Memoria de la práctica 1

Manel Lurbe Sempere

Manuel José Martínez Baños

Contenido

Script para la obtención de resultados

Este es el Script asociado a la fase de simulación, encargado de lanzar cuatro ficheros de resultados ".res" por cada carga de prueba disponibles (test-args.i386, test-printf.i386, test-math.i386, test-sort.i38), obteniendo como finalización de la ejecución 4 ficheros.

Cada fichero de resultado (.res), es obtenido por el lanzamiento sobre un programa de prueba ya compilado del cual se lanzan cuatro cargas con diferente tamaño de cola de instrucciones, con los tamaños de banco de registros físicos solicitados.

Ahora adjuntamos parte del código del script para detallar su funcionamiento:

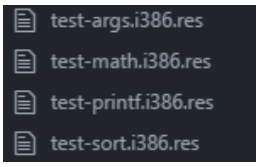
```
tempfile=$(mktemp)
#Creamos 5 ficheros .res de resultados por cada programa de prueba
for workload in test-args.i386 test-printf.i386 test-math.i386 test-sort.i386
do
  resfile="$workload.res"
  rm -f $resfile
  echo "Guardando en : $resfile"
  for iqsize in 4 8 16 32 64 #cola de instrucciones
  do
    for rfsize in 32 36 44 60 92 #banco de registros fisico
    do
      ./m2s -iq_size $iqsize -rf_int_size $rfsize -rf_fp_size $rfsize $workload > /dev/null 2> $tempfile
      line=$(cat $tempfile | grep sim.ipc)
      ipc=$(echo $line | awk '{print $2}')
      echo $ipc >> $resfile
      echo "$workload: simulacion con BANCO DE REGISTRO de tamanyo $rfsize con COLA INST: $iqsize  terminada"
    done
    echo >> $resfile
  done
  echo
done
echo
```

Ilustración 1 Script Ejercicio1.sh

Como vemos en la imagen, por cada programa de prueba se genera un fichero ".res" con su nombre, donde se almacena todos los resultados.

Vemos que recorremos dos **bucles for** por cada tipo de programa de prueba (for iqsize y for rfsize). En el último for, lanzamos la instrucción donde modificamos el tamaño de la cola de instrucciones con "-iq_size" adjuntando "**$iqsize"** los valores del primer for(4,8,16,32,64) y el banco de registro de enteros y como flotante con "-rf_fp_size" y "-rf_int_size", respectivamente. Poniendo el valor la variable "$rfsize" con los valores del segundo for.

Al final se obtendrán estos 4 ficheros:



Si analizamos la información de uno de ellos en detalle:

```
#Informacion de puntos de una linea
  #cola de instrucciones  diferente 4
  [ipc valor]
  0.6292      #banco de registros fisico diferente 32
  0.6483      #banco de registros fisico diferente 36
  0.643
  0.6788
  0.6501

#cola de instrucciones  diferente 8
0.7501
0.6471
0.7591
0.7225
0.7155
0.7269
```

Ilustración 2 Imagen recortado sobre un fichero "resultado.res" de ejemplo

Tenemos una serie de tantos bloques como diferentes valores **de cola de instrucciones** elegidas y tantos valores distintos por cada bloque como valores diferentes de **banco de registros**.

Script generación del gráfico

El **script** que corresponde a la fase de representación del gráfico es ejercicio1.plot. Este **Script** obtiene el contenido de los ficheros test-args.i386.res, test-printf.i386.res, test-math.i386.res, test-sort.i386.res y llamará por cada uno a **gnuplot** para generará una serie de graficas con los nombres: test-args.i386.eps, test-printf.i386.eps, test-math.i386.eps, test-sort.i386.eps.

