Oprofile

Oprofile este un sistem de profiling disponibil ca modul pentru kernel-ul de Linux (şi integrat în variantele mai noi ale acestuia), capabil să analizeze atât kernel-ul, cât şi aplicaţiile utilizator. Foloseşte tehnica de sampling şi se bazează pe informaţiile oferite de contoarele din CPU.

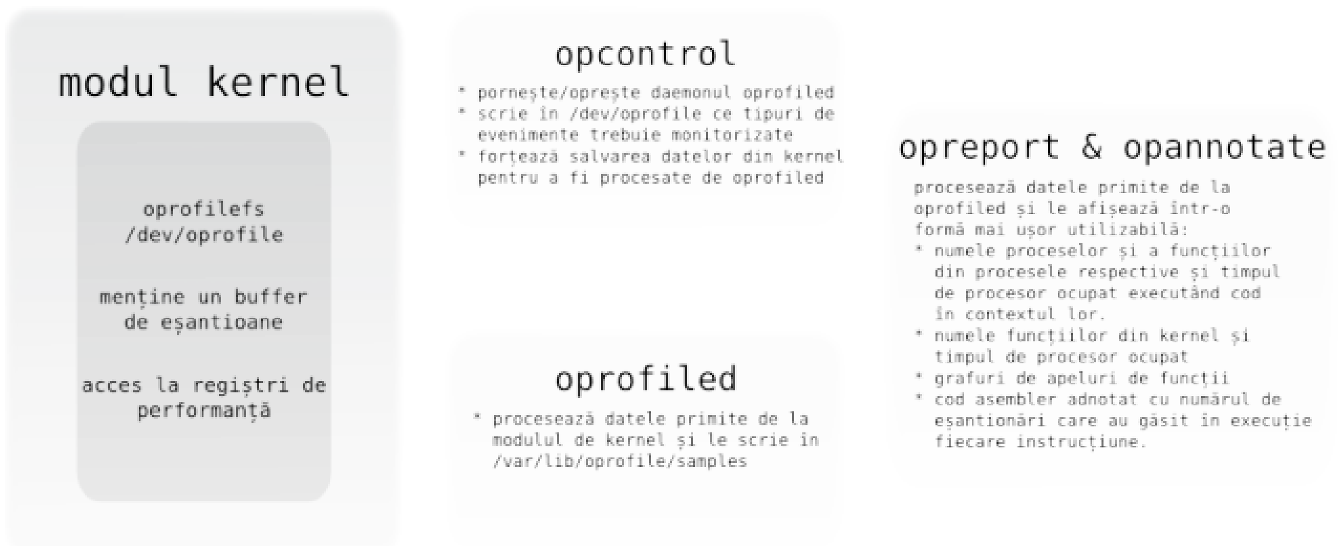
Avantaje:

- overhead redus
- profiling al întregului sistem (inclusiv secţiuni "delicate" din kernel)
- prezentarea efectelor la nivel de hardware

Dezavantaje:

- necesita drepturi de utilizator privilegiat (root)
- nu poate fi folosit pentru cod compilat dinamic sau interpretat (Java, Python, etc)

Arhitectura şi modul de funcţionare



Procesoarele au nişte countere speciale pe care le decrementează de fiecare dată când are loc un eveniment de un anumit tip (cache miss, tlb miss, branch miss-predictions, etc.). Când un astfel de counter se ajunge la valoarea zero, se trimite o întrerupere NMI care este interceptată de modulul de kernel al oprofile. Modulul de kernel resetează counterul la o valoare fixată de utilizator şi salvează într-un buffer din kernel date despre locaţia care a generat întreruperea (proces/kernel, id-ul threadului, instrucţinea curentă care a generat decrementarea counterului, etc.).

