TD 03 – Parallélisation du Bucket Sort

Luiza Gonçalves Soares
École Nationale Supérieure des Techniques Avancées – ENSTA Paris
Palaiseau, France
luiza.goncalves@ensta-paris.fr

1 Objectif

L'objectif de ce TD est d'implémenter une version parallèle de l'algorithme *Bucket Sort* à l'aide de MPI. Le programme est écrit en Python en utilisant la bibliothèque `mpi4py`.

2 Initialisation de MPI

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

Cette partie initialise l'environnement MPI. Le communicateur global `MPI.COMM_WORLD` permet la communication entre tous les processus. Chaque processus récupère son identifiant (`rank`) ainsi que le nombre total de processus (`size`).

3 Génération des données

```
N = 20
if rank == 0:
    data = np.random.random(N).tolist()
    print(f"Master (rank 0) generated: {data}\n")
else:
    data = None
```

Le processus de rang 0 joue le rôle de processus maître. Il génère un tableau de nombres réels aléatoires compris entre 0 et 1. Les autres processus ne génèrent pas de données et attendent la distribution.

4 Création et distribution des buckets

```
if rank == 0:
    buckets = [[] for _ in range(size)]
    for number in data:
        index = int(number * size)
        if index == size:
            index = size - 1
        buckets[index].append(number)
else:
    buckets = None
```

Le processus maître répartit les valeurs dans plusieurs *buckets*. Chaque bucket correspond à un intervalle de valeurs. L'indice du bucket est calculé à partir de la valeur du nombre et du nombre total de processus.

```
local_bucket = comm.scatter(buckets, root=0)
```

La fonction `MPI.scatter` envoie un bucket à chaque processus. Ainsi, chaque processus reçoit une partie différente des données.

5 Tri parallèle

```
local_bucket.sort()
print(f"Rank␣{rank}␣sorted␣its␣bucket:␣{local_bucket}")
```

Chaque processus trie localement son bucket de manière indépendante. Cette étape correspond à la partie parallèle de l'algorithme.

6 Rassemblement des résultats

```
all_sorted_buckets = comm.gather(local_bucket, root=0)
```

Les buckets triés sont renvoyés au processus 0 à l'aide de la fonction `MPI.gather`.

```
if rank == 0:
    final_table = []
    for b in all_sorted_buckets:
        final_table.extend(b)
    print(f"\nFinal␣sorted␣table␣gathered␣on␣Rank␣0:␣{
        final_table}")
```

Le processus maître concatène les buckets dans l'ordre des rangs afin d'obtenir le tableau final trié. Cette concaténation est correcte car chaque bucket contient des valeurs appartenant à un intervalle disjoint.

7 Conclusion

Cette implémentation respecte les étapes demandées dans l'énoncé :
— génération des données par le processus 0,
— distribution des données entre les processus,
— tri effectué en parallèle,
— rassemblement du résultat final sur le processus maître.

Elle illustre l'utilisation de MPI pour la parallélisation d'un algorithme de tri.