El código del ejercicio1.plot es el siguiente:

```
for rfsize in "test-printf.i386" "test-math.i386" "test-args.i386" "test-sort.i386"
do

  resfile="$rfsize.res"
  figfile="$rfsize.eps"
  echo "Guardando $rfsize FIGURA EN  en : $figfile"
  plotfile=$(mktemp)
  cat << EOF > $plotfile
  set term postscript eps
  set title "$rfsize"
  set size 0.5, 0.5
  set key under
  set xlabel 'BANCO DE REGISTROS'
  set ylabel 'IPC'
  set xrange [-0.5:4.3]
  set xtics ( '32' 0, '36' 1, '44' 2, '60' 3, '92' 4)

plot '$resfile' every :::0::0 w linespoints t 'cola_instrucciones_4',   \
     '$resfile' every :::1::1 w linespoints t 'cola_instrucciones_8', \
     '$resfile' every :::2::2 w linespoints t 'cola_instrucciones_16',   \
     '$resfile' every :::3::3 w linespoints t 'cola_instrucciones_32',   \
     '$resfile' every :::4::4 w linespoints t 'cola_instrucciones_64'
EOF
  gnuplot $plotfile > $figfile
  rm -f $plotfile
done
```

Ilustración 3 Código para generar las 4 graficas, una por cada resultado.res

Como vemos en la implantación, $rfsize equivale ahora al nombre del fichero de **resultado.res**, en este caso Tenemos 4.

Por cada uno, se genera un fichero ".**eps**" con el comando "set term postscript **eps**", cuyo nombre está en el nombre de variable **figfile**. Ahora, si nos fijamos en un documento ".res" donde están los resultados del cual se genera la gráfica, vemos que tenemos por cada cola de instrucciones unos valores resultantes, que es el IPC, por cada tipo de cola de instrucciones.

Por tanto, en el **eje x** (set xlabel) tendremos los valores del "Banco de registros" y en el **eje y**(set ylabel) los valores de IPC.

Cada línea corresponderá por tanto a un valor de cola de instrucción y cada punto de la línea, corresponderá a un valor de banco de registro.

Por tanto, vemos que tenemos 4 líneas por las colas de instrucciones (4,8,16,32,64) cada una identificada por unos "linespoints" distintos. Cada línea tendrá 5 "linepoints" (32,36,44,60,92) que están indicados en la orden "set xtics ('**32**' 0, '**36**' 1, '**44**' 2, '**60**' 3, '**92**' 4)".

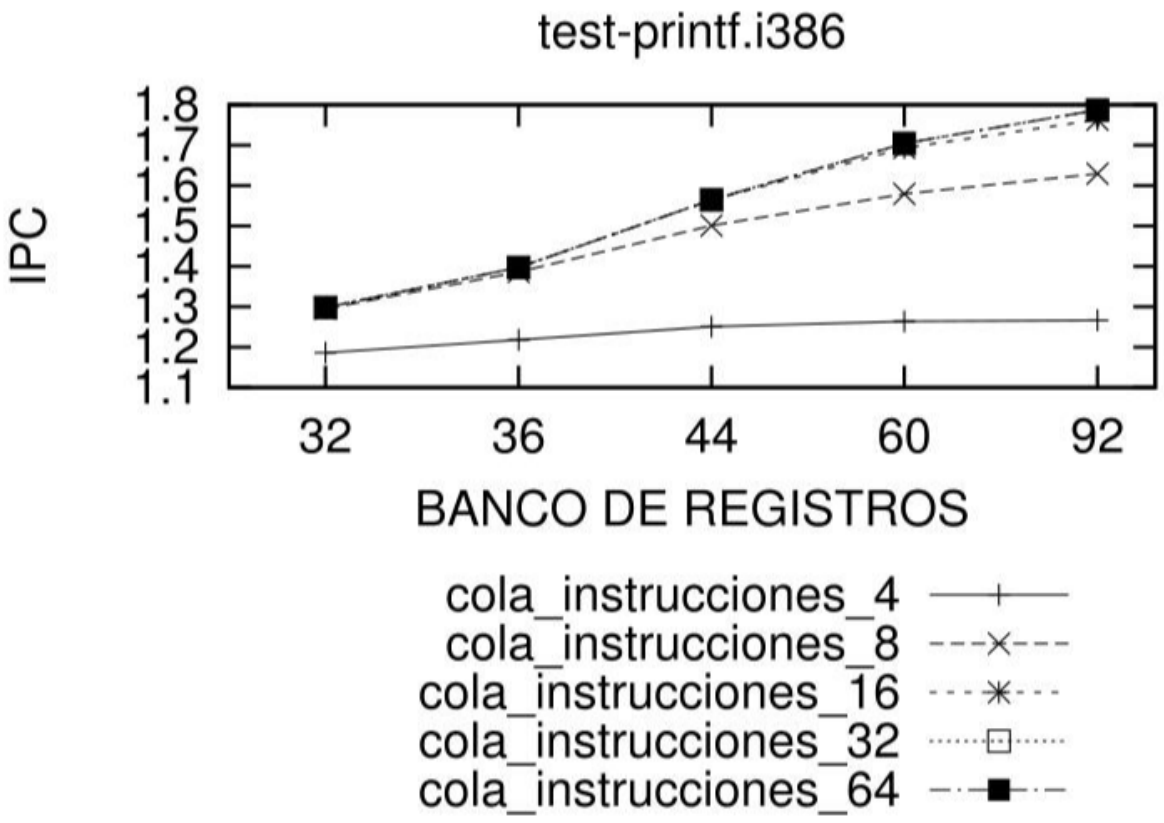
Análisis de los gráficos del Ejercicio 1

