

laboratoire 4 - Perf & Valgrind

auteur : Alexandre Gabrielli

date :

matériel

j'effectue ce laboratoire sur un raspberry Pi 3 Model B Rev 1.2 donc voici les informations essentielles:

Hardware : BCM2835

Revision : a02082

Serial : 0000000040be5365

le fabricant est donc Sony Uk

et les informations sur le processeur:

processor : 1

model name : ARMv7 Processor rev 4 (v7l)

BogoMIPS : 38.40

Features : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32

CPU implementer : 0x41

CPU architecture: 7

CPU variant : 0x0

CPU part : 0xd03

CPU revision : 4

version de linux

debian 10.3

version du kernel

4.19.97-v7+

installation

perf sur debian pose quelque problème , après avoir installer linux tools j'ai dû installer une autre version de perf car le package 4.19 n'a pas été écrit sur debian et je n'ai pas envie de l'écrire moi même.

du coup j'ai installé linux-perf-4.9 et ensuite modifier le fichier /usr/bin/perf et modifier la ligne exec "perf_4.\$version" "\$@" par exec "perf_4.9" "\$@" cela implique que perf n'est pas à 100% fonctionnel et peut nécessiter d'utiliser les paramètres --no-demangle ou --call-graph=lib works.

perf

pour commencer à profiler nos deux fonctions nous allons commencer par utiliser perf en mesurant les événements suivants:

- task-clock pour voir l'utilisation du cpu

- context-switches pour voir s'il y a beaucoup de changement de contexte
- page-faults afin de voir si on a bien géré nos data
- branches et branch-misses pour voir si on est limité par des branch mis-prediction

nous allons lancé perf sur ./sort array 100000 et ./sort list 100000

array

```

pi@raspberrypi: ~/Documents/hpc20_student/lab04/code/src
Fichier  Édition  Onglets  Aide

33115.920576      cpu-clock:u (msec)      #      0.998 CPUs utilized

33.170078058 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ perf stat -e task-clock,context-switches,page-faults,branches,branch-misses ./sort array 100000

Performance counter stats for './sort array 100000':

34868.852761      task-clock:u (msec)      #      0.999 CPUs utilized
          0      context-switches:u      #      0.000 K/sec
        239      page-faults:u      #      0.007 K/sec
5'001'098'392      branches:u      #      143.426 M/sec
508'213      branch-misses:u      #      0.01% of all branches

34.901596307 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $

```

discutions array

on voit que nous avons 0 % de cache miss mais notre programme prend beaucoup de temps, on utilise beaucoup de temps cpu et nous avons énormément de branches car nous utilisons un tri par insertion et nous devons donc effectuer beaucoup de comparaison.

en changeant la premières version de notre code (ci dessous):

```

/* Arrange a array in increasing order of value */
void array_sort(uint64_t *data, const size_t len) {
    int i = 0, j = 0, tmp;
    for (i = 0; i < len; i++) { // loop n times - 1 per element
        for (j = 0; j < len - i - 1; j++) { // last i elements are sorted
already
            if (data[j] > data[j + 1]) { // swop if order is broken
                tmp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = tmp;
            }
        }
    }
}

```

par un tri nécessitant moins de comparaison comme un tri par sélection (code ci-dessous):

```

/* Arrange a array in increasing order of value */
void array_sort(uint64_t *data, const size_t len) {
    /*tri par selection*/
    int passage = 0;
    bool permutation = true;
    int en_cours;

    while ( permutation) {
        permutation = false;
        passage ++;
        for (en_cours=0;en_cours<20-passage;en_cours++) {
            if (data[en_cours]>data[en_cours+1]){
                permutation = true;
                // on echange les deux elements
                int temp = data[en_cours];
                data[en_cours] = data[en_cours+1];
                data[en_cours+1] = temp;
            }
        }
    }
}

```

on obtiens de bien meilleur performance.

