2024/25

Nom : DADA SIMEU CÉDRIC DAREL

Email : cedric-darel.dada@ensta-paris.fr

Titre : Compte rendu TP3

STIC

ENSTA Paris, Institut Polytechnique de Paris

Table des matières

1 Architecture matérielle de l'ordinateur 4
2 Présentation de l'algorithme et choix des intervalles dans les buckets 6
2.1 Génération des données 6
2.2 Distribution des données 6
2.3 Tri local 6
2.4 Détermination des intervalles(bornes) des buckets 6
2.5 Redistribution des données 6
2.6 Tri final et collecte 6
3 Analyse de la complexité 6
4 Expérimentation et Résultats 7
4.1 Configuration matérielle 11
4.2 Analyse et interprétation des résultats 11

Table des figures

1 Résultat de la commande lscpu 4
2 Résultat de la commande lstopo : Nous pouvons visualiser les tailles des caches 5
3 Temps d'exécution vs nombre de processus. 8
4 Déséquilibrage des charges vs nombre de processus. 9
5 Comparaison entre temps de communication et temps de calcul local pour $N = 10^6$. 9
6 Speedup vs nombre de processus. 10
7 Efficiency vs nombre de processus. 10

1 Architecture matérielle de l'ordinateur

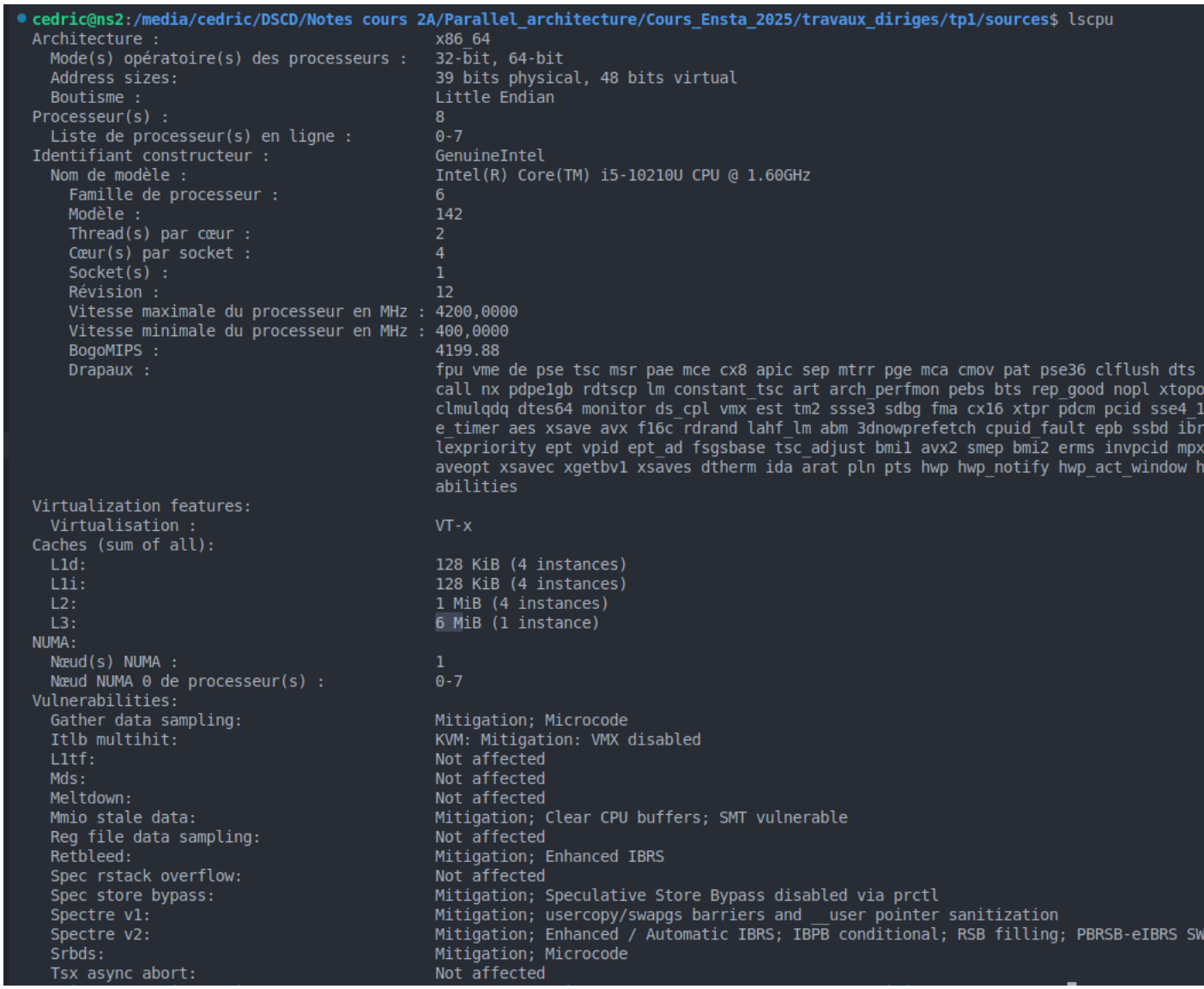

FIGURE 1 – Résultat de la commande lscpu

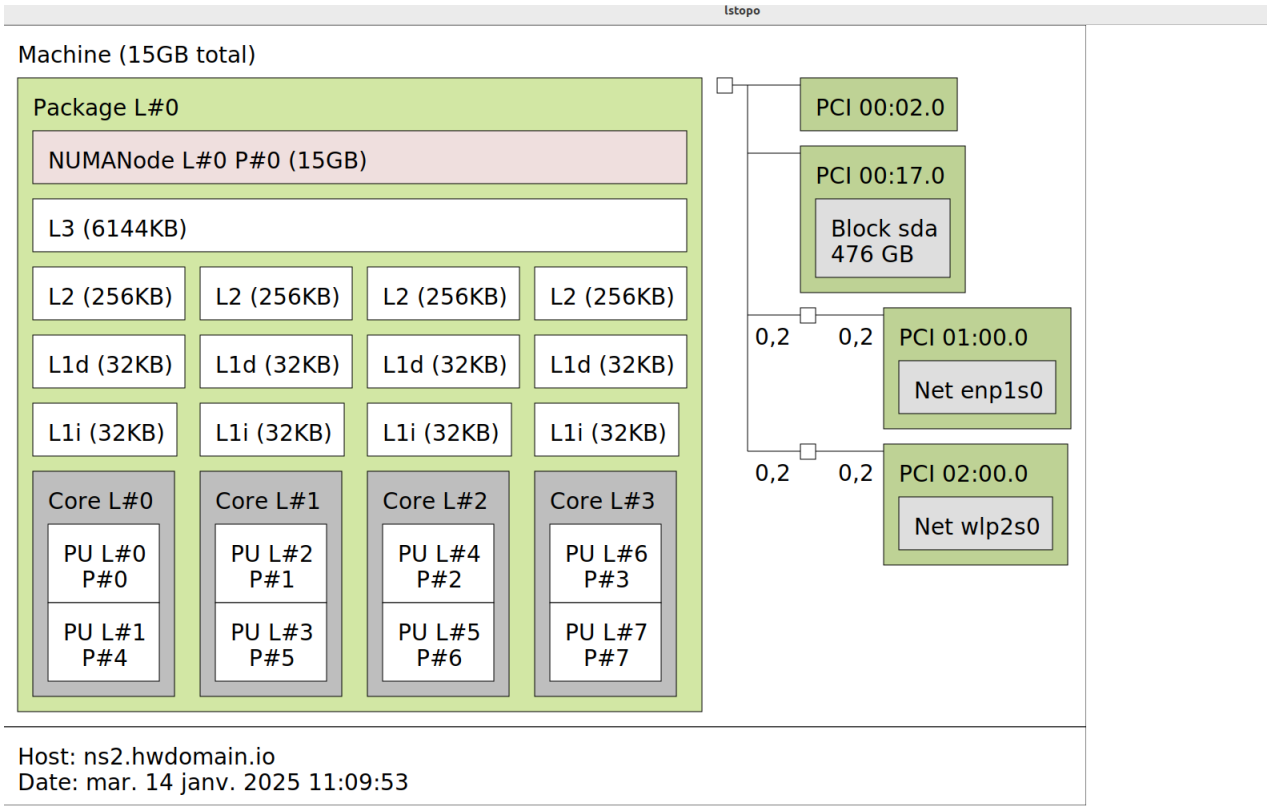

FIGURE 2 – Résultat de la commande lstopo : Nous pouvons visualiser les tailles des caches

2 Présentation de l'algorithme et choix des intervalles dans les buckets

2.1 Génération des données

— Le processus 0 génère un tableau de nombres aléatoires.

