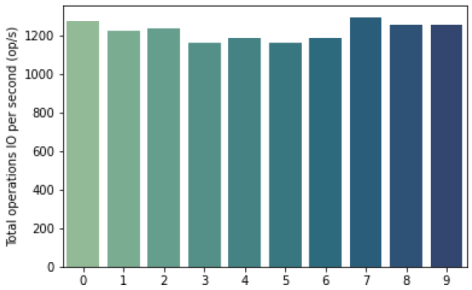

FFT Score
0 1 2 3 4 5
0 1 2 3 4 5 6 7 8 9

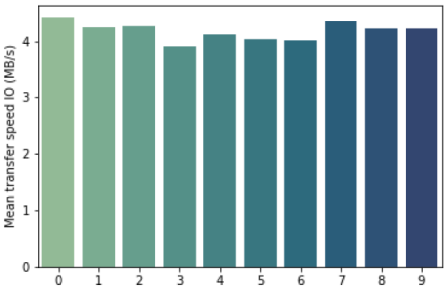

Raytracing Score
0.0
0.5
1.0
1.5
2.0
0 1 2 3 4 5 6 7 8 9

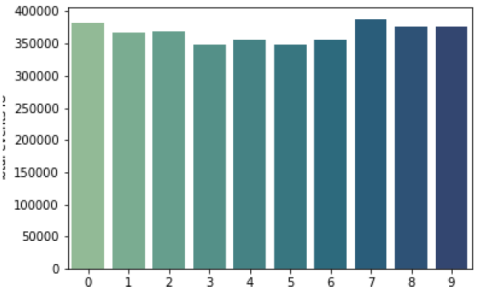

Total events CPU
0
5000
10000
15000
20000
25000
30000
0 1 2 3 4 5 6 7 8 9


1e8
Total operations Memory
0.0
0.2
0.4
0.6
0.8
1.0
0 1 2 3 4 5 6 7 8 9


Mean transfer speed Memory (MB/s)
0
2000
4000
6000
8000
10000
0 1 2 3 4 5 6 7 8 9

