Systèmes et Supports d'Exécution pour le Calcul Parallèle et Distribué

Rapport pour les TP2 && TP3

Master2 Systèmes Distribués Réseaux et Parallélisme

Charge de TP : M. Denis

Étudiant: Lou Ruding

TP2

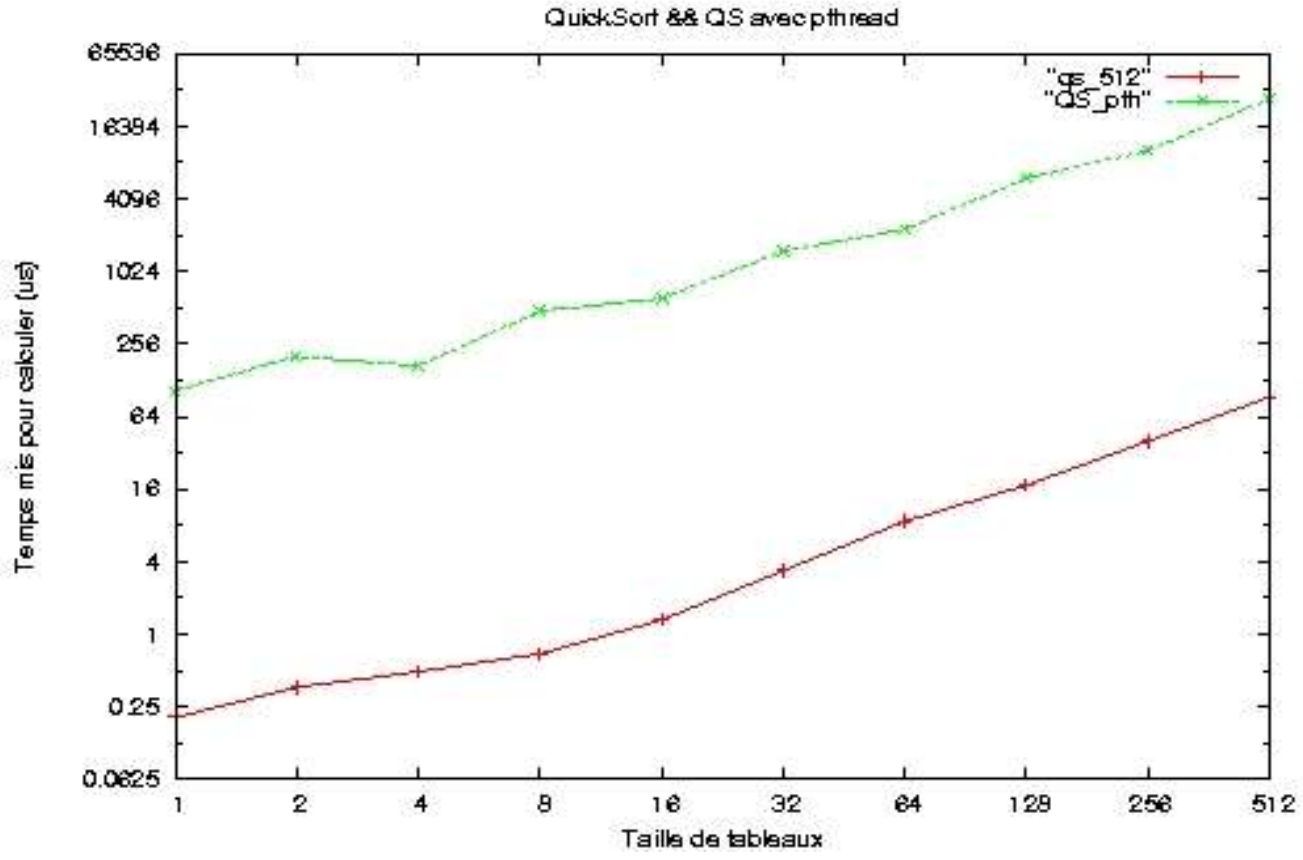
Question 2

Conclusion: Quand il y a trop de pthreads ont été créés,il y aurait un problème de "Erreur de segmentation".
Il y a une limitation de création de pthreads dans la machine en même temps.

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/Pour rendre/q2$ ./qs_thread
1               104.308
2               202.051
4               168.023
8               480.357
16              606.878
32              1499.99
64              2282.56
128             6038.17
256             10249.7
512             27659.6
Erreur de segmentation
```

La machine il s'arrête ici, sans finir le calcul.
Aussi quand la taille du tableau n'est pas très grande, le temps des créations des threads est très important par rapport le temps de calcul. Donc c'est moins intéressant de créer les threads pour les deux appels récursifs.

FIGURE 2-1



D'après l'image (2-1) ci-dessus, on voit très bien avec le pthread il ne peux pas trier les tableaux très grand, aussi avec les tableaux qu'il peut trier, le pthread a mis beaucoup plus de temps pour calculer. Donc c'est pas très rentable de l'utiliser.

Question 3

Pour contrôler le nombre des niveaux (n) de threads créés, quand n appartient à {n| 0 < n < 8 où n est un entier} il ressemble pertinentes. Parce que quand n > 8,il y a un problème de "Erreur de segmentation". Comme même ça dépend la taille de tableaux qu'on va traiter aussi, ici les tableaux sont inférieur de taille 1000000.

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q3$ ./niv_pth_qs
combien de niveaux de threads maximum( >0 ) créer pour trier ce
tableau :  9
1          96.7352
2          188.496
4          197.217
8          532.599
16          1293.03
32          2234.6
64          3117.42
128          11001.6
256          12872.2
512          11891.5
1024          18428.3
2048          24267.5
4096          22373.2
8192          23528.3
16384          24504.4
32768          34720.9
65536          64845.8
131072          218793
262144          777696
524288          3.00813e+06
bye
```

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q3$ ./niv_pth_qs
combien de niveaux de threads maximum( >0 ) créer pour trier ce
tableau :  10
1          110.69
2          165.893
4          287.961
```

```
8           553.128
16           1190.13
32           1731.63
64           4125.19
128           10432.9
256           20891.6
512           16817.3
1024           25573.1
Erreur de segmentation
```

Donc n = 9, c'est le max pour n.

Et pour la version en contrôlant le nombre (m) de threads j'ai trouvé que le nombre critique est 253.Donc les valeurs de m le plus pertinentes est { m | 0 < m < 254 où m est un entier}

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q3$ ./nbr_pth_qs
combien de threads maximum( >0 ) créer pour trier ce tableau :  252
1           139.94
2           114.466
4           168.073
8           293.393
16           422.117
32           1060.15
64           2299.62
128           5824.67
256           10315.9
512           24096.4
1024           31741
2048           40670.3
4096           50359.4
8192           54600.4
16384           93233.4
32768           105835
65536           191745
131072           319013
262144           859224
524288           3.06026e+06
bye
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q3$ ./nbr_pth_qs
combien de threads maximum( >0 ) créer pour trier ce tableau :  253
1           90.9772
2           118.215
4           118.452
8           288.39
16           612.59
32           895.364
```

```
64           2046.62
128           5967.17
256           10541.1
512           25583.7
1024           43349.4
2048           39915.9
4096           62285.7
8192           56314.2
16384           87068.1
32768           88326.4
65536           196056
Erreur de segmentation
```

Donc m = 253 c'est le max pour m.

Et à mon avis la version en contrôlant le nombre de threads équilibre mieux la charge. parce que le niveaux c'est très large, et ça peut-être que l'un coté de niveaux il y a beaucoup plus de threads que l'autre coté de même niveau. Aussi pour contrôler le nombre de threads,c'est beaucoup plus précis pour contrôler le thread.

