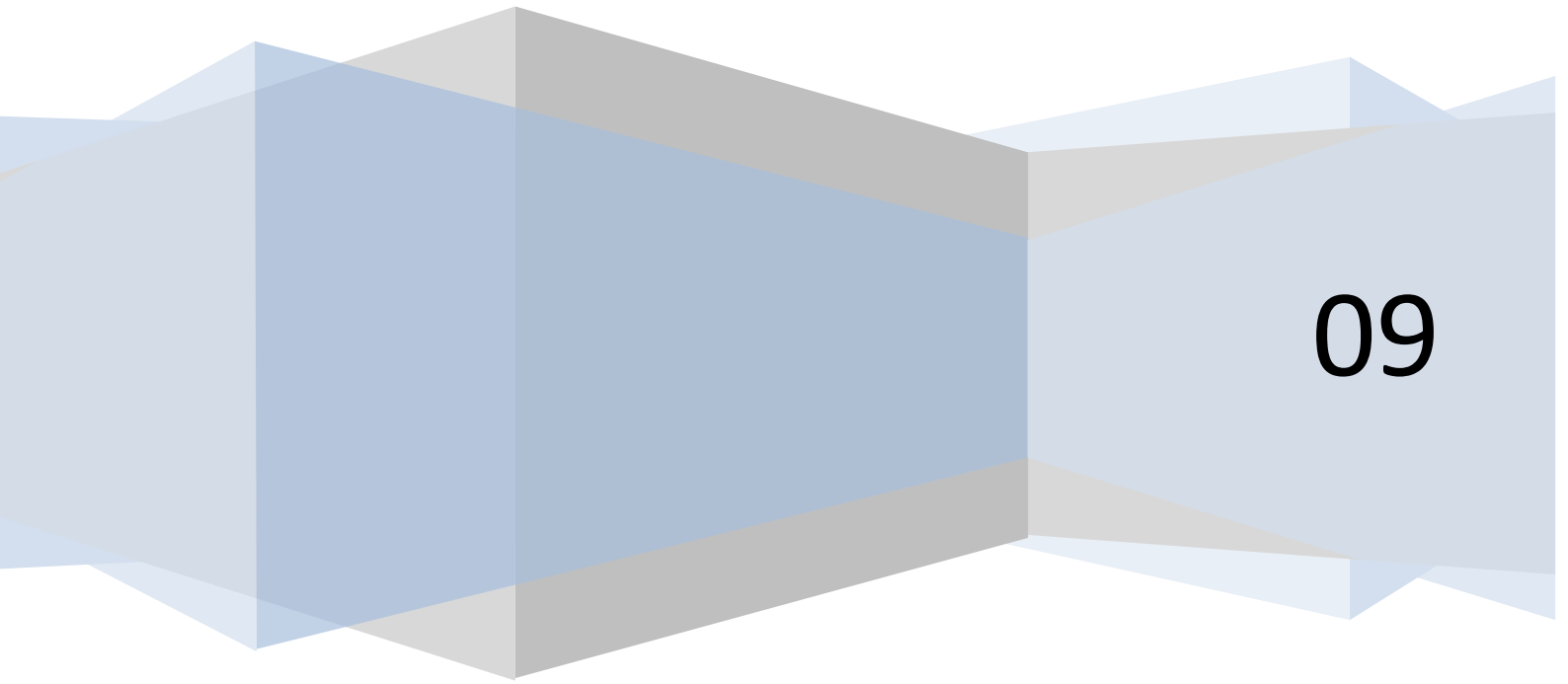
LAB4

# Complemento de ILP y Previo de TLP

## Laboratorio de Arquitectura e Ingeniería de Computadores

Valentin Puente



09

Revision 1.2: 21/04/2010 VM updated

# 1 INTRODUCCIÓN Y OBJETIVOS

El objetivo fundamental de esta práctica es realizar algunos experimentos sobre arquitecturas VLIW y multicore, y de realizar posibles recomendaciones arquitecturales a la vista de los resultados. La práctica servirá para completar algunos aspectos de ejecución superescalar no vistos en la práctica anterior y cómo introducción a la arquitectura que vamos a emplear en la práctica siguiente

A diferencia del las prácticas anteriores, esta práctica solo tienen una parte guiada y obligatoria para todo el mundo. En compensación, su contribución a la nota media de las prácticas será la mitad que las otras.