Oprofile e format din 3 componente majore:

- **modul de kernel** - un driver special care este notificat de la fiecare întrerupere NMI generată de procesor
- **daemon** - intermediar între modulul kernel şi toolurile userspace. Preia date din kernel şi le scrie (după procesare) în `/var/lib/oprofile/samples/`.
- **utilitare user space**:
  - `opcontrol` - permite configurarea evenimentelor monitorizate şi a frecvenţei cu care sunt eşantionate, a proceselor pentru care se face monitorizarea, etc.
  - `opreport` - sumar al monitorizării
  - `opannotate` - adnotează pe surse sau pe codul dezasamblat al programului numărul de evenimente care au fost eşantionate la fiecare instrucţiune.

Utilizare

Configurare

Primul pas este verificarea existenţei modulului oprofile:

```
muttley:~# modinfo oprofile
```

Apoi este necesară încărcarea efectivă:

```
muttley:~# modprobe oprofile
```

sau

```
muttley:~# opcontrol --init
```

Înainte de a începe colectarea datelor de profiling, trebuie configurat şi pornit daemonul `oprofiled`.
Pentru a afla informaţii despre kernel, `oprofile` are nevoie de calea către imaginea vmlinux (necompresată) a kernelului activ:

```
muttley:~# opcontrol --vmlinux=/boot/vmlinux-`uname -r`
```

Dacă nu se doreşte analizarea kernelului, ci doar a unor aplicaţii, se foloseşte:

```
muttley:~# opcontrol --no-vmlinux
```

Pentru a afla tipurile de evenimente disponibile si detalii despre acestea (inclusiv valoarea minima a contorului asociat) se foloseste:

```
student@muttley:~$ opcontrol --list-events
```

sau

```
student@muttley:~$ ophelp
```

`opcontrol` poate monitoriza maximum două tipuri de evenimente în acelaşi timp (deoarece procesoarele nu au mai multe countere de performanţă care pot fi folosite simultan). Pentru a alege evenimentele dorite se foloseşte opţiunea `?event`. Sintaxa acesteia este:

```
# opcontrol --event=<TIP_EVENIMENT>:<COUNT>:<UNIT-MASK>:<KERNEL-SPACE-COUNTING>:<USER-SPACE-COUNTING>
```

Unde:

- TIP_EVENIMENT - reprezintă numărul evenimentului care se doreşte monitorizat
- COUNT - reprezintă numărul de evenimente hardware de acest tip după care procesorul emite o întrerupere NMI.
- UNIT-MASK - o valoare numerică ce poate modifica comportamentul pentru counterul curent. Dacă nu este specificat se foloseşte o valoarea implicită (poate fi determinată cu `ophelp`).
- KERNEL-SPACE-COUNTING - poate lua două valori, 0/1, şi specifică dacă se activează sau nu counterul atunci când se rulează cod kernel. Implicit are valoarea 1.
- USER-SPACE-COUNTING - paote lua două valori, 0/1, şi specifică dacă se activează sau nu counterul atunci când se rulează cod user. Implicit are valoarea 1.

De exemplu: pe procesoare Core 2 duo, counterul DTLB_MISSES poate fi folosit cu oricare combinaţie a următoarelor valori:

- 0x01: ANY Memory accesses that missed the DTLB.
- 0x02: MISS_LD DTLB misses due to load operations.
- 0x04: L0_MISS_LD L0 DTLB misses due to load operations.
- 0x08: MISS_ST TLB misses due to store operations.

```
# se doreşte măsurarea toate evenimentele de tip DTLB_MISSES (0xf = 0x1|0x2|0x4|0x8)
# care au avut loc în timp ce rulam cod user space
# ignorând cele care au avut loc în kernel space.
# Se doreşte cuantificarea fiecărui al 10000-lea eveniment de acest tip.
muttley:~# opcontrol --event=DTLB_MISSES:10000:0x0f:0:1
```

Pentru a vedea care sunt parametrii cu care este configurat în acest moment oprofile se poate folosi

```
muttley:~# opcontrol --status
Daemon not running
Event 0: PREF_RQSTS_DN:45000:0:1:1
Separate options: library thread
vmlinux file: none
Image filter: /home/gringo/labs/so/profiling/cache_trashing/with_padding
Call-graph depth: 6
Buffer size: 65536
```