The screenshot shows a terminal window on a Raspberry Pi. The user has compiled a program named 'sort.o' using 'gcc' with various optimization flags. They then run 'perf stat -e task-clock,k,context-switches,page-faults,branches,branch-misses ./sort array 100000'. The output shows performance counter stats for './sort array 100000':

Value	Counter	Unit	Additional Info
12.077239	task-clock:u	(msec)	# 0.900 CPUs utilized
0	context-switches:u		# 0.000 K/sec
237	page-faults:u		# 0.020 M/sec
727'522	branches:u		# 60.239 M/sec
10'027	branch-misses:u		# 1.38% of all branches

Below the stats, it shows '0.013418487 seconds time elapsed'. The prompt shows the user is in the directory '~/Documents/hpc20_student/lab04/code/src'.

on voit que nous avons plus de branch-misses (environ 1.4%) mais nous effectuons beaucoup moins de branches (environ 700'000 contre plus de 5'000'00'000) et nécessitons moins de temps cpu (12ms contre ~120'000 ms).

le résultat est donc un programme beaucoup plus rapide malgré le fait que nous avons beaucoup plus de branch-misses.

essayons maintenant d'amélioré encore cella avec le tri rapide:

```

void array_sort(int *tableau, int taille) {
    int mur, courant, pivot, tmp;
    if (taille < 2) return;
    // On prend comme pivot l'element le plus a droite
    pivot = tableau[taille - 1];
    mur = courant = 0;
    while (courant < taille) {
        if (tableau[courant] <= pivot) {
            if (mur != courant) {
                tmp = tableau[courant];
                tableau[courant] = tableau[mur];
                tableau[mur] = tmp;
            }
            mur++;
        }
        courant++;
    }
    array_sort(tableau, mur - 1);
    array_sort(tableau + mur - 1, taille - mur + 1);
}

```

on obtiens

```

pi@raspberrypi: ~/Documents/hpc20_student/lab04/code/src
Fichier  Édition  Onglets  Aide
edness: 'int' and 'size_t' {aka 'const unsigned int'} [-Wsign-compare]
    } while (i < len && printf(", "));
    ^
gcc -O3 -std=c11 -Wall -Wextra -pedantic -g -I../include -o sort.o -c sort.c
gcc -o sort list_util.o array_util.o sort.o
pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ perf stat -e task-clock,context-switches,page-faults,branches,branch-misses ./sort array 100000

Performance counter stats for './sort array 100000':

      23.000886      task-clock:u (msec)      #    0.913 CPUs utilized
           0         context-switches:u      #    0.000 K/sec
        239         page-faults:u           #    0.010 M/sec
    727'394         branches:u              #   31.625 M/sec
       9'867        branch-misses:u         #    1.36% of all branches

    0.025180770 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $

```

on voit que les deux derniers tri sont assez similaire, pour les séparer augmentons de manière significatif et de voir la différence entre les deux :

selection sort

```
pi@raspberrypi: ~/Documents/hpc20_student/lab04/code/src
Fichier  Édition  Onglets  Aide
Performance counter stats for './sort array 100000':

      23.542031      task-clock:u (msec)      #    0.913 CPUs utilized
           0        context-switches:u      #    0.000 K/sec
       239         page-faults:u           #    0.010 M/sec
    727'661        branches:u             #   30.909 M/sec
    10'101        branch-misses:u         #    1.39% of all branches

    0.025789530 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ perf stat -e task-clock,context-switches,page-faults,branches,branch-misses ./sort array 1000000000

Performance counter stats for './sort array 1000000000':

   10529.749043      task-clock:u (msec)      #    0.866 CPUs utilized
           0        context-switches:u      #    0.000 K/sec
    195'393         page-faults:u           #    0.019 M/sec
   703'504'822       branches:u             #   66.811 M/sec
   4'125'847        branch-misses:u         #    0.59% of all branches

   12.164896812 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $
```