## 1.1 INTRODUCCION DE LOS NUEVOS ARQUITECTURAS

En esta práctica introduciremos nuevos targets que Simics es capaz de simular. No se poseen modelos arquitecturalmente precisos para estos sistemas, lo que hace más difícil realizar estudios complejos con ellas. No obstante, este laboratorio servirá fundamentalmente para introducir su uso.

El primer target, denominado *Vasa*, es un sistema basado en Intel Itanium ejecutando un Linux Red Hat. Como ya sabemos, el procesador del sistema es un VLIW. Su eficiencia, en parte, depende del compilador y su capacidad para planificar grupos de instrucciones en bloques de 128 bytes.

El segundo target modela un computador basado en un procesador Sun Microsystem UltraSPARCT1, denominado Niagara, ejecutando OpenSolaris. Este procesador usa una arquitectura SPARCv9. Posee 8 cores, cada uno de ellos con capacidad para ejecutar 4 threads. El sistema emplea realiza de “entrelazado” de cada uno de los 4 threads de forma fina en el core, de modo que el sistema operativo cree disponer de hasta 32 procesadores virtuales. Simics tiene diferentes versiones disponibles, desde 1 thread a 32 threads.

## 1.2 PUNTOS CALIFICABLES

Deberás entregar en formato ***pdf*** en través de la tarea abierta en WebCT un documento que responda a los siguientes apartados de la práctica:

# 2 PARTE DIRIGIDA

## 2.1 EXPLORACIÓN DE UN SISTEMA VLIW

Antes de proceder con este target, tenemos que hacerlo disponible desde el simulador. En el

El nuevo target se puede iniciar simplemente con desde tu workspace de Simics:

```
host$ ./simics targets/ia64-460gx/vasa-common.simics
```

La máquina disponible no tiene ni compilador no acceso al sistema de ficheros del *host*, por lo que nos dedicaremos a estudiar programas preexistentes.

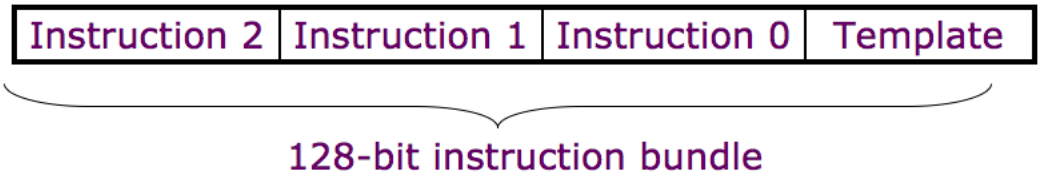



**Ilustración 1 Representación de los bundles en Simics,**

El Itanium es un procesador VLIW. El formato de cada "paquete" de instrucciones viene esquematizado en la Ilustración 1. Cada paquete o *bundle* contiene 3 instrucciones de 41 bits y un *template* de 5 bits que representa la relación de este paquete con los *bundles* adyacentes y que representan un grupo. Los grupos son conjuntos de instrucciones que pueden ejecutarse en paralelo. La primera instrucción en el *bundle* se denomina slot-0, la segunda slot-1, etc. Algunas veces alguno de los slots se emplean para operandos inmediatos del las otras instrucciones del *bundle*. El comando de simics **step-cycle** ejecuta un slot del *bundle* cada vez. Simics está pensado para procesar en cada instante una instrucción por procesador, luego la instrucción original debe ser dividida para poder ejecutarse. Dado que cada instrucción no tiene definida una dirección de acceso claramente identificada, Simics accede a ella usando la dirección de comienzo del bundle combinado con el número de slot. Esta es la información que aparece cuando se desensambla el código. Dado que los *bundles* están alineados a 16-bytes, los 4 bits menos significativos del bundle son nulos. Estos 4 bits los utilizara para codificar el número de slot de la instrucción. Por ejemplo, las siguientes instrucciones representan los tres slots del bundle situado en la dirección **0xe0000000044031f0**.

```
<v:0xe0000000044031f0>  <p:0x00000000044031f0>  1988005022040090a054080000000020  mov.m  r17  =
ar.fpsr

<v:0xe0000000044031f1> <p:0x00000000044031f1> 1988005022040090a054080000000020 mov.m ar.fpsr =
r18

<v:0xe0000000044031f2> <p:0x00000000044031f2> 1988005022040090a054080000000020 nop.b 0x0
```

Ejecuta los siguientes comandos de uno en uno en el target. Para cada uno de ellos, interrumpe cuanto antes la ejecución con un ^C en la consola de simics. Esto pausara la ejecución de la aplicación, como ya sabemos. El método es mucho más impreciso (y con resultados no repetibles) que el uso de las *magic*. Sin embargo, será aceptable para el tipo de análisis que vamos a realizar aquí.