Pornire

După configurare, trebuie pornit daemonul:

```
muttley:~# opcontrol --start
```

Oprire

```
muttley:~# opcontrol --shutdown
```

Colectarea şi analizarea datelor

Pentru a nu fi influenţaţi de eventuale date rămase de la o rulare anterioară, trebuiesc curaţate datele salvate:

```
# se şterg toate datele din buffer
muttley:~# opcontrol --reset
```

sau se salvează bufferul curent în sesiunea cu numele `sesiunea_anterioara`

```
muttley:~# opcontrol --save=sesiunea_anterioara
```

Odată pornit daemonul, `oprofile` va colecta date despre toate executabilele ce rulează pe acea maşină (sau dacă a fost configurat cu `?image` va colecta date doar pentru executabilele din calea specificată). Acum trebuie executat şi binarul pe care dorim să îl măsurăm. Deoarece `oprofile` încearcă să-şi minimizeze impactul asupra performanţelor sistemului, acesta va amâna pe cât posibil scrierea datelor din bufferele interne pe disk. Dacă toolurile userspace `opreport` sau `opannotate` nu găsesc informaţii de profiling pe disk vor raporta o eroare:

```
student@muttley:~$ opannotate --source
opannotate error: No sample file found: try running opcontrol --dump or specify a session containing sample files
```

Putem să forţăm scrierea pe disc a bufferelor curente cu:

```
student@muttley:~$ opcontrol --dump
```

Exemplu de rulare

Avem următorul program care calculează numărul de numere divizibile cu 2 şi cu 3 din intervalul $0..2^{30}$ (exerciţiu pur teoretic 😃 .

simple.c

```
#include <stdio.h>

int main()
{
  int start = 0, stop  = (1 << 30);
  int div2 = 0, div3 = 0;
  int i;

  for (i = start; i < stop; i++) {
    if (i % 2 == 0)
      div2 ++;
    if (i % 3 == 0)
      div3 ++;
  }

   <a href="http://www.opengroup.org/onlinepubs/009695399/functions/printf.html">printf</a>("div2=%d, div3=%d\n",
div2, div3);
  return 0;
}
```

Dacă rulăm programul obţinem:

```
student@muttley:~$ time ./simple
div2=536870912, div3=357913942

real    0m13.828s
user    0m12.525s
sys     0m0.020s
```

Dorim să îmbunătăţim performanţele programului şi îl paralelizăm.

O variantă ar fi să incrementăm variabilele `div2` şi `div3` din două threaduri diferite protejând accesul la variabilele `div2` şi `div3` cu un mutex. Această implementare va fi mult mai ineficientă decât cea prezentată mai sus, majoritatea timpului de rulare fiind petrecut în rutinele de preluare şi eliberare a mutexului.

O paralelizare mai bună se poate implementa folosind variabile separate pentru fiecare thread şi integrarea rezultatelor parţiale la final (folosind o paradigma de programare de tipul map-reduce).

