Memoria de la práctica 2

Manel Lurbe Sempere

Manuel José Martínez Baños

Contenido

Ejercicio 0

Cache config L1u L2u

cacheconfig.l1u_l2u

```
[ CacheGeometry l1topo ]
Sets = 16
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 l1 ]
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 = l1
ICache = l1

```

Cache config L1s L2s

cacheconfig.l1s_l2s

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

[ CacheGeometry l2topo ]
Sets = 16
Assoc = 4
BlockSize = 64
Latency = 20

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

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

[ Net net-2 ]
Topology = Bus
LinkWidth = 32

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

[ Cache il1 ]
Geometry = l1topo
LoNet = net-1

[ Cache dl2 ]
Geometry = l2topo
HiNet = net-0
LoNet = net-2

[ Cache il2 ]
Geometry = l2topo
HiNet = net-1
LoNet = net-2

[ MainMemory ]
HiNet = net-2
BlockSize = 64
Latency = 200

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

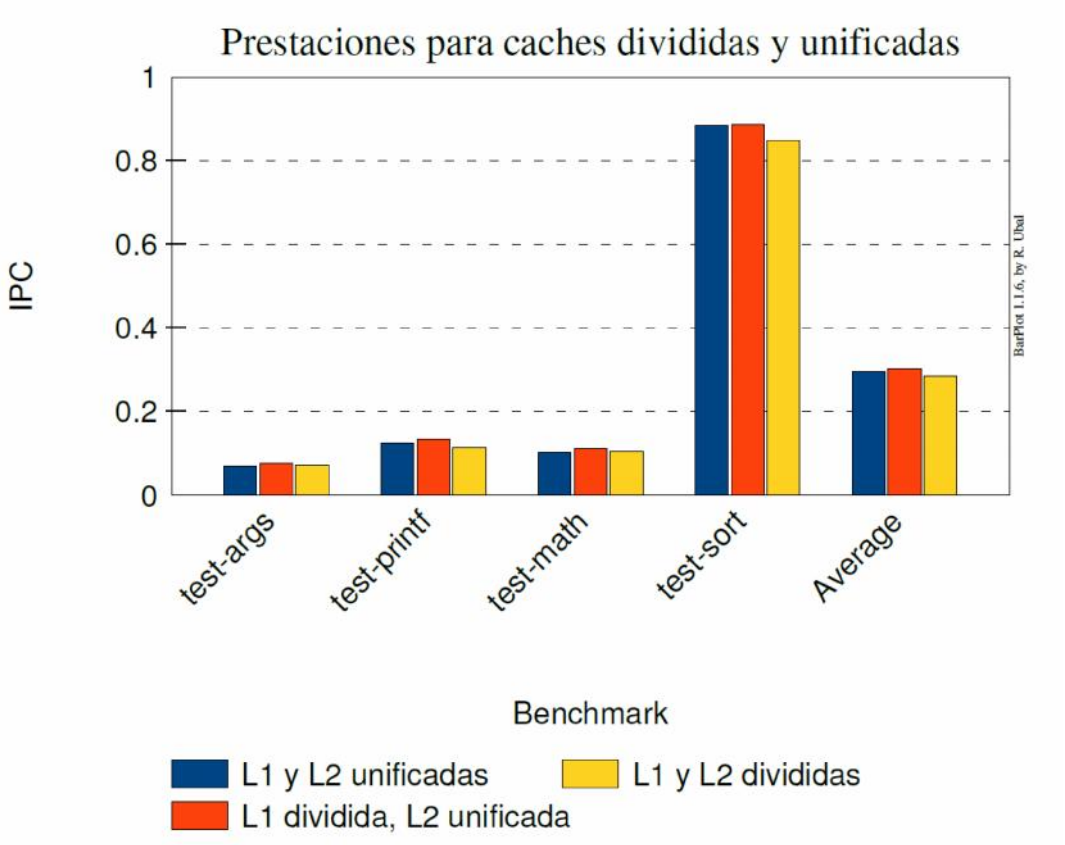
Ejercicio 1

Resultados obtenidos

cache.res

```
0.06841 0.07484 0.07161
0.1238 0.1337 0.1139
0.1029 0.1109 0.105
0.8832 0.8855 0.849
```

