Taller de SIMD

Organización del Computador II

En este taller vamos a trabajar con algunas instrucciones SSE (Streaming SIMD Extensions) de Intel, que nos permitirán hacer operaciones de manera más eficiente. Vamos a aplicar un filtro a un archivo de audio en formato WAV, cuya implementación en C tendrán que estudiar. El filtro que usaremos es de respuesta impulsiva finita (FIR) dada su versatilidad y facilidad de implementación. En la siguiente parte del taller, tendrán que implementar el filtro con instrucciones SIMD.

Al corregir cada **checkpoint** todos los miembros del grupo deben estar presentes, salvo aquellas instancias en que puedan justificar su ausencia previamente. Si al finalizar la práctica de la materia algún miembro cuenta con mas de un 20 % de ausencia no justificada en la totalidad de los checkpoints se considerará la cursada como desaprobada.

1. Introducción

Un filtro FIR de orden N, se puede definir mediante la ecuación:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \tag{1}$$

$$= \sum_{i=0}^{N} b_i \cdot x[n-i], \tag{2}$$

donde:

- $x[n]$ es la señal de entrada
- $y[n]$ es la señal de salida
- N es el orden del filtro
- b_i es el iésimo coeficiente del filtro

La forma directa del filtro FIR puede verse en la figura **??**

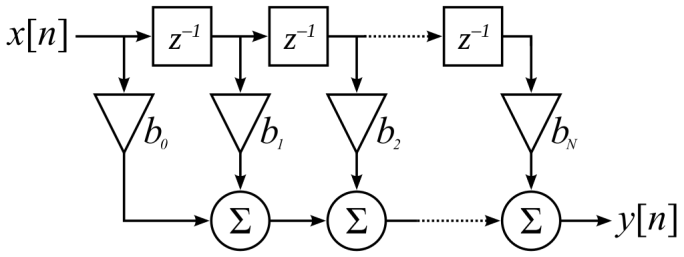

Figura 1: Forma Directa del filtro FIR

2. Análisis de código

Para compilar el programa, dentro del directorio de código, simplemente hacer

```
$ make
```

El programa recibe como parámetro un archivo wav, y se puede ejecutar de la siguiente manera:

```
$ ./oppenheim <archivo wav>
```

Vean la ayuda del programa, para entender los parámetros que le pueden pasar, haciendo:

```
$ ./oppenheim --help
```

Disponen de 2 archivos en la carpeta **wavs**: uno más liviano, `muestra.wav` y uno más pesado, `caballo.wav`. Para las pruebas rápidas pueden usar el archivo más liviano.

Además, disponen de una carpeta **filters** donde se almacenan los distintos filtros que se pueden aplicar. [1]

Datos técnicos: los archivos WAVs son formato PCM signed de 16 bits, stereo, con una frecuencia de muestreo de 44100 Hz.

Al iniciar, se inicializan dos filtros (uno para el canal izquierdo y uno para el derecho) y se inicializan dos buffers (`rbuffer` y `wbuffer`) que albergarán una cantidad de muestras fijas, para cada canal.

En el loop, se va leyendo de a `BLOCK_SIZE` muestras hasta terminar el archivo, que luego se filtran y escriben en el archivo wav destino. Una barra muestra el progreso de la operación. Si bien alentamos a que profundicen en el código para un mejor entendimiento, vamos a poner el foco en la implementación del filtro en cuestión, archivos **filtros.c** y **filtros.h**

Respondan las siguientes preguntas, referentes a la función `fir_filter`:

a) ¿Para qué se realiza la siguiente operación? Explique qué hace esa línea y cuál es el objetivo de dicha operación

```
memcpy(&filtro->buffer[filtro->length - 1], in, length * sizeof(int16_t));
```

Hint: puede hacer `man memcpy` en la terminal para más información

b) ¿Qué función tiene el loop externo? ¿Y el interno?

c) ¿En qué línea se está haciendo la operación MAC (multiply-accumulate)? ¿Por qué `acc` es de 32 bits?

d) ¿Para qué se realiza la siguiente operación?

```
memmove(&filtro->buffer[0], &filtro->buffer[length], (filtro->length - 1) * sizeof(int16_t));
```

e) Realice un diagrama donde se resuma la operación de la implementación en C del filtro.

Checkpoint 1

[1] ¡Pueden agregar más filtros si lo desean! Respeten el formato, y modifiquen apropiadamente el archivo `filtros.h` y el `main.c` para que lo puedan pasar por línea de comandos. Para diseñar filtros pueden usar: `http://t-filter.engineerjs.com/` que anda muy bien y exporta los coeficientes de forma adecuada. Hay muchas herramientas para diseño de filtros. una en python es pyfda

3. Implementación del filtro con instrucciones SIMD

En este checkpoint tienen que realizar la implementación del filtro con instrucciones SIMD. Para ello, tienen que implementar la función:

```
size_t fir_filter_simd(FIR_t*filtro, int16_t *in, unsigned length, int16_t *out);
```

en el archivo `simd.asm`. Como referencia, les dejamos una nota de aplicación de Intel, donde se detalla la estructura general que puede tomar el filtro. Tengan en consideración que esa implementación difiere de la que necesitamos, porque no tiene en cuenta las muestras anteriores en un procesamiento de a bloques (`chunks`)

Checkpoint 2