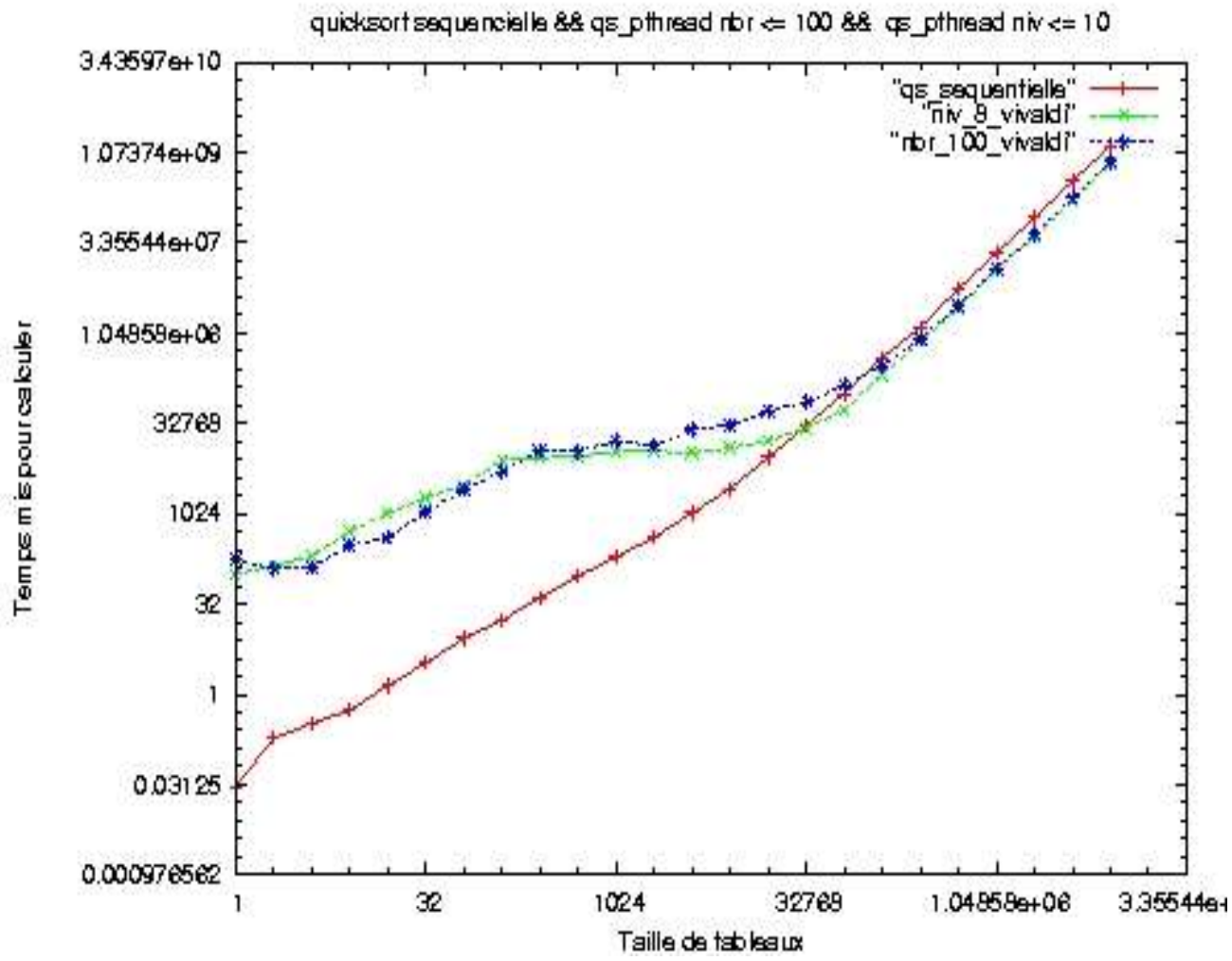
Et pour obtenir un bon parallélisme on n'a pas besoin de créer deux threads pour les appels récursifs. C'est-à-dire quand un thread s'occupe de la fonction quicksort() et il fini de mettre en place un pivot, il arrive le moment où il devais faire des appels récursifs pour trier les deux coté de ce pivot. Au lieu de créer deux nouveaux threads pour les deux appels, on ne crée qu'un thread qui va s'occuper l'un des deux, et on laisse le thread courant faire l'autre appel.

C'est ce que j'ai fait aussi,dans la question 2, vous nous demandez de laisser les threads s'occuper les appels récursifs, et pour la question 3 , limitation de nombre de threads aussi j'ai fais comme ça(ne créer pas deux nouveau threads). Sauf que pour la version contrôlant le nombre de niveaux , j'ai créé deux threads pour les deux appels chaque fois.

Par contre en m'amusant, j'ai fait une test sur le nbr 100 par hasard, et je trouve que c'est plus vite. Avant je donnais la nombre de threads maximum pour trouver le nombre de threads le plus grand où il peut supporter. Donc je met l'exécution de nb <= 100 dans le chemin suivant. j'ai fait un dessin comparer les 3 version (quicksort séquentielle, avec nbr de pthread <= 100 et avec niv de pthread < 9) , pour voir qui est le plus performante.

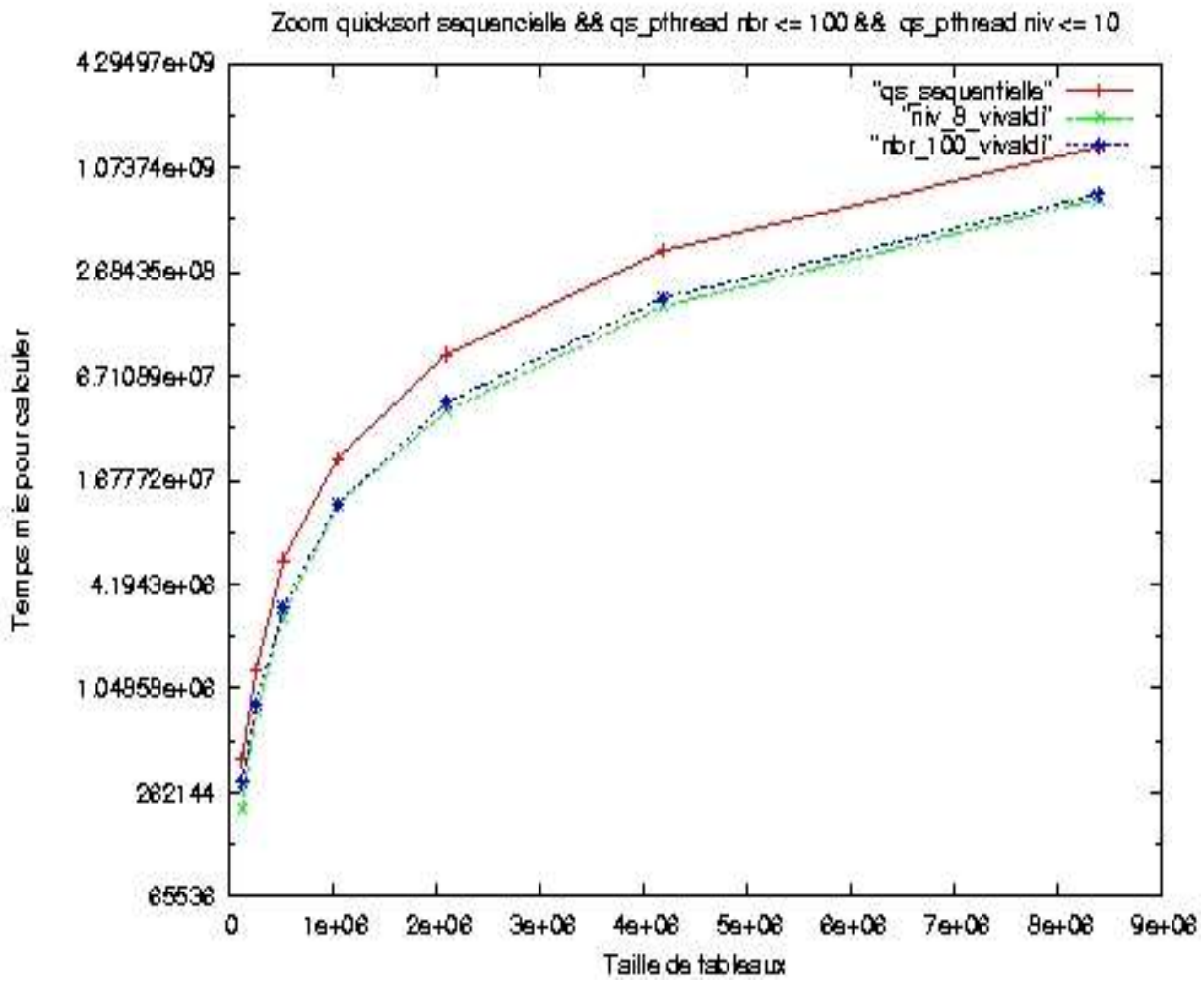
Voyons la Figure 3-1.