quick sort

```
pi@raspberrypi: ~/Documents/hpc20_student/lab04/code/src
Fichier  Édition  Onglets  Aide
' {aka 'long long unsigned int'} and 'int' [-Wsign-compare]
    if (tableau[courant] <= pivot) {
        ^~
array_util.c: In function 'print_array':
array_util.c:52:16: warning: comparison of integer expressions of different signedness: 'int' and
'size_t' {aka 'const unsigned int'} [-Wsign-compare]
    } while (i < len && printf(", "));
               ^
gcc -O3 -std=c11 -Wall -Wextra -pedantic -g -I../include -o sort.o -c sort.c
gcc -o sort list_util.o array_util.o sort.o
pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ perf stat -e task-clock,context-switches,page-faults,branches,branch-misses ./sort array 1000000000

Performance counter stats for './sort array 1000000000':

   55999.243023      task-clock:u (msec)      #    0.999 CPUs utilized
           0        context-switches:u      #    0.000 K/sec
    211'616         page-faults:u           #    0.004 M/sec
   6'121'305'702     branches:u             #  109.311 M/sec
   1'287'897'689     branch-misses:u         #   21.04% of all branches

   56.039863504 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $
```

on peut voir que si nous augmentons la taille de l'array le quick sort fait beaucoup plus de page-faults sûrement due à la récursion. Nous préférons donc utiliser le sélection sort

list

```
pi@raspberrypi: ~/Documents/hpc20_student/lab04/code/src
Fichier  Édition  Onglets  Aide

508'213      branch-misses:u      #      0.01% of all branches

34.901596307 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ perf stat -e task-clock,context-switches,page-faults,branches,branch-misses ./sort list 100000

Performance counter stats for './sort list 100000':

118251.395468      task-clock:u (msec)      #      1.000 CPUs utilized
          0      context-switches:u      #      0.000 K/sec
          626      page-faults:u      #      0.005 K/sec
5'005'566'439      branches:u      #      42.330 M/sec
1'546'024'240      branch-misses:u      #      30.89% of all branches

118.281727191 seconds time elapsed

pi@raspberrypi:~/Documents/hpc20_student/lab04/code/src $
```

discutions list

on peut voir que nous avons autant de branches que notre premier code de array mais en plus nous avons énormément de branch-misses. comme nous ne pouvons pas effectuer facilement un tri par sélection ou quick sort nous allons remplacer notre premier code par un tri par insertion que nous dont nous avons trouver [le code](#) sur internet

```
/* Arrange a list in increasing order of value */
void list_sort(struct list_element *start) {
    int swapped, i;
    struct list_element *ptr1;
    struct list_element *lptr = NULL;

    /* Checking for empty list */
    if (start == NULL)
        return;

    do {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr) {
            if (ptr1->data > ptr1->next->data) {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}
```

deviens :

```

/ function to sort a singly linked list using insertion sort
void list_sort(struct list_element **head_ref)
{
    // Initialize sorted linked list
    struct list_element *sorted = NULL;

    // Traverse the given linked list and insert every
    // node to sorted
    struct list_element *current = *head_ref;
    while (current != NULL)
    {
        // Store next for next iteration
        struct list_element *next = current->next;

        // insert current in sorted linked list
        sortedInsert(&sorted, current);

        // Update current
        current = next;
    }

    // Update head_ref to point to sorted linked list
    *head_ref = sorted;
}

/* function to insert a new_node in a list. Note that this
function expects a pointer to head_ref as this can modify the
head of the input linked list (similar to push())*/
void sortedInsert(struct list_element** head_ref, struct list_element* new_node)
{
    struct list_element* current;
    /* Special case for the head end */
    if (*head_ref == NULL || (*head_ref)->data >= new_node->data)
    {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
    else
    {
        /* Locate the node before the point of insertion */
        current = *head_ref;
        while (current->next!=NULL &&
            current->next->data < new_node->data)
        {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}

```

on peut voir que le résultat est beaucoup plus satisfaisant:

▼ ▲

```
Performance counter stats for './sort list 1000':
```

```
0.009842661 seconds time elapsed
```

```
Performance counter stats for './sort list 10000':
```

```
0.471846994 seconds time elapsed
```