— La génération est faite avec np.random.rand(N) où N est le nombre total d'éléments à trier

— np.random.seed(42), nous garantit que nos différents tests seront réalisés avec les memes valeurs de la liste, ce qui rend les interprétations possibles.

2.2 Distribution des données

Les données sont distribuées entre les processus via Scatterv. Chaque processus reçoit une portion égale ou presque égale des données.

2.3 Tri local

Chaque processus trie localement sa portion de données avec local_data.sort(), sachant que la méthode sort utilise le tri rapide par défaut.

2.4 Détermination des intervalles(bornes) des buckets

— Chaque processus sélectionne des échantillons régulièrement espacés dans ses données locales.

— Ces échantillons sont rassemblés sur le processus 0 avec comm.gather

— Sur le processus 0, les échantillons sont triés, et des bornes sont définies en divisant l'espace des valeurs en segments égaux.

— Les bornes sont ensuite diffusées à tous les processus avec comm.bcast.

2.5 Redistribution des données

Les données sont redistribuées selon les bornes des buckets :

— Chaque processus détermine combien d'éléments il doit envoyer à chaque autre processus.

— Les données sont redistribuées avec Alltoallv.

2.6 Tri final et collecte

— Après redistribution, chaque processus trie localement ses nouvelles données.

— Les résultats sont rassemblés sur le processus 0 avec comm.gather.

3 Analyse de la complexité