Ahora vamos a ver cada uno de los documentos “.eps”, estos también pueden ser .png si así lo deseamos, solo sería modificar nuestro script para que nos lo generara en ese formato.

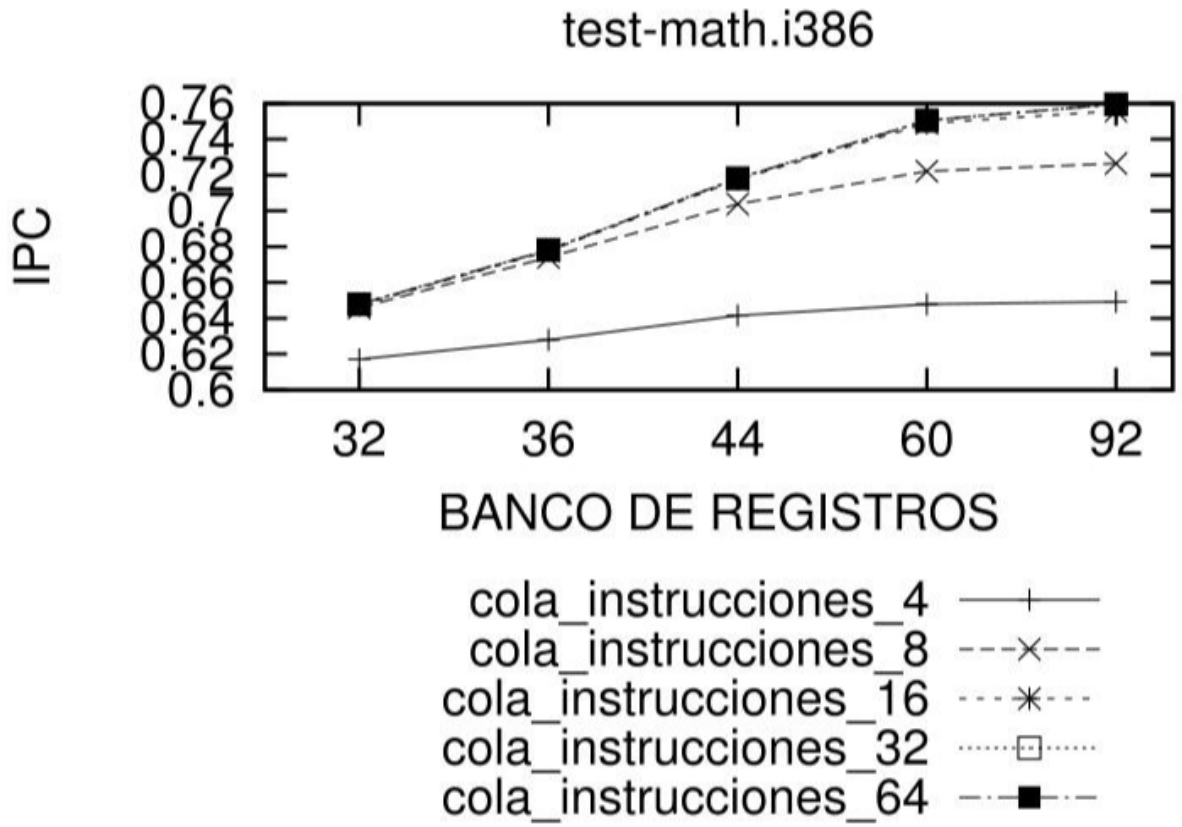
Vamos a analizar las gráficas siguiendo el orden implementado en el script para generarlas:

