analyse de trois rapides

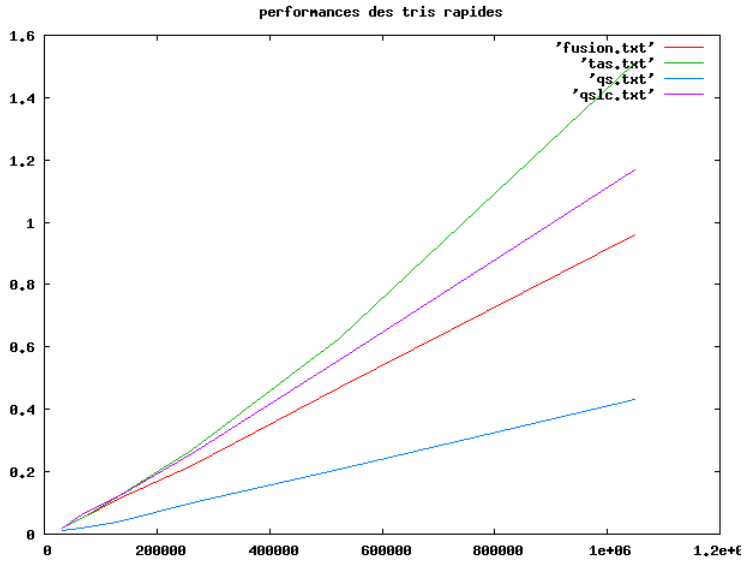
19 octobre 2012

Résumé

L'objectif est de mettre en évidence quelques points importants concernant les tris rapides.

1 tris rapides

Il s'agit de trier des tableaux de nombres entiers. On pourra supposer que la taille des tableaux est une puissance de 2.



1. Implanter les tri rapides : `quicksort`, `qsort`, `fusion`, par `tas`.
2. Comparer les performances de ces trois approches.
3. Déterminer les "facteurs cachés" des trois implantations en prenant comme unité celui du tri `qsort`.

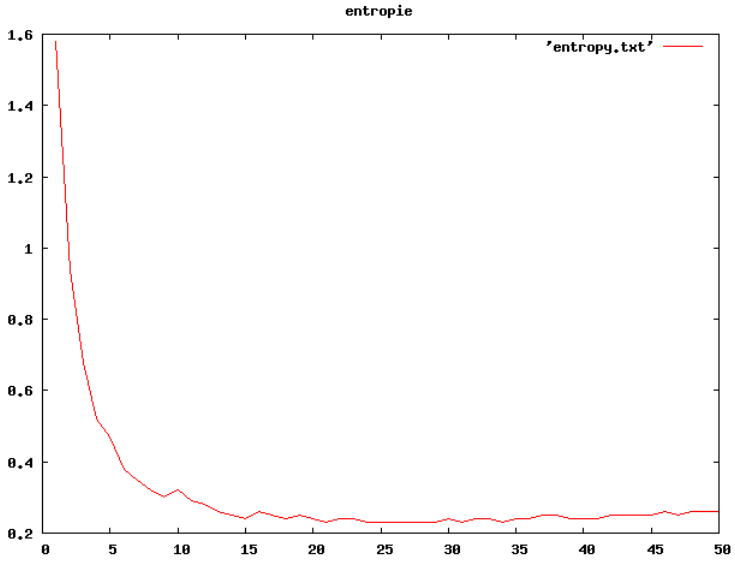


FIG. 1 – `quic sort` a du mal avec les instances ordonnées !

2 cas défavorable

Nous savons que que le tri rapide `quick sort` trie une table de taille n en un temps proportionnel à $n \log n$. Il s'agit d'un résultat en moyenne, les pires instances sont les tableaux. . .triés !

1. Ecrire une fonction `melange(p, n)` qui construit un tableau de taille n obtenu en appliquant pn permutations aléatoires dans un tableau trié.
2. Faire des mesures de temps de calcul en fixant n, puis en faisant varier p dans $[0, \frac{1}{2}]$.

3 optimisation

Une optimisation classique des algorithmes récursifs est de stopper la récursion à une profondeur limite pour éviter de backtracker avec des petites instances.

