TD2 - OpenMP Cartographie et Réduction

Ce TD a consisté en l'optimisation d'un programme de calcul de comptage d'occurences de lettres dans une matrice de lettres aléatoire à l'aide d'OpenMP. Les résultats des temps d'exécution de ces programmes et l'évaluation de performance résultante sont résumés ci-dessous.

L'ordinateur utilisé pour l'évaluation de performances est un des nœuds « dahu » du site de Grenoble de Grid'5000 :
CPU : 2x Intel Xeon Gold 6130 à 16 cœurs/CPU, 192 Gio de RAM, Stockage : 240 Go SSD + 480 Go SSD + 4.0 To HDD, Réseau : 10 Gbps + 100 Gbps Omni-Path.

Dans les données de passage à l'échelle fort, la valeur avec 0 cœur correspond en réalité à une compilation sans l'option « -fopenmp », donc non parallélisée et s'exécutant sur un seul cœur.

La totalité de la réalisation du TD peut être retrouvée sur le dépôt GitHub public : https://github.com/Pyhro36/TD2-OpenMP. La branche « master » a été la branche de travail. Les branches « question21, question22, question23 ... » sont les branches de rendu décrites ci-dessous. Les fichiers et méthodes ayant servis à l'évaluation des performances sont décrits dans le fichier « README.md » dans les branches correspondantes.

2.1 Parallel for

Le code pour chaque question de cette partie est contenu sur une branche correspondante dans le dépôt GitHub : « question2X » pour la **question 2.X** (**question 2.1** : question21,... **question 2.4** : question24,... **question 2.7** : question27). Les fichiers et méthodes concernés par la réponse à la question sont décrits dans le fichier « README.md ». Il suffit donc de faire un checkout de chaque branche pour accéder au code, dans le fichier « main.cpp ».

Pour la **question 2.4**, le code utilisé avec OpenMP dans la méthode « parallelLetterCount(char** mat, int height, int width, int* result) » pour faire le calcul sur le problème complet a été choisi car il est fonctionnellement correct d'après la norme OpenMP et permet expérimentalement de résoudre de manière la plus optimisée le problème. Idem pour la **question 2.6**.

Pour les **question 2.5** à **question 2.7**, on nomme « hauteur » (« height ») le nombre de lignes et « largeur » (« width ») le nombre de colonnes de la matrice générée aléatoirement et dont on veut compter les occurrences de lettres. On nomme « sous-hauteur » (« underHeight ») le nombre de lignes et « sous-largeur » (« underWidth ») le nombre de colonnes des sous-matrices issues du découpage du problème en sous-problèmes.

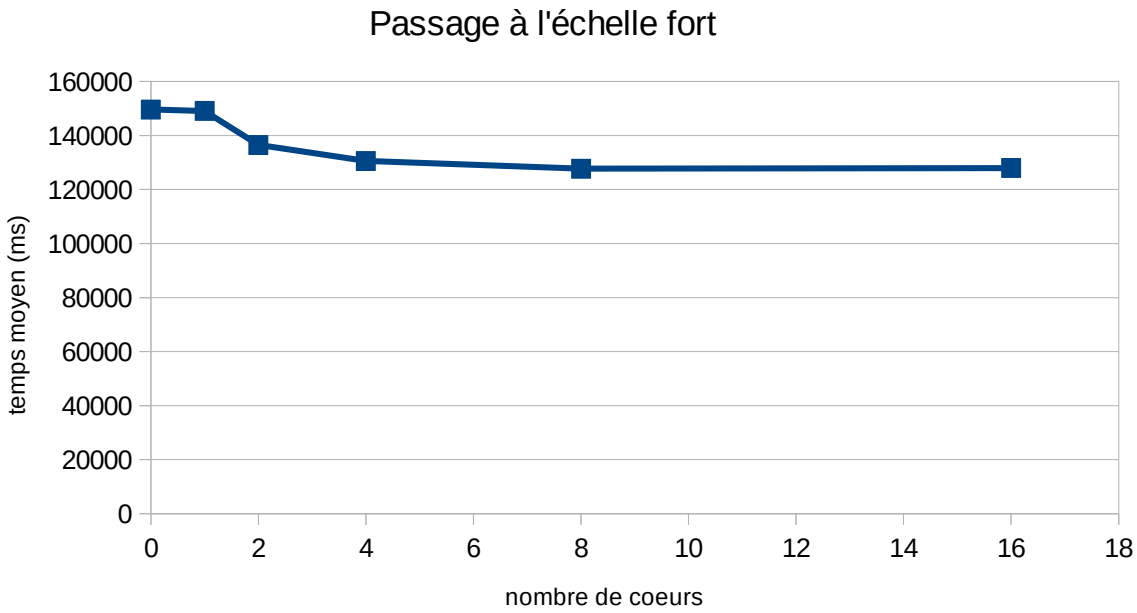
Pour la **question 2.7**, pour les passages à l'échelle, les rapports entre chacune des dimensions des matrices et celles des sous-matrices ont été choisis dans le script « perf_eval.sh » de manière à optimiser au plus le calcul, d'après les expériences de tests amont.

Pour l'évaluation des performances, on fait varier le nombre de coeurs et en plus les dimensions

de la matrice et des sous-matrices pour le passage à l'échelle faible. Pour chaque paramétrage, on effectue 7 exécutions et on en récupère dans les tableaux ci-dessous les temps d'exécution (t1 à t7). On fait ensuite la moyenne pour chaque paramétrage et on trace ainsi les graphes de passage à l'échelle fort et faible.

