



OS202 – Systèmes Parallèles

TD3 : Parallel Bucket Sort (MPI)

Comparaison avec Quicksort séquentiel

MENESES GAMBOA Carlos

Table des matières

1	Résumé	2
2	Configuration expérimentale	2
2.1	Machine utilisée	2
2.2	Compilation et exécution	2
3	Algorithme et instrumentation	2
3.1	Pipeline Parallel Bucket Sort	2
3.2	Métriques déjà disponibles dans le code	3
3.3	Métriques ajoutées pour un rapport professionnel	3
4	Définitions formelles des métriques	3
4.1	Performance globale	3
4.2	Coût de communication	3
4.3	Qualité d'équilibrage des buckets	4
4.4	Migration et débit	4
5	Protocole de mesure utilisé	4
6	Résultats expérimentaux	4
6.1	Baseline Quicksort (séquentiel)	4
6.2	Comparaison globale Bucket Sort MPI vs Quicksort	5
6.3	Communication et équilibrage	5
6.4	Décomposition par phase (cas représentatif)	5
7	Analyse et discussion	5
7.1	Constats principaux	6
7.2	Interprétation des métriques retenues	6
8	Menaces à la validité	6
9	Conclusion	6

1 Résumé

Ce rapport présente l'implémentation MPI de *Parallel Bucket Sort* (variante *Sample Sort*) et son évaluation expérimentale face à un tri *Quicksort* séquentiel sur le même jeu de données. Le code mesure non seulement le temps global, mais aussi des métriques fines : coût par phase MPI, qualité de l'équilibrage des buckets, volume réel de données migrées entre processus et débit de tri (*throughput*).

Objectif du rapport : produire une comparaison défendable entre version parallèle et baseline séquentielle, avec des métriques directement utiles pour discuter *scalabilité*, *efficacité* et *coûts de communication*.

2 Configuration expérimentale

2.1 Machine utilisée

TABLE 1 – Caractéristiques de la machine d'expérimentation.

Champ	Valeur
Architecture	x86_64
Processeur	Intel Core i5-1135G7 @ 2.40GHz
CPU logiques	8
Cœurs physiques	4
Hyperthreading	Oui (2 threads/cœur)
Hyperviseur	WSL2 (Microsoft)

2.2 Compilation et exécution

- Compilation : `make`
- Exécution générique : `mpirun -np p ./bucket_sort N debug quicksort_runs`
- Paramètres recommandés pour benchmark : `debug=0, quicksort_runs=5`
- Exemple : `mpirun -np 4 ./bucket_sort 1000000 0 5`

3 Algorithme et instrumentation

3.1 Pipeline Parallel Bucket Sort

Le code suit la séquence standard en 8 étapes :

1. initialisation MPI et paramétrage de N ,
2. génération des données (rank 0) et `MPI_Scatter`,
3. tri local + échantillonnage régulier,
4. tri des échantillons globaux + choix des pivots,
5. calcul des `send_counts/send_displs`,

6. redistribution variable via `MPI_Alltoallv`,
7. tri local final des buckets reçus,
8. collecte finale via `MPI_Gatherv`.

3.2 Métriques déjà disponibles dans le code

Le programme `Bucket_Sort.cpp` instrumente :

- temps par phase (*min/avg/max*) entre processus,
- temps total parallèle (*max* entre processus),
- ratio communication/temps total,
- volume total envoyé/reçu,
- qualité d'équilibrage des buckets (*min/avg/max*, *avg/max*),
- comparaison avec Quicksort (temps, speedup, efficacité).

3.3 Métriques ajoutées pour un rapport professionnel

Les métriques ci-dessous sont particulièrement pertinentes pour l'analyse :

- **Quicksort multi-runs** : temps *min* et *avg* sur *k* exécutions (stabilité du baseline),
- **Données migrées** : part de données qui changent réellement de processus pendant `Alltoallv`,
- **Dispersion de charge** : écart-type, coefficient de variation (CV), ratio *max/min*,
- **Débit** : millions d'éléments triés par seconde (Melem/s), parallèle et séquentiel.

4 Définitions formelles des métriques

4.1 Performance globale

Soit p le nombre de processus MPI.

$$T_{par}(p) = \max_{r \in [0, p-1]} T_r$$

$$S_p = \frac{T_{seq}}{T_{par}(p)}, \quad E_p = \frac{S_p}{p}$$

où T_{seq} est le temps Quicksort séquentiel (utiliser T_{seq}^{min} pour limiter le bruit).