FIGURE 3-1



Voilà d'après la figure, on voit très bien qu'avec les tableaux de taille inférieur de 10000 , le quicksort séquentiel est plus efficace que les autres versions de parallélisation. Mais attention, quand la taille des tableaux est supérieure de 100000, les versions de parallélisation sont plus efficace, on peut voir le graphe juste au dessous (zoom sur la partie où les versions de parallélisation prennent moins de temps que quicksort séquentiel, l'axe de temps n'est pas en échelle logarithmique) FIGURE 2-1. Aussi on peut consulter sur les résultats de l'exécutions en suite.

FIGURE 3-2



Taille	QS séquentiel	Parallélisation	temps gain
131072	405899	298103	107796
262144	1.32115e+06	836733	484417
524288	5.77642e+06	3.07414e+06	2.70228e+06
1048576	2.24604e+07	1.22851e+07	1.01753e+07
2097152	8.87129e+07	4.69318e+07	4.17811e+07
4194304	3.55635e+08	1.89598e+08	1.66037e+08
8388608	1.41155e+09	7.55318e+08	6.56232e+08

Conclusion

D'après les graphes, on sait que on n'a pas travaillé sur la parallélisation pour rien, parce que on a vu que la parallélisation marche plus vite que les versions séquentielles.

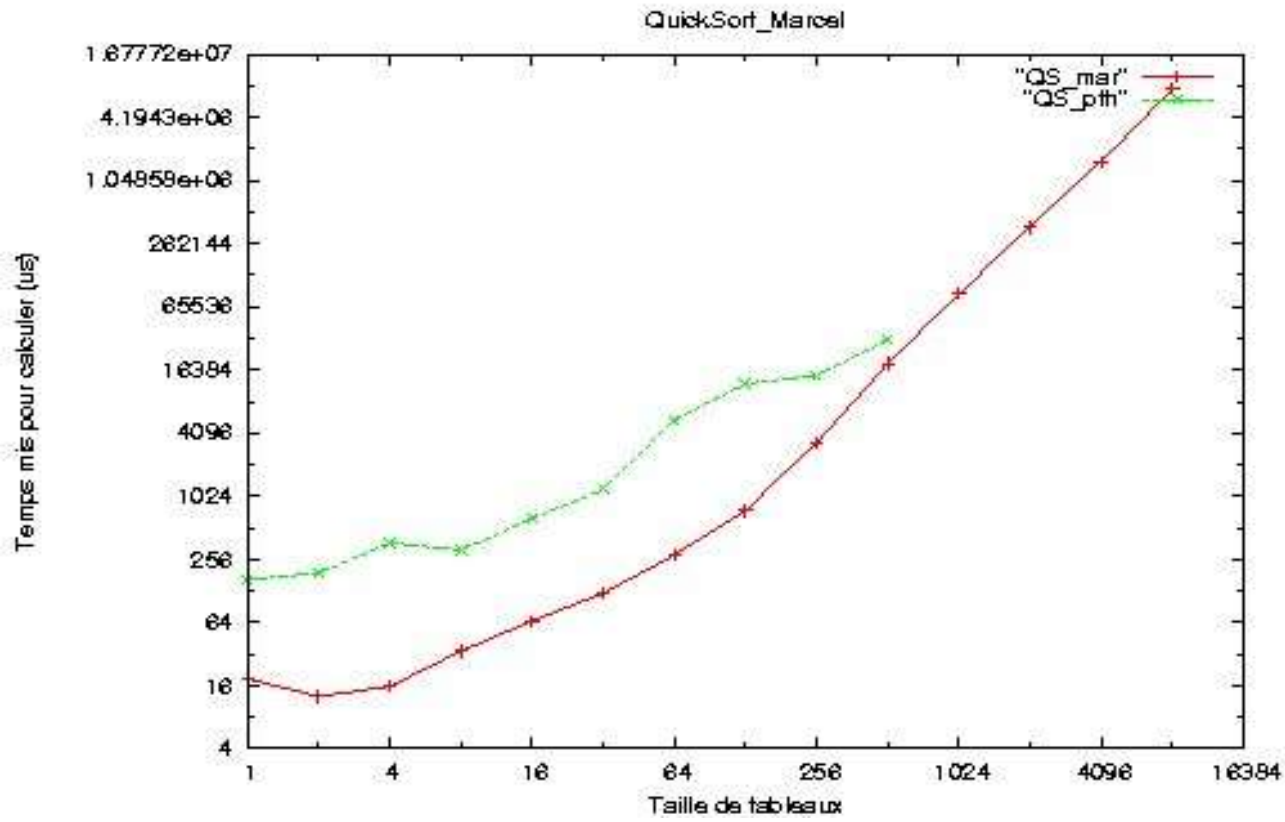
Question 4

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q4$ ./qsort-marcel
1               18.4153
2               12.5691
4               15.6855
8               33.8922
16              66.0167
32              121.092
64              281.047
128             744.608
256             3295.73
512             18859.9
1024            87529.1
2048            374462
4096            1.57693e+06
8192            7.8933e+06
```

D'abord pour la première version avec qui on ne limite pas du tout le pthread ni le marcel, je vois bien avec le graphe ci-dessous (Figure 4-1) que le marcel est beaucoup plus puissant. Pour trier les tableaux, marcel peut aller plus loin,c'est-à-dire que avec marcel il peux calculer jusqu'au tableau avec taille 8192. avec le thread on arrive à la taille 512.

Et aussi avec les tableaux de différentes tailles qui peuvent être triés par thread ,taille <= 512, le marcel a mis beaucoup moins de temps pour trier les tableaux que les threads. On peut voir dans le graphe , la ligne vert est faite pour pthread, il va moins loin, et aussi il prend plus de temps pour trier la même taille de tableau que le marcel (ligne rouge).

FIGURE 4-1



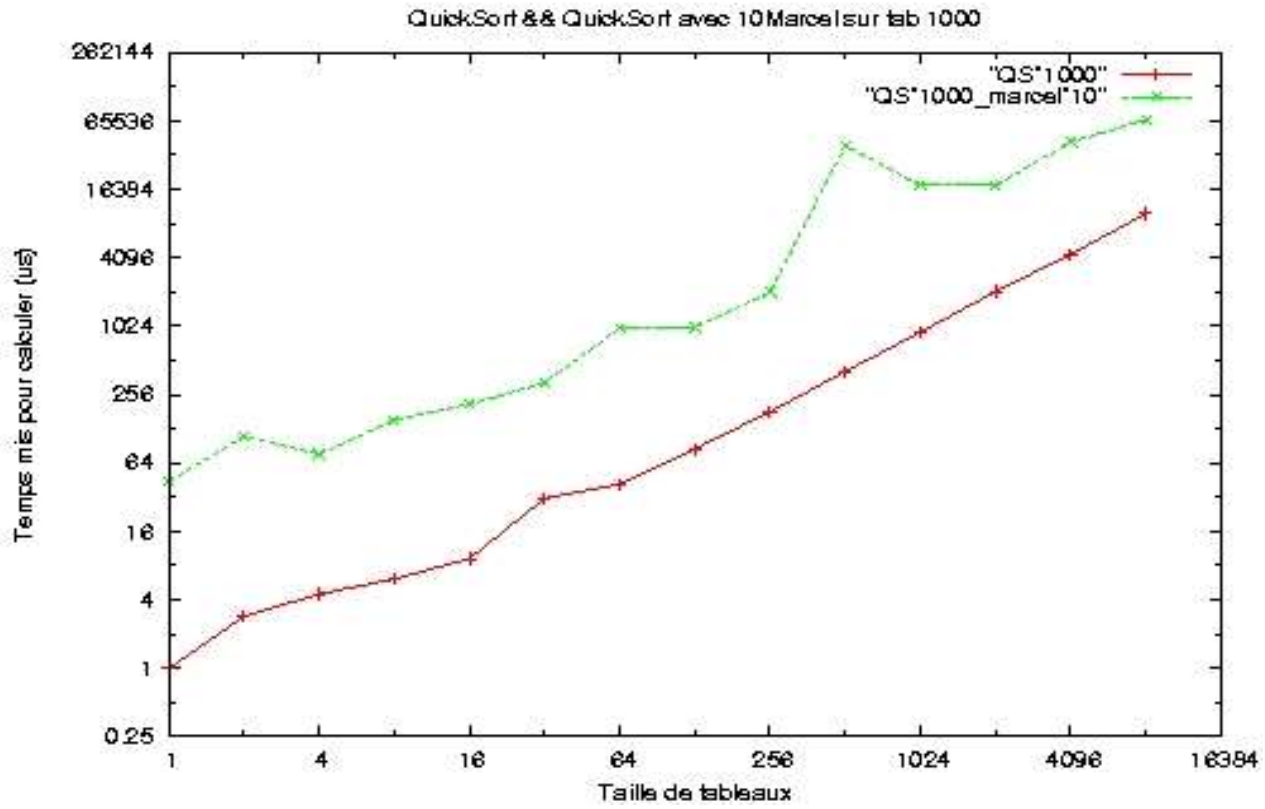
Et pour la version en contrôlant le nombre de marcels, j'ai obtenu des résultats plus représantatble, voici ci-dessous:

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q4$ ./nbr_mar_qs
la taille maximum de tableaux :10000
combien de marcels maximum( >=0 ) créer pour trier ce tableau :  0
1               0.987031
2               2.85562
4               4.47258
8               6.07879
16              9.14963
32              31.1627
64              41.7268
128             85.0411
256             180.301
512             408.015
1024            906.87
2048            2060.66
4096            4377.39
8192            9993.46
bye
```