D'après le cours, la complexité du bucket sort parrallèle est donnée par :

$$T_{\text{total}} = O(N + \frac{N}{k} log_2(\frac{N}{k}))$$

Où k = n (nombre de buckets = nombre de processus).

Pour une distribution équilibrée des données :

— Le cout local de tri est $O(\frac{N}{n} log_2(\frac{N}{n}))$

— Le cout de redistribution est proportionnel à $O(\frac{N}{n})$

4 Expérimentation et Résultats

A Savoir 4.1. *Les données ayant permi la réalisation des différents graphes et tableaux qui suivent sont disponibles dans le fichier tests.txt*

— Le tableau 1 résume le temps total d'exécution pour chaque combinaison de N(taille du tableau à trier) et n(nombre de processus).

— Dans, le tableau 3 le speedup est calculé comme étant : $S(n) = \frac{T(2)}{T(n)}$

— Dans le tableau, 4, $E(n) = \frac{S(n)}{n}$

— Le graphique 3 montre le temps d'exécution total pour chaque valeur de N en fonction du nombre de processus.

— Le graphique 7 montre l'efficiency pour chaque valeur de N en fonction du nombre de processus.

— Le graphique 6 montre le speedup pour chaque valeur de N en fonction du nombre de processus.

— Le graphique 4 présente le déséquilibrage des charges pour chaque valeur de N en fonction du nombre de processus.
Ce déséquilibrage est donné par la formule Déséquilibrage $(\%) = \frac{\max(\text{local\_sort}) - \min(\text{local\_sort})}{\max(\text{local\_sort})} \times 100$

TABLE 1 – Temps d'exécution total pour différentes valeurs de N et n.

