Arquitecturas Avanzadas
Práctica 2

Jerarquía de memoria

Rafael Ubal
Mónica Serrano
Alejandro Valero
Julio Sahuquillo

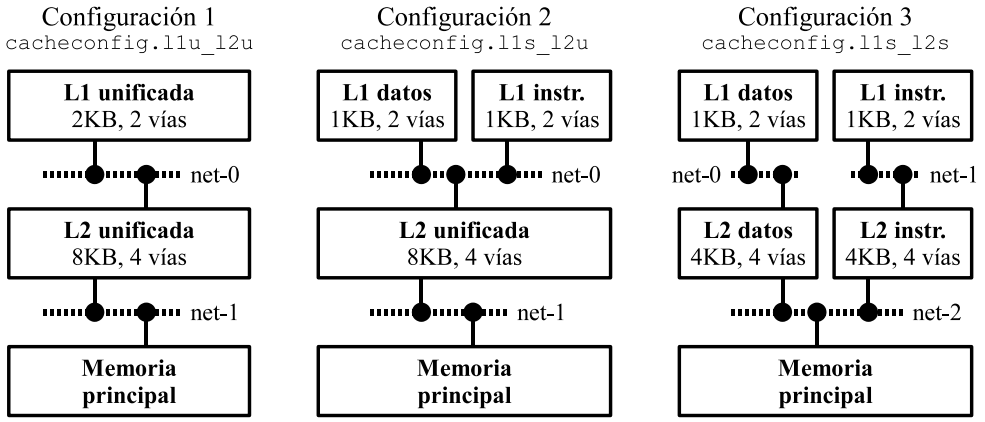
Objetivos

- Familiarizarse con las propiedades que caracterizan la jerarquía de memoria de un procesador, así como sus posibilidades de configuración.
- Conocer nuevas alternativas y herramientas gráficas para la presentación de resultados que faciliten su interpretación.
- Ejercitar la evaluación de prestaciones con casos de estudio relacionados con la jerarquía de memoria.

1 Caches de datos e instrucciones

En la primera parte de esta práctica se propone la jerarquía de memoria como estructura sobre la que evaluar diferentes alternativas de diseño. Multi2Sim da una gran flexibilidad en la configuración de la jerarquía de memoria. Entre otras características, las memorias cache pueden definirse como caches exclusivamente de datos, caches exclusivamente de instrucciones, o caches unificadas (tanto instrucciones como datos).

En este caso de estudio, se propone evaluar las prestaciones de los diseños resultantes de combinar dos niveles de caches divididas o unificadas. Las fases de simulación y representación vienen dadas como referencia para los siguientes ejercicios. Las combinaciones posibles de configuración son tres, puesto que un nivel 1 unificado seguido de un nivel 2 dividido no está permitido en Multi2Sim. La siguiente figura muestra los esquemas para cada configuración:



Las caches se conectan entre sí y con memoria principal mediante redes de interconexión, representadas con las líneas punteadas y etiquetadas con el prefijo *net-*. En las configuraciones 1 y 2 hacen falta dos redes de interconexión, mientras que la configuración 3 requiere tres redes.

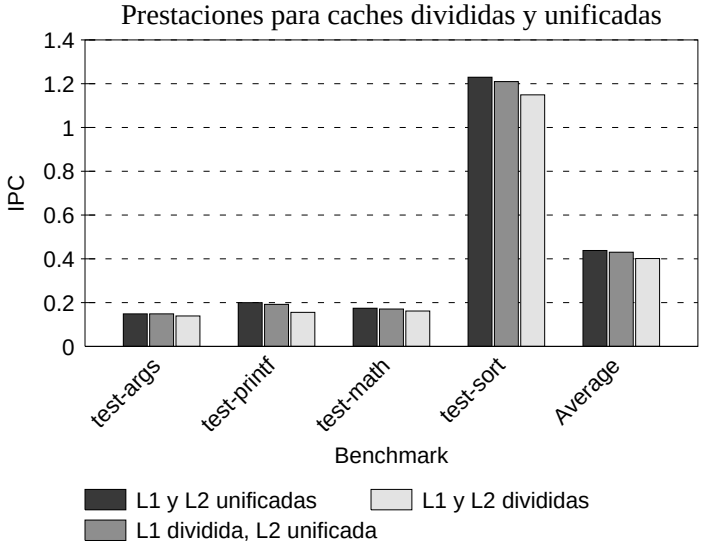
1.1 La herramienta gráfica BarPlot

En los casos de evaluación de prestaciones en los que se varía el tamaño de una estructura, como ocurría en la práctica 1, es conveniente la generación de gráficas de puntos unidos por rectas. Con este esquema, se observa fácilmente la evolución de un determinado índice de prestaciones.