4.2 Coût de communication

$$R_{comm} = \frac{T_{comm}}{T_{par}}$$

avec T_{comm} approché par la somme des phases MPI de communication (`Scatter`, `Allgather`, `Alltoall`, `Alltoallv`, `Gather`, `Gatherv`).

4.3 Qualité d'équilibrage des buckets

Pour les tailles finales de buckets b_i :

$$\bar{b} = \frac{1}{p} \sum_{i=1}^p b_i, \quad LB_{avg/max} = \frac{\bar{b}}{\max_i b_i}$$

$$\sigma_b = \sqrt{\frac{1}{p} \sum_{i=1}^p (b_i - \bar{b})^2}, \quad CV_b = \frac{\sigma_b}{\bar{b}}, \quad I_{max/min} = \frac{\max_i b_i}{\min_i b_i}$$

4.4 Migration et débit

$$\text{MovedElements} = N - \sum_{r=0}^{p-1} \text{send_counts}_r[r]$$

$$\text{MovedRatio} = \frac{\text{MovedElements}}{N}$$

$$\text{Throughput}_{par} = \frac{N}{T_{par}}, \quad \text{Throughput}_{seq} = \frac{N}{T_{seq}^{min}}$$

5 Protocole de mesure utilisé

1. Grille testée : $N \in \{10^5, 5 \times 10^5, 10^6\}$, $p \in \{1, 2, 3, 4\}$.
2. Pour chaque couple (N, p) , exécuter 3 runs MPI indépendants et conserver :
 - T_{par}^{avg} , ainsi que T_{par}^{min} et T_{par}^{max} ,
 - les moyennes des métriques de communication et d'équilibrage.
3. Baseline séquentielle : Quicksort interne mesuré avec `quicksort_runs=5` à chaque run.
4. Paramètres fixes : `debug=0`, même machine, même binaire.

Traceabilité : les logs bruts des runs sont archivés dans `bench_logs/`, et les agrégats dans `benchmark_results_avg.csv`.

6 Résultats expérimentaux

6.1 Baseline Quicksort (séquentiel)

TABLE 2 – Résultats Quicksort utilisés comme baseline de comparaison.

N	$T_{qs}^{min,avg}$ (s)	$T_{qs}^{avg,avg}$ (s)	Throughput qs (Melem/s)
100000	0.007714	0.008712	12.976599
500000	0.038869	0.041491	12.881275
1000000	0.084858	0.090081	11.787215

6.2 Comparaison globale Bucket Sort MPI vs Quicksort

TABLE 3 – Résultats globaux agrégés de Bucket Sort MPI (3 runs par couple N, p).

N	p	T_{par}^{avg} (s)	$T_{par}^{[min, max]}$ (s)	$T_{qs}^{min, avg}$ (s)	$T_{qs}^{avg, avg}$ (s)	S_p^{avg}	E_p^{avg}	Thr. par (Melem/s)
100000	1	0.010732	[0.009461, 0.012589]	0.007714	0.008712	0.726922	0.726922	9.456722
100000	2	0.009170	[0.007909, 0.011446]	0.007072	0.007537	0.795059	0.397530	11.214214
100000	3	0.006773	[0.005600, 0.008311]	0.007333	0.008143	1.102069	0.367356	15.164331
100000	4	0.007694	[0.006240, 0.010011]	0.007554	0.011704	1.028315	0.257079	13.551377
500000	1	0.058392	[0.052807, 0.066316]	0.038869	0.041491	0.669541	0.669541	8.642737
500000	2	0.045987	[0.044571, 0.047428]	0.039806	0.042134	0.866392	0.433196	10.879650
500000	3	0.040056	[0.037802, 0.041754]	0.040640	0.044117	1.016815	0.338938	12.504567
500000	4	0.039284	[0.033652, 0.042567]	0.042400	0.049501	1.091964	0.272991	12.871378
1000000	1	0.124194	[0.117234, 0.137921]	0.084858	0.090081	0.687553	0.687553	8.098800
1000000	2	0.144780	[0.114693, 0.175667]	0.096668	0.110025	0.684422	0.342211	7.118998
1000000	3	0.086940	[0.081992, 0.089712]	0.099635	0.107229	1.145459	0.381820	11.521473
1000000	4	0.091129	[0.087795, 0.097336]	0.096957	0.108809	1.064616	0.266154	10.998145

6.3 Communication et équilibrage

TABLE 4 – Métriques les plus pertinentes sur les meilleures configurations par taille.