N	$n=2$	$n=3$	$n=4$	$n=8$	$n=16$
10^3	0.007857	0.008237	0.091724	0.017107	0.294403
10^4	0.015728	0.016078	0.085148	0.034283	0.525791
10^5	0.094280	0.084637	0.216848	0.159568	1.188889
10^6	0.965147	0.715270	0.878063	0.688540	1.198288
10^7	10.276568	6.984677	8.705777	6.583885	13.667255
10^8	10.181517	7.492297	9.834438	8.913589	22.307478

TABLE 2 – Déséquilibrage des charges en fonction de N et n.

N	$n=2$	$n=3$	$n=4$	$n=8$	$n=16$
10^3	11.11%	15.79%	15.38%	38.46%	77.78%
10^4	10.00%	17.65%	20.00%	38.46%	77.78%
10^5	1.79%	17.65%	15.38%	25.00%	40.00%
10^6	1.49%	11.76%	15.38%	25.00%	40.00%
10^7	1.49%	11.76%	15.38%	25.00%	40.00%
10^8	1.79%	17.65%	15.38%	25.00%	40.00%

TABLE 3 – Speedup pour différentes valeurs de N et n.

N	$n=2$	$n=3$	$n=4$	$n=8$	$n=16$
10^3	1.00	0.95	0.09	0.57	0.03
10^4	1.00	0.98	0.18	0.46	0.03
10^5	1.00	1.11	0.44	0.60	0.08
10^6	1.00	1.48	1.10	1.40	0.81
10^7	1.00	1.47	1.18	1.56	0.75
10^8	1.00	1.36	1.04	1.15	0.45

TABLE 4 – Efficiency pour différentes valeurs de N et n.

N	$n=2$	$n=3$	$n=4$	$n=8$	$n=16$
10^3	0.50	0.32	0.02	0.07	0.00
10^4	0.50	0.33	0.05	0.06	0.00
10^5	0.50	0.37	0.11	0.08	0.01
10^6	0.50	0.49	0.28	0.18	0.05
10^7	0.50	0.49	0.29	0.20	0.05
10^8	0.50	0.45	0.26	0.14	0.03

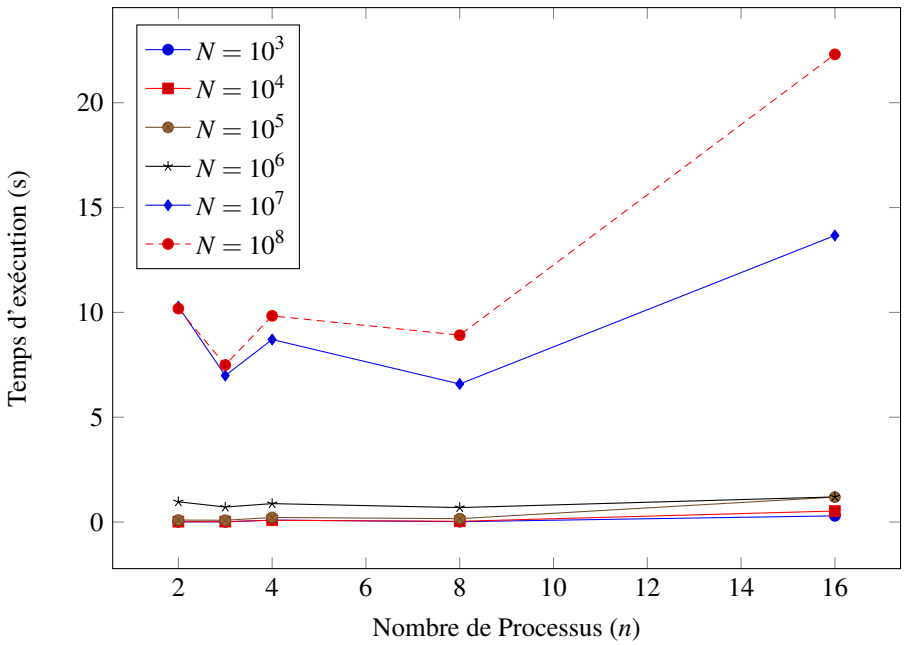

FIGURE 3 – Temps d'exécution vs nombre de processus.