Sin embargo, este tipo de gráficas no es intuitivo a la hora de evaluar diferentes diseños hardware que no implican una variación del tamaño de una estructura. Así ocurre con el presente caso de estudio, en el que se busca obtener las prestaciones de diferentes configuraciones de la jerarquía de memoria.

En este tipo de estudios, es deseable representar los resultados en gráficas de barras, todas ellas clasificadas en varios grupos con el mismo número de barras. Generalmente, la barra i dentro de los diferentes grupos se corresponde con los resultados de un determinado diseño, mientras que cada grupo de barras representa la evaluación de todos los posibles diseños utilizando una determinada carga computacional. En estos casos, se suele incorporar un último grupo de barras correspondiente a la media de los anteriores, que da información acerca del comportamiento de cada diseño para todos los benchmarks en promedio.

La siguiente gráfica es una anticipación de la representación gráfica que buscamos en este caso de estudio:



Una figura como ésta podría generarse también con `gnuplot`, pero esta herramienta proporciona una interfaz algo tediosa para la creación de gráficas de barras. Para estas prácticas, proponemos el uso de la herramienta `barplot`, que es un *script* de Python con una interfaz mucho más sencilla y diseñada especialmente para este propósito. La herramienta está incluida en el paquete de prácticas descargado. La sintaxis de ejecución de `barplot` es la siguiente:

```
./barplot.py <plotfile> <figure.eps>
```

donde `<plotfile>` es el fichero de texto que contiene los datos a representar, y `<figure.eps>` es la gráfica en formato EPS que genera la aplicación. El formato de los ficheros de entrada de `barplot` es éste:

```
<Key1> = <Value1>
<Key2> = <Value2>
...

d11 d12 ... d1m
d21 d22 ... d2m
...
dm1 dm2 ... dmn
```

Los pares `<Key>=<Value>` indican ciertos atributos de la gráfica. Se puede obtener una enumeración de todos los posibles parámetros ejecutando la orden `barplot` sin argumentos. Por otro lado, los valores $d_{i,j}$ forman una matriz con los datos a representar. Cada fila i de esta matriz se corresponde con el grupo de barras i, mientras que el elemento j dentro de la fila i indica la altura de la barra j dentro del grupo i.

1.2 Fase de simulación

Para generar la gráfica anterior, necesitamos lanzar simulaciones utilizando las cuatro cargas computacionales disponibles (args, test-printf, test-math y sort) y modelando las tres configuraciones de la jerarquía de memoria mostradas anteriormente. La jerarquía de memoria se especifica en Multi2Sim con el argumento `-cacheconfig <file>`, donde `<file>` es un fichero de texto que contiene la configuración de cada componente.

En el paquete de prácticas, se proporciona el archivo de configuración `cacheconfig.l1s_l2u` correspondiente a la configuración de memoria con el nivel 1 dividido, y el nivel 2 unificado. Su formato se describe a continuación:

```
[ CacheGeometry l1topo ]
Sets = 8
Assoc = 2
BlockSize = 64
Latency = 2

[ CacheGeometry l2topo ]
Sets = 32
Assoc = 4
BlockSize = 64
Latency = 20

[ Net net-0 ]
Topology = Bus
LinkWidth = 32

[ Net net-1 ]
Topology = Bus
LinkWidth = 32

[ Cache dl1 ]
Geometry = l1topo
LoNet = net-0

[ Cache il1 ]
Geometry = l1topo
LoNet = net-0

[ Cache l2 ]
Geometry = l2topo
HiNet = net-0
LoNet = net-1

[ MainMemory ]
HiNet = net-1
BlockSize = 64
Latency = 200

[ Node 0 ]
Core = 0
Thread = 0
DCache = dl1
ICache = il1
```

Como se puede observar, el fichero está compuesto por secciones con nombres entre corchetes, dentro de las que se encuentran elementos de tipo `<variable> = <valor>`. En primer lugar, las secciones `[ CacheGeometry <nombre> ]` definen perfiles o topologías de cache. En este fichero, se definen dos topologías, llamadas `l1topo` y `l2topo`, que se usarán para las cache de nivel 1 y 2, respectivamente. La primera de ellas define una cache de 8 conjuntos, 2 vías y bloque de 64 bytes (1KB), mientras que la segunda define 32 conjuntos, 4 vías y bloque de 64 bytes (8KB). Se recuerda que el tamaño total de la cache viene dado por la multiplicación del número de conjuntos, la asociatividad y el tamaño de bloque. La definición de la topología incluye el tiempo de acceso, que será de 2 y 20 ciclos, respectivamente.