valgrind

array

[illegible]


```

.  /* Allocate an array of size "len" and fill it
.  * with random data.
.  * Return the array pointer */
.  uint64_t *array_init(const size_t len) {
5      srand((unsigned) 1991);
Negative repeat count does nothing at /usr/bin/callgrind_annotate line 821, <INPUTFILE> line 14.
685,698 => ???:srand (1x)
1,061 => ???:_dl_runtime_resolve (1x)
.      uint64_t *ptr = (uint64_t *) malloc(len * sizeof(uint64_t));
2,170 => ???:_dl_runtime_resolve (2x)
4,338 => ???:malloc (1x)
Negative repeat count does nothing at /usr/bin/callgrind_annotate line 821, <INPUTFILE> line 15.
669,052 => ???:rand (10000x)
.      int i;
.      for (i = 0; i < len; i++) {
.          ptr[i] = rand();
.      }
.      return ptr;
.  }
.
-- line 23 -----
-- line 25 -----
.  void array_sort(uint64_t *data, const size_t len) {
.      /*tri par selection*/
.      int passage = 0;
.      bool permutation = true;
.      int en_cours;
.
.      while (permutation) {
.          permutation = false;
13          passage++;
288          for (en_cours = 0; en_cours < 20 - passage; en_cours++) {
2,172 => array_util.c:array_sort'2 (1x)
.              if (data[en_cours] > data[en_cours + 1]) {
.                  permutation = true;
.                  // on echange les deux elements
525                  int temp = data[en_cours];
1,389                  data[en_cours] = data[en_cours + 1];
246                  data[en_cours + 1] = temp;
.              }
.          }
.      }
.  }

```

comme notre code d'array est plutôt simple on ne voit pas trop de problème qui apparait avec calgrind, il ne semble pas y avoir de goulet d'étranglement bien défini.

List

priya@princeton.edu, <http://www.princeton.edu/~hpz06/students/priya/>

pi@raspberrypi

```
Use of uninitialized value in division (/) at /usr/bin/callgrind_annotate line 1210.
```

on voit très bien qu'il y a beaucoup de D1 misses dans ce programme et on voit très bien que le goulet d'étranglement se trouve a la ligne

```
if (*head_ref == NULL || (*head_ref)->data >= new_node->data)
```

je vais donc essayer de retenir directement l'adresse et change cette fonction comme suis :

```
void sortedInsert(struct list_element** head_ref, struct list_element* new_node)
{
    struct list_element* current;
    struct list_element* tmp = *head_ref;
    /* Special case for the head end */
    if (tmp == NULL || (tmp)->data >= new_node->data)
    {
        new_node->next = tmp;
        tmp = new_node;
    }
    else
    {
        /* Locate the node before the point of insertion */
        current = tmp;
        while (current->next!=NULL &&
            current->next->data < new_node->data)
        {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
    }
}
```