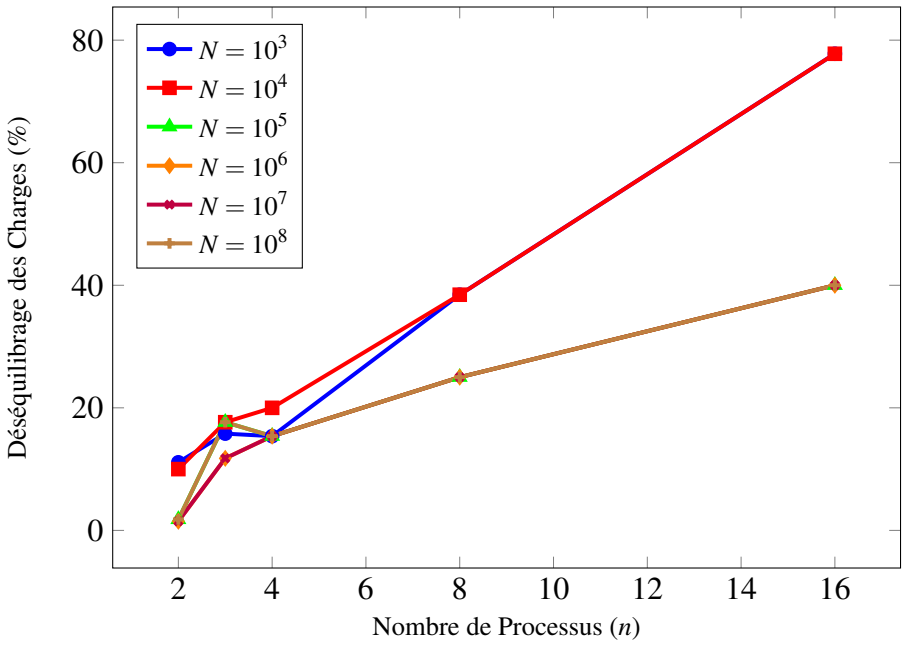

FIGURE 4 – Déséquilibrage des charges vs nombre de processus.

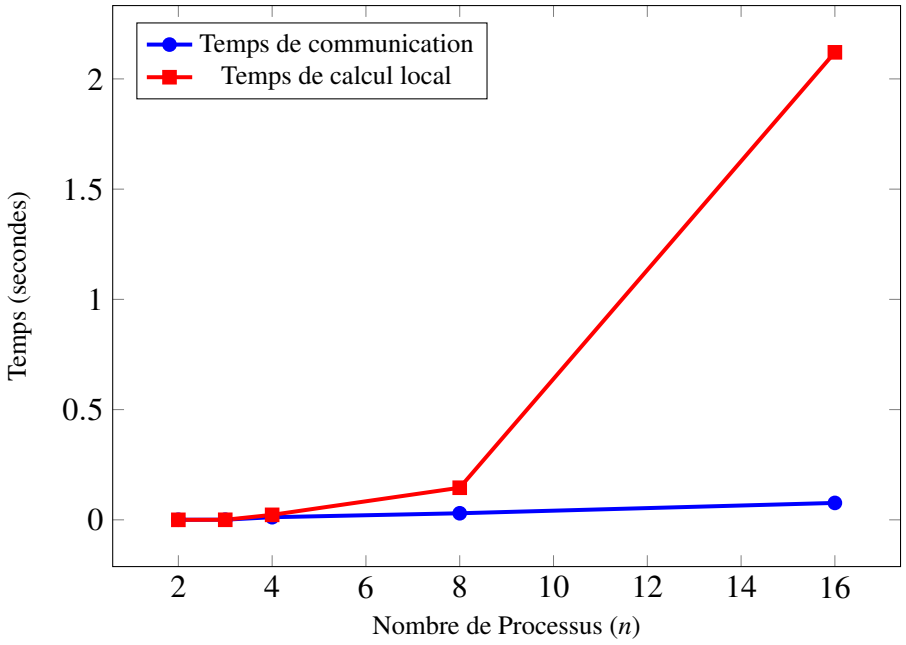

FIGURE 5 – Comparaison entre temps de communication et temps de calcul local pour $N = 10^6$.