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q4$ ./nbr_mar_qs
la taille maximum de tableaux :10000
combien de marcels maximum( >=0 ) créer pour trier ce tableau :  10
1               43.7007
2               109.694
4               76.3586
8               152.549
16              209.957
32              322.868
64              976.839
128             1001.62
256             2025.26
512             39987
1024            17660.9
2048            17738.9
4096            42485.1
8192            66682.7
bye
```

pour les deux tests j'ai fait un dessin aussi pour eux. Voir Figure 4-2. Le marcel ne tri pas plus vite les tableaux que le quicksort naïve. Mais on n'est comme même content de voir que par rapport le figure 2-1 au debut(question 2), la ligne de marcel est plus proche de la ligne quicksort naïve que la ligne pthread dans Figure 2-1.

FIGURE 4-2



Au cours de teste j'ai trouvé que avec marcel on ne peut pas trier les tableaux très grands, sinon il y a des problèmes comme

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q4$ ./nbr_mar_qs
la taille maximum de tableaux :1000000
combien de marcels maximum( >=0 ) créer pour trier ce tableau :
1000000
1               45.7713
2               58.1137
4               66.3627
8               148.995
16              207.699
32              375.489
64              831.009
128             1602.54
256             4923.07
512             7510.4
1024            24208.4
2048            51260.3
4096            116911
8192            418509

Unhandled exception STORAGE_ERROR: No space left on the heap in task
16692
FILE : source/marcel_alloc.c, LINE : 111
Abandon
```

Ça c'est un nouveau problème.

Quand je faisais marcher la version contrôlant le nombre de niveaux de marcels, il y a aussi des problèmes comme ci-dessus.

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/Pour rendre/q4$ ./niv_mar_qs
Taiile de tableaux maximum :1000000
combien de niveaux de marcels maximum( >=0 ) créer pour trier ce
tableau :  5
1               40.2343
2               115.588
4               111.058
8               260.91
16              284.331
32              514.813
64              777.637
128             1152.88
256             1332.43
512             785.472
```

```
1024            935.374
2048            1302.29
4096            1492.18
8192            2114.21
16384           5675.85
32768           18222.7
65536           56496.8
OOPS!!! Signal 11 catched on thread 0x3fe1fc40 (1102053384)
OOPS!!! current lwp is 2
OOPS!!! Entering endless loop so you can try to attach process to gdb
(pid = 16055)
OOPS!!! Signal 11 catched on thread 0x3fdbfc40 (675)
OOPS!!! current lwp is 2
OOPS!!! Entering endless loop so you can try to attach process to gdb
(pid = 16055)

Unhandled exception PROGRAM_ERROR in task 682
FILE : source/marcel_work.c, LINE : 28
Abandon
```

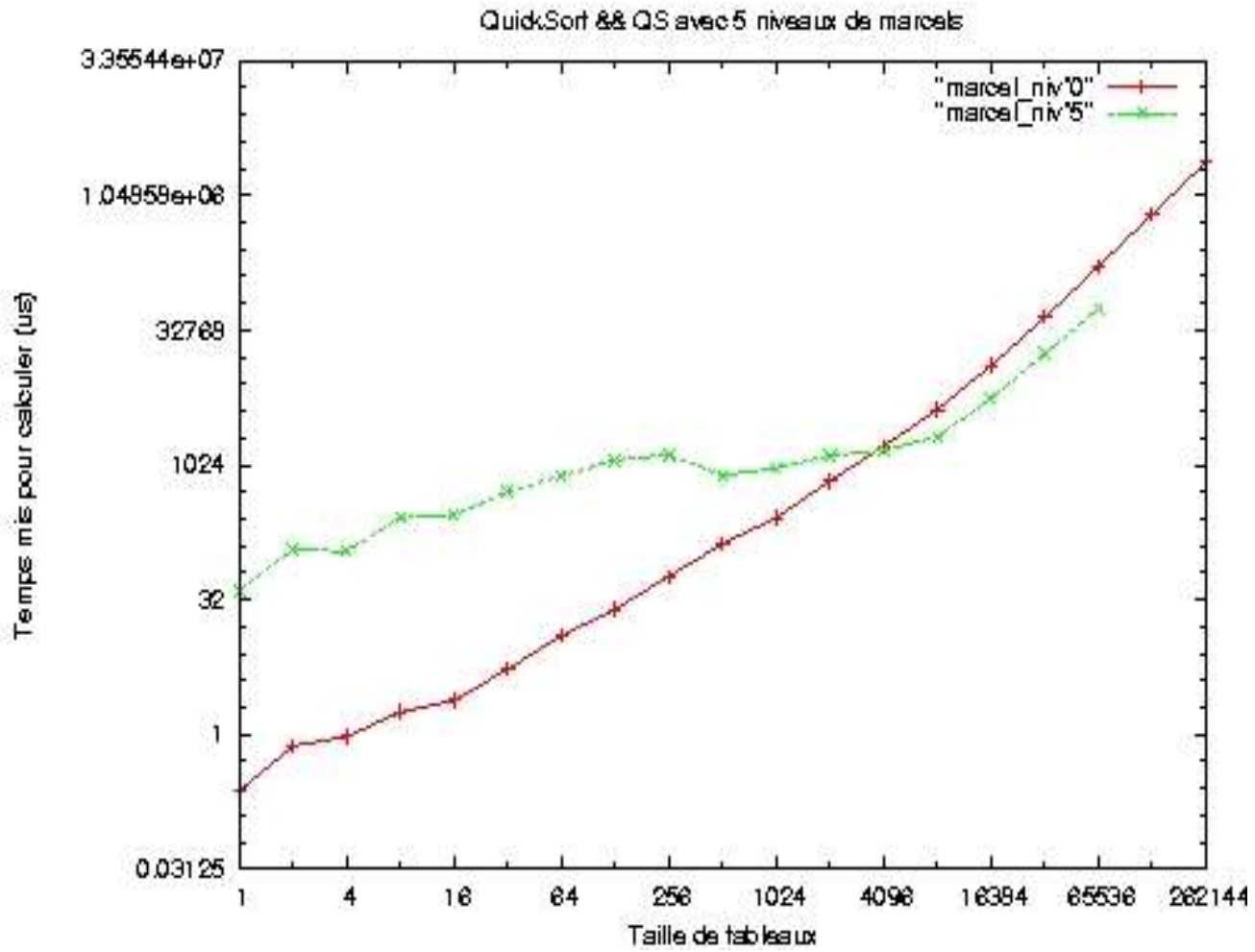
Après j'ai tester encore une fois avec taille maximum de tableaux 1000000, et 0(aucun) niveau de marcel.