Grafico obtenido



Comentario sobre los resultados obtenidos

Viendo el grafico anterior vemos claramente que en todos los test la mejor configuración es L1 dividida y L2 unificada, seguidamente encontramos la configuración L1 y L2 unificadas, y por último tenemos la que resulta ser la que menos prestaciones nos ha demostrado, L1 y L2 divididas.

En el caso de las caches divididas L1 y L2 podemos decir que tienen el peor rendimiento de las 3 configuraciones porque tienen un espacio más limitado de cache que una unificada y si por ejemplo la cache de instrucciones no se llena y la de datos si, la cache de datos no va a poder usar el espacio vacío de la caché de instrucciones, cosa que en una unificada sí que se podría hacer de tal forma que si son necesarios menos bloques de datos y más de código estos se reparten de forma eficiente y transparente.

Las caches totalmente unificadas tampoco son la mejor opción ya que puede darse el caso de que haya muchos reemplazos en L1 y L2 por parte de las instrucciones sobre los datos y viceversa, por eso la mejor opción es buscar el punto intermedio que es L1 dividida y L2 unificada, así podemos ofrecer más almacenamiento a la caché que lo necesite (L2) y aseguremos un espacio reservado por lo menos para instrucciones y para datos (L1), de este modo en L1 no hay tanta “interferencias” de tipos(mezcla de instrucciones y datos).

En resumen, podemos decir, que la memoria en L1 es una zona más critica que en L2, por tanto, interesa, tener una L1 dividida para instrucciones y datos. Pero una L2 unificada, para que esta sea más grande y pueda almacenar diferentes tipos de datos, que pueden ser necesitados por otros recursos. Evitar de este modo lo máximo posibles fallos en la L2.

Así logramos, la mayoría de las veces, que en caso de fallo en L1 , acceda a L2 y aquí haya un “acierto” por tanto se produzca un ciclo de para.

Ejercicio 2

Cache config Dual Core

cachedualcore

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

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

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

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

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

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

[ Cache dl1-1 ]
Geometry = l1topo
LoNet = net-0

[ Cache il1-1 ]
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 = dl1-0
ICache = il1-0

[ Node 1 ]
Core = 1
Thread = 0
DCache = dl1-1
ICache = il1-1
```

Cache config Dual Procesor

cachedualprocesor

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

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

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

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

[ Net net-2 ]
Topology = Bus
LinkWidth = 32

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

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

[ Cache dl1-1 ]
Geometry = l1topo
LoNet = net-1

[ Cache il1-1 ]
Geometry = l1topo
LoNet = net-1

[ Cache l2-0 ]
Geometry = l2topo
HiNet = net-0
LoNet = net-2

[ Cache l2-1 ]
Geometry = l2topo
HiNet = net-1
LoNet = net-2

[ MainMemory ]
HiNet = net-2
BlockSize = 64
Latency = 200

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

[ Node 1 ]
Core = 1
Thread = 0
DCache = dl1-1
ICache = il1-1
```

Script de simulación

ejer2_proceso.simul

```
#!/bin/sh

resfile="resultados_ejer2.res"
tempfile=$(mktemp)
rm -f $resfile
for workload in mezcla1 mezcla2 mezcla3 mezcla4; do
  for config in cachedualcore cachedualprocesor; do
    ./m2s -cores 2 -cacheconfig $config -ctxconfig $workload >/dev/null 2>$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

```

Mezcla 1

mezcla1

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

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

Mezcla 2

mezcla2

```
1  [ Context 0 ]
2  Exe = test-math.i386
3
4  [ Context 1 ]
5  Exe = test-printf.i386
6
```