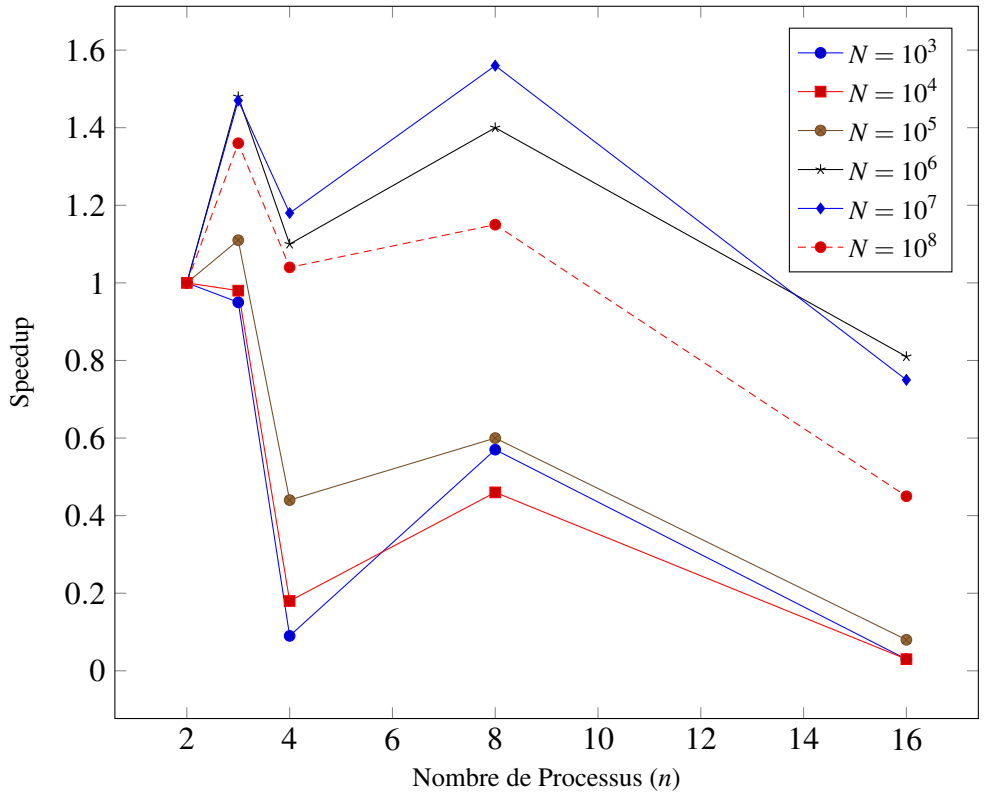

FIGURE 6 – Speedup vs nombre de processus.