Los comandos a ejecutar son

```
target# bzip2 /var/log/lastlog;bunzip /var/log/lastlog.bz2

target# yes

target# grep Swap /var/log/dmesg
```

Ejecuta bloques de 30 instrucciones (es decir 10 *bundles*) cada vez con el tracer habilitado. Esto lo puede hacer con:

```
simics>si 30
```

Para al menos 20 *bundles* , establece:

- ¿Cuál es el porcentaje de *bundles* que tienen una, dos o tres instrucciones útiles?
- ¿Cual es IPC teórico del sistema si asumimos que se realiza el fech y ejecución de dos bundles por ciclo (es decir, como funciona realmente el Itanium)? ¿Cuál es realmente el IPC práctico, a juzgar por el muestreo realizado?
- ¿Cuál crees que la relación entre los ***nops*** y las otras instrucciones en el bundle o adyacentes?
- ¿Cuándo aparecen instrucciones predicadas?[1] ¿En promedio, cuantas instrucciones se predican por bundle?

## 2.2 EJECUTANDO CÓDIGO EN UN CMP (CHIP MULTI-PROCESSOR)

En esta sección, cómo ya se comento en la introducción, deberás ejecutar código en sistema con un procesador multicore basado en Sun UltraSparc T1 (Niagara). Cada uno de los 8 cores del procesador tiene soporte para ejecución simultanea de 4 *threads*. Esto quiere decir, que en cada ciclo puede seleccionar una instrucción de los 4 *threads* disponibles y la ejecutara. El sistema operativo maneja el contexto de cada *thread* como una CPU diferente. Por lo tanto, desde su punto de vista el sistema poseerá 32 CPUs. En el segundo tema de paralelismo a nivel de *thread*, veremos cómo es posible implementar esa funcionalidad.

En esta parte de la práctica prepararemos los checkpoints preciosos para llevar a cabo la práctica siguiente. Para ello comenzaremos con la instalación de los discos virtuales de simics. A continuación ejecutaremos unas cuantas cargas multiprogramadas. Una carga multi-programada es un caso típico de aplicación thorughput computing. Para 1 y 32 contextos hardware repetir lo siguiente:

[1] Las instrucciones predicadas son aquellas que llevan un registro de predicación entre paréntesis antes del nemónico de la instrucción.

1. Arrancar Simics sin ningún argumento. Observa donde has copiado las imágenes en los pasos anteriores para descubrir el path al target. El sistema operativo que está usando el target es Solaris 10. La instalación es mínima; no incluye ni compilador ni muchas de las herramientas comunes en Linux.

2. Establecer el número de CPUs (1 o 32). Para fijar Por ejemplo, para fijar 1 CPU haremos:

```
simics>$num_cpus=1
```

3. Arrancar el target. Esto va a ser muy largo para la máquina con 32 contextos.

```
simics> run-command-file targets/niagara-simple/niagara-simple-solaris-common.simics
```

4. Copiar los benchmarks facilitados en WebCT.

5. Crear un checkpoint. Requerirá al menos 250MB de espacio en disco disponible para la configuración mayor.

6. Ejecutar varias veces (de forma concurrente) cada benchmark de los facilitados y observar cómo se comporta el sistema[2]. En Solaris podéis el comando "**prstat**" para ver los procesos en ejecución. Es equivalente al "**top"** de Linux.

7. Utiliza los comandos que consideres oportunos para descubrir cuantos procesos caen en cada core en cada una de las configuraciones evaluadas.

8. Una vez finalizado el proceso con 1 CPUs, repetir con 32. Recuerda que hacen falta otros 250MB para guardar los checkpoints. Si no tuvieses suficiente espacio, borra los anteriores. Procura conservar los de 32 CPUs para la práctica siguiente.

# 3 REFERENCIAS

[1].- Documentación de OpenSolaris http://opensolaris.org/os/community/documentation/.

[2] Ten cuidado con los ficheros de entrada en algunos de los benchmarks. Cada proceso deberá de manejar el suyo.

[2].- Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun, "Niagara: A 32-Way Multithreaded SPARC Processor",IEEE MICRO Magazine, March-April 2005, and presented at Hot Chips 16, August 2004.