Las dos siguientes secciones `[ Net <nombre> ]` definen las redes de interconexión. En este caso, se crean dos de ellas, llamadas `net-0` y `net-1`, respectivamente. Ambas tendrán

una topología de bus y un ancho de banda de 32 bytes por ciclo. Por tanto, un bloque de cache se transferirá en dos ciclos.

Las secciones `[ Cache <nombre> ]` instancian memorias cache. La cache L1 de datos se llama `dl1`, tiene la topología `l1topo` y se conecta al nivel inferior por la red `net-0`. La cache `il1` es la cache L1 de instrucciones, también con topología `l1topo` y conectada a la red `net-0` por abajo. Finalmente, la cache L2 unificada es `l2`, tiene la topología `l2topo`, se conecta por arriba a la red `net-0`, y por abajo a la red `net-1`.

La sección `[ MainMemory ]` define los parámetros de la memoria principal. En este caso, se indica que se accede a ella a través de la red `net-1`, que su tamaño de bloque es de 64 bytes, y la latencia de acceso es de 200 ciclos.

Por último, la sección `[ Node 0 ]` especifica las vías de acceso a la jerarquía de memoria por parte del nodo de computación situado en el *core* 0 y *thread* 0 del procesador. Con los nombres de cache asignados a las variables `DCache` e `ICache` indicamos que el nodo debe acceder a los datos en la cache con nombre `dl1` y tiene que recoger el código a ejecutar de la cache `il1`.

La tabla siguiente lista todas las posibles variables que se pueden usar en cada sección del archivo de configuración de cache.

Sección	Variable

Geometry `<name>`	`Sets`
	`Assoc`
	`Blocksize`
	`Latency`
	`Policy`
	`ReadPorts`
	`WritePorts`

`Net <name>`	`LinkWidth` `Topology`

`Cache <name>`	`Geometry`
	`HiNet`
	`LoNet`

`MainMemory`	`HiNet`
	`Latency`
	`BlockSize`

`Node <num>`	`Core`
	`Thread`
	`DCache`
	`ICache`

Ejercicio 0

Se pide al alumno crear los archivos de configuración de memoria para las dos configuraciones restantes (`cacheconfig.l1u_l2u` y `cacheconfig.l1s_l2s`). Todas las configuraciones se deben especificar para un sólo nodo de computación. Asimismo, todas las memorias deben mantener un tamaño de bloque fijo de 64 bytes. También asumiremos que el tiempo de acceso para un determinado nivel de la jerarquía es el mismo que hemos considerado en la configuración dada como ejemplo. Finalmente, todas

las redes de interconexión serán de tipo bus y el ancho de banda estará fijado a 32 bytes por ciclo.

A continuación se muestra el contenido del *script* con nombre `cache.simul`, también disponible en el paquete de prácticas, utilizado para generar los resultados:

```
#!/bin/sh

resfile="cache.res"
tempfile=$(mktemp)
rm -f $resfile
for workload in test-args.i386 test-printf.i386 test-math.i386 test-sort.i386; do
    for config in l1u_l2u l1s_l2u l1s_l2s; do
        ./m2s -cacheconfig cacheconfig.$config $workload >/dev/null 2>$tempfile
        line=$(cat $tempfile | grep sim.ipc)
        ipc=$(echo $line | awk '{print $2}')
        echo -n "$ipc " >> $resfile
        echo -n "."
    done
    echo >> $resfile
done
rm -f $tempfile
echo
```

En el código anterior se puede observar que los datos generados y almacenados en `cache.res` ya se producen con el formato requerido por `barplot` posteriormente. Concretamente, los resultados correspondientes a un mismo *benchmark* se vuelcan en una misma línea con la orden `echo -n`, y antes de proceder con la siguiente iteración, se añade un salto de línea.

1.2.1 Otros parámetros de configuración

Caches perfectas

Los parámetros de configuración `-iperfect` y `-dperfect` permiten especificar una cache perfecta de instrucciones y de datos, respectivamente. Para ello, se ha de establecer su valor a `t` (*true*).

Información adicional sobre la simulación de la memoria

Para conocer más al detalle la simulación de toda la jerarquía de memoria del sistema, Multi2sim ofrece la posibilidad de generar dos tipos de archivos con información que puede ser útil, por ejemplo, para la depuración del código del simulador o para el análisis de las prestaciones obtenidas. Los parámetros de configuración implicados son `-report:cache` y `-debug:cache`.