complex.c

```
#define _GNU_SOURCE
#include <sched.h>
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

#define BIGNUM           (1 << 30) // limita superioară a intervalului de prelucrare
#define NR_CPUS           2
#define NR_THREADS        NR_CPUS
#define PER_CPU_INTERVAL (BIGNUM / NR_CPUS) //lungimea intervalului pe care va căuta fiecare thread

struct thd_data {
  int div2;
  int div3;
};

struct thd_data per_cpu_vec[NR_THREADS];
pthread_t threads[NR_THREADS];

/**
 * Se impune threadului `thdid` să ruleze doar pe procesorul `thdid % NR_CPUS`.
 * Dacă există procesoare >= threaduri, fiecare thread va rula pe un procesor separat.
 */
static void set_affinity(int thdid)
{
  int rc;
  cpu_set_t cmask;
  CPU_ZERO(&cmask);
  CPU_SET(thdid % NR_CPUS, &cmask);
  rc = sched_setaffinity(0 /*current thead*/, sizeof(cpu_set_t), &cmask);
  if (-1 == rc)
    perror("sched_setaffinity error");
}


static void * thd_func(void *param)
{
  int thdid = (int)param;
  int i;
  /* Fiecare thread operează pe o porţiune distinctă a intervalului. */
  int start = thdid       * PER_CPU_INTERVAL;
  int stop  = (thdid + 1) * PER_CPU_INTERVAL;

  set_affinity(thdid);


  for (i = start; i < stop; i++) {
    if (i % 2 == 0)
      per_cpu_vec[thdid].div2 ++;
    if (i % 3 == 0)
      per_cpu_vec[thdid].div3 ++;
  }

  return NULL;
}


int main()
{
  int rc, i, sum;

  /* se creează threadurile */
  for (i = 0; i < NR_THREADS; i++) {
    rc = pthread_create(&threads[i], NULL, thd_func, (void*)i);
    if (-1 == rc) {
      perror("error creating thread");
      exit(-1);
    }
  }


  /* se aşteaptă ca toate threadurile să-şi termine execuţia */
  for (i = 0; i < NR_THREADS; i++) {
    rc = pthread_join(threads[i], NULL);
    if (-1 == rc) {
      perror("error joining thread");
      exit(-1);
    }
  }
```

```
        /* se integrează rezultatele parţiale ale fiecărui thread */
        sum = 0;
        for (i = 0; i < NR_THREADS; i++) {
          sum += per_cpu_vec[i].div2;
        }
        <a href="http://www.opengroup.org/onlinepubs/009695399/functions/printf.html">printf</a>("div2=%d, ", sum);

        sum = 0;
        for (i = 0; i < NR_THREADS; i++) {
          sum += per_cpu_vec[i].div3;
        }
        <a href="http://www.opengroup.org/onlinepubs/009695399/functions/printf.html">printf</a>("div3=%d\n", sum);
        return 0;
      }
```

Programul paralelizat este mai complex, performanţele lui fiind:

```
student@muttley:~$ time ./complex
div2=536870912, div3=357913942

real    0m38.865s
user    1m14.913s
sys     0m0.016s
```

Pentru programul paralelizat

- timpul total de rulare este cu 350% peste cel al variantei neparalelizate.
- timpul de procesor este cu 700% peste cel al variantei neparalelizate, folosind 2 core-uri în acelaşi timp

Intuitiv timpul de procesor ar fi trebuit să crească, dar timpul de rulare ar fi trebuit să scadă. Vom investiga problema folosind oprofile.

