Decodificación MPEG2

Alberto Suarez, *Ingeniero, Hewlett Packard,* Yu Shan Hsieh, *Ingeniero, Hewlett Packard*

***Abstract*—Artículo científico sobre benchmarking en la decodificación de mpeg2.**

***Index Terms*—Decodificación, mpeg2, Benchmark**

I. Introducción

EL propósito de este paper es analizar varios aspectos de rendimiento que tiene un programa de decodificación de mpeg2 tales como consumo de potencia, desempeño de la CPU, tiempo de ejecución, etc. Para poder hablar sobre esos aspectos, es importante primero entender qué es la decodificación de mpeg2 y cómo se realiza. Para esto se analizará los códigos fuentes que Posteriormente se detallará las pruebas que se van a realizar y el testbench sobre la cual se realizará las pruebas. Luego que hará el análisis de resultados y finalmente las conclusiones que se pueden derivar de estos experimentos.

Setiembre 27, 2013

II. Metodología de optimización

A. Descripción del benchmark

B. Herramientas

1) Simplescalar: Simplescalar[2] es un simulador de arquitectura de computadoras Open Source escrito en C. Este simulador contiene una serie de herramientas que modela un sistema computador virtual con CPU, cache y jerarquías de memoria. Utilizando Simplescalar, el usuario puede modelar lo que ocurre cuando un programa corre sobre una variedad de arquitecturas desde procesadores sencillos hasta procesadores con scheduling dinámico, cache no-bloqueante y con predicción de saltos. Simplescalar soporta set de instrucciones tipo PISA y se compone de tres herramientas principales:

- **simplesim** Simulador Simplescalar.
- **simpletools** Compilador GNU GCC de PISA para Simplescalar.
- **simpleutils** Herramientas para el compilador que incluyen el ensamblador de PISA, linker, etc.

2) Wattch: Wattch[3] es un simulador que estima la potencia consumida de un CPU. Tiene modelos de potencia integrados en Simplescalar y utiliza una version modificada de sim-outorder de Simplescalar para recolectar resultados.

3) GCC: GNU Compiler Collection o GNU C Compiler es un compilador de C para sistemas operativos GNU.

4) Mpeg2 decoder: El decodificador viene en un suite que trabaja tanto la parte de codificación (encode) como la de decodificación. En nuestro caso, solo vamos a trabajar la parte de decode. Dentro del directorio de src/mpeg2dec se encuentra el codigo en C del decoder junto con el Makefile. El archivo principal es mpeg2dec.c y está compuesto por las siguientes funciones principales:

```
static void Initialize_Decoder _ANSI_ARGS_((void));
static int Decode_Bitstream _ANSI_ARGS_((void));
```

La función de *Decode_Bitstream* llama a la función video_sequence() que se encarga de ir por cada imagen y decodificarlo.

C. Configuración

En esta sección se va a hace un análisis preliminar de la configuración de la arquitectura (sistema de referencia a utilizar).

El código de decodificación MPEG2 vemos que contiene en su mayoría contiene variables de tipo integer, punteros y referencias a memoria para procesar los buffers de datos a decodificar. Existen apenas tres variables de tipo double, y se ejecutan unas cuantas operaciones con las mismas. El codigo se basa en llamadas a funciones y luego a otras cuantas mas logrando cerca de unos 5 niveles de anidación de las llamadas, lo cual representan saltos multiples que pueden afectar el IPC. Dedido a la muy poca utilización de doubles se espera una poca utilización de ALUs de punto flotante.

D. Función Costo

Definición de la función costo. Debido al uso tan extenso que se hace de la codificación/decodificación MPEG2, se opta por definir nuestra función de costo en base al rendimiento del chip asi como la energía consumida por el mismo, ya que nos parece son los parámetros que se deberían de maximisar para esta aplicación. El rendimiento es necesario para permitir una decodificación rápida y eficiente, la energía es importante por motivos del costo de consumo de la misma para realizar las decodificaciones.

```
Funcion de costo:     F(x) = P(x) * D(x)
P(x) = Potencia de x (Potencia total consumida por el chip)
D(x) = Rendimiento IPC del codigo.
```