```
rlou@vivaldi:~/master2/SSECPD/TP2_SSECPD/q4$ ./niv_mar_qs
Taiile de tableaux maximum :1000000
combien de niveaux de marcels maximum( >=0 ) créer pour trier ce
tableau :  0
1               0.233099
2               0.743006
4               0.946968
8               1.77556
16              2.41476
32              5.35036
64              12.7563
128             24.918
256             58.1419
512             134.807
1024            264.103
2048            675.504
4096            1657.29
8192            4265.28
16384           13234.2
32768           45370.9
65536           169757
```

```
131072          646828
262144          2.47837e+06
OOPS!!! Signal 11 catched on thread 0xbffdfc40 (-7136)
OOPS!!! Entering endless loop so you can try to attach process to gdb
(pid = 16125)
```

Il s'arrête ici. mais je susi content de voir la dessin(4-3) faite selon les deux teste ci-dessus. À la fin marcel il a mis moins de temps pour trier les tableaux (>4096) , donc marcel nous a montré sa puissance comme même.

FIGURE 4-3



TP3

Question 1

1.quel est le surcoût de la parallélisation par threads?

D'après les testes et à mon avis,le surcoût de la parallélisation par threads est le temps de la création et les temps pour mettre en route le mutex et pour déverouiller.

2.quel est le gain obtenu sur machine SMP ?

Sur les machine multiprocessor, la version avec les threads parallèles devait marcher plus vite, mais je n'ai pas réaliser, normalement ça sera deux fois plus vite que quicksort naif, parce que le hyperthreading permet de marcher deux threads simultanément. Mais dans dans le TP2,avec la graphe 3-2,qui est zoom sur la partie critique de la graphe précédent, je pense que si c'est sur une machine SMP, il faut que la taille soit supérieure de certaine taile, (à mon avis > 100000, on pourra gagner du temps pour trier. Et aussi les données juste sous la graphe 3-2 nous montre la puissance de parallélisation sur SMP à partir certaine taille de tableaux.

3.conclusions ?

les testes sur les machines multiprocessor normalement marchait beaucoup mieux que sur la machine monoprocessor. Mais ça n'a pas marché jusqu'au bout, avec la taille de tableaux inférieur de 1000. Je pense que c'est peut-être parce que deux threads avancent simultanément pour trier un même tableau, donc il y a un problème.

Question 2

Mise au point