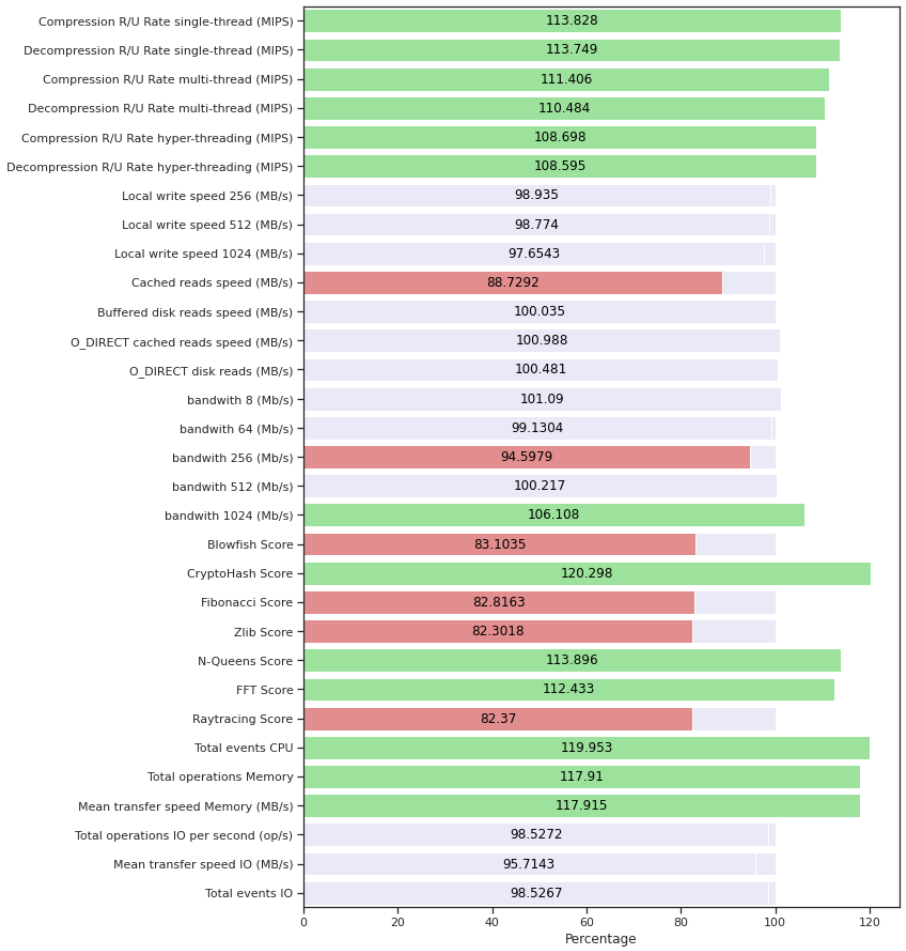

Total operations IO per second (op/s)
0
200
400
600
800
1000
1200
0 1 2 3 4 5 6 7 8 9

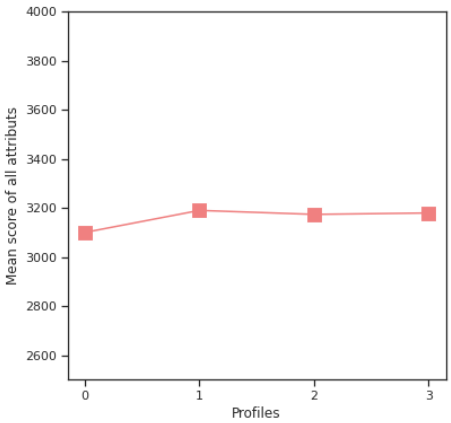

Mean transfer speed IO (MB/s)
0
1
2
3
4
0 1 2 3 4 5 6 7 8 9

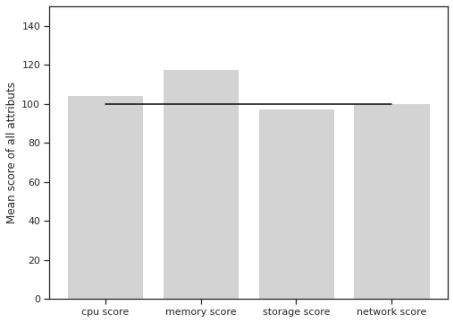

0
50000
100000
150000
200000
250000
300000
350000
400000
0 1 2 3 4 5 6 7 8 9

A.4.2 Exemples de graphes de type benchmark

* Ces graphes ont été utilisés afin de visualiser et calculer les différents scores exprimés lors de la section *Résultats expérimentaux*.







Bibliographie & Webographie

[1] watelectronics. *Cluster Computing : Architecture Its Types*. URL: https://www.watelectronics.com/cluster-computing-architecture-its-types/. (accessed: 23.07.2022).

[2] geeksforgeeks. *An Overview of Cluster Computing*. URL: https://www.geeksforgeeks.org/an-overview-of-cluster-computing/. (accessed: 23.07.2022).

[3] tutorialspoint. *What is Cluster Computing?* URL: https://www.tutorialspoint.com/what-is-cluster-computing. (accessed: 23.07.2022).

[4] esds. *Cluster Computing: Definition, Architecture, and Algorithms*. URL: https://www.esds.co.in/blog/cluster-computing-definition-architecture-and-algorithms/. (accessed: 23.07.2022).

[5] educba. *What is Cluster Computing?* URL: https://www.educba.com/what-is-cluster-computing/. (accessed: 23.07.2022).

[6] wisdomplexus. *What is Cluster Computing and how it is different from Cloud Computing?* URL: https://wisdomplexus.com/blogs/what-is-cluster-computing/. (accessed: 23.07.2022).

[7] Sanath Kumar. *Building a Beowulf Cluster in just 13 steps*. URL: https://www.linux.com/training-tutorials/building-beowulf-cluster-just-13-steps/. (accessed: 23.07.2022).

[8] David Meador. *Beowulf Clusters*. URL: https://www.tutorialspoint.com/Beowulf-Clusters. (accessed: 23.07.2022).

[9] ucdavis. *What is a Beowulf?* URL: http://yclept.ucdavis.edu/Beowulf/aboutbeowulf.html. (accessed: 23.07.2022).

[10] beowulf.org. *Beowulf FAQ*. URL: https://beowulf.org/overview/faq.html. (accessed: 23.07.2022).

[11] Jonathan Teague. *Beowulf Clustering*. URL: https://www.uu.edu/dept/compscience/seminar/Teague.pdf. (accessed: 23.07.2022).

[12] Magnus Reinholdsson Ellen-Louise Bleeker. *Creating a Raspberry Pi-Based Beowulf Cluster*. Faculty of Health, Science and Technology, 2017.

[13] wikipedia. *IT-Benchmarking*. URL: https://fr.wikipedia.org/wiki/IT-Benchmarking. (accessed: 23.07.2022).

[14] wikipedia. *Benchmark (computing)*. URL: https://en.wikipedia.org/wiki/Benchmark_(computing). (accessed: 23.07.2022).

[15] PC Plus. *How to benchmark your PC*. URL: https://www.techradar.com/news/computing/pc/how-to-benchmark-your-pc-954580. (accessed: 23.07.2022).

[16] Wikipedia. *Raspberry Pi*. URL: https://fr.wikipedia.org/wiki/Raspberry_Pi. (accessed: 23.07.2022).

[17] Adem Alpaslan Altun Erdem Ağbahca. *Performance Test of MPI on Raspberry Pi 2 Beowulf Cluster*. 2016.

[18] Dimitrios Papakyriakou. *Benchmarking Raspberry Pi 2 Beowulf Cluster*. 2018.

[19] Khin Nyein Myint Myo Hein Zaw and Win Thanda Aung. *Parallel and Distributed Computing Using MPI on Raspberry Pi Cluster*. 2020.

[20] Eric Wilcox Pooja Jhunjhunwala Karthik Gopavaram Jorge Herrera. *Pi-Crust: A Raspberry Pi Cluster Implementation*. 2015.

[21] Aparicio Carranza Mahendra Ganesh Harrison Carranza Casimer DeCusatis. *Performance Evaluation of a Raspberry Pi Bramble Cluster for Penetration Testing*. 2019.

[22] Simon J. Cox James T. Cox Richard P. Boardman Steven J. Johnston Mark Scott Neil S. O'Brien. *Iridis-pi: a low-cost, compact demonstration cluster*. 2012.

[23] A. Petitet R. C. Whaley J. Dongarra A. Cleary. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. URL: `https://www.netlib.org/benchmark/hpl/`. (accessed: 23.07.2022).

[24] Wikipedia. *Message Passing Interface*. URL: `https://en.wikipedia.org/wiki/Message_Passing_Interface`. (accessed: 23.07.2022).

[25] mpich.org. *MPICH Overview*. URL: `https://www.mpich.org/about/overview/`. (accessed: 23.07.2022).

[26] Lisandro Dalcin. *MPI for Python*. URL: `https://mpi4py.readthedocs.io/en/stable/`. (accessed: 23.07.2022).

[27] Computer Hope. *NFS*. URL: `https://www.computerhope.com/jargon/n/nfs.htm`. (accessed: 23.07.2022).

[28] Wikipedia. *Secure Shell*. URL: `https://en.wikipedia.org/wiki/Secure_Shell`. (accessed: 23.07.2022).

[29] hprc. *TEXAS AM HIGH PERFORMANCE RESEARCH COMPUTING*. URL: `https://hprc.tamu.edu/`. (accessed: 23.07.2022).