Mezcla 3

mezcla3

```
1  [ Context 0 ]
2  Exe = test-sort.i386
3
4  [ Context 1 ]
5  Exe = test-args.i386
6  Args = arg1 arg2
```

Mezcla 4

mezcla4

```
1  [ Context 0 ]
2  Exe = test-sort.i386
3
4  [ Context 1 ]
5  Exe = test-math.i386
6
```

Script de obtención de resultados

ejer2_proceso.plot

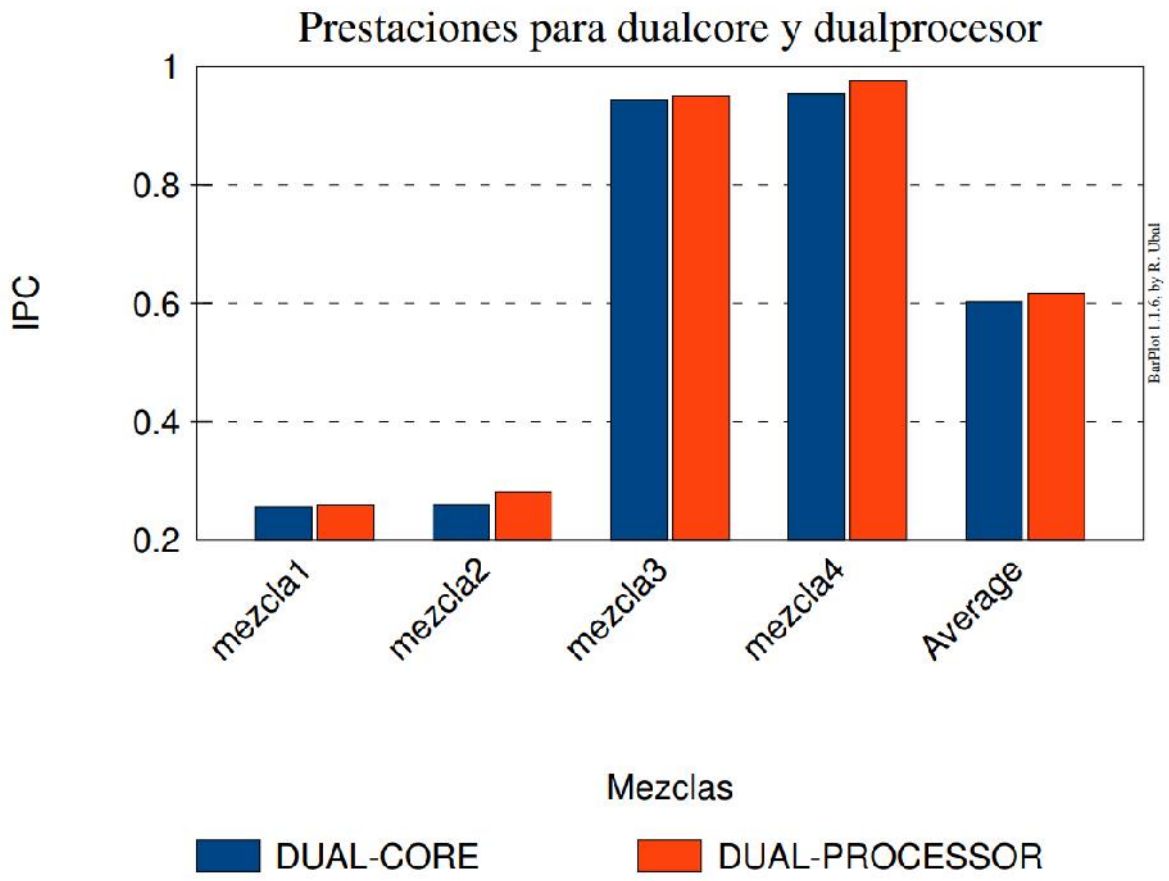
```
#!/bin/sh
plotfile=$(mktemp)
cat << EOF >> $plotfile
XTics = 'mezcla1' 'mezcla2' 'mezcla3' 'mezcla4'
Key = 'DUAL-CORE' 'DUAL-PROCESSOR'
Averages = True
XLabel = 'Mezclas'
YLabel = 'IPC'
Title = 'Prestaciones para dualcore y dualprocesor'
EOF
cat resultados_ejer2.res >> $plotfile
./barplot.py $plotfile resultados_ejer2.eps
rm -f $plotfile
```