Fichier Édition Onglets Aide

```
ii@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ make test
/sort list 3000
/sort array 3000
ii@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ valgrind --tool=callgrind --simulate-cache=yes ./sort list 10000
==27233== Callgrind, a call-graph generating cache profiler
==27233== Copyright (C) 2002-2011, and GNU GPL'd, by Josef Weidendorfer et al.
==27233== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==27233== Command: ./sort list 10000
==27233==
==27233-- Warning: Cannot auto-detect cache config on ARM, using one or more defaults
==27233-- For interactive control, run 'callgrind_control -h'.
==27233-- WARNING: Serious error when reading debug info
==27233-- When reading debug info from /lib/arm-linux-gnueabi/hf/ld-2.28.so:
==27233-- Ignoring non-Dwarf2/3/4 block in .debug_info
==27233-- WARNING: Serious error when reading debug info
==27233-- When reading debug info from /lib/arm-linux-gnueabi/hf/ld-2.28.so:
==27233-- Last block truncated in .debug_info; ignoring
==27233-- WARNING: Serious error when reading debug info
==27233-- When reading debug info from /lib/arm-linux-gnueabi/hf/libc-2.28.so:
==27233-- Ignoring non-Dwarf2/3/4 block in .debug_info
==27233-- WARNING: Serious error when reading debug info
==27233-- When reading debug info from /lib/arm-linux-gnueabi/hf/libc-2.28.so:
==27233-- Last block truncated in .debug_info; ignoring
==27233==
==27233== Events      : Ir Dr Dw Ilmr Dlmw ILmr DLmr Dlmw
==27233== Collected : 3581477 930282 600179 1156 6637 4071 897 1723 3950
==27233==
==27233== I  refs:      3,581,477
==27233== I1 misses:    1,156
==27233== L1i misses:     897
==27233== I1 miss rate:   0.3%
==27233== L1i miss rate:  0.2%
==27233==
==27233== D  refs:      1,530,461 (930,282 rd + 600,179 wr)
==27233== D1 misses:    10,708 ( 6,637 rd + 4,071 wr)
==27233== L1d misses:     5,673 ( 1,723 rd + 3,950 wr)
==27233== D1 miss rate:   0.6% ( 0.7% + 0.6% )
==27233== L1d miss rate:  0.3% ( 0.1% + 0.6% )
==27233==
==27233== LL refs:       11,864 ( 7,793 rd + 4,071 wr)
==27233== LL misses:     6,570 ( 2,620 rd + 3,950 wr)
==27233== LL miss rate:   0.1% ( 0.0% + 0.6% )
ii@raspberrypi:~/Documents/hpc20_student/lab04/code/src $ callgrind_annotate --auto=yes callgrind.out.27233
```

```

Fichier  Édition  Onglets  Aide
pi@ras

nextNext = next->next;
free(next);
next = nextNext;
}
}

/*this function print the list*/
void print_list(struct list_element *head) {

-- line 41 -----
-- line 64 -----
// function to sort a singly linked list using insertion sort
void list_sort(struct list_element **head_ref)
{
    // Initialize sorted linked list
    struct list_element *sorted = NULL;

    // Traverse the given linked list and insert every
    // node to sorted
    struct list_element *current = *head_ref;
    while (current != NULL)
    {
        // Store next for next iteration
        struct list_element *next = current->next;

        // insert current in sorted linked list
        sortedInsert(&sorted, current);

        // Update current
        current = next;
    }

    // Update head_ref to point to sorted linked list
    *head_ref = sorted;
}

/* function to insert a new_node in a list. Note that this
function expects a pointer to head_ref as this can modify the
head of the input linked list (similar to push())*/
void sortedInsert(struct list_element** head_ref, struct list_element* new_node)
{
    struct list_element* current;
    struct list_element* tmp = *head_ref;
    /* Special case for the head end */
    if (tmp == NULL || (tmp->data >= new_node->data))
    {
        new_node->next = tmp;
        tmp = new_node;
    }
    else
    {
        /* Locate the node before the point of insertion */
        current = tmp;
        while (current->next!=NULL &&
            current->next->data < new_node->data)
-- line 107 -----

```

on peut voir que le résultats est surprenant, la ligne n'est plus le goulet d'étranglement qu'il était avant et le nombre de miss dans la d1 a très très fortement diminuer, on vérifie avec perf pour voir si la différence de temps est significatif

```

Performance counter stats for './sort list 10000':

    10.785780      task-clock:u (msec)      #    0.813 CPUs utilized
         0         context-switches:u          #    0.000 K/sec
        100       page-faults:u                #    0.009 M/sec
    374'122       branches:u                    #   34.687 M/sec
    26'455        branch-misses:u               #    7.07% of all branches

    0.013268839 seconds time elapsed

pi@rasberryvni:~/Documents/hnc20_student/lab04/code/src $

```

on voit bien que l'on a extrêmement réduit le temps cpu, le nombre de branches , même si on a gagner quelque branch-misses on est beaucoup plus rapide avec cette version.

conclusion

J'ai pu grâce a ce laboratoire comprendre comment utilisé valgrind et perf pour detecter les goulets d'étranglement, grâce au piste donné par les deux outils j'ai pu dans un premier temps sélectionner un meilleur algorithme de tri et dans un seconde nettement amélioré mes performances en particulier pour la liste chaîné.

On peut voir que la dernière transformation de code effectuée sur la fonction `sortedInsert` qui semble de prime abord anodin a permis un énorme gain de performance, sûrement dû au fait qu'à cause des pointeurs le compilateur ne pouvait pas prouver certaines choses pour pouvoir faire des raccourcis mais nous verrons cela plus avant dans le cours.

J'ai pu remarquer aussi que c'était inutile de vouloir à tout pris réduire les branch-misses car cela peut rajouter beaucoup de branch et de task-clock et pas rentable en terme de performance, le but est donc d'avoir un bon ratio entre branches / branch-misses et task.clock.