Trabajo practico final - Organizacion del Computador 2 - UBA Manuel Ferreria - LU 199/10 - Septiembre 2012

==========

Meta:

Implementar por software algunos mecanismos que implementan las microarquitecturas de los procesadores modernos, para poder evaluar rendimiento de estos mecanismos de forma sintetica, sin tener que construirlos realmente.

El proyecto se concentro en dos de los temas mas criticos a la hora de performance en la microarquitectura, el hit rate de la cache y el predictor de saltos.

Ambos simuladores fueron construidos alrededor de las herramientas que provee Intel, a traves de la libreria PIN. (http://www.pintool.org/)

Notas:

En la carpeta "pin/" se encuentra la libreria completa, tal como se descarga de la pagina de Intel, para que el docente pueda explorar todos los otros ejemplos presentes y ver la potencia de la libreria.

==========

Instrucciones de compilacion:

Para compilar los simuladores hechos para este TP, basta con:

cd tpOrga2/pin ./make_tp

========== Simulador de Cache multinivel: ----------

a) Para ejecutar:

Desde la carpeta tpOrga2/pin/:

```
./cachesim.sh <parametros> -- <ejecutable>
```

Por ejemplo, para correr un "ls -la" con 6 vias de asociatividad usando LRU

```
./cachesim.sh -V 6 -lru 1 -- ls -la
```

Y los resultados se pueden ver en "cache.out"

Los parametros posibles son: -L [default 4] Especifica la cantidad de bits que tiene una direccion de memoria que se usan para indexar dentro de la linea (tamaño de la linea) -L1 [default 16] Especifica la cantidad de bits que tiene una

direccion en la L1 (tamaño de la cache) -L2 [default 20] Especifica la cantidad de bits que tiene una direccion en la L2 (tamaño de la cache) -V [default 4] Especifica la cantidad de vias que hay en una linea de cache -lru [default 0] Utiliza desalojo LRU de la linea en vez de FIFO -o [default cache.out] Especifica el nombre del archivo de salida del informe

b) Detalles

El codigo del simulador de cache se encuentra en: tpOrga2/pin/source/tools/Tp/cache.cpp

El simulador esta pensado para mantener un equivalente de la memoria cache writeback, pero en memoria RAM.

Es decir, cuenta con las estructuras de datos necesarias para ser una cache (el bit de presente, el bit de dirty y el tag asignado a esa linea).

Existe una funcion unica para acceder a la cache, "accessCache", y sus parametros indican si es lectura o escritura, y en que nivel de la cache tiene que buscar el dato en la direccion de memoria.

Esta funcion se encarga de direccionar la linea en el nivel de cache accedido y ve si matchea el tag en todas las vias. Si no lo hace, intenta hacer un read en un nivel superior, a ver si lo encuentra. Si se pasa el maximo nivel de cache, entonces hace el equivalente de buscar en la RAM. Luego, si necesita, libera una via de esa linea, y guarda el dato en la cache de mas abajo. Esto permite que el proximo acceso sea a la L1, maximizando la eficiencia.

Si encuentra el dato en un nivel, simplemente contabiliza ese hit, para computarlo en el hitrate final.

c) Notas sobre el simulador de cache

El simulador esta armado para ser totalmente extensible a la cantidad de niveles de memoria necesarios, de uno a cinco, o mas incluso.

El unico cambio necesario seria agregar los parametros para customizarlo, que se podria hacer facilmente.

========== Predictor de saltos: ----------

a) Para ejecutar: Desde la carpeta tpOrga2/pin/

./jmpsim.sh –

Por ejemplo, para correr un "ls -la" con una bht de 2^4 entradas

```
./jmpsim.sh -s 4 -- ls -la
```

Y los resultados se pueden ver, por default, en "jmp.out"

Los parametros posibles son: -s [default 12] Especifica la cantidad de bits de la tabla BHT (tamaño, la cantidad de bits de una direccion que se toman para indexar una tabla con maquina de estados) -o [default jmp.out] Especifica el nombre del archivo de salida del reporte

b) Detalles

El codigo del simulador de prediccion de saltos se encuentra en: tpOrga2/pin/source/tools/Tp/jmp.cpp

El simulador esta pensado para intentar entender mecanismos basicos de prediccion de salto, para evitar stalls de pipeline cuando existen branches condicionales.

El mecanismo interno es sencillo. Se crea un binding usando PIN al programa que se ejecuta por parametro, de modo que cada vez que haya una instruccion que sea un jump

Se implementaron varios mecanismos de prediccion, algunos obvios y sin memoria, 'Siempre Salta' y 'Nunca Salta' como prueba de como ocurren los saltos en un programa; algunos mas interesantes como 'Salta si y solo si es a una direccion anterior' que resulta ser un predictor decente para no tener logica alguna; y dos mas avanzados: '2 bit Saturation counter' y '2 bit Histeresis counter', que funcionan mediante maquinas de estado indexadas por el instruction pointer.

Algunos de estos metodos fueron tratados en el libro de Henessy-Patterson "Computer Architecture".

========== Programas interesantes para probar ----------

Las comillas son para mostrarlas aca, en los simuladores van sin comillas

"ls -la" "traceroute google.com" "bash" (y usarla normalmente. Para terminar, desloguearse) "firefox" "locate archivo" (incluso si no existe, hay que usar sudo para el script) "ssh @" "svn checkout http://tpsh.dc.uba.ar/svn/materias-orga2/2012-cuat2 orga2"