Une implantation efficace du tri des tablaeux de petites taille permet l'améliorer sensiblement les performances du tri rapide.

```
void A ( z )
{
if ( size( z ) < thresold  )
    finale( x )
else {
    x, y := split( z )
    A( x );
    A( y )
}
```

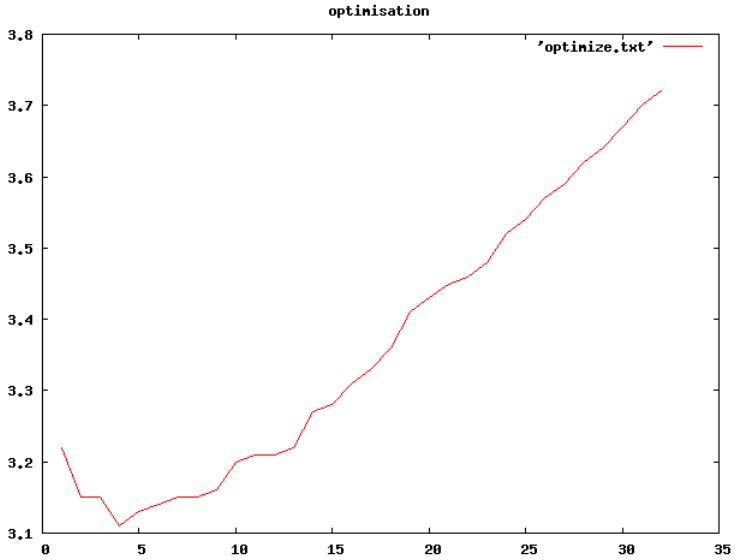


FIG. 2 – Efficacité du tri par sélection sur les feuilles

1. Modifier votre tri rapide pour trier les tables de petites tailles avec un tri non récursifs : insertion, ou sélection.
2. Faire des mesures de temps de calcul pour mettre en évidence le paramétrage optimal de cette heuristique.
3. Comparer à la vaveur de seuil utilisée dans la fonction `qsort` de la bibliothèque `glibc`.
4. Implanter un tri rapide efficace.

Les effets des options de compilation -Ox i.e. les optimisations de `gcc` sont illustrées par la table ci-contre.

-O2 Optimiser encore plus. GCC effectue pratiquement toutes les optimisations supportées qui n'impliquent pas un compromis espace/vitesse. Le compilateur n'effectue pas de déroulement des boucles ou de mise en ligne de fonctions.

-O3 Optimiser toujours plus. -O3 active toutes les optimisations spécifiées par -O2 et active également les options -finline-functions et -frename-registers.

	-O0	-O1	-O2	-O3	O4
0	3.94	2.35	2.07	2.04	2.04
1	3.91	2.35	2.06	2.04	2.02
2	3.83	2.34	2.01	2.02	2.02
3	3.83	2.34	2.03	2.02	2.03
4	3.81	2.35	2.01	2.03	2.03
5	3.82	2.35	2.03	2.05	2.04
6	3.81	2.36	2.02	2.04	2.05
7	3.84	2.37	2.04	2.05	2.07
8	3.86	2.38	2.05	2.08	2.06

4 parallélisation

Si vous n'avez pas eu de difficulté à répondre aux questions précédentes, vous pouvez vous lancer dans l'analyse de la parallélisation de `quick sort`. Il s'agit d'utiliser les processus légers (`thread`) de la biblothèque `glibc`, sur une machine SMP.

```
void A ( int *t, g,  d )
{
if ( d - g < seuil  )
    finale( t, g, d)
else {
    k = split(t, g, d
    A( t, g, k - 1);
    A( t, k+1, d  )
}
```

```
void P ( int *t, g, d, p )
{
if ( p-- == 0 )
   A( t, g, d );
else {
   k = split(t, g, d );
   create
     P(t, g, k - 1, p);
   P( t, k+1, d, p)
   join;
}
```