Partea computaţională (for-ul) foloseşte puţine date şi ar trebui să încapă în cache-ul L1. Vom verifica dacă se fac accese în cache-ul L2.

```
# se şterg datele unei rulări anterioare
muttley:~# opcontrol --reset
# se monitorizează toate cererile în cache-ul L2 (L2_RQSTS) făcute de executabilul "complex"
muttley:~# opcontrol --no-vmlinux --image=./complex --event=L2_RQSTS:7500
# se porneşte serverul oprofiled
muttley:~# opcontrol --start
# se rulează aplicaţia monitorizată
muttley:~# ./complex
div2=536870912, div3=357913942
# se închide serverul
muttley:~# opcontrol --dump; opcontrol --shutdown

# se preiau datele
muttley:~# opreport
CPU: Core 2, speed 1833 MHz (estimated)
Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000
   L2_RQSTS:90000|
  samples|      %|
------------------
     1660 100.000 p1


# se afişează informaţii per-simbol
muttley:~# opreport --symbols
CPU: Core 2, speed 1833 MHz (estimated)
Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000
samples  %        symbol name
1660     100.000  thd_func


# se afişează detalii per instrucţiune şi se afişează informaţii despre fişierul şi linia din care provine fiecare instrucţiune.
muttley:~# opreport --debug-info --details
CPU: Core 2, speed 1833 MHz (estimated)
Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000
vma      samples  %        linenr info                 symbol name
08048600 1660     100.000  cache_trashing.c:40         thd_func
  0804863a 5         0.3012  cache_trashing.c:51
  08048648 597      35.9639  cache_trashing.c:52 --       per_cpu_vec[thdid].div2 ++;
  08048658 1         0.0602  cache_trashing.c:53
  0804865f 8         0.4819  cache_trashing.c:53
```

```
  08048681 2         0.1205  cache_trashing.c:53
  08048684 1         0.0602  cache_trashing.c:53
  08048694 1044     62.8916  cache_trashing.c:54 --       per_cpu_vec[thdid].div3 ++;
  0804869e 1         0.0602  cache_trashing.c:50
  080486a2 1         0.0602  cache_trashing.c:50


# se adnotează pe codul sursă date de profiling
muttley:~# opannotate --source
...
     2  0.1205 :  for (i = start; i < stop; i++) {
     5  0.3012 :    if (i % 2 == 0)
   597 35.9639 :      per_cpu_vec[thdid].div2 ++;
    12  0.7229 :    if (i % 3 == 0)
  1044 62.8916 :      per_cpu_vec[thdid].div3 ++;
...


# se adnotează pe codul dezasamblat al programului date de profiling
muttley:~# opannotate --assembly
...
     5  0.3012 : 804863a:       test   %eax,%eax
               : 804863c:       jne    8048652 <thd_func+0x52>
               : 804863e:       mov    -0x4(%ebp),%eax
               : 8048641:       mov    0x8049a44(,%eax,8),%edx
   597 35.9639 : 8048648:       add
struct thd_data {
  int div2;
  int div3;
};
struct thd_data per_cpu_vec[NR_THREADS];
x1,%edx
: 804864b:       mov    %edx,0x8049a44(,%eax,8)
: 8048652:       mov    -0x8(%ebp),%eax
: 8048655:       mov    %eax,-0x18(%ebp)
1  0.0602 : 8048658:       movl
struct thd_data {
  int div2;
  int div3;
  char padding[NR_PADDING_BYTES];
};
struct thd_data per_cpu_vec[NR_THREADS];
x55555556,-0x1c(%ebp)
8  0.4819 : 804865f:       mov    -0x1c(%ebp),%eax
...
2  0.1205 : 8048681:       mov    %edx,-0x14(%ebp)
1  0.0602 : 8048684:       cmpl
muttley:~# time ./with_padding
div2=536870912, div3=357913942
real    0m6.432s
user    0m12.793s
sys     0m0.008s


muttley:~# opreport
CPU: Core 2, speed 1833 MHz (estimated)
Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 7500
    L2_RQSTS:7500|
  samples|      %|
------------------
        1 100.000 with_padding
x0,-0x14(%ebp)
: 8048688:       jne    804869e <thd_func+0x9e>
: 804868a:       mov    -0x4(%ebp),%eax
: 804868d:       mov    0x8049a48(,%eax,8),%edx
1044 62.8916 : 8048694:       add
```

```
# se şterg datele unei rulări anterioare muttley:~# opcontrol --reset # se monitorizează toate cererile în cache-ul L2 (L2_RQSTS) făcute de executabilul "complex" muttley:~# opcontrol --no-vmlinux --image=./complex --event=L2_RQSTS:7500 # se porneşte serverul oprofiled muttley:~# opcontrol --start # se rulează aplicaţia monitorizată muttley:~# ./complex div2=536870912, div3=357913942 # se închide serverul muttley:~# opcontrol --dump; opcontrol --shutdown # se preiau datele muttley:~# opreport CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 L2_RQSTS:90000| samples| %| ------------------ 1660 100.000 p1 # se afişează informaţii per-simbol muttley:~# opreport --symbols CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 samples % symbol name 1660 100.000 thd_func # se afişează detalii per instrucţiune şi se afişează informaţii despre fişierul şi linia din care provine fiecare instrucţiune. muttley:~# opreport --debug-info --details CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 vma samples % linenr info symbol name 08048600 1660 100.000 cache_trashing.c:40 thd_func 0804863a 5 0.3012 cache_trashing.c:51 08048648 597 35.9639 cache_trashing.c:52 -- per_cpu_vec[thdid].div2 ++; 08048658 1 0.0602 cache_trashing.c:53 0804865f 8 0.4819
```

