TRABAJO AC

Fase 1

Cristóbal Solar Fernández

José Antonio García García

Hugo Leite Pérez

1 Aplicación Singlethread

Asignación de memoria mediante el uso de malloc.

Inicializamos tres vectores “vectorU”, “vectorW” y “vectorT” con números de 32 bits aleatorios entre 1 y -1.

CountPos (contar positivos): recorremos el vectorW y contamos la cantidad denumeros positivos que contiene.

Sub2 (resta y división): recorremos el vector U restando la posición siguiente