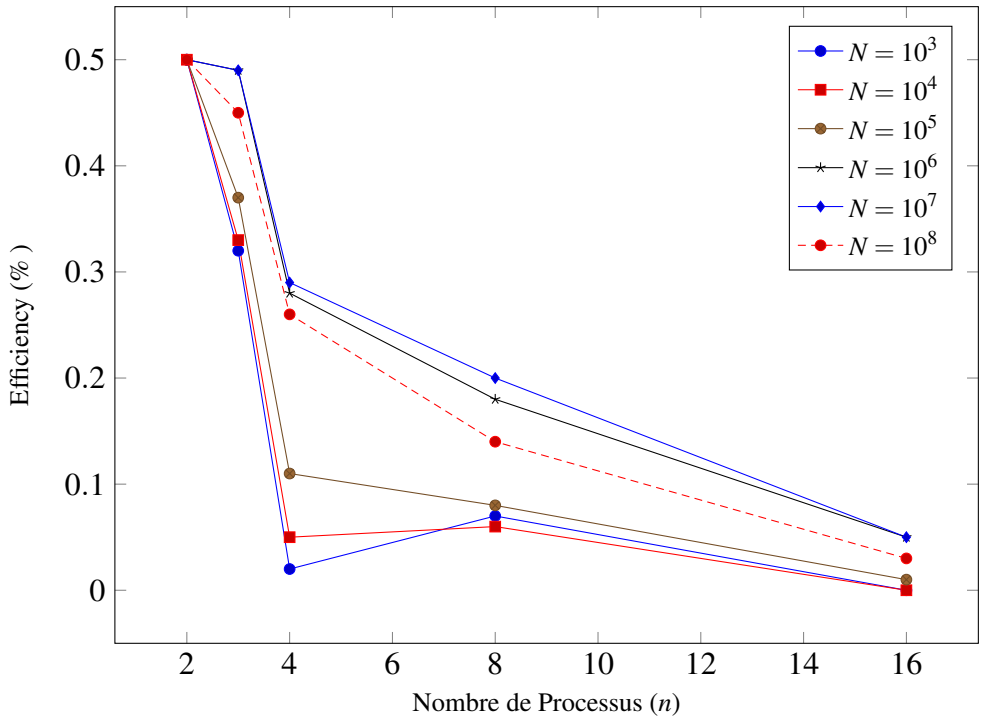

FIGURE 7 – Efficiency vs nombre de processus.

4.1 Configuration matérielle

Le mode d'exécution –oversubscribe était nécesaire pour exécuter plus de 4 processus MPI sur notre machine qui ne contient que 4 coeurs physiques. (voir fig 1).

Impact de l'oversubscription

L'oversubscription signifie que plusieurs processus partagent le même cœur physique, ce qui peut entraîner :

— Contestation des ressources : Les processus doivent partager les caches et les unités de calcul, augmentant potentiellement les délais d'accès mémoire et réduisant les performances.

— Overhead du multitâche : Le système d'exploitation doit constamment basculer entre les threads, introduisant un coût supplémentaire lié au context-switching.

4.2 Analyse et interprétation des résultats

Temps d'exécution total

Le temps total d'exécution augmente lorsque n>4, surtout pour de grandes valeurs de N. Cela est dû à :

— Communication accrue : Plus il y a de processus, plus les communications inter-processus (scatter, redistribute, gather) deviennent coûteuses.

— Oversubscription : Au-delà de 4 processus, les threads partagent les mêmes cœurs physiques, réduisant l'efficacité du parallélisme.

Speedup et Efficiency

— Pour $N < 10^5$, le speedup diminue rapidement lorsque n>4, car les communications dominent les calculs locaux.

— Pour $N \geq 10^6$, le speedup reste raisonnable jusqu'à n=8, mais chute pour n=16 en raison de l'oversubscription et des communications excessives.

— L'efficiency chute significativement pour n>8, indiquant un mauvais équilibre entre les calculs et les communications.

— Pour $N \geq 10^6$, le déséquilibrage reste modéré jusqu'à n=4, mais augmente pour n>4, en partie à cause de l'oversubscription.

Impact des surcouts de communication

Les phases de communication (scatter, redistribute, gather) représentent une part importante du temps total d'exécution, surtout pour n>4.

— Pour $N = 10^3$ ou 10^4, **les communications dominent les calculs locaux**, rendant l'utilisation de nombreux processus inefficace.

— Pour $N \geq 10^6$, les communications restent importantes, mais leur impact relativement aux temps de calcul diminue à mesure que N augmente, car les calculs locaux (local_sort) deviennent plus coûteux.