Travaux Pratique 3

Contexte : Les données indiquées dans ce rapport ont été obtenues avec un Processeur Ryzen 5 6 cœurs et 12 processeurs logiques sous Windows. Tous les temps de calcul sont en microsecondes.

Exercice 1 : STL MAP

J'ai testé la fonction transform de la STL avec les stratégies d'exécution seq et par_unseq. J'ai obtenu les temps suivants :

Size	TseqUnaire	TparUnaire	EfficacitéUnaire	TseqBinaire	TparBinaire	EfficacitéBinaire
1000	0	2	0	0	4	0
10000	1	4	0,020833333	0	12	0
100000	11	12	0,076388889	10	29	0,028735632
1000000	115	76	0,126096491	107	160	0,055729167
10000000	4474	4532	0,082266843	5905	6073	0,081028048
100000000	46263	44555	0,086527887	59445	60103	0,082421011

On constate que dans l'ensemble, la stratégie séquentielle est la plus intéressante pour les calculs réalisés car elle est généralement plus rapide. La parallélisation n'est pas intéressante, l'efficacité est très faible. Je suppose que c'est parce que l'opération effectuée (mettre au carré et additionner deux nombres) est très simple et donc très rapide en séquentielle. Le coût de la parallélisation avec l'instanciation des threads et la synchronisation n'est pas amorti. Si l'opération était plus compliquée, je pense que l'efficacité augmenterait fortement.

Exercice 2 : STL REDUCE

Temps obtenu avec la fonction reduce :

Size	Tseq	Tpar	Efficacité
1000	0	2	0
10000	1	4	0,020833333
100000	11	13	0,070512821
1000000	117	65	0,15
10000000	3548	2858	0,103452298
100000000	35489	28002	0,10561448

On voit clairement que l'efficacité est légèrement meilleure que pour la transform mais reste faible et donc la parallélisation n'a toujours pas beaucoup d'intérêt dans le cas de cette opération.

Temps obtenu avec la fonction transform suivi par une fonction reduce :

Size	Tseq	Tpar	Efficacité
1000	0	8	0
10000	6	26	0,019230769
100000	71	105	0,056349206
1000000	2101	1965	0,089100933
10000000	27115	25179	0,089740789
100000000	277000	261000	0,08844189

L'efficacité est très faible, on retrouve des résultats proches de ceux de la transform.

Temps obtenu avec la fonction transform_reduce :

Size	Tseq	Tpar	Efficacité
1000	0	2	0
10000	3	13	0,019230769
100000	32	30	0,088888889
1000000	247	76	0,270833333
10000000	5860	2855	0,17104495
100000000	58504	29143	0,16729003

Ici, l'efficacité est meilleure qu'avant, sans pour autant être très élevé. On constate aussi que l'algorithme est bien plus rapide que d'utiliser les fonctions transform et reduce. Je suppose que c'est parce qu'il n'a pas besoin de stocker le résultat de transform. Dès qu'un résultat de transform est calculé, il peut être utilisé pour le reduce.

Exercice 3 : TRANSFORM

Pour les exercices suivants, j'ai utilisé les pools de threads et analysé les temps de calculs des patrons implantés avec une stratégie par bloc et une stratégie par modulo.

Size	TparUnaireBloc	TparBinaireBloc	TparUnaireModulo	TparBinaireModulo
1000	88	89	76	98
10000	94	102	87	96
100000	99	110	152	176
1000000	221	295	1278	1336
10000000	4594	6052	20397	31139
100000000	47325	63824	516000	735000

En exécutant mon programme plusieurs fois, j'ai pu observer des différences de temps de calcul assez grandes pour des tailles assez basses (environ 100000 et moins). Le temps peut alterner de 20 à 100 µs. Tout d'abord, on peut observer que la stratégie par bloc semble plus intéressante que la stratégie modulo, surtout pour des valeurs très grandes. Ensuite, on voit que les temps d'exécution pour la stratégie par bloc sont assez proches des valeurs obtenues pour l'exercice 1, avec la transform de la STL.

Exercice 4 : GATHER, SCATTER

Size	TparGatherBloc	TparScatterBloc	TparGatherModulo	TparScatterModulo
1000	102	108	92	109
10000	85	98	103	96
100000	117	108	230	177
1000000	276	339	1529	1474
10000000	9471	10872	49734	50502
100000000	97815	113000	722000	787000

On remarque la même chose qu'à l'exercice précédent, la stratégie par bloc est plus rapide que la stratégie par modulo.

Exercice 5 : REDUCE

Size	TparBloc	TparModulo
1000	105	95
10000	94	99
100000	181	214
1000000	1620	1998
10000000	16807	23579
100000000	158000	281000

Pour le reduce, la stratégie par bloc est toujours plus efficace pour les grandes tailles de données. Cependant, on voit clairement une différence notable entre le temps d'exécution de mon algorithme et celui de la STL.

Conclusion

On peut conclure des données que globalement la parallélisation n'est pas forcément intéressante pour des tâches simples et rapides. De plus, dans le cas où la parallélisation est importante, la STL fournit déjà un certain nombre d'outils performants, en tout cas, plus performant que ce que je suis capable d'implanter.