



## Contadores Hardware

Universidad Nacional de La Plata



Facultad de Informática



# Agenda

2

## Contadores Hardware



# Contadores Hardware

3

- Hardware performance counter, Performance monitoring counters (PMC) o en castellano **Contadores hardware**: son un conjunto de registros de propósito especial incorporados en los procesadores modernos para llevar la cuenta de actividades que ocurren en el hardware de bajo nivel:
  - Cantidad de fallos de cache en los distintos niveles
  - Cantidad de instrucciones retiradas por ciclo
  - Cantidad de saltos
  - Cantidad de accesos a memoria
  - Etc.

# Contadores Hardware

## Historia

4

- Uno de los primeros procesadores en implementar los contadores hardware y una instrucción (RDPMC) para accederlos fue el Intel Pentium.
- Esta información no estaba documentada (“intencionalmente???”).
- En 1994, Terje Mathisen estaba haciendo ingeniería inversa y “por casualidad”, descubrió un conjunto de registros. Tardó un tiempo en darse cuenta su propósito, al descubrirlo escribió un artículo en la revista Byte:

1994-07-01

*Pentium secrets: the Pentium collects lots of information about code execution and now you can get access to it. (deciphering undocumented registers to aid in code optimization) (Core Technologies: CPUs)*

*“Pentium Secrets: Undocumented features of the Intel Pentium can give you all the information you need to optimize Pentium code”*  
*Mathisen, Terje - Byte Magazine, July 1994, Page 191*

# Contadores Hardware

5

- La cantidad de registros de contadores hardware dependen de cada arquitectura y son muy limitados (mucho menos de 10 registros, salvo excepciones).
- Pueden obtenerse en los manuales de los procesadores o microarquitecturas (difíciles de encontrar):

Arquitectura	Contadores
Intel P6: Pentium Pro, Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon	2 x 40-bit
Intel NetBurst: Pentium 4, Celeron, Pentium D, Celeron D, Pentium Extreme Edition, Xeon (2001 – 2006)	18 x 40-bit
Intel Atom - Intel Core	2 x programables (el evento puede modificarse mediante una dirección) 3 x de función fija (el evento no puede modificarse, siempre cuenta el mismo evento)

# Contadores Hardware

6

## Ventajas

- Acceso de bajo overhead
- Se obtiene información muy detallada
- No es necesario modificar el código fuente
- Algunas herramientas permiten su uso a nivel de SO (scheduling)

## Desventajas

- Cuando se quieren contar más eventos que los contadores disponibles se deben hacer varias ejecuciones. Algunas herramientas permiten multiplexación de eventos.

- La misma información puede obtenerse con técnicas basadas en software, conocidas como instrumentación o profiling. Un ejemplo es la herramienta PIN del mismo Intel. Sin embargo, esto genera mayor overhead.

# Agenda

7

Perf



# Contadores Hardware – Herramientas

## Perf

8

- Perf (tambien conocida perf\_events o perf tools) es la herramienta más utilizada de análisis de rendimiento basada en contadores hardware para Linux.
- Tiene soporte para la mayoría del hardware disponible (Intel, AMD, ARM, Sparc, PowerPC etc).
- Desde 2014 soporta Running Average Power Limit (RAPL) para medir consumo energético en procesadores Intel.
- Está disponible en todos los repositorios de Linux y se instala directamente con cualquier gestor de paquetes (apt, aptitude, yum etc). Una vez instalado se gestiona a través del comando:

perf

# Contadores Hardware – Herramientas Perf

9

- Cada arquitectura tiene su propio conjunto de contadores y no todos los eventos se pueden contar.
  - Para ver la listas de eventos disponibles en la arquitectura en que nos encontramos debemos ejecutar:

# Contadores Hardware – Herramientas

## Perf

10

- **Hardware event:** medidas de eventos microarquitecturales que provienen del procesador en si mismo (Performance Monitoring Unit – PMU). Estos eventos se conocen también como PMU hardware events. Varían de una arquitectura a otra, son específicos de cada CPU y están documentados por el fabricante. (Ej: L1-cache-misses).
- **Software event:** son contadores propios del kernel. (Ejemplo: context-switches).
- **Hardware cache event:** son nombres de algunos eventos generales que incluyen la mayoría de las arquitecturas. Si la arquitectura no lo reconoce con ese nombre, pero lo soporta, debe buscarse la dirección del evento en el manual y correr en modo Raw.
- **Raw:** puede ocurrir que un evento específico no aparezca en la lista pero el manual del hardware indica que existe. El propio manual debería proporcionar la dirección del registro la cual se puede utilizar con el comando: `# perf stat -e r003c 1s`
- **Tracepoint event:** implementados por la infraestructura de ftrace (para profiling y depuración).