Resultados obtenidos

resultados_ejer2.res

```
0.2556 0.2584
0.2588 0.281
0.9423 0.9497
0.9528 0.9757
```

Grafico obtenido



Comentario sobre los resultados obtenidos

En el grafico anterior vemos claramente que una configuración Dual-Processor saca mejores prestaciones que una configuración Dual-Core, ya que en la primera configuración tenemos caches individuales para cada procesador y en la segunda configuración tenemos una jerarquía de caches compartida para ambos procesadores. Esto provoca que haya reemplazos de un procesador sobre el otro derivando en una pérdida de prestaciones.

Como hemos visto en el apartado anterior test-sort tiene más IPC por eso en la mezcla 3 y 4, donde aparece test-sort, hace aumentar considerablemente el IPC en estas dos mezclas.

Ejercicio 3

Script de simulación

```
resfile="resultados_ejer3.res"
tempfile=$(mktemp)
rm -f $resfile

resfile2="resultados_ejer3_INS.res"
rm -f $resfile2

resfile3="resultados_ejer3_REMP.res"
rm -f $resfile3
#test-printf.i386 test-math.i386 test-sort.i386
for workload in test-args.i386  test-printf.i386 test-math.i386 test-sort.i386; do
  for config in ejer3LRU_cacheconfig.l1u_l2u ejer3FIFO_cacheconfig.l1u_l2u ejer3Random_cacheconfig.l1u_l2u; do
    ./m2s -cacheconfig $config -report:cache report_cache $workload   >/dev/null 2>$tempfile

    line=$(cat $tempfile | grep l1.hitratio)
    hitratio=$(echo $line | awk '{print $2}')
    echo -n "$hitratio " >> $resfile
    echo -n "."

    line2=$(cat $tempfile | grep sim.inst)
    num_instruccions=$(echo $line2 | awk '{print $2}')
    echo -n "$num_instruccions " >> $resfile2
    echo -n "."

#cat report_cache.txt  | grep -m1 "Evictions ="
#Filtrem la primera linea del report_cache que correspont al paramete Evictions de l1
    remplazos=$(cat report_cache | grep -m1 "Evictions ="| awk '{print $3}')
    echo $remplazos
    echo -n "$remplazos " >> $resfile3
    echo -n "."


    rm -f rc.$config
  done
  echo >> $resfile
  echo >> $resfile2
  echo >> $resfile3

done
rm -f $tempfile

echo
```

Script de obtención de resultados

ejer3_remplazo.plot

```
#!/bin/sh
plotfile=$(mktemp)
cat << EOF >> $plotfile
XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
Key = 'LRU' 'FIFO' 'Random'
Averages = True
XLabel = 'Benchmark'
YLabel = 'HIT RATE'
Title = 'Prestaciones para diferentes tipos de reemplazo'
EOF
cat resultados_ejer3.res >> $plotfile
./barplot.py $plotfile resultados_ejer3.eps
rm -f $plotfile

plotfile=$(mktemp)
cat << EOF >> $plotfile
XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
Key = 'LRU' 'FIFO' 'Random'
Averages = True
XLabel = 'Benchmark'
YLabel = 'Num.Ins'
Title = 'Num. instrucciones realizadas por los programas'
EOF
cat resultados_ejer3_INS.res >> $plotfile
./barplot.py $plotfile resultados_ejer3_INS.eps
rm -f $plotfile

plotfile=$(mktemp)
cat << EOF >> $plotfile
XTics = 'test-args' 'test-printf' 'test-math' 'test-sort'
Key = 'LRU' 'FIFO' 'Random'
Averages = True
XLabel = 'Benchmark'
YLabel = 'Num.Remplazos'
Title = 'Para Diferentes Politicas de Remplazo, Num.Remplazos realizados'
EOF
cat resultados_ejer3_REMP.res >> $plotfile
./barplot.py $plotfile resultados_ejer3_REMP.eps
rm -f $plotfile
```

