Taller 2: Extensión SIMD: SSEx

Malcolm Davis, Miembro Estudiantil, IEEE
mdavis.cr@ieee.org

Resumen— En este documento se encuentran las respuestas a las preguntas propuestas en el taller 2 del curso Arquitectura de Computadores II relacionadas con el Set de Extensiones SIMD(SSE).

Index Terms— TEC, Arquitectura De Computadores II, SSE, SIMD.

1. Pregunta 1: Definición SSE

¿Qué es el set SSE, cuál es su utilidad y qué aplicaciones tiene?
SSE o Streaming SIMD Extensions es un set de instrucciones que permiten procesar más de un dato en una instrucción(procesamiento vectorial de datos). Con respecto a las aplicaciones, como mencionan en [1] alguna de las aplicaciones comunes tienen que ver con el procesamiento de datos 3D como en vídeo juegos, programas de edición de vídeo y modelado 3D.

2. Pregunta 2: Compilando SSE en C

¿Cómo realiza la compilación de un código c (.c) que utilice el set SSEx de Intel?
Para compilar el código se utilizan las banderas de compilación *-mssex* donde *x* representa la versión de SSE. Además, se necesita incluir los headers necesarios para la versión. Para los ejercicios de este taller se modificó el Makefile para simplificar el proceso de compilación.

```
# Declaration of variables
CC = gcc
CC_FLAGS = -lm -fopenmp -w -msse4

# Main Target
all: hello max matMult

hello: hello.o
	$(CC) hello.c  $(CC_FLAGS) -o hello

max: max.o
	$(CC) max.c $(CC_FLAGS) -o max

matMult: matMul.o
	$(CC) matMul.c $(CC_FLAGS) -o matMult


# To remove generated files
clean:
	rm -f hello max matMult
```

3. Pregunta 3: Variables y Memoria en SSE

¿Qué importancia tienen la definición de variables y el alineamiento de memoria al trabajar con un set SIMD vectorial, como SSE?
Las operaciones vectoriales realizan sobre 1 o más vectores alguna operación, esto significa que se tiene que cargar n elementos continuos para aplicar una operación, si los datos que se van a utilizar no son uniformes, se corre el riesgo de desperdiciar procesamiento o de un error al intentar acceder a una memoria prohibida [2].

4. Ejercicio 1: Hola Mundo SSE

Con base en el código hello_simd.c

```
// SSE headers
#include <emmintrin.h> //v3
#include <smmintrin.h> //v4

#include <stdio.h>


int main(){
//****Nota: este código requiere SSE4****//
printf("Hola Mundo desde SSE \n");

// Little endian, stored in 'reverse'
__m128i vector1 = _mm_set_epi32(4, 3, 2, 1);
__m128i vector2 = _mm_set_epi32(8, 7, 6, 5);

// Addition
// result = vector1 + vector 2
__m128i result = _mm_add_epi32(vector1,
vector2);

//Vector Printing
int data = 0;
int i;
printf("Result ********************** \n");
data = _mm_extract_epi32(result,0);
printf("%d \t", data);

data = _mm_extract_epi32(result,1);
printf("%d \t", data);

data = _mm_extract_epi32(result,2);
printf("%d \t", data);

data = _mm_extract_epi32(result,3);
```

```
printf("%d \t", data);
printf("\n");
return 1;
}
```

4.1. Realice un análisis del código, ¿qué operación realiza el mismo?, ¿qué instrucciones SIMD se están utilizando y con qué fin? ¿Qué versión de SSE utiliza el código?

El código imprime un hola mundo y hace una suma vectorial. Utiliza 3 instrucciones vectoriales, _mm_set_epi32, _mm_add_epi32 y _mm_extract_epi32. La primera se utiliza para inicializar un vector con enteros de 32 bits, la segunda para sumar dos vectores de 4 enteros de 32 bits y la tercera para extraer un elemento del vector. Hay diferentes versiones porque por ejemplo la instrucción set está en la versión 2 pero el extract está en la versión 4.1

4.1.1. Realice una compilación del código y ejecútelo. Escriba la salida de consola de la aplicación.

En la figura 1 se puede observar la salida de la ejecución del programa.

```
./hello
Hola Mundo desde SSE
Result **********************
6       8       10      12
```

Figura 1. Salida de hello_simd

5. Ejercicio 1: Máximo Vectorial

Realice un programa que calcule un vector con el valor máximo de cada columna de una matriz de 2x8 (fxc) en la que los valores son números enteros de 16 bits (shorts). El programa debe recibir los datos del usuario e imprimir el vector resultante.

Para realizar esto se utilizó la función _mm_max_epi16 que retorna un vector __m128i con los valores máximos entre 2 vectores comparados. El código completo puede ser encontrado en los archivos que se adjuntan con este documento).

Como se observa, a diferencia del hello_simd.c no se utiliza el método _mm_set_epix para inicializar el vector sino los vectores nativos de c. Esto para la generalidad. Además, se implementa un programa que acepta n cantidad de vectores o ninguno(utilizando valores aleatorios para los vectores). Por último, se utilizó OpenMP para paralelizar los ciclos que no tienen dependencias. Un ejemplo de la salida del programa con valores aleatorios se puede observar en la figura 2.

```
./max
Vector 1: 21598 23547   30805   14310   13307   4960    8993    14214
Vector 2: 11985 20046   7004    8122    31331   5720    15690   22054
Result ***********************
Result: 21598   23547   30805   14310   :
```

Figura 2. Salida de max

6. Ejercicio 3: Multiplicación de Matrices

Realice un programa que calcule el resultado de la multiplicación de dos matrices de 4x4, en la que los datos corresponden a números de punto flotante en precisión simple.

Para realizar esto se utilizó la función _mm_mul_ps que retorna un vector __m128 con los valores resultantes de la multiplicación entre 2 vectores. El código completo puede ser encontrado en los archivos que se adjuntan con este documento).

Como se observa, al igual que el programa anterior, no se utiliza _mm_set_epix para inicializar el vector sino los vectores nativos de c y se agrega trabajan como matrices. Esto para la generalidad. El programa puede generar valores aleatorios para matrices de 4x4 o utilizar los valores ingresados en terminal. Por último, se utilizó OpenMP para paralelizar los ciclos que no tienen dependencias. Un ejemplo de la salida del programa con valores aleatorios se puede observar en la figura 3.

```
./matMult 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Matrix 1:
1.000000        2.000000        3.000000        4.000000
5.000000        6.000000        7.000000        8.000000
9.000000        10.000000       11.000000       12.000000
13.000000       14.000000       15.000000       16.000000

Matrix 2:
17.000000       18.000000       19.000000       20.000000
21.000000       22.000000       23.000000       24.000000
25.000000       26.000000       27.000000       28.000000
29.000000       30.000000       31.000000       32.000000


Result **********************
Result:
250.000000      260.000000      270.000000      280.000000
618.000000      644.000000      670.000000      696.000000
986.000000      1028.000000     1070.000000     1112.000000
1354.000000     1412.000000     1470.000000     1528.000000
```

Figura 3. Salida de matMult

Referencias

[1] B. Ward, *"What are SSE instruction sets and what do they do? - PCMech"*. PCMech, 2017.Disponible: https://www.pcmech.com/articl. [Accedido: 19-abril-2018]

[2] G. Koharchik, *.An Introduction to GCC Compiler Intrinsics in Vector Processing"*. Linuxjournal.com, 2012.Disponible: https://www.linuxjournal.com/content/introduction-gcc-compiler-intrinsics-vector-processing.. [Accedido: 19-abril-2018]