# Contadores Hardware – Herramientas

## Perf

11

- Se pueden obtener ciertas estadísticas generales de la ejecución de un comando o aplicación. Ejemplo, `ls` :

```
# perf stat ls
...
Performance counter stats for 'ls':

      1.642676  task-clock-msecs      #      0.866 CPUs
          0      context-switches     #      0.000 M/sec
          0      CPU-migrations      #      0.000 M/sec
        254      page-faults         #      0.155 M/sec
    2608232      cycles            #   1587.795 M/sec
    2573440      instructions       #      0.987 IPC
    21813      cache-references    #     13.279 M/sec
     4013      cache-misses        #      2.443 M/sec

  0.001897479  seconds time elapsed
```

# Contadores Hardware – Herramientas

## Perf

12

- Se puede obtener un evento específico de la ejecución de un comando o aplicación. Este evento se obtiene previamente ejecutando perf list. Ejemplo, `ls` :

```
# perf stat -e instructions,cycles,cache-misses ls
...
Performance counter stats for 'ls':

      2483173  instructions          #      1.010  IPC
      2458463  cycles                #      0.000 M/sec
          255  cache-misses          #      0.000 M/sec

  0.001792414  seconds time elapsed
```

# Contadores Hardware – Herramientas

## Perf

13

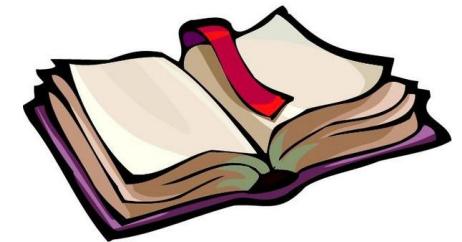
- Perf soporta **multiplexación**: permite contar más eventos que los contadores disponibles. Se utiliza una técnica de Round Robin para que cada evento tenga la chance de acceder a los contadores:
  - Si la arquitectura tiene C contadores disponibles, los primeros C eventos se asignan a estos por un tiempo.
  - Pasado ese tiempo se asignan los siguientes C eventos.
- Sólo es aplicable a eventos PMU.
- Multiplexación significa que cada evento no se mide todo el tiempo. Al final de cada ejecución perf estima el valor del evento basándose en el tiempo total asignado y el tiempo de ejecución. (**Es una estimación y no un valor real!!!**)

$$\text{final\_count} = \text{raw\_count} \cdot \frac{\text{time\_enabled}}{\text{time\_running}}$$

# Agenda

14

PAPI



# Contadores Hardware – Herramientas

## PAPI

15

- **PAPI**<sup>1</sup>(Performance Application Programming Interface): Es una API que provee un conjunto de bibliotecas comunes y fáciles de usar para acceder a los contadores de hardware en varias plataformas. Cuenta con dos interfaces de acceso a los contadores:
  - **Interfaz de alto nivel:** para obtener mediciones sobre eventos simples y comunes a la mayoría de las arquitecturas (preset-events)
  - **Interfaz de bajo nivel:** pensada para eventos más complejos y dependientes de la arquitectura (native-events).
- Está basada en perf pero se utiliza dentro del código fuente de las aplicaciones (instrumentación):
  - Existe un conjunto de comandos “papi\_” (NO son para obtener medidas)
  - Dentro del código fuente de la aplicación se debe incluir papi.h y se utilizan funciones “PAPI\_”
- No viene como parte de Linux, se debe descargar e instalar.