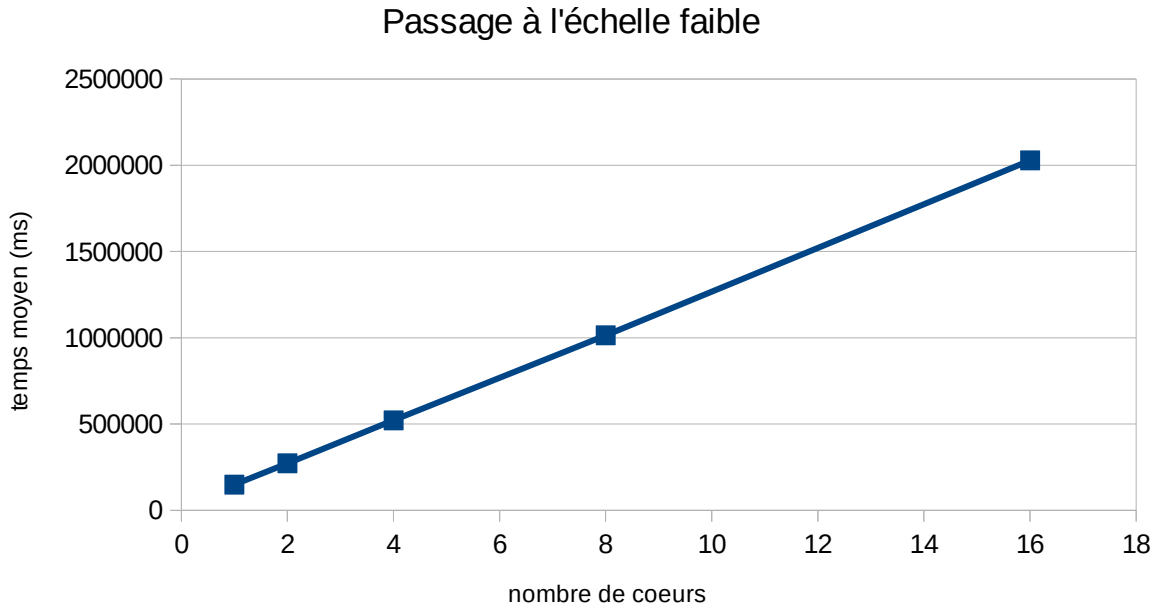
Passage à l'échelle fort

Nombre de coeurs	Hauteur	Largeur	Sous-Hauteur	Sous-Largeur	t1(ms)	t2(ms)	t3(ms)	t4(ms)	t5(ms)	t6(ms)	t7(ms)	Moyenne(ms)
0	100	100000	10	1000	147911,482	148031,767	151868,856	154793,355	148526,517	147825,549	148077,743	149576,467
1	100	100000	10	1000	148102,174	149871,684	149592,036	149417,245	147865,483	149155,624	149580,217	149083,494
2	100	100000	10	1000	136464,432	136361,439	136429,054	136837,918	136461,118	136569,284	136369,051	136498,899
4	100	100000	10	1000	130266,36	130834,299	130386,896	130498,364	130591,386	131188,126	130182,903	130564,047
8	100	100000	10	1000	127720,712	128052,467	127717,089	127853,91	127671,441	127509,013	127394,863	127702,785
16	100	100000	10	1000	128733,759	127795,776	126181,576	132899,34	126605,234	127127,782	126314,381	127951,121



Passage à l'échelle faible

Nombre de coeurs	Hauteur	Largeur	Sous-Hauteur	Sous-Largeur	t1(ms)	t2(ms)	t3(ms)	t4(ms)	t5(ms)	t6(ms)	t7(ms)	Moyenne(ms)
1	100	100000	10	1000	149247,78	149167,435	149417,153	148262,98	149134,482	149267,927	149113,175	149087,276
2	100	200000	10	2000	272296,138	272696,067	272278,733	272176,47	272523,011	272181,394	272371,016	272360,404
4	200	200000	20	2000	520771,016	520549,825	520684,264	518788,485	522894,115	524493,488	521335,98	521359,596
8	200	400000	20	4000	1013441,573	1017840,834	1013451,147	1011614,696	1014908,474	1015492,975	1012300,669	1014150,052
16	400	400000	40	4000	2030661,593	2018650,417	2026880,113	2030412,594	2026799,054	2030321,417	2031796,252	2027931,634



Conclusions

Globalement, on constate que la compilation avec OpenMP permet une amélioration des performances par rapport à une compilation normale non parallélisée.

Cependant, par exemple dans le cas des passages à l'échelle forts, on constate que l'amélioration des performances ne suit pas l'augmentation du nombre de coeurs. Le fait est qu'avec l'augmentation du nombre de cœurs, il y a un écart de plus en plus important au résultat attendu si le passage l'échelle fort était parfait (division par 2 du temps de calcul). Il est dû au temps de calcul nécessaire à la création, la destruction et la synchronisation des threads OpenMP, augmentant avec le nombre de cœurs, et très complexes à gérer dans le cas des nombreuses boucles imbriquées de l'algorithme présentant un grand nombre de tâches.

Avec le passage à l'échelle faible, on constate que la méthode par découpage ne semble pas adaptée via OpenMP au passage à l'échelle faible en particulier à cause de sa trop grande complexité.