Cache config FIFO

ejer3FIFO_cacheconfig.l1u_l2u

```
[ CacheGeometry l1topo ]
Sets = 16
Assoc = 2
BlockSize = 64
Latency = 2
Policy = FIFO

[ 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 l1 ]
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 = l1
ICache = l1
```

Cache config LRU

ejer3LRU_cacheconfig.l1u_l2u

```
[ CacheGeometry l1topo ]
Sets = 16
Assoc = 2
BlockSize = 64
Latency = 2
Policy = LRU

[ 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 l1 ]
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 = l1
ICache = l1
```

Cache config Random

ejer3Random_cacheconfig.l1u_l2u

```
[ CacheGeometry l1topo ]
Sets = 16
Assoc = 2
BlockSize = 64
Latency = 2
Policy = Random

[ 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 l1 ]
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 = l1
ICache = l1
```

Resultados obtenidos

resultados_ejer3.res

```
0.2935 0.2707 0.2689
0.3053 0.2712 0.2866
0.3829 0.3417 0.3428
0.8293 0.8002 0.8067
```

1 Hit Rate

resultados_ejer3_REMP.res

```
1732 1803 1890
112769 119147 123339
9363 9988 10417
40834 42173 43091
```

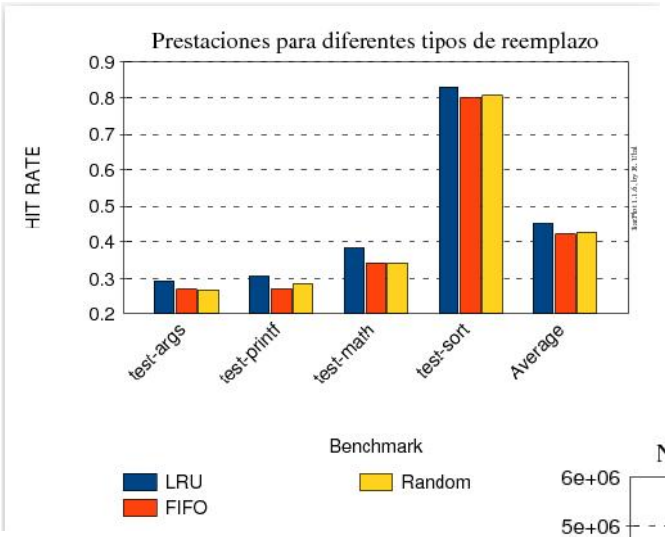
2 Reemplazo

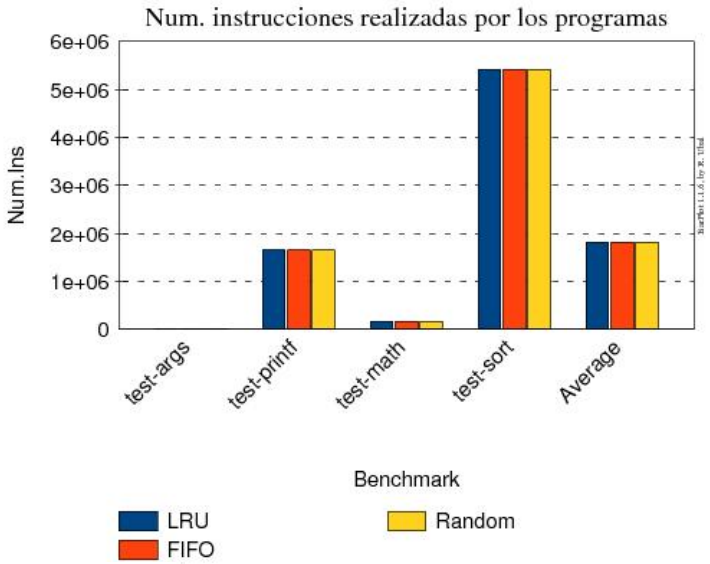
resultados_ejer3_INS.res

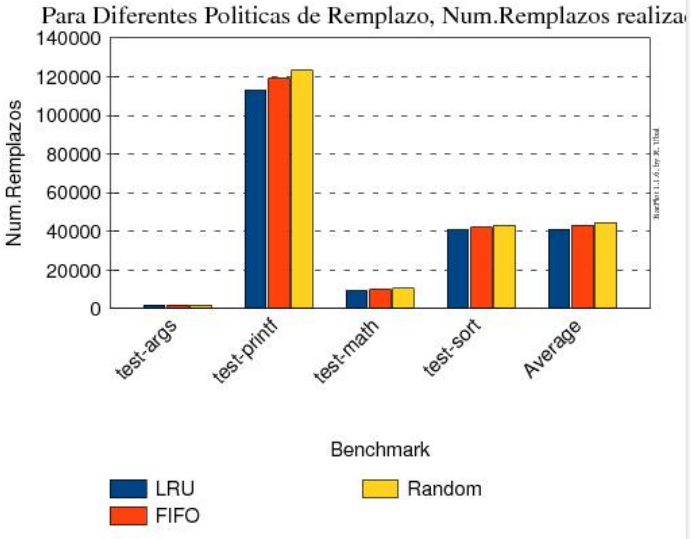
```
20636 20636 20636
1660412 1660412 1660412
171496 171496 171496
5426759 5426759 5426759
```

3 Instrucciones

Gráficos obtenidos







Comentario sobre los resultados obtenidos

En el ejercicio 3, si visualizamos la gráfica donde tenemos en cuenta el hit-rate, vemos que, en todo el uso de la política, LRU sale ganadora y muy ajustadas entre ellas la random y FIFO después.

