Arquitectura de computadores y diseño de circuitos digitales
Proyecto Unidad 2
Gabriel Sanhueza

Especificaciones:

Este proyecto fue compilado en macOS Sierra en la versión del jdk 1.8.0_112 y se creó un script con diversas configuraciones que también se incluyen en el directorio del proyecto, así como la salida de este.

La ejecución se debe hacer desde consola desde el directorio local escribiendo java –jar <nombre del archivo> <opciones>. Ej:

```
java -jar CacheSimulator.jar -bs 4 -cs 32 -fa -wt -wna -split traces/spice.trace
```

Para ejecutar el script desde consola (Al menos en MacOS) se usa la siguiente línea:

```
./run.sh
```

Opciones de entrada:

-bs N -- tamaño de bloque basado en el número de palabras (words) que contiene
-cs N -- tamaño del cache, N es el número de líneas que contiene.
-wt -- si el cache es Write-Through (por defecto, debería ser Write-Back)
-fa -- el cache es fully associative (LRU política para reemplazar bloques)
-sa N -- el cache es set-associative (LRU política para reemplazar bloques), N es el tamaño del conjunto (por defecto el cache debería ser mapeo directo)
-wna -- el cache usa Write-No-Allocate (por defecto, debería ser Write-Allocate)
-split -- el cache está separado en dos secciones, una para instrucciones y la otra para los datos

Preguntas:

¿Cuál es la mejor arquitectura de cache para spice.trace?

La mejor arquitectura para spice.trace es con fully asociative con split. Los resultados de la ejecución con esta arquitectura son los siguientes:

BlockSize = 4

cacheSize = 32

wt = false

fa = true

sa = 16 // 32/16 = 2 que es el numero del conjunto en que se separan. 1 conjunto de datos y otro de instrucciones.

wna = false

split = true

fileDir = traces/spice.trace

Numero de referencias a instrucciones: 782764

Numero de referencias a datos: 217237

Numero de faltas a instrucciones: 576445

Numero de faltas a datos: 82793

Numero de palabras copiadas desde RAM: 2636952

Numero de palabras copiadas a RAM: 139448

miss instruction rate: 73,642% | hit rate : 26,358%

miss data rate: 38,112% | hit rate : 61,888%

¿Cuál es la mejor arquitectura de cache para cc.trace?

La mejor arquitectura para cc.trace es fully associative con Split. Los resultados de la ejecución con esta arquitectura son los siguientes:

BlockSize = 4

cacheSize = 32

wt = false

fa = true

sa = 16

wna = false

split = true

fileDir = traces/cc.trace

Numero de referencias a instrucciones: 757341

Numero de referencias a datos: 242661

Numero de faltas a instrucciones: 666884

Numero de faltas a datos: 92579

Numero de palabras copiadas desde RAM: 3037852

Numero de palabras copiadas a RAM: 167936

miss instruction rate: 88,056% | hit rate : 11,944%

miss data rate: 38,152% | hit rate : 61,848%

¿Cuál es la mejor arquitectura de cache para tex.trace?

La mejor arquitectura para tex.trace es con fully asociative con split. Los resultados de la ejecución con esta arquitectura son los siguientes:

BlockSize = 4

cacheSize = 32

wt = false

fa = true

sa = 16

wna = false

split = true

fileDir = traces/tex.trace

Numero de referencias a instrucciones: 597309

Numero de referencias a datos: 235168

Numero de faltas a instrucciones: 485394

Numero de faltas a datos: 7489

Numero de palabras copiadas desde RAM: 1971532

Numero de palabras copiadas a RAM: 14948

miss instruction rate: 81,263% | hit rate : 18,737%

miss data rate: 3,185% | hit rate : 96,815%

¿Por qué hay diferencias entre los programas? ¿Si tuviera que diseñar un cache para un computador que ejecutará solo estos programas, que arquitectura usaría?

Hay diferencias entre los programas debido a la cantidad y tipo de instrucciones que poseen. Si tuviera que diseñar un computador que solo ejecutara estos programas usaría fully asociative con Split. Otras arquitecturas que también resaltaron sobre las demás pero no tanto como las aquí mencionadas son:

Para spice.trace write throught, full associative, write no allocate, Split.

Para cc.trace set associative, Split.

Para tex.trace set associative, Split.

Cabe recalcar que las arquitecturas escogidas como mejores solo sobresalieron en el acceso a datos.