```
cache_trashing.c:53 08048681 2 0.1205 cache_trashing.c:53 08048684 1 0.0602 cache_trashing.c:53 08048694 1044 62.8916 cache_trashing.c:54 -- per_cpu_vec[thdid].div3 ++; 0804869e 1 0.0602 cache_trashing.c:50 080486a2 1 0.0602 cache_trashing.c:50 # se adnotează pe codul sursă date de profiling muttley:~# opannotate --source ... 2 0.1205 : for (i = start; i < stop; i++) { 5 0.3012 : if (i % 2 == 0) 597 35.9639 : per_cpu_vec[thdid].div2 ++; 12 0.7229 : if (i % 3 == 0) 1044 62.8916 : per_cpu_vec[thdid].div3 ++; ... # se adnotează pe codul dezasamblat al programului date de profiling muttley:~# opannotate --assembly ... 5 0.3012 : 804863a: test %eax,%eax : 804863c: jne 8048652  : 804863e: mov -0x4(%ebp),%eax : 8048641: mov 0x8049a44(,%eax,8),%edx 597 35.9639 : 8048648: add $0x1,%edx : 804864b: mov %edx,0x8049a44(,%eax,8) : 8048652: mov -0x8(%ebp),%eax : 8048655: mov %eax,-0x18(%ebp) 1 0.0602 : 8048658: movl $0x55555556,-0x1c(%ebp) 8 0.4819 : 804865f: mov -0x1c(%ebp),%eax ... 2 0.1205 : 8048681: mov %edx,-0x14(%ebp) 1 0.0602 : 8048684: cmpl $0x0,-0x14(%ebp) : 8048688: jne 804869e  : 804868a: mov -0x4(%ebp),%eax : 804868d: mov 0x8049a48(,%eax,8),%edx 1044 62.8916 : 8048694: add $0x1,%edx : 8048697: mov %edx,0x8049a48(,%eax,8) 1 0.0602 : 804869e: addl $0x1,-0x8(%ebp) 1 0.0602 : 80486a2: mov -0x8(%ebp),%eax
```

```
x1,%edx
: 8048697:       mov    %edx,0x8049a48(,%eax,8)
1  0.0602 : 804869e:       addl
```

```
# se şterg datele unei rulări anterioare muttley:~# opcontrol --reset # se monitorizează toate cererile în cache-ul L2 (L2_RQSTS) făcute de executabilul "complex" muttley:~# opcontrol --no-vmlinux --image=./complex --event=L2_RQSTS:7500 # se porneşte serverul oprofiled muttley:~# opcontrol --start # se rulează aplicaţia monitorizată muttley:~# ./complex div2=536870912, div3=357913942 # se închide serverul muttley:~# opcontrol --dump; opcontrol --shutdown # se preiau datele muttley:~# opreport CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 L2_RQSTS:90000| samples| %| ------------------ 1660 100.000 p1 # se afişează informaţii per-simbol muttley:~# opreport --symbols CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 samples % symbol name 1660 100.000 thd_func # se afişează detalii per instrucţiune şi se afişează informaţii despre fişierul şi linia din care provine fiecare instrucţiune. muttley:~# opreport --debug-info --details CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 90000 vma samples % linenr info symbol name 08048600 1660 100.000 cache_trashing.c:40 thd_func 0804863a 5 0.3012 cache_trashing.c:51 08048648 597 35.9639 cache_trashing.c:52 -- per_cpu_vec[thdid].div2 ++; 08048658 1 0.0602 cache_trashing.c:53 0804865f 8 0.4819 cache_trashing.c:53 08048681 2 0.1205 cache_trashing.c:53 08048684 1 0.0602 cache_trashing.c:53 08048694 1044 62.8916 cache_trashing.c:54 -- per_cpu_vec[thdid].div3 ++; 0804869e 1 0.0602 cache_trashing.c:50 080486a2 1 0.0602 cache_trashing.c:50 # se adnotează pe codul sursă date de profiling muttley:~# opannotate --source ... 2 0.1205 : for (i = start; i < stop; i++) { 5 0.3012 : if (i % 2 == 0) 597 35.9639 : per_cpu_vec[thdid].div2 ++; 12 0.7229 : if (i % 3 == 0) 1044 62.8916 : per_cpu_vec[thdid].div3 ++; ... # se adnotează pe codul dezasamblat al programului date de profiling muttley:~# opannotate --assembly ... 5 0.3012 : 804863a: test %eax,%eax : 804863c: jne 8048652  : 804863e: mov -0x4(%ebp),%eax : 8048641: mov 0x8049a44(,%eax,8),%edx 597 35.9639 : 8048648: add $0x1,%edx : 804864b: mov %edx,0x8049a44(,%eax,8) : 8048652: mov -0x8(%ebp),%eax : 8048655: mov %eax,-0x18(%ebp) 1 0.0602 : 8048658: movl $0x55555556,-0x1c(%ebp) 8 0.4819 : 804865f: mov -0x1c(%ebp),%eax ... 2 0.1205 : 8048681: mov %edx,-0x14(%ebp) 1 0.0602 : 8048684: cmpl $0x0,-0x14(%ebp) : 8048688: jne 804869e  : 804868a: mov -0x4(%ebp),%eax : 804868d: mov 0x8049a48(,%eax,8),%edx 1044 62.8916 : 8048694: add $0x1,%edx : 8048697: mov %edx,0x8049a48(,%eax,8) 1 0.0602 : 804869e: addl $0x1,-0x8(%ebp) 1 0.0602 : 80486a2: mov -0x8(%ebp),%eax
```

```
x1,-0x8(%ebp)
1  0.0602 : 80486a2:       mov    -0x8(%ebp),%eax
```