E. EspacioDiseno

Resultados preliminares y Espacio de Diseño. De la primer corrida obtenemos los siguientes datos:

```
Total Power Consumption: 56.2826
sim_IPC         1.8649 # instructions per cycle
```

```
sim_CPI          0.5362 # cycles per instruction
il1.misses       277707 # total number of misses
dl1.misses       96318 # total number of misses
ul2.misses       20691 # total number of misses
itlb.misses      32 # total number of misses
dtlb.misses      114 # total number of misses
il1.hits         178888369 # total number of hits
dl1.hits         32954311 # total number of hits
ul2.hits         384866 # total number of hits
itlb.hit         179166044 # total number of hits
dtlb.hits        33333723 # total number of hits
il1.miss_rate    0.0015 # miss rate (i.e., misses/ref)
dl1.miss_rate    0.0029 # miss rate (i.e., misses/ref)
ul2.miss_rate    0.0510 # miss rate (i.e., misses/ref)
itlb.miss_rate   0.0000 # miss rate (i.e., misses/ref)
dtlb.miss_rate   0.0000 # miss rate (i.e., misses/ref)
```

Se aprecia que la cantidad de cache misses es muy pequeña en comparación con los hits, todos lo miss rates son menores o iguales al 5 por ciento, dada esta característica no nos vamos a enfocar en tratart de mejorar el hit rate de las caches, sino en intentar otras configuraciones para intertar bajar la potencia total y el CPI. Dada la cantidad de saltos se opta por variar las politicas del branch predictor, la configuracion por default ya cuenta con suficientes ALUs para integers por lo que no vamos a intentar variar este componente. Como se trabaja mucho con datos de memoria y punteros tambien se opta por variar los componenetes para actualizar los registros así como el tamaño del queue para los Loads y Stores.

Nuestro espacio de diseño va a ser entonces: branch preditor, el register update unit (RUU) size, load/store queue (LSQ) size, y la politica de reemplazo de bloques en las memorias cache(FIFO, LFU).

F. Simulaciones a realizar

Cálculo de la cantidad total de simulaciones a realizar. Las simulaciones preliminares nos dan un estimado de 279 segundos para correr cada simulación, lo cual corresponde aproximadamente a 4 minutos. En un lapso de 45 minutos teoricamente se pueden correr alrededor de 11 simulaciones. Dado que son pocas las iteraciones que se pueden correr en un lapso de 45 minutos, nos tenemos que limitar a correr 12 simulaciones, lo que implica que solo se van a probar 12 combinaciones posibles. A continuación se muestra las pruebas y su configuracion:

1) **Test 1**. bpredm=comb, lsq=8, ruu=16, cache=lfu
2) **Test 2**. bpredm=comb, lsq=8, ruu=32, cache=lfu
3) **Test 3**. bpredm=comb, lsq=8, ruu=32, cache=fifo
4) **Test 4**. bpredm=comb, lsq=16, ruu=16, cache=lfu
5) **Test 5**. bpredm=comb, lsq=16, ruu=32, cache=lfu
6) **Test 6**. bpredm=comb, lsq=16, ruu=32, cache=fifo
7) **Test 7**. bpredm=2lev, lsq=8, ruu=16, cache=lfu
8) **Test 8**. bpredm=2lev, lsq=8, ruu=32, cache=lfu
9) **Test 9**. bpredm=2lev, lsq=8, ruu=32, cache=fifo
10) **Test 10**. bpredm=2lev, lsq=16, ruu=16, cache=lfu
11) **Test 11**. bpredm=2lev, lsq=16, ruu=32, cache=lfu
12) **Test 12**. bpredm=2lev, lsq=16, ruu=32, cache=fifo

Las simulaciones se lanzan con el script *./run_benchmark.sh* que llama a Wattch junto con el binario cross-compilado de Simplescalar.