LRU es la política con mejores prestaciones porque reemplaza los bloques que más antiguos que han sido usados. Este mecanismo se basa en pura heurística. Depende de un contador para poder conocer cuando han sido usados. Para después descartar aquellos con el contador menor.

LRU por tanto, tiene mayor ventaja que el resto, porque esos boques remplazados, tiene una probabilidad mayor de que no se vuelvan a necesitar.

Pero LRU tiene el inconveniente de que es más difícil de implementar, a parte, de que en L2 su uso es bastante mas ineficiente.

Si nos vamos más al detalle. Tenemos dos gráficas más, la que nos muestra el Número de instrucciones por cada programa y otra el número de reemplazamos.

Como vemos la relación está clara, a mayor cantidad de número de instrucciones, su número de reemplazos es mayor y por tanto los aciertos crecen, pero no siempre es así.

Hay que tener en cuenta que si un programa tiene muy pocas instrucciones, como es el caso de test-args, sus reemplazos serán muy escasos y su hit-rate también. No le da tiempo a acertar, porque de primeras la mayoría de veces falla al principio al no tener los valores por tanto al tener pocas instrucciones no necesitará cargar más “bloques” que contienen un conjunto de instrucciones.

En el caso de test-print el número de reemplazos es muy alto por lo que su tasa de acierto es muy baja, debido a la cantidad de veces que falla y debe reemplazar. Además, vemos que test-sort tiene un numero de instrucciones mayor que por ejemplo test-print y en cambio, el número de reemplazos es menor. Podemos concluir, que no siempre el número de instrucciones determina un número de reemplazos mayor y por tanto peor hit-rate.