Comment gère-t-on la dépendance entre les tâches ?

Quant à la dépendance entre les tâches, la dépendance est que une tâche, doit être faite après quelques tâche, par exemple une tâche de trier un tableau dépend une l'autre tâche de trier un sous-tableau. Mais pour notre cas ,je crois on ne s'occupe de dépendance entre les tâches, parce que les tâches de sous-tableaux sont générées après les traitements de leur tableau-parant. Donc il n'y a aucun souci pour la dépendance.

Comment initialise-t-on la pile? Comment détecte-t-on la fin du quicksort

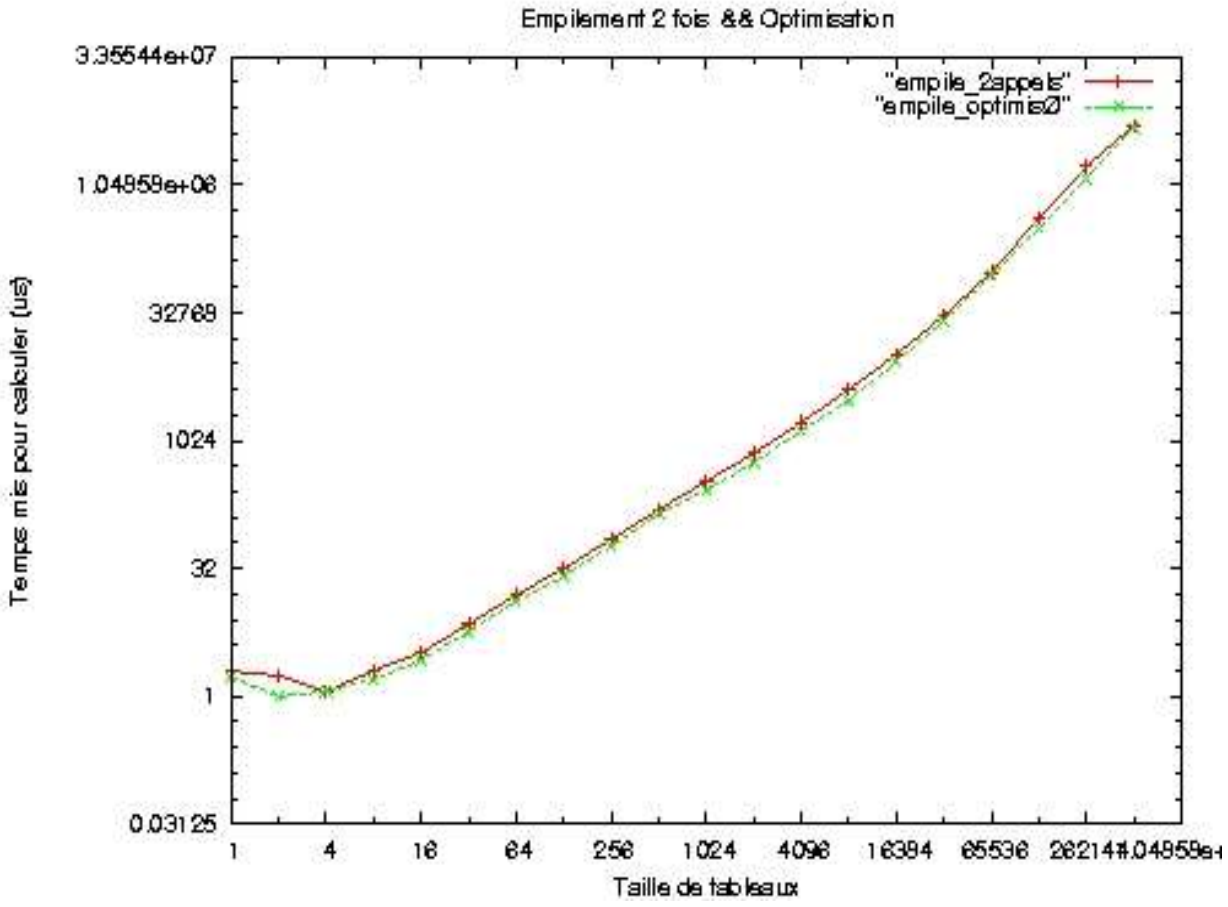
Pour initialise la pile des tâche, il suffit de créer un tableau,

et de créer un itérateur qui point toujours sur la tâches mise le plus récent du pile. Donc au début il point sur la position 0. Quand il traite un tableau, lorsqu'il finit une permutation d'un pivote, il peux bouger l'itérateur plus loin de position 0 et mettre les deux cotés(pour trier après) de ce pivote dans le pile. Donc quand il veut traiter les tâches dans le pile, il peut regarder ce que le itérateur pointe et il la prend, et bouge l'itérateur vers la position 0. Quand l'itérateur point position 0, c'est-à-dire il n'y a plus de tâche à fair.

Optimisation de la récursion terminale

J'ai fait un dessin pour comparer la version avec lequel on met 2 appels récursifs dans le pile, et la version avec lequel on met 1 appel récursif dans le pile et fait l'autre appel directement.

FIGURE 2-1



D'après le dessin(Figure 2-1), on voit qu'on a gagné juste un peu de temps de calcul.

Et j'ai dessiner un dessin pour comparer la version sans optimisation, les versions multi-thread et la version séquentielle.

FIGURE 2-2

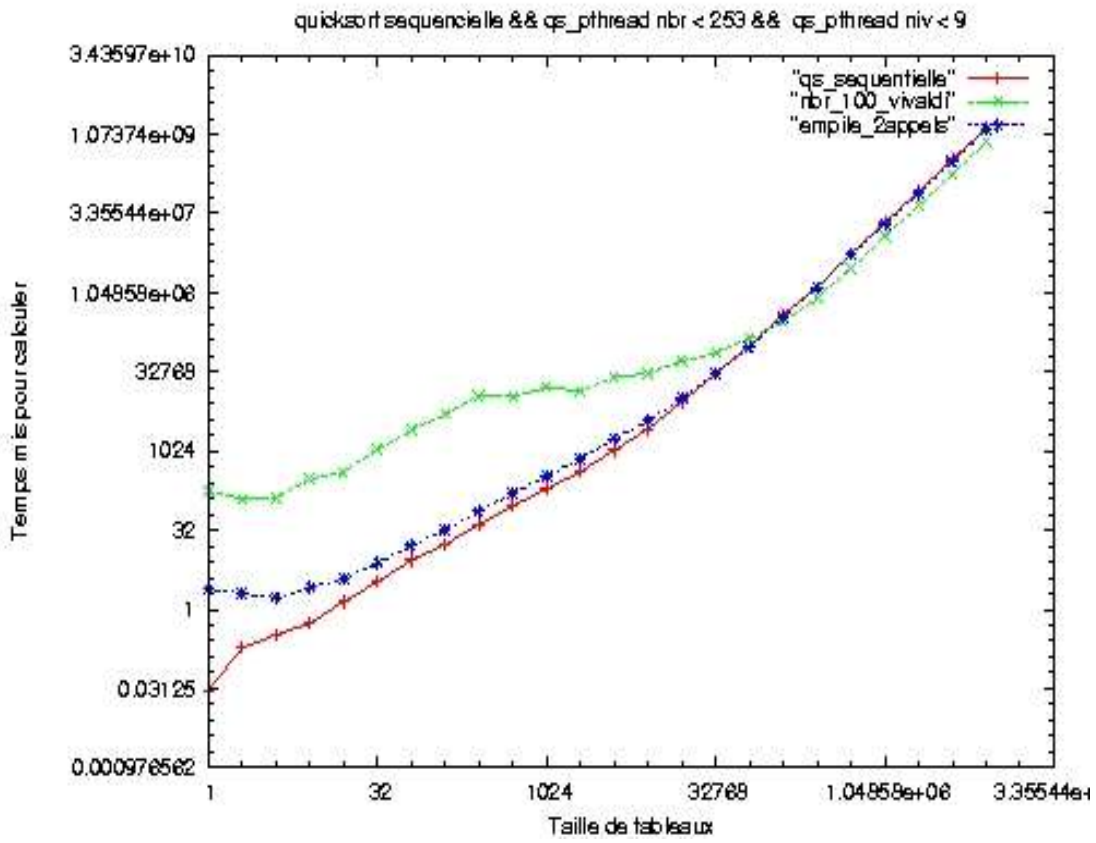
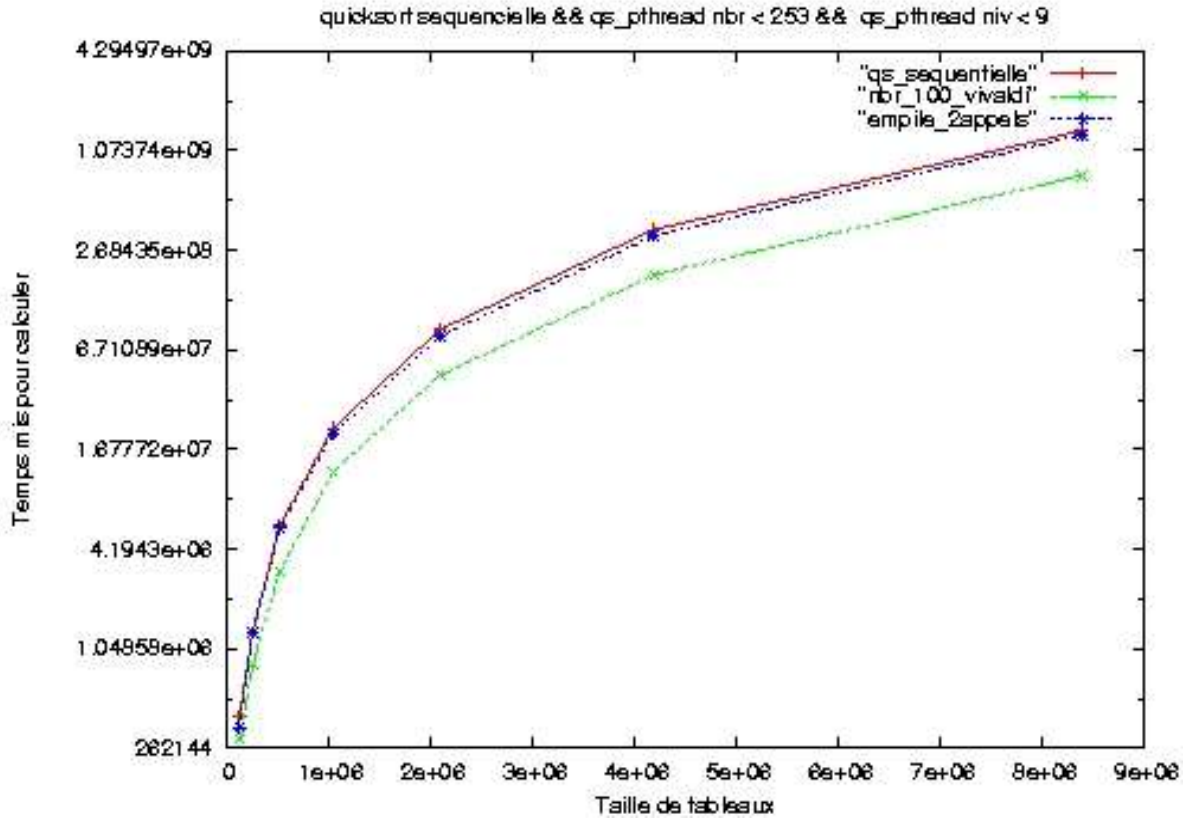


FIGURE ZOOM 2-2



Conclusion

D'après la comparaison avec les trois versions, je peut insister que la parallélisation est comme même plus puissante que les autres. Mais avec le façons de empiler les tâches, ça a l'air mieux que version séquentielle. Donc après, on va passer à la version de empilement et dépilement des taches en parallélisation. Comme je suis prévenu dans le TP, donc je suis motivé de passer à la section suivante.

Question 3

Pile de travail commune

Comment gère-t-on les dépendances ? La tâche prête à exécuter est-elle forcément celle en sommet de pile ?