"test-printf.i386" "test-math.i386" "test-args.i386" "test-sort.i386".

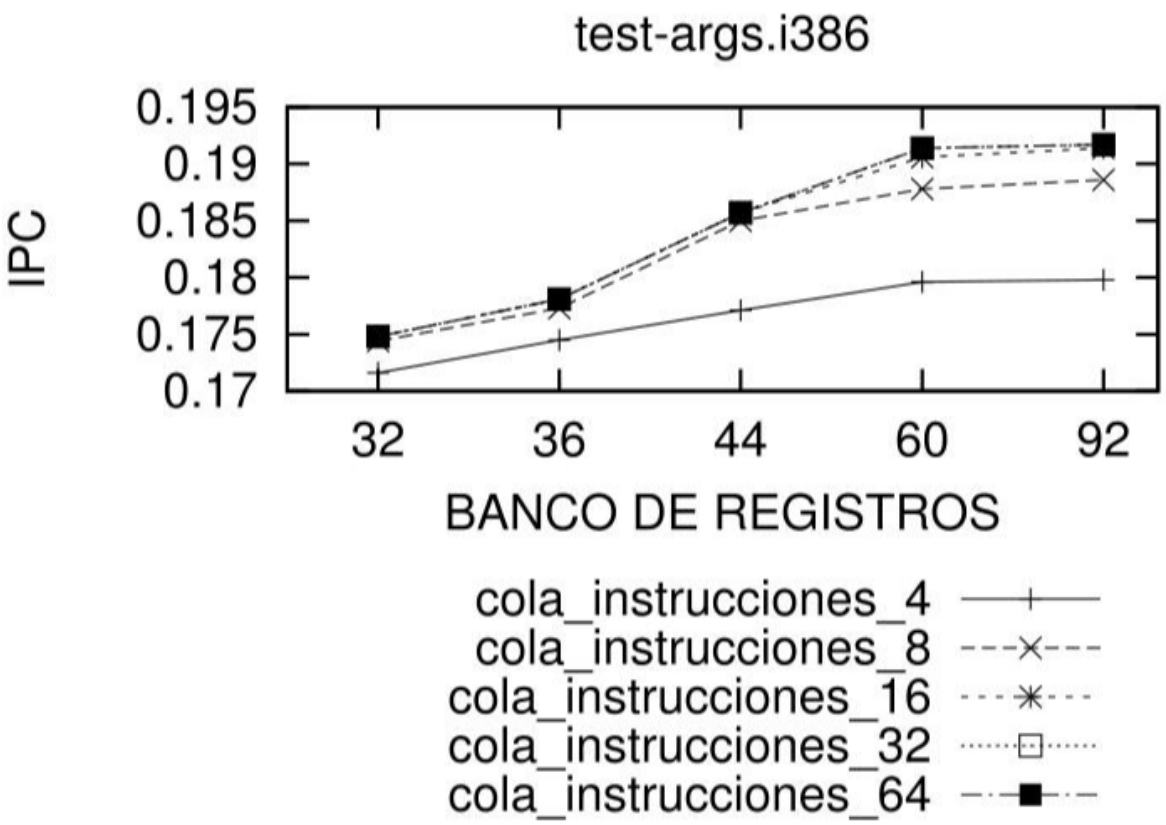
Grafica resultados de test-printf.i386



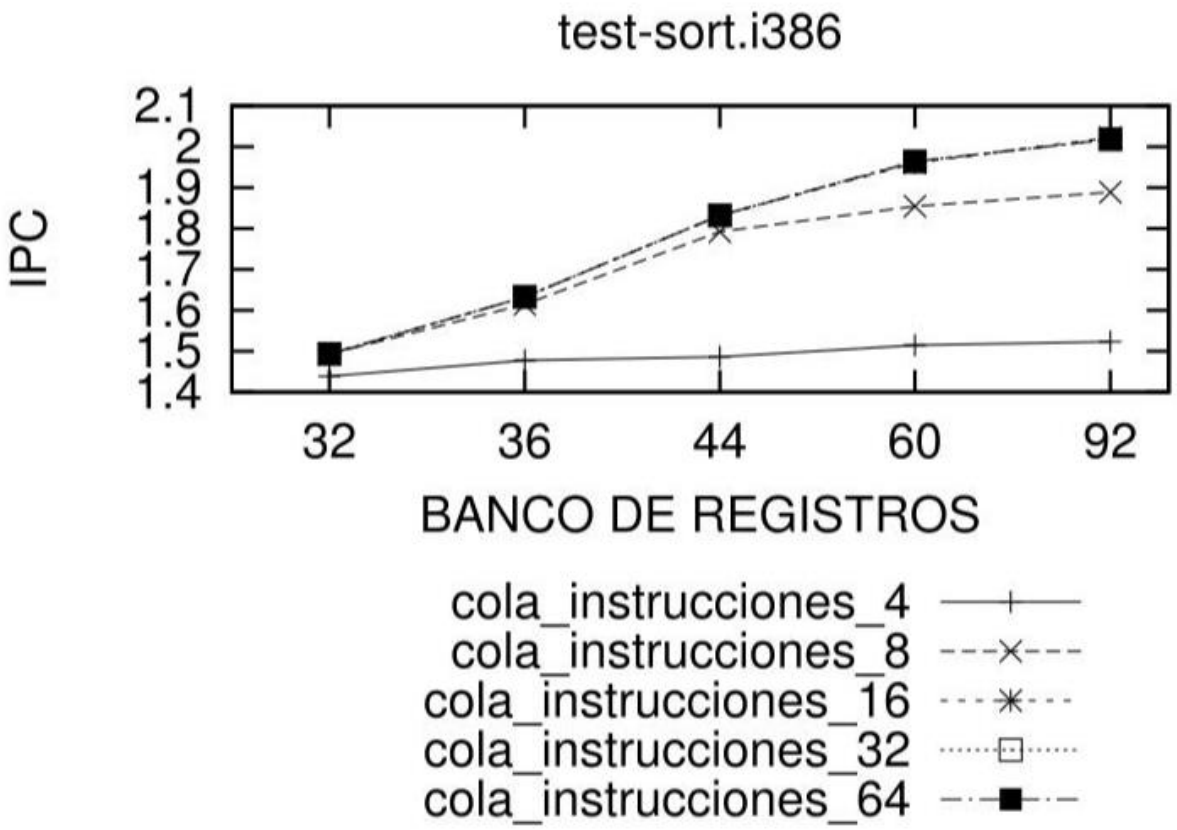
Grafica resultados test-math.i386



Grafica resultados test-args.i386



Grafica resultados test-sort.i386



Comentario de los gráficos

En los gráficos podemos ver la relación entre las distintas configuraciones de las colas de instrucciones, el número de registros habilitados y el IPC (instrucciones por ciclo). Como se puede observar las pendientes de cada línea son diferentes, siendo más acusadas cuando la cola de instrucción llega a un valor de 16 o superior en los test, estabilizándose la pendiente cuando la cola de instrucciones pasa de 32 entradas y se aproxima a 64. Además, es normalmente inferior en la cola de instrucciones con valores cercanos a 4, lo que significa que llega a un límite en el cual por más que aumentemos el tamaño de la cola de instrucciones ya no surge efecto dicho aumento no habiendo apenas mejoría en el IPC.