Din rezultatele monitorizării se observă că se fac un număr mare de accese în cache-ul L2 când se accesează datele per-cpu, deşi datele în loop încap în cache-ul L1.

```
struct thd_data { int div2; int div3; }; struct thd_data per_cpu_vec[NR_THREADS];
```

Procesorul pe care a fost rulat exemplul anterior deţine câte un cache L1 pentru fiecare core şi un cache L2 partajat. Dacă mai multe core-uri modifică date care corespund unei aceleiaşi linii din cache-ul L2, de fiecare dată când unul din core-uri scrie în acea linie, linia va fi invalidată în cache-ul L1 al celorlalte core-uri. Astfel celelalte threaduri vor trebui să recitească din L2 toată linia curentă, cache-ul L1 ne mai ajutând în acest caz, toate acesele făcându-se la o viteză mai mică chiar decât cea a cache-ului L2.

Dacă se modifică structura de date specifică fiecărui thread, se poate elimina problema de "false sharing".

```
struct thd_data { int div2; int div3; char padding[NR_PADDING_BYTES]; }; struct thd_data per_cpu_vec[NR_THREADS];
```

```
muttley:~# time ./with_padding div2=536870912, div3=357913942 real 0m6.432s user 0m12.793s sys 0m0.008s muttley:~# opreport CPU: Core 2, speed 1833 MHz (estimated) Counted L2_RQSTS events (number of L2 cache requests) with a unit mask of 0x7f (multiple flags) count 7500 L2_RQSTS:7500| samples| %| ------------------ 1 100.000 with_padding
```

În acest caz a avut loc doar un singur eşantion de citire în L2 (în realitate au avut loc între 7500 şi 2*7500-1 evenimente de acest tip).

- timpul total de rulare este 58% din cel al variantei neparalelizate
- timpul de procesor este 112% din cel al variantei neparalelizate.

From:
http://elf.cs.pub.ro/so/wiki/ - **Sisteme de Operare**

Permanent link:
http://elf.cs.pub.ro/so/wiki/laboratoare/resurse/oprofile

Last update: **2011/05/08 13:51**