G. Resultados

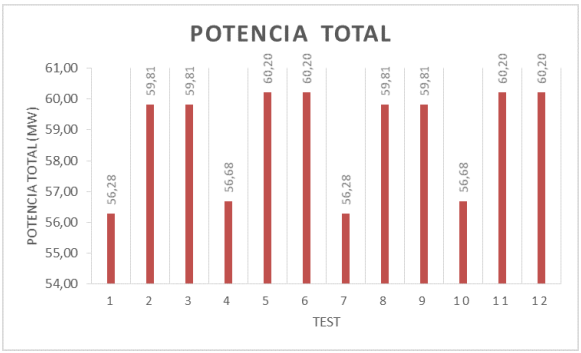

Fig. 1. Resultados de Potencia total

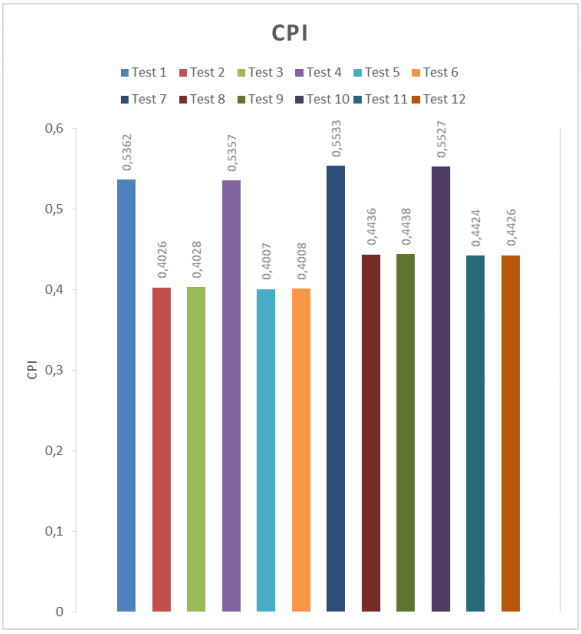

Fig. 2. Resultados de IPC

H. Análisis de Resultados

A partir de los resultados se observa que al reducir el tamaño de la RUU se obtiene mejor potencia total (más baja). Esto es de imaginar ya que el algoritmo de decodificación de mpeg2 hace bastante uso de de buffers y de operaciones en registros. Al mismo tiempo, al reducir la RUU de 32 a 16, aumenta la cantidad de ciclos necesarios para ejecutar una instrucción (CPI). Noten que este aumento en la CPI tiene mayor peso sobre la función costo que la potencia reducida. En otras palabras, la mejora que se obtuvo en potencia no fue lo suficiente para compensar el incremento en la CPI. Al incrementar la CPI se incrementa también el tiempo de ejecución del programa lo cual no es deseable.

El segundo factor que mas influenció fue el modo del Branch Predictor, y las variables utilizadas son *2Lev* y *comb* con el último siendo el método por default del compilador.

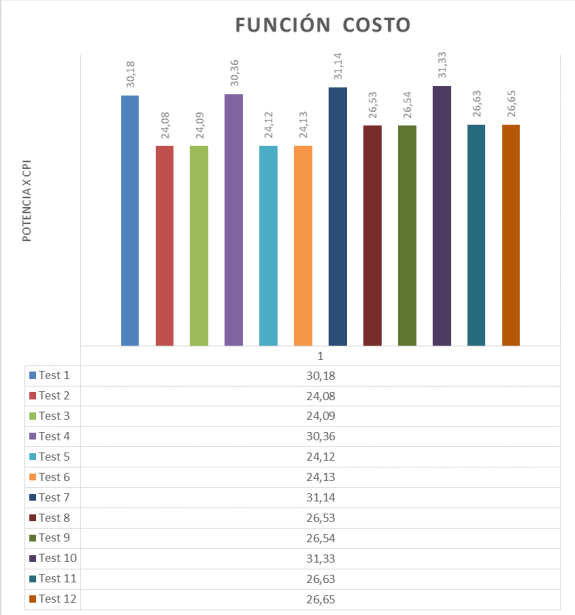

Fig. 3. Gráfica de Función Costo

Note con el método *comb* se produce mejores tasas de CPI de forma consistente en comparación con *2Lev*.

Los cambios en la estrategia de reemplazo de los bloques de la cache y el tamao del queue de LoadStore no aportan variaciones significativas de la función costo aunque se observa una ligera mejora en la CPI al utilizar LFU en comparación con utilizar FIFO. Esto se debe a que el código hace un uso intensivo de la memoria y las acciones realizadas con estos datos permiten calendarizar otras operaciones entre lecturas de memoria.

III. Conclusiones

Conclusiones.

Acknowledgment

References

[1] MPEG Software Simulation Group, MPEG-2 Encoder/Decoder, Version 1.2, July 19, 1996
[2] Simplescalar LLC, http://www.simplescalar.com/, 2395 Timbercrest Court Ann Arbor, MI 48105
[3] Wattch, http://www.eecs.harvard.edu/ dbrooks/wattch-form.html, Version 1.02d.

Alberto Suarez Ingeniero Electrónico, Hewlett Packard

Yu Shan Hsieh Bachillerato en Ingeniería Eléctrica, UCR Ingeniero Electrónico, Hewlett Packard