Si, por ejemplo, entramos a comparar dos graficas de resultados, de diferentes test, podemos ver en "test-sort", que a diferencia de "test-math", es más dependiente del número de registro (valor del banco de registros). Se nota en el cambio de valores de IPC cuando el banco de registro aumenta o disminuye, pasando por ejemplo de 1.5 de IPC con 32 ha 2 con 60, un aumento considerable de 0,5. Sobre todo si lo compráramos con el test en math (llamada a librerías matemáticas) donde pasar de un banco de registros de 32 a 60 apenas hay un aumento de 0,10 en el valor de IPC.

Esto se cumple, cuando pasamos a 8 el valor de colas de instrucciones, con la cola de instrucciones 4 apenas hay aumento, dándonos a entender que dicha cola nos está provocando un cuello de botella.

Podemos concluir que con una cola de instrucciones de tamaño 16-32 en los test, ya no suelen necesitar más entradas en la cola de instrucciones, es decir que con una cola de tamaño superior no se obtiene ninguna mejoría en su ejecución, sobrando espacio. La cantidad de registros ayuda a mejorar el IPC, cuando mayor sea, mejor resultados de IPC obtendremos. Pero se observa que se estanca o disminuye la mejora sobre el IPC cuando sobrepasamos de 60 entradas en el banco de registros.

En resumen, en la mayoría de veces el aumento de entradas en el banco, se obtendrán mejores prestaciones en la ejecución, esta mejoría se debe a que cuantos más registros tengamos libres más instrucciones podrán entrar en ejecución cada ciclo ya que habrá más registros libres para renombrar y no habrá que esperar a liberarlos.

Cuestiones sobre el depurador del pipeline

Cuestión 1: ¿En qué ciclo finaliza el acceso a la cache la primera load?

La primera load empieza en el 284 y termina en el 571 porque como es la primera vez que carga un dato falla en las cachés L1, L2 y L3 porque ninguna tiene el dato aún y llega a memoria principal, si no fuese simulado los ciclos serían mayores ya que tendríamos que acceder a disco duro ya que la memoria principal tampoco tendría el dato (fallaría) y este tardaría mucho más en devolvernos los datos.

Cuestión 2: Respecto a la microinstrucción de salto con código de secuencia 55, ¿acierta el predictor de saltos?

La instrucción de salto numero 55 acierta la predicción ya que las siguientes instrucciones no se cancelan después de la etapa “commit”.

Cuestión 3: Indica el código de secuencia de la primera microinstrucción de salto (el nemotécnico utilizado es branch) en la que la predicción es incorrecta.

La primera instrucción que falla la predicción en el código es:

079 branch zps/- Di ... ... I Wb C

Cuestión 4: ¿Por qué el diagrama no empieza en el ciclo 1, sino que tarda más de 100 ciclos en empezar?

No comienza en el ciclo 1 porque hay que traer las instrucciones de memoria principal a los niveles superiores L1 y L2 de caché que haga falta. Si no estuviéramos utilizando un simulador seguramente comenzaría más tarde la ejecución de la primera instrucción ya que tendríamos que traerlas desde el disco duro que al ser más lento que la memoria principal y caché tardaría más ciclos, el hecho de estar en un entorno simulado nos asegura que las instrucciones están cargadas en memoria principal.

Resultados Ejercicio 2

Resultado	Procesador escalar	Procesador superscalar
Ciclos	1664166	1034048
Instrucciones "Commit"	1500000	1500000
IPC	0.9014	1.451
L1 Instrucciones	0.9557	0.9557
L1 Datos	0.9986	0.9984

Valores de configuración para el procesador escalar:

decode width = 1, dispatch width = 1, issue width = 1, commit width = 1

El speedup obtenido sería:

$$SpeedUp = \frac{IPC(superescalar)}{IPC(escalar)} = \frac{1{,}451}{0{,}9014} = 1{,}6097$$

En vista del resultado obtenido podemos deducir que es un 60,97 % más rápido siendo superescalar.

Se observa en la tabla que para el mismo número de instrucciones el procesador superescalar utiliza menos ciclos, con lo que confirma que habrá un aumento en las prestaciones.