Pour cette fois-ci , comme avant je ne crois pas on devais s'occuper sur les dépendances. Parce que les tâches empilé dans le pile sont mis après la taches de leurs parant. Donc maintenant il y a deux threads qui cherches des travailles dans le pile , et les traites, et aussi ils empilent des taches descendant.

figure a

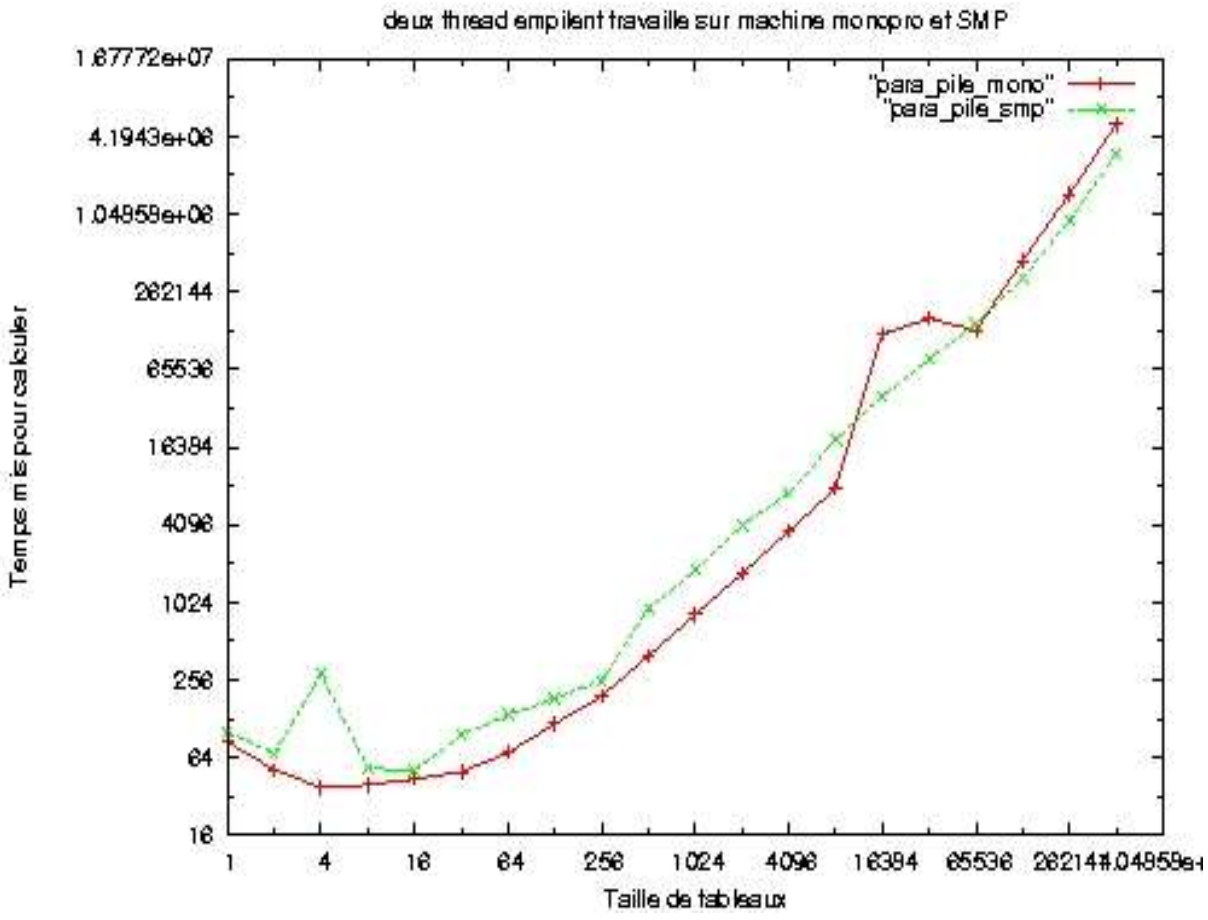
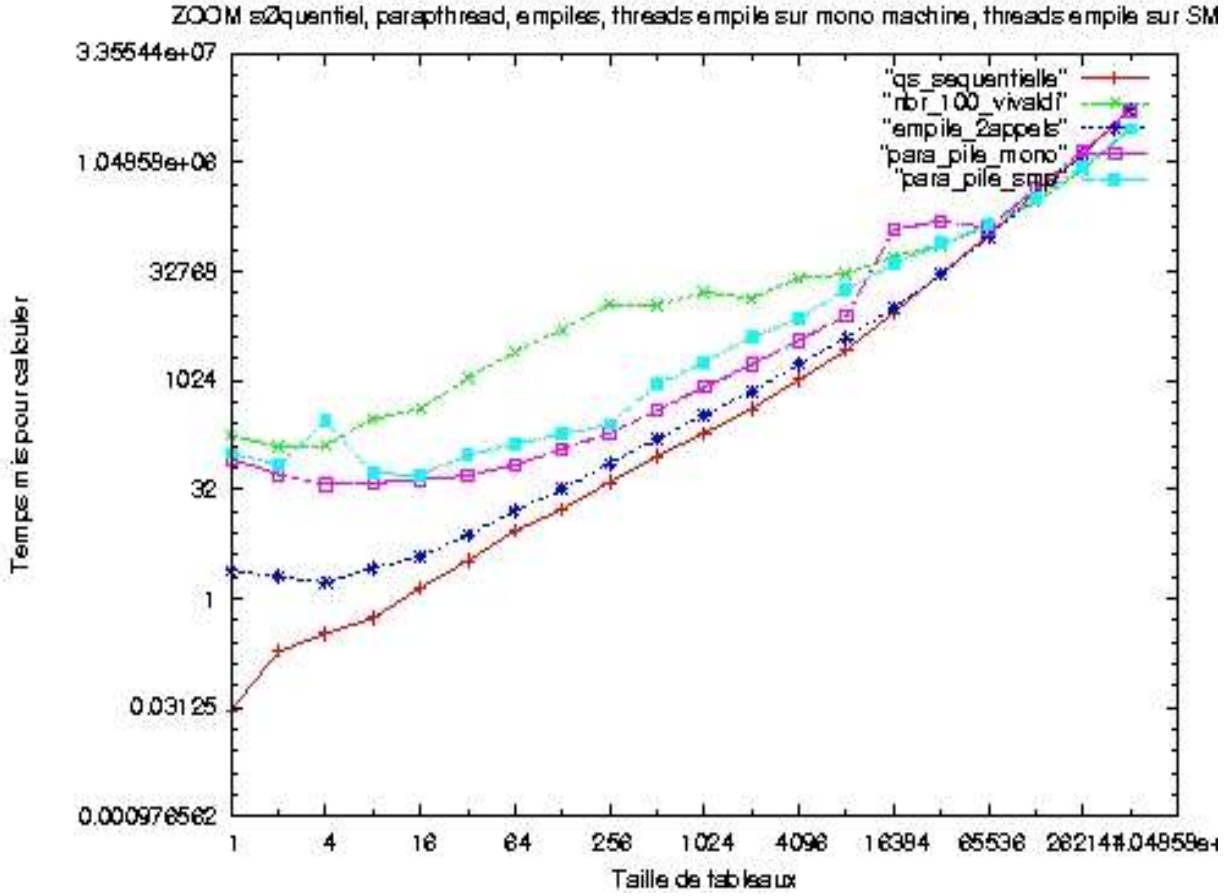
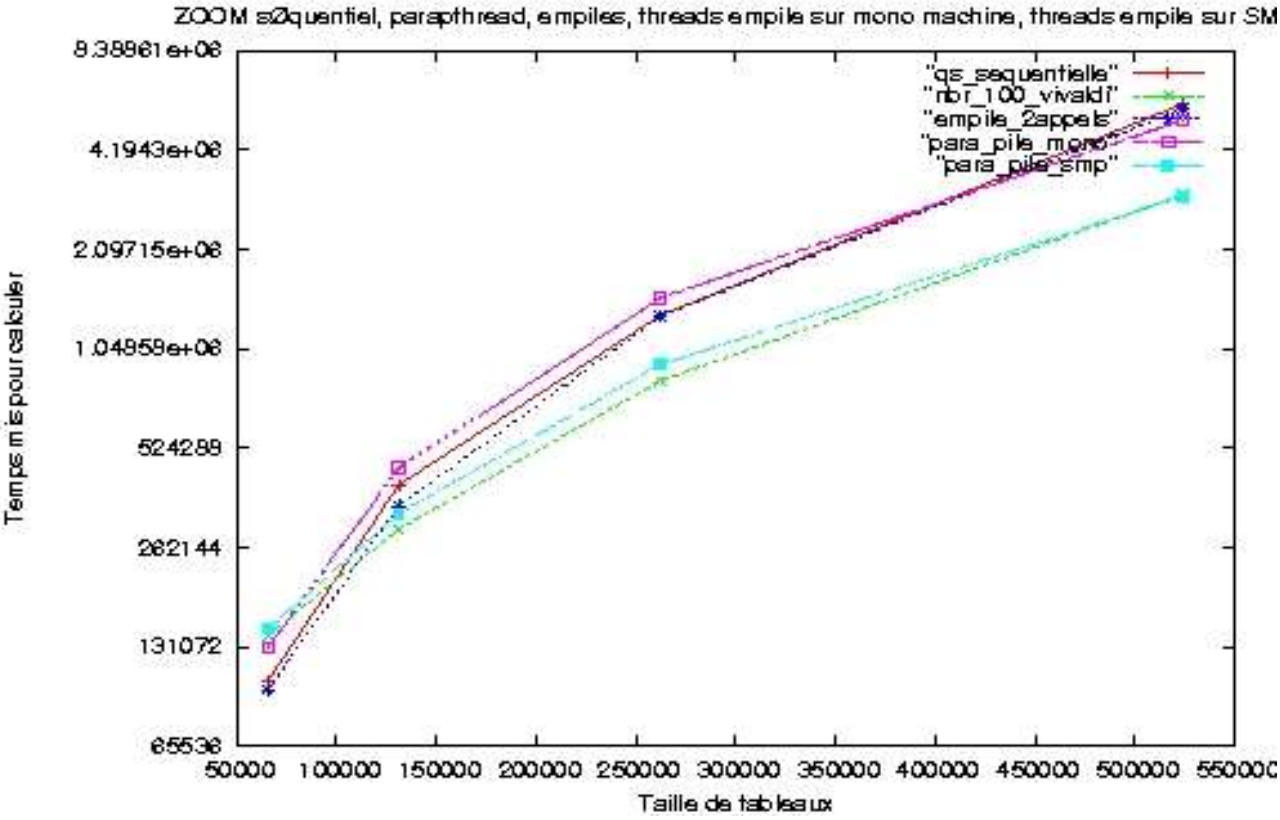


figure b


ZOOM sØquentiel, parapthread, empiles, threads empile sur mono machine, threads empile sur SM
"qs_sequentielle"
"nbr_100_vivaldi"
"empile_2appels"
"para_pile_mono"
"para_pile_smp"
Temps mis pour calculer
Taille de tableaux
3.35544e+07
1.04858e+06
32768
1024
32
1
0.03125
0.000976562
1
4
16
64
256
1024
4096
16384
65536
262144
1.04858e+06


ZOOM sØquentiel, parapthread, empiles, threads empile sur mono machine, threads empile sur SM
"qs_sequentielle"
"nbr_100_vivaldi"
"empile_2appels"
"para_pile_mono"
"para_pile_smp"
Temps mis pour calculer
Taille de tableaux
8.38861e+06
4.1943e+06
2.09715e+06
1.04858e+06
524288
262144
131072
65536
50000
100000
150000
200000
250000
300000
350000
400000
450000
500000
550000

Voyons les graphes précédents. Figure a , c'est deux threads empilent les taches dans une pile commune pour trier les tableaux,une exécution sur une machine mono-processeur, et l'autre sur SMP. Et sur la marche SMP, ça marche plus vite à partir certaine taille de tableaux,(à peu près 100000).Comme avant on fait des testes et on compare les versions parallèles par thread avec la version séquentielle dans TP2, la parallélisation peut montrer sa puissance (avantage) à partir certaine taille de tableaux. Donc ici aussi. Et aussi j'ai fait un graphe pour comparer les version dans le TP2 aussi , voyons la Figure b . On peux avoir la même conclusion, c'est que la parallélisation marche mieux sur les tableaux avec la taille plus de 100000 éléments. Pour plus précis, on peux regarder le graphe suivant, c'est zoom de la partie critique.