<sup>1</sup><http://icl.cs.utk.edu/papi>

# Contadores Hardware – Herramientas

## PAPI

16

- Para ver la listas de eventos disponibles en nuestra arquitectura se debe ejecutar el siguiente comando:

```
# papi_avail
...
      Name      Code   Avail Deriv Description (Note)
PAPI_L1_DCM 0x80000000 Yes  No  Level 1 data cache misses
PAPI_L1_ICM 0x80000001 Yes  No  Level 1 instruction cache misses
PAPI_L2_DCM 0x80000002 Yes  Yes Level 2 data cache misses
PAPI_L2_ICM 0x80000003 Yes  No  Level 2 instruction cache misses
PAPI_L3_DCM 0x80000004 No   No  Level 3 data cache misses
PAPI_L3_ICM 0x80000005 No   No  Level 3 instruction cache misses
...
```

# Contadores Hardware – Herramientas

## PAPI

17

- Instrumentación con PAPI dentro del código de una aplicación:

```
#include <papi.h>
main(){
    int EventSet = PAPI_NULL;
    long_long values[3];
    PAPI_library_init(PAPI_VER_CURRENT); //Primera inicialización
    PAPI_create_eventset(&EventSet); //Crea el conjunto de eventos
    //Agrega eventos
    PAPI_add_event(EventSet, PAPI_TOT_INS);
    PAPI_add_event(EventSet, PAPI_TOT_CYC)
    PAPI_add_event(EventSet, PAPI_LST_INS)
    for (counter = 0; counter < stoppoint; counter++){ //Ejecuta varias veces algun tipo de computo
        PAPI_library_init(PAPI_VER_CURRENT); //Inicializa los contadores, los pone en cero
        PAPI_start(EventSet); //Comienza a contar
        //Realizar Computo
        PAPI_read(EventSet, values); //Lee los valores de los contadores al finalizar la ejecución
        printf ("Instrucciones:%lld Ciclos:%lld Load Store%lld\n", values[0], values[1], values[2]);
        PAPI_stop(EventSet, values) //Detiene PAPI
    }
}
```

Todas las funciones PAPI retornan códigos de error que permiten conocer si la invocación fue o no exitosa.

# Contadores Hardware – Herramientas

## PAPI

18

- Ejemplo **PAPI\_simple.c** : cuenta ciertos eventos al calcular varias veces un factorial.
- Compilar:

```
gcc -I$PAPI_PATH/include -O0 PAPI_simple.c $PAPI_PATH/lib/libpapi.a -o PAPI_simple
```

- Ejecutar:

```
# ./PAPI_simple
...
Factorial de 0 es 1
Instrucciones ejecutadas: 3098 Ciclos: 7317 Instucciones Load Store 1287

Factorial de 1 es 1
Instrucciones ejecutadas: 1445 Ciclos: 1629 Instucciones Load Store 604

Factorial de 2 es 4
Instrucciones ejecutadas: 1453 Ciclos: 1287 Instucciones Load Store 611
...
```

# Agenda

19

## PMCTrack



# Contadores Hardware – Herramientas

## PMCTrack

20

- **PMCTrack**<sup>1</sup>: Es una herramienta de monitorización de rendimiento orientada a sistemas operativos (Linux).
- Diseñada para ayudar a los desarrolladores del kernel a implementar algoritmos de planificación en Linux que aprovechan los datos de los contadores hardware (PMC) para realizar optimizaciones en tiempo de ejecución.
- Tiene soporte para la mayoría de las arquitecturas.
- La instalación requiere aplicar un parche al kernel de Linux y recompilarlo.

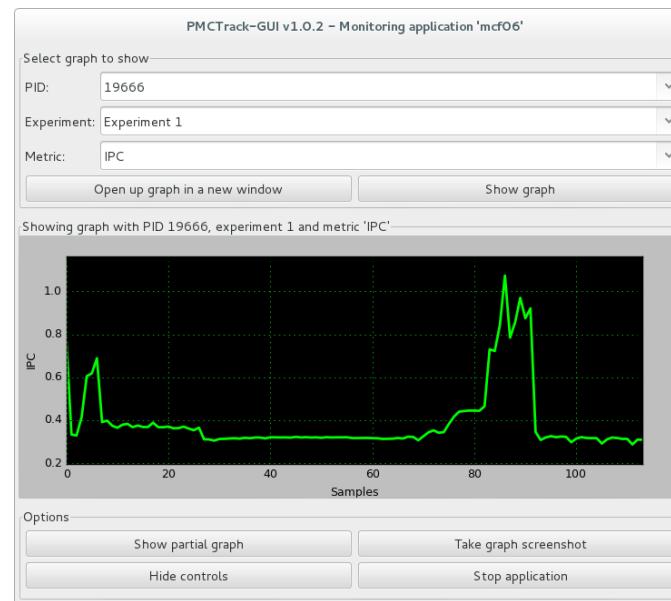
<sup>1</sup><https://pmctrack.dacya.ucm.es>

# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

21

- Aunque está orientado al SO, también se puede utilizar a nivel de usuario combinando funcionalidades similares a perf y PAPI.
- Se puede gestionar por línea de comando o por una interfaz gráfica basada en Python (PMCTrack-GUI)



# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

22

- El comando `pmc-events` permite acceder a información de la arquitectura:

```
# pmc-events -I
[PMU 0]
pmu_model=x86_intel-core.haswell-ep
nr_fixed_pmc=3
nr_gp_pmc=8
```

- Y de los contadores disponibles:

```
# pmc-events -L
[PMU 0]
instr_retired_fixed
unhalted_core_cycles_fixed
unhalted_ref_cycles_fixed
instr
cycles
unhalted_core_cycles
...
```

# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

23

- El comando `pmctrack` también permite acceder a información de los contadores de forma similar a perf. Soporta tres modos de uso:
  - **Time-Based Sampling (TBS)**: Los valores de los contadores se recolectan cada ciertos intervalos regulares de tiempo.
  - **Event-Based Sampling (EBS)**: Los valores de los contadores se recolectan cada vez que un contador alcanza determinado valor.
  - **Time-Based system-wide monitoring mode**: Una variante de TBS, pero la información monitoreada la provee cada CPU en el sistema y no una aplicación.
- Incluye el concepto de **contadores virtuales**: contadores que no son parte de los PMC sino que sus valores se obtienen de otros registros o sensores en el hardware (Por ejemplo: temperatura).

# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

24

```
#pmctrack -T 2 -c instr,llc_misses ./miAplicacion
[Event-to-counter mappings]
pmc0=instr
pmc3=llc_misses
[Event counts]
nsample    pid      event        pmc0        pmc3
      1  10767    tick  2017968202  30215772
      2  10767    tick  1220639346  24866936
      3  10767    tick  1204660012  24726068
      4  10767    tick  1524589394  20013147
      5  10767    tick  1655802083  9520886
      6  10767    tick  2555712483  18420142
      7  10767    tick  2222232510  19594864
...
...
```

- Ejemplo del modo TBS donde la opción -T 2 significa que obtiene el valor de los contadores cada 2 segundos.
- Los contadores se pueden especificar mediante el nombre o mediante su dirección (Por ejemplo: 0x08)

# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

25

```
# pmctrack -c instr:ebs=500000000,llc_misses ./miAplicacion
[Event-to-counter mappings]
pmc0=instr
pmc3=llc_misses

[Event counts]
nsample    pid      event        pmc0        pmc3        virt0
      1  10839      ebs  500000078  892837  526489
      2  10839      ebs  500000047  9383500 1946166
      3  10839      ebs  500000050  9692922 2544250
      4  10839      ebs  500000007  10017122 2818298
      5  10839      ebs  500000012  9907918 3055236
      6  10839      ebs  500000011  10335579 3108215
      7  10839      ebs  500000046  10735151 3118713
...
...
```

- Ejemplo del modo EBS donde se obtiene el valor de los contadores cuando el contador de instrucciones (instr) alcance el valor 500000000.

# Contadores Hardware – Herramientas

## PMCTrack – Modo Usuario

26

- También es posible caracterizar el rendimiento de un fragmento de código C específico (idem PAPI) mediante la biblioteca Libpmctrack (API).
- El programador puede obtener el valor de los contadores de cualquier parte del código simplemente encerrándolo entre las invocaciones a las funciones pmctrack\_start\_count() y pmctrack\_stop\_count().

```
#include <pmctrack.h>
int main(int argc, char *argv[]){
    pmctrack_desc_t* desc;
    char* virtual_cfg=NULL; //Contadores virtuales no utilizados (NULL)
    const char* strcfg[]={ "pmc1=0x11,pmc2=0x08"}; // string de la opcion -c

    desc=pmctrack_init(100); //Inicializa el descriptor de pmctrack
    pmctrack_config_counters(desc,strcfg,virtual_cfg,0)); //Configura los contadores
    pmctrack_start_counters(desc); //Comienza a contar
    // Realizar computo
    pmctrack_stop_counters(desc); //Detiene el conteo
    pmctrack_print_counts(desc, stdout, 0); //Imprime el valor de los contadores
    pmctrack_destroy(desc); //Finaliza pmctrack
}
```