`-report:cache <filename>`

Esta opción especifica el archivo donde se volcará un informe estadístico detallado de la simulación de la jerarquía de memoria. Dicho informe contiene estadísticas tales como el número de accesos a cada cache, el total de aciertos y fallos y su ratio, los reemplazos (*evictions*), las lecturas y escrituras y sus aciertos y fallos, etc.

`-debug:cache <filename>`

Mediante este parámetro se le indica al simulador el archivo que contendrá la evolución cronológica de todos los eventos producidos por cada una de las operaciones de memoria ejecutadas. Para cada evento, se detalla el ciclo en el que ocurre, la instrucción que lo desencadena, la dirección de memoria accedida, la cache implicada, si se produce un acierto o un fallo, el bloqueo de directorios, entre otros.

Como ejemplo, vamos a generar archivos de cache de los dos tipos descritos para el benchmark `test-args` con la configuración de caches definida en los ficheros `cacheconfig.l1u_l2u` y `cacheconfig.l1s_l2u`.

```
#!/bin/sh
tempfile=$(mktemp)
./m2s -cacheconfig cacheconfig.l1u_l2u -debug:cache dc.l1u_l2u.test-args
-report:cache rc.l1u_l2u.test-args test-args.i386 >/dev/null 2>$tempfile
echo -n "."
./m2s -cacheconfig cacheconfig.l1s_l2u -debug:cache dc.l1s_l2u.test-args
-report:cache rc.l1s_l2u.test-args test-args.i386 >/dev/null 2>$tempfile
echo -n "."
rm -f $tempfile
```

1.3 Fase de representación

Para generar el archivo de entrada de `barplot` es necesario introducir la sección de cabeceras, seguida de la sección de datos. Como en la fase de simulación ya se ha tenido la precaución de extraer los resultados con el formato apropiado, la generación de la sección de datos simplemente supone adjuntar el fichero `cache.res`. Éste es un posible listado del *script* correspondiente a la fase de representación, llamado `cache.plot`:

```
#!/bin/sh

plotfile=$(mktemp)
cat << EOF >> $plotfile
XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
Key = 'L1 y L2 unificadas' 'L1 dividida, L2 unificada' 'L1 y L2 divididas'
Averages = True
XLabel = 'Benchmark'
YLabel = 'IPC'
Title = 'Prestaciones para caches divididas y unificadas'
EOF

cat cache.res >> $plotfile
./barplot.py $plotfile cache.eps
rm -f $plotfile
```

Las variables de `barplot` utilizadas para generar la gráfica son las siguientes:

- `XTics`: etiquetas de los bloques de barras. Debe haber tantas como filas tenga la matriz de datos.
- `Key`: leyenda. Debe haber tantas entradas como columnas tenga la matriz de datos.
- `Averages`: si es `True`, se incluye un bloque de barras con los valores medios.
- `XLabel`, `YLabel`: etiquetas de los ejes X e Y.
- `Title`: título de la gráfica.

1.4 Fase de análisis

Ejercicio 1

Se propone la finalización de este caso de estudio por parte del alumno. Se deben incluir breves comentarios acerca de los resultados mostrados en la gráfica `cache.eps`, en la que se comparan tres alternativas de diseño de la jerarquía de memoria. ¿Cuál puede ser el motivo de la tendencia que se observa en las barras al dividir las memorias cache?

2 Configuraciones de memoria *Dual Core* y *Dual Processor*

En un *Dual Core* tenemos dos nodos de procesamiento, cada uno con su memorias cache L1 de datos e instrucciones privadas, los cuales comparten una única memoria cache L2. Por otra parte, un *Dual Processor* tiene la misma configuración en el primer nivel de cache (es decir, caches L1 de datos e instrucciones privadas para cada nodo de procesamiento), pero en este caso en segundo nivel de memoria no está unificado sino que hay una cache L2 privada para cada uno de los nodos.

Ambas configuraciones corresponden a procesadores multinúcleo. Este tipo de procesadores (tal y como sucede con los multihilo), proporcionan al Sistema Operativo la visión de tener múltiples procesadores lógicos donde se pueden ejecutar varias tareas al mismo tiempo. En Multi2Sim, las tareas software se llaman contextos, y su número está limitado por número de hilos multiplicado por el número de núcleos del procesador modelado.

