Laboratoire 5: Profiling

Département: **TIC**

Unité d'enseignement: **HPC**

Auteur(s):

- **CECCHET Costantino**

Professeur:

- **DASSATTI Alberto**

Assistant:

- **DA ROCHA CARVALHO Bruno**

Date:

- **15/05/2024**

[Page de mise en page, laissée vide par intention]

HIGHWAY

Ce projet est un projet de recherche de pattern dans un fichier comme les outils `pt` et `ag`.

La commande de benchmark utilisée tout le long du labo sera :

```
./hw help
```

Cette commande sera exécutée depuis le dossier racine du projet.

time mesure

```
costi@Cos:~/Desktop/highway$ time ./hw help

...

real    0m0.014s
user    0m0.009s
sys 0m0.018s
```

Voici les résultats de la commande `time` pour la commande `./hw help`.

On peut voir pas grand chose, nous allons procéder à une analyse plus poussée.

perf stat fast fetch

Nous allons utiliser `perf stat` pour voir les statistiques de performance de la commande `./hw help`.

```
 Performance counter stats for './hw help':

             10.97 msec task-clock                       #    2.418 CPUs utilized
                45      context-switches                 #    4.103 K/sec
                11      cpu-migrations                   #    1.003 K/sec
               418      page-faults                      #   38.109 K/sec
        26,614,572      cycles                           #    2.426 GHz
        15,505,893      instructions                     #    0.58  insn per cycle
         3,283,758      branches                         #  299.377 M/sec
            78,552      branch-misses                    #    2.39% of all branches

       0.004536999 seconds time elapsed

       0.004070000 seconds user
       0.006783000 seconds sys
```

Ici on voit que le programme utilise 2.418 CPUs, ce qui est un bon signe, cela signifie que le programme est bien parallélisé.

Il effectue peu de context-switches et de cpu-migrations, ce qui est aussi un bon signe.

Les fautes de pages sont assez élevées, mais cela est normal pour un programme qui lit beaucoup de fichiers car il doit charger les pages en mémoire.

Les instructions par cycle sont de 0.58, ce qui n'est pas très bon, ceci pourrait être une piste d'optimisation.

perf record

```
$ perf record -g ./hw help
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.007 MB perf.data (2 samples) ]

$ perf report
```

Grâce à perf record et perf report on peut voir les fonctions qui sont les plus utilisées et les fonctions qui sont les plus appelées.

Nous remarquons que la fonction **search_buffer** est la fonction qui est la plus appelée.

hotspot

En utilisant hotspot on peut voir ou le programme passe le plus de temps.

Dans le fichier search.c on a la fonction **search_buffer** qui est appeler le plus souvent.

Cette fonction est applée dans le thread **search_thread** qui est un des deux threads qui sont crées., le deuxième thread sert a l'affichage des résultats.

Cette partie du thread l.169 **worker.c** appelle la fonction search qui cherche un pattern dans un buffer, pour ce faire une fois le buffer chargé, on appelle la fonction **search_buffer** qui est la fonction la plus appelée, qui fait la recherche du pattern dans le buffer.

Apres analyse cette fonction est deja très bien optimiser même les sous appelle de fonction le sont, mais il serait possible de mieux paralléliser le code en plusieurs threads.

Il serait aussi possible de changer l'algorithme de recherche pour un algorithme plus rapide tel que **Boyer-Moore** ou **Knuth-Morris-Pratt**.

On pourrait aussi créer une fonction **memchr()** qui pourrait être optimisée pour notre cas spécifique.

Ces optimisations reste très compliquées et demandent beaucoup de temps et de ressources.

ce programme est déjà très bien optimisé.

conclusion

Le programme est déjà très bien optimisé, il est difficile de trouver des optimisations qui pourraient être faites.

Mais nous avons appris à utiliser des outils de profiling qui nous permettent de voir ou le programme passe le plus de temps et de voir les fonctions qui sont les plus appelées.

Ceci pourrait permettre par la suite d'optimiser certaines parties du code, qui ne sont pas evidentes à première vue.

Environnement d'execution

Le système décrit dispose d'un processeur Intel Core i7-8550U avec 8 threads, répartis sur 4 cœurs physiques. La fréquence du processeur est de 1,80 GHz avec une fréquence mesurée de 1432,548 MHz.

Voici plus ample information sur le processeur:

```
$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 142
model name  : Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
stepping    : 10
microcode   : 0xf4
cpu MHz     : 1432.548
cache size  : 8192 KB
physical id : 0
siblings    : 8
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
fpu     : yes
fpu_exception   : yes
cpuid level : 22
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflus
vmx flags   : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1gb flexpriority tsc_offse
bugs        : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_mult
bogomips    : 3999.93
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual

$ likwid-topology
--------------------------------------------------------------------------------
CPU name:   Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
CPU type:   Intel Kabylake processor
CPU stepping:   10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:        1
Cores per socket:   4
Threads per core:   2
--------------------------------------------------------------------------------
HWThread    Thread      Core        Socket      Available
0       0       0       0       *
1       0       1       0       *
2       0       2       0       *
3       0       3       0       *
```

```
4       1       0       0       *
5       1       1       0       *
6       1       2       0       *
7       1       3       0       *
--------------------------------------------------------------------------------
Socket 0:       ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:          1
Size:           32 kB
Cache groups:       ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:          2
Size:           256 kB
Cache groups:       ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
--------------------------------------------------------------------------------
Level:          3
Size:           8 MB
Cache groups:       ( 0 4 1 5 2 6 3 7 )
--------------------------------------------------------------------------------
********************************************************************************
NUMA Topology
********************************************************************************
NUMA domains:       1
--------------------------------------------------------------------------------
Domain:         0
Processors:     ( 0 4 1 5 2 6 3 7 )
Distances:      10
Free memory:        542.145 MB
Total memory:       7695.82 MB
--------------------------------------------------------------------------------
```