Donc la plus puissant ,c'est que la version de parallélisation faite par thread en limitant le nombre de threads inférieur de 100, et la version que je viens de réaliser dans cet exercice, threads empilement en parallèle sur SMP. En faite pour le dernière exercice, on peut prévoir ça sera encore mieux que cette version, parce que avec cette version deux threads accèdent dans une même pile pour chercher des travaux et empiler des travaux, sur les machine SMP, c'est un peu gênant de laisser deux threads avançant simultanément de accéder dans un même tableaux, c'est ce que j'ai trouvé quand j'ai fait TP2, aussi j'en ai mentionné dans ce rapport aussi.

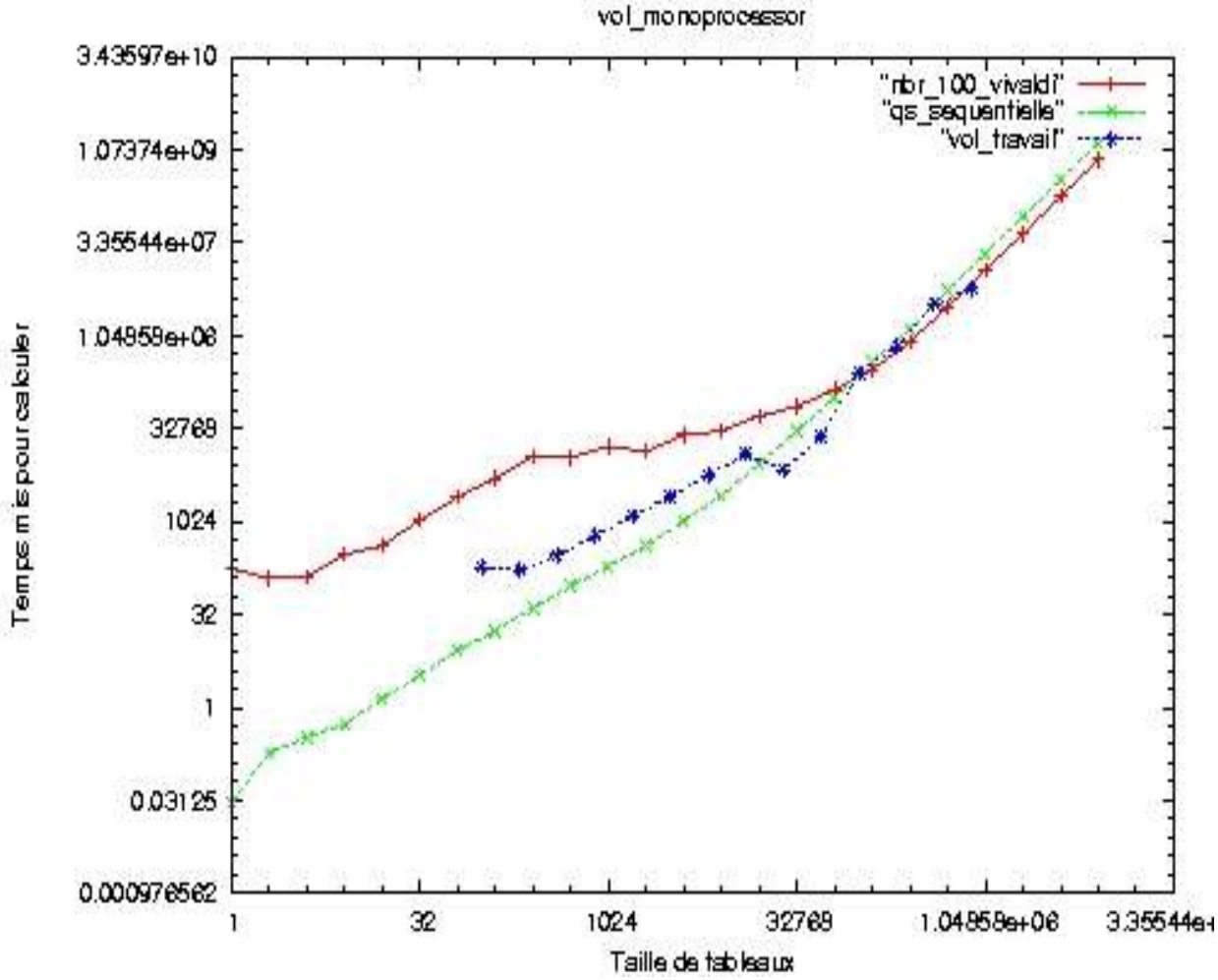
Une pile propre de travail par thread

Pour le dernier challenge, je n'ai pas pu réalisé, je suis désolé. Au début je fais comme ça : Je laisse le tableaux entière au thread main de trier, et elle va empiler des taches dans sa pile . Et pthread commence en regardant dans la pile propre de pthread, et puis essaie de voler. Donc le problème c'est que à certain moment, l'exécution échoue de Erreur de segmentation. Et après en examinant, je trvouve que le problème peut-être se pose au moment où un thread veut voler des taches de l'autre, et aussi comme dans l'autre pile dans laquelle il veut voler des taches il y'en a aucune, donc le thread doit attendre en se bouclant que l'autre thread qui est en train de traiter une tache empile une tache dans sa pile. Peut-être il y a problème si le thread en attente trop se boucle longtemps.

Comme j'ai pas de chance , il reste moi tout seul, donc c'est difficile de avoir plusieurs idée différentes. Et j'ai demandé au autres pour avoir des idée, mais comme je utilise pas list.h pour la pile, donc c'est un peu loin pour eux. Et aussi mon programme pas très propre, c'est-à-dire la manière, donc c'est un peu génant de comprendre par les autres. Donc pour cette version, j'ai essayé de bien ranger, et mettre très propre. Donc c'est la plus propre à mon avis. J'ai n'ai plus rien à faire pour améliorer, et juste ce matin

j'ai penser à distribuer les taches au deux threads , pour qu'il ne commence pas par attendre voler dans l'autre pile. Mais ça marche un peu sur la machine mono-processor, donc j'ai dessiner juste au dessous. Mais sur la machine SMP, elle n'arrive pas terminer une tache, aussi j'ai examiner un peu pour voir, c'est aussi peut-être que il y a des problème quand il se boucle pour attendre.

En tout cas il faut que je termine pour rendre, mais après je voudrais bien que vous pouvez m'indiquer comment je peux faire pour il marche s'il vous plait.



Fin de rapport.