Al modelar un único núcleo e hilo, Multi2Sim recibe en la propia línea de órdenes el nombre de la aplicación a ejecutar, tal y como habíamos visto hasta ahora. Sin embargo, cuando varios archivos ejecutables se asignan cada uno a un núcleo (o hilo), es necesario crear un archivo de configuración de contextos, llamado por ejemplo `ctxconfig`, con el siguiente aspecto:

```
[ Context 0 ]
Exe = test-args.i386
Args = arg1 arg2

[ Context 1 ]
Exe = test-sort.i386
```

Utilizando el parámetro `-ctxconfig` y asignándole el fichero anterior como valor, se ejecutarían las cargas `test-args` y `test-sort`, cada una de ellas en un núcleo. Además, habría que notificar a Multi2Sim que se pretende modelar un procesador multinúcleo de dos núcleos. Ésta es la orden completa:

```
./m2s -ctxconfig ctxconfig -cores 2
```

Ejercicio 2

Se pide simular las configuraciones de *Dual Core* y *Dual Processor* en Multi2Sim. Respecto a *Dual Core*, se asumirá una cache L2 el doble de grande (16KB) respecto a las caches L2 de *Dual Processor*. Para ello se aumentará el número de conjuntos manteniendo constante el tamaño de bloque y asociatividad.

Por otra parte, se pide generar 4 ficheros de contexto de acuerdo a las mezclas de cargas que se observan en la tabla inferior (aparte de test-args, ninguna carga utiliza argumentos de entrada). Ejecuta las mezclas tanto para *Dual Core* como *Dual Processor* y obtén el IPC del sistema. Representa los resultados en una gráfica de barras incluyendo los valores medios. El eje Y debe mostrar el IPC, mientras que el eje X debe mostrar las mezclas, cada una de las cuales tendrá dos barras (*Dual Core* y *Dual Processor*).

Mezclas	Benchmark 1	Benchmark 2
Mezcla1	test-args	test-printf
Mezcla2	test-math	test-printf
Mezcla3	test-sort	test-args
Mezcla4	test-sort	test-math

3 Algoritmo de reemplazo en la cache L1

Las memorias cache de correspondencia asociativa por conjuntos con n vías son aquellas en las que un mismo conjunto puede contener hasta n bloques de datos. Inicialmente, las vías vacías van ocupándose con los bloques nuevos traídos desde el nivel inferior de la jerarquía de memoria. Cuando ya no hay ninguna vía libre, actúa el algoritmo de reemplazo, que decide cuál de los bloques existentes va a ser sustituido por el nuevo.

En Multi2Sim, el algoritmo de reemplazo de bloque se especifica en el fichero de configuración de la jerarquía de memoria, concretamente en la sección que define una geometría de cache. La variable asociada es `Policy`, y los valores que puede tomar son `LRU`, `FIFO` y `Random`. Si no se especifica, su valor por defecto es `LRU`.

Ejercicio 3

Se pide realizar un estudio comparativo entre los tres algoritmos de reemplazo de bloque implementados en Multi2Sim en un sistema mono-core. La jerarquía de memoria base debe tener dos niveles de cache, y cada memoria cache debe estar unificada para datos e instrucciones, como queda definido en el fichero `cacheconfig.l1u_l2u`. La variación de la política de reemplazo sólo debe afectar a la cache L1.

El estudio debe incluir las fases de simulación, representación y análisis. Se debe generar una gráfica de barras, en la que cada grupo represente un *benchmark* y cada barra un algoritmo de reemplazo. La métrica de prestaciones a utilizar en este caso es la tasa de acierto en la cache (estadística `l1.hitratio`), aunque el número total de instrucciones ejecutadas o el número de reemplazos pueden ser útiles también a la hora de justificar los resultados (estadísticas `sim.inst` y `l1.evicts`).

Realización y entrega de la memoria de la práctica

La entrega correspondiente a esta práctica consiste en una breve memoria de los ejercicios realizados. A continuación se detalla la información requerida de cada ejercicio:

- Ejercicio 0: adjuntar el contenido de las configuraciones de memoria `cacheconfig.l1u_l2u` y `cacheconfig.l1s_l2s`.
- Ejercicio 1: adjuntar el contenido del fichero de resultados `cache.res` y la gráfica `cache.eps`. Razonar los resultados obtenidos.
- Ejercicio 2: adjuntar el contenido de los ficheros de configuraciones de memoria *Dual Core* y *Dual Processor*, así como los scripts de simulación y obtención de resultados. Mostrar los resultados obtenidos, la gráfica correspondiente y la discusión de resultados.
- Ejercicio 3: adjuntar los scripts de simulación y obtención de resultados. Mostrar los resultados obtenidos, la gráfica correspondiente y la discusión de resultados.

La entrega de la memoria de esta práctica se realizará individualmente por parte de cada alumno el primer día que empiece la siguiente práctica.