N	p	Comm (%)	Moved (%)	$LB_{avg/max}$	CV_b
100000	3	4.254775	66.660667	0.991995	0.008950
500000	4	4.710001	75.029000	0.989770	0.007825
1000000	3	4.759588	66.629667	0.997648	0.002322

6.4 Décomposition par phase (cas représentatif)

TABLE 5 – Profil des phases pour $N = 10^6$, $p = 4$ (run représentatif, $T_{par} = 0.088257$ s).

Phase	min (s)	avg (s)	max (s)	Part max/total (%)
Step2_Generation	0.030578	0.030581	0.030583	34.65
Step2_Scatter	0.001312	0.001313	0.001313	1.49
Step3_LocalSort	0.032427	0.032430	0.032432	36.75
Step3_Sampling_Allgather	0.000043	0.000044	0.000044	0.05
Step4_Splitters	0.000002	0.000002	0.000002	0.00
Step5_BucketPartitioning	0.000034	0.000037	0.000038	0.04
Step5_Alltoall_Counts	0.000022	0.000023	0.000024	0.03
Step6_Alltoallv_Data	0.000766	0.000779	0.000792	0.90
Step7_FinalLocalSort	0.014365	0.014369	0.014373	16.29
Step8_Gather_Counts	0.000013	0.000021	0.000023	0.03
Step8_Gatherv_Data	0.001276	0.001279	0.001284	1.45

7 Analyse et discussion

7.1 Constats principaux

- **Seuil d'intérêt du parallèle** : pour $N = 10^5$, le gain reste limité et sensible à l'overhead MPI; pour $N \geq 5 \times 10^5$, les configurations $p = 3$ et $p = 4$ deviennent compétitives (speedup > 1 dans plusieurs cas).
- **Meilleurs compromis mesurés** : $N = 10^5 \Rightarrow p = 3$ ($S = 1.102$), $N = 5 \times 10^5 \Rightarrow p = 4$ ($S = 1.092$), $N = 10^6 \Rightarrow p = 3$ ($S = 1.145$).
- **Équilibrage robuste** : CV_b reste faible (≤ 0.009 pour $p \leq 4$) et $LB_{avg/max}$ est proche de 1, ce qui indique que la sélection de splitters produit des buckets bien distribués.
- **Coût de redistribution** : la part de données migrées suit la tendance attendue d'un tri global ($\approx (1 - \frac{1}{p}) \times 100\%$ pour données aléatoires), avec un impact modéré tant que $p \leq 4$.

7.2 Interprétation des métriques retenues

- T_{par}^{avg} + intervalle $[T_{par}^{min}, T_{par}^{max}]$: mesure de performance + stabilité.
- S_p^{avg} , E_p^{avg} : qualité réelle du parallélisme au regard du baseline séquentiel.
- Comm (%), Moved (%) : coût de communication et intensité de redistribution.
- $LB_{avg/max}$, CV_b : qualité d'équilibrage des partitions finales.

8 Menaces à la validité

- Variabilité système (processus concurrents, fréquence CPU dynamique).
- Effets spécifiques à l'environnement virtualisé (WSL2).
- Les métriques sont des moyennes sur 3 runs; une campagne plus longue (10+ runs) améliorerait encore la robustesse statistique.

9 Conclusion

Cette étude montre que *Parallel Bucket Sort MPI* devient pertinent lorsque la taille de problème est suffisante et que le nombre de processus reste adapté à la machine :

- pour petites tailles ($N = 10^5$), le gain est fragile et dominé par l'overhead,
- pour tailles intermédiaires/grandes ($N = 5 \times 10^5, 10^6$), les configurations $p = 3$ ou $p = 4$ donnent les meilleurs compromis,
- l'équilibrage des buckets est globalement bon (CV_b faible), ce qui valide la stratégie de sampling/splitters,
- la communication reste maîtrisée pour $p \leq 4$, cohérent avec les limites matérielles de la machine testée.

Les métriques retenues (T_{par} , S_p , E_p , Comm%, Moved%, $LB_{avg/max}$, CV_b) sont suffisantes pour justifier de manière professionnelle **où** le temps est dépensé, **pourquoi** la scalabilité se limite, et **quand** l'approche parallèle dépasse Quicksort.