La parallélisation `P` de l'algorithme `A` est obtenue en lançant des processus légers dans les noeuds de petites profondeurs.

```
PTHREAD_CREATE(3)          Manuel du programmeur Linux

  pthread_create - Creer un nouveau thread

SYNOPSIS
  int pthread_create(pthread_t * tid, pthread_attr_t * attr,
      void * (*start_routine)(void *), void * arg);

DESCRIPTION
  pthread_create() cree un nouveau thread s'executant
  simultanement avec le thread appelant.

PTHREAD_JOIN(3)            Manuel du programmeur Linux

  pthread_join - Attendre la fin d'un autre thread

SYNOPSIS
  int pthread_join(pthread_t tid, void **thread_return);

DESCRIPTION
  pthread_join()  suspend  l'execution du thread appelant
jusqu'a ce que le thread identifie par tid acheve son
execution, soit en appelant pthread_exit(3) soit apres avoir
ete annule.
```

```
$ lscpu
Architecture:          x86_64
CPU(s):                4
```

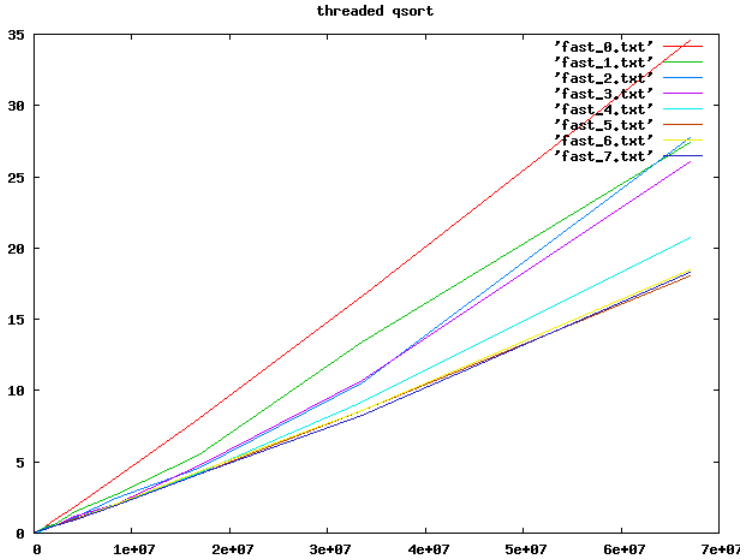


FIG. 3 – threads 4 CPUs (2 coeurs)

```
Thread(s) par coeur :   2
Coeur(s) par support CPU :2
Support(s) CPU :        1
Noeud(s) NUMA :         1
ID du vendeur :         GenuineIntel
Famille CPU :           6
Modele :                28
Version :               2
CPU MHz :               1599.996
L1d cache :             24K
L1i cache :             32K
L2 cache :              512K
```

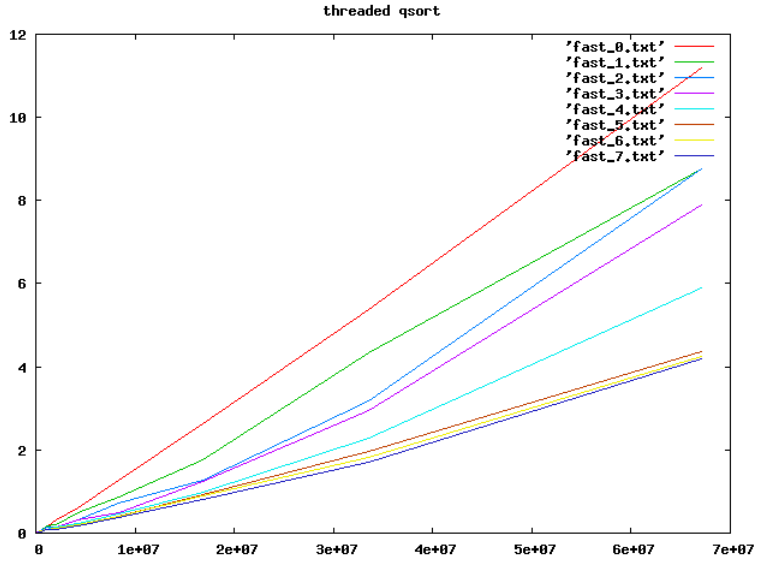


FIG. 4 – threads 24 CPUs, 12coeurs