Programmation sur architectures parallèlles

8INF856 - Devoir 2

Étudiants : **Lapu Matthias, Pellevoizin Jules, Guebel Samuel**

1 Nom d'usager et mot de passe

— Nom d'usager : mpiuser18
— Mot de passe : bwQyjfZ

2 Résultat des tests et analyse

Afin d'analyser notre programme, nous avons lancé un "benchmark" en fonction de la taille du tableau ainsi que du nombre de threads. Pour ce faire, nous faisons varier le nombre de threads ainsi que le nombre de valeurs afin de trouver à partir de quelle valeur de n les algorithmes parallèles sont plus efficaces que l'algorithme séquentiel. Nous obtenons les résultats moyens suivants :

Nombre de thread	**Taille du tableau où pthread devient plus rapide**	**Pour OpenMP**
2	256	8
4	512	512
8	4096	2048
16	8192	1024
24	8192	65536
48	16384	1024

On observe qu'OpenMP gère moins efficacement les cas avec un grand nombre de threads.

En utilisant l'algorithme séquentiel et en ayant fixé $n = 100000$, nous obtenons un temps moyen de 0.026164 s. Pour les mêmes valeurs de n et pour les algorithmes parallèles utilisant pthread et OpenMP, nous obtenons les résultats moyens suivants :

Comparaison des différents algorithmes en fonction du nombre de threads			
Algorithme	Nombre de threads	Valeur de n	Temps d'exécution
pthread OpenMP	2	100000	0.015622 s 0.024946 s
pthread OpenMP	4	100000	0.008017 s 0.018750 s
pthread OpenMP	8	100000	0.005285 s 0.018998 s
pthread OpenMP	16	100000	0.005226 s 0.020818 s
pthread OpenMP	24	100000	0.005324 s 0.020852 s
pthread OpenMP	48	100000	0.005306 s 0.019490 s

Cela confirme bien les résultats du "benchmark". Ainsi, pour un n assez grand, les 2 algorithmes sont plus rapides que l'algorithme séquentiel dès 2 threads.

2.1 Algorithme

Pour les deux algorithmes parallèles nous avons calculé une **profondeur** de récursivité. A partir de cette profondeur le code arrête de créer des threads. Nous la calculons en faisant le logarithme en base 2 du nombre de thread maximal utilisé.

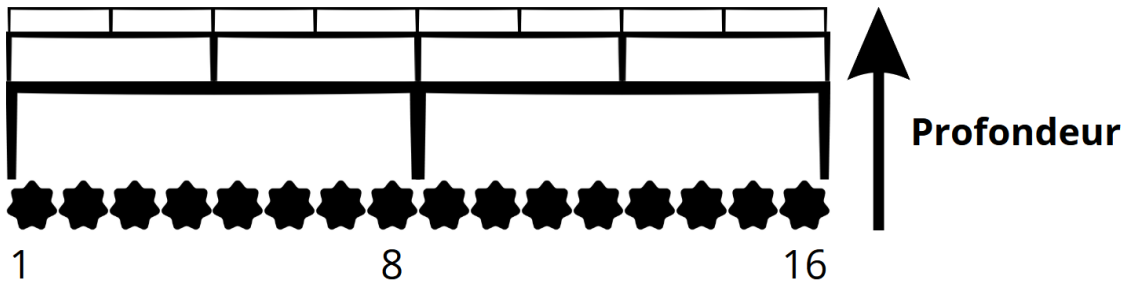

Figure 1 - Exemple avec une liste de taille 16

Sans cette limite de récursivité et en imposant une limite de nombre de threads, le programme pouvait ne pas s'arrêter. En effet, le principe du tri fusion est d'atteindre la comparaison de seulement deux éléments de la liste. Or quand la taille de la liste est trop grande le programme doit redevenir séquentiel dans chaque thread. Le tri fusion consiste à scinder la liste en deux récursivement il est donc évident que $t = d^2$ où d est la profondeur atteinte avec un nombre de threads égal à t.

3 Compilation

Pour notre projet, nous avons utilisé **Make**. Dans le répertoire source, il est donc possible de lancer facilement les commandes indiqués dans l'énoncé.

```
make
```

Permet de lancer les commandes demandées par l'énoncé.

```
make benchmark
```

Permet de lancer le programme pour pouvoir déterminer à partir de quelle valeur de N, le temps parallèle est inférieur au temps séquentiel. Indiquant donc que le temps parallèle est meilleur. Ces valeurs sont des moyennes et ont été représentés dans la partie 2.

```
make array
```

Cette commande permettra de créer un tableau de taille N de taille 10, 1000 et 1000000. Nous n'avons pas testé avec la taille de 2000000000 car celle-ci est beaucoup trop lourde pour nos machines ainsi que pour le cluster MPI. Nous n'avons donc pas pu créer de fichier de cette taille.

```
./src/create_array.sh <size_of_array>
```

Permet de créer un fichier, où il y aura le nombre d'élément indiqué en paramètre dans le fichier séparé d'un espace. Le premier chiffre sera la taille totale du tableau.

Afin de compiler, nous avons utilisé différents "flags". En effet, il était nécessaire de rajouter le "flag" -fopenmp pour l'entièreté des fichiers contenants des algorithmes étant donné qu'il était nécessaire de récupérer le temps d'exécution avec omp_get_wtime. Pour compiler les fichiers, nous avons originellement utilisé ces commandes :

```
gcc -c ./src/utils.c -o ./bin/utils.o
gcc ./bin/utils.o ./src/d2s.c -fopenmp -o ./bin/d2s
gcc ./bin/utils.o ./src/d2p.c -lpthread -fopenmp -o ./bin/d2p
gcc ./bin/utils.o ./src/d2omp.c -fopenmp -o ./bin/d2omp
```

Cependant, afin de respecter l'énoncé, nous avons finalement enlevé "utils" de la ligne de compilation, mais nous l'avons gardé dans notre dossier de rendu. Finalement, pour compiler nous utilisons donc les commandes suivantes :

```
gcc  ./src/d2s.c -fopenmp
gcc  ./src/d2p.c -lpthread -fopenmp
gcc  ./src/d2omp.c -fopenmp
```