[30] Raspberry Pi. *Raspberry Pi*. URL: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/`. (accessed: 23.07.2022).

[31] amazon. *Xute Raspberry Pi 4 Modèle B 4 Go RAM Starter Kit avec 64 Go*. URL: `https://www.amazon.fr/Xute-Alimentation-Interrupteur-Ventilateur-Refroidissement/dp/B09W9BPWKT`. (accessed: 23.07.2022).

[32] labists. *LABISTS Raspberry Pi 4 4GB Complete Starter Kit with 32GB Micro SD Card*. URL: `https://labists.com/products/labists-raspberry-pi-4g-ram-32gb-card`. (accessed: 23.07.2022).

[33] Wikipedia. *Ansible*. URL: `https://fr.wikipedia.org/wiki/Ansible_(logiciel)`. (accessed: 23.07.2022).

[34] journaldunet. *NFS en informatique : définition et focus sur la version NFSv4*. URL: `https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445280-nfs-en-informatique-definition-et-focus-sur-la-version-nfsv4/`. (accessed: 23.07.2022).

[35] Raspberry Pi. *The config.txt file*. URL: `https://www.raspberrypi.com/documentation/computers/config_txt.html#overclocking-options`. (accessed: 23.07.2022).

[36] ubuntu. *ubuntu manpage*. URL: `https://manpages.ubuntu.com/manpages/focal/en/man1/hardinfo.1.html`. (accessed: 23.07.2022).

[37] gentoo. *Sysbench*. URL: `https://wiki.gentoo.org/wiki/Sysbench`. (accessed: 23.07.2022).

[38] Martin Rowan. *Raspberry Pi 4 - Planned USB 3.0 Firmware isn't the Silver Bullet for the Thermal Issues*. URL: `https://www.martinrowan.co.uk/2019/07/raspberry-pi-4-power-temperature-at-idle/`. (accessed: 23.07.2022).

[39] Raspberry Pi. *Overclocking Problems*. URL: https://www.raspberrypi.com/documentation/computers/config_txt.html#overclocking-problems. (accessed: 23.07.2022).

[40] Q-engineering. *Safe overclocking of the Raspberry Pi 4 to 2 GHz*. URL: https://qengineering.eu/overclocking-the-raspberry-pi-4.html. (accessed: 23.07.2022).

[41] Rob Lauer. *Optimizing Raspberry Pi Power Consumption*. URL: https://blues.io/blog/tips-tricks-optimizing-raspberry-pi-power/. (accessed: 23.07.2022).

[42] Wikipedia. *EEPROM*. URL: https://en.wikipedia.org/wiki/EEPROM. (accessed: 23.07.2022).

[43] Q-engineering. *Safe overclocking of the Raspberry Pi 4 to 2 GHz*. URL: https://qengineering.eu/overclocking-the-raspberry-pi-4.html. (accessed: 23.07.2022).

[44] Tony George. *Ubuntu 22.04 Tips Tricks*. URL: https://teejeetech.com/2022/04/23/ubuntu-22-04-tips-tricks/. (accessed: 23.07.2022).

[45] techterms. *Swap File*. URL: https://techterms.com/definition/swap_file. (accessed: 23.07.2022).

[46] ubuntu. *zRAM*. URL: https://doc.ubuntu-fr.org/zram. (accessed: 23.07.2022).

[47] arif-ali. *Compile and Running (HPL) HowTo*. URL: https://github.com/arif-ali/raspberrypi-hpl. (accessed: 23.07.2022).

[48] wikichip. *Cortex-A72 - Microarchitectures - ARM*. URL: https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a72. (accessed: 23.07.2022).

[49] advancedclustering. *HOW DO I TUNE MY HPL.DAT FILE?* URL: https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/. (accessed: 23.07.2022).

[50] netlib. *HPL Tuning*. URL: https://netlib.org/benchmark/hpl/tuning.html. (accessed: 23.07.2022).