Analyse et Optimisation des Performances avec MPI et Télémétrie

Chadha Sakka

December 22, 2024

1 Introduction

Ce rapport analyse les performances d'un benchmark MPI à l'aide du test **OSU MPI Bi-Directional Bandwidth Test v7.5**. Ce test évalue la **bande passante bidirectionnelle** entre deux nœuds MPI, mesurant ainsi leur capacité à échanger simultanément des données.

L'objectif principal est d'instrumenter le code pour collecter des mesures en temps réel et d'intégrer ces données dans un outil de télémétrie (**Chronograph**) via le protocole **StatsD**. Cela permet d'identifier les variations de performances et de détecter d'éventuels goulots d'étranglement.

En combinant **tests pratiques**, **analyse graphique** et **interprétation des résultats**, cette étude propose des pistes d'amélioration pour garantir des performances optimales dans un contexte **HPC** (High-Performance Computing).

2 Méthodologie

2.1 Configuration Docker et Logicielle

Le projet repose sur un environnement basé sur **Docker** (27.3.1), garantissant une **isolation**, une **portabilité** et une **reproductibilité** des tests. Le code a été modifié pour inclure un module de télémétrie via **StatsD** et une visualisation avec **Chronograph**.

- **Image Docker** : `mpi-python:latest`, construite à partir d'un Dockerfile incluant Open MPI.
- **Télémétrie :** Chronograph connecté à StatsD via UDP.
- **Infrastructure :** Deux nœuds MPI configurés avec IP statiques sur un réseau Docker bridge.
- **Conteneurs Docker :** 4 conteneurs configurés avec MPI et les outils d'analyse.

2.2 Modifications apportées au code

Le fichier `osu_bibw.c` a été modifié pour :

1. Ajouter une fonction `chrono()` pour mesurer et envoyer les résultats au serveur StatsD.
2. Intégrer des appels à `chrono()` après chaque test pour un suivi en temps réel.

2.3 Commandes exécutées

Les commandes suivantes ont été utilisées pour compiler et exécuter le benchmark MPI :

1. **Compilation du code MPI :**

```
make clean
make
```

2. **Exécution du benchmark :**

```
mpirun --mca btl_tcp_if_include 10.0.1.0/24 -n 2 -host 10.0.1.7,10.0.1.4 osu_bibw
```

3 Graphique PlantUML

3.1 Exemple de Prompt IA avec PlantUML

Pour illustrer l'utilisation de l'IA dans ce projet, nous avons expérimenté avec deux modèles d'IA : **ChatGPT 4.0** et **LLaMA 3.1**. L'objectif était de générer un diagramme de séquence décrivant la télémétrie applicative avec une base de données **InfluxDB**.

3.2 Prompt Utilisé

> *Peux-tu générer un diagramme de séquence en PlantUML décrivant une télémétrie applicative avec l'envoi de données depuis un processus MPI vers un serveur StatsD ? Le diagramme doit inclure :*
> *Un utilisateur lançant un benchmark MPI et appelant la fonction `chrono()`.*
> *Une fonction `chrono()` envoyant les données via UDP vers un serveur StatsD.*
> *Un serveur StatsD stockant les données dans une base InfluxDB.*
> *Un outil Chronograph récupérant les données depuis InfluxDB pour visualisation.*
> *Un bloc d'alternative (alt) pour gérer les cas de succès ou d'échec de transmission avec des réponses appropriées.*

3.3 Diagramme UML : Flux de Télémétrie avec MPI et StatsD

```
@startuml
actor Utilisateur
participant "Processus MPI" as MPI
participant "Fonction chrono()" as CHRONO
participant "Serveur StatsD" as STATSD
database "Base InfluxDB" as INFLUX
participant "Chronograph" as CHRONO_G

Utilisateur -> MPI : Lance le benchmark
MPI -> CHRONO : Appelle chrono() avec la duree d execution
CHRONO -> STATSD : Envoie les donnees via UDP
STATSD -> INFLUX : Stocke les donnees
CHRONO_G -> INFLUX : Recupere les donnees
Utilisateur <-- MPI : Affiche les resultats

alt Succes de la transmission
    STATSD -> CHRONO : Accuse de reception OK
else Echec de la transmission
    STATSD -> CHRONO : Erreur d envoi
end
@enduml
```

3.4 Comparaison des Résultats

Le modèle **ChatGPT 4.0** a fourni un code conforme aux attentes et facile à intégrer dans PlantUML. Le code généré a permis de produire un diagramme de séquence clair et structuré car nous avons pu introduire un exemple de graphique PlantUML. En revanche, le modèle **LLaMA 3.1** n'a pas généré un résultat satisfaisant. Le code produit présentait des erreurs de syntaxe et un manque de structure, ce qui a nécessité plusieurs corrections manuelles.

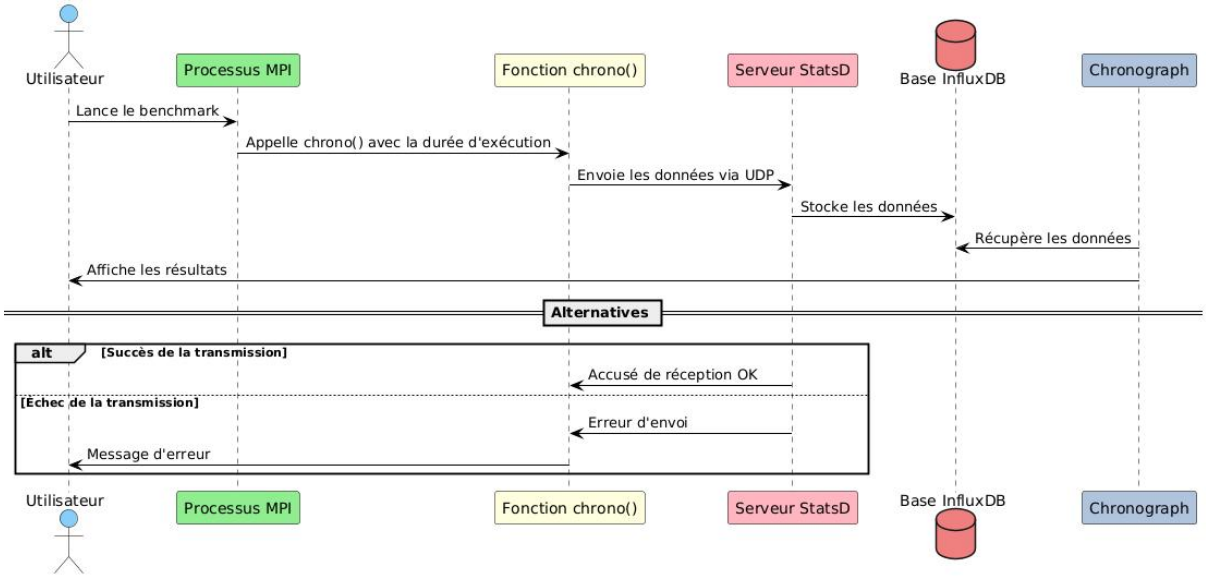

Figure 1: Diagramme UML illustrant le processus de télémétrie applicative

Ce diagramme illustre les interactions entre les composants de la chaîne de télémétrie. L'application envoie ses mesures au collecteur, qui les traite et les stocke dans une base de données InfluxDB. Les données peuvent être consultées ultérieurement via des requêtes, fournissant une vue détaillée pour l'analyse et la visualisation.

4 Résultats des Tests de Bande Passante Bidirectionnelle

Le tableau 1 présente les résultats obtenus avec le test de bande passante bidirectionnelle.

Table 1: Bande passante bidirectionnelle (MB/s)

Taille (B)	**Bande passante (MB/s)**
1	0.15
2	0.30
4	0.70
8	1.31
16	2.50
32	5.02
64	9.60
128	20.80
256	41.93
512	75.95
1024	153.46
2048	287.83
4096	560.27
8192	915.57
16384	1255.66
32768	1665.74
65536	1275.21
131072	1960.09
262144	2439.59
524288	2494.66
1048576	2282.57
2097152	2278.36
4194304	2128.23

4.1 Analyse des Résultats

Les résultats obtenus révèlent une performance remarquable avec un débit maximal atteignant **2494.66 MB/s** pour des messages de taille moyenne (524288 octets). Les petites tailles de messages présentent un débit limité (**0.15 MB/s** pour 1 octet), principalement en raison des latences liées à l'établissement des connexions et aux surcharges protocolaires.

En revanche, à mesure que la taille des messages augmente, la bande passante s'améliore considérablement grâce à une meilleure utilisation des tampons mémoire et des mécanismes d'optimisation de MPI. Cependant, une légère dégradation des performances est observée pour les très grandes tailles de messages (**2128.23 MB/s** pour 4 Mo). Cette diminution pourrait être attribuée à des goulots d'étranglement dans la gestion des tampons MPI ou à des limitations liées au réseau sous-jacent.

Des optimisations supplémentaires, telles que l'ajustement de la taille des tampons et l'amélioration des configurations réseau (MTU et QoS), pourraient stabiliser davantage les performances et réduire les variations observées.

5 Visualisation des Performances de la Bande Passante MPI sur Chronograph

Figure 2: Performances fluctuantes avec des variations marquées.

Le graphique ci-dessus illustre des variations marquées dans les performances mesurées. Ces fluctuations peuvent être attribuées à plusieurs facteurs, notamment :
- Des **conflits de ressources réseau**, où d'autres processus utilisent simultanément la bande passante.
- L'effet de **multiplexage réseau**, qui entraîne des délais liés à l'ordonnancement des paquets de données.
- Des **goulots d'étranglement dans les tampons MPI**, dus à une gestion inefficace de la mémoire tampon, provoquant des retards dans l'envoi et la réception des données.

Figure 3: Performances stables sur Chronograph avec des variations limitées.

Le second graphique montre une stabilité relative des performances, oscillant autour de 4000 à 5000 MB/s, contrairement au premier graphique qui présente des variations marquées. Pourtant, les deux

graphiques ont été obtenus sur une intervalle de temps identique d'une heure. Les variations mineures observées peuvent être dues à des **fluctuations réseau temporaires** ou à des **interférences minimales** dans la gestion des tampons MPI. Cette stabilité suggère un environnement réseau correctement configuré et une allocation efficace des tampons mémoire. Ces résultats sont particulièrement adaptés aux scénarios nécessitant des performances prévisibles et reproductibles.

6 Conclusions

Les résultats de ce benchmark démontrent la capacité du système MPI à fournir des performances élevées, avec un débit maximal dépassant **2.4 GB/s**. L'analyse a mis en évidence une amélioration significative des performances avec l'augmentation de la taille des messages, tout en signalant une légère dégradation pour les très grands messages en raison de limitations réseau et protocolaires.

L'utilisation de **Docker** n'a pas introduit d'impact significatif sur les performances, bien que des fluctuations aient été observées dans certains cas. Ces variations peuvent être attribuées à des interférences réseau ou à une configuration sous-optimale des tampons MPI.

Pour améliorer davantage les résultats, plusieurs axes peuvent être explorés :

- **Optimisation des tampons MPI :** Ajuster leur taille pour minimiser les goulots d'étranglement.
- **Optimisation réseau :** Configurer la taille maximale des paquets (MTU) et utiliser des mécanismes de qualité de service (QoS).
- **Approfondissement de la télémétrie :** Ajouter des mesures supplémentaires pour suivre la latence et détecter les pics d'utilisation.
- **Analyse en temps réel :** Automatiser l'intégration avec des outils tels que Grafana ou Chronograph pour un suivi continu.

En conclusion, ce travail a permis de démontrer la faisabilité d'une solution combinant benchmarks MPI et télémétrie en temps réel. Il ouvre la voie à des optimisations futures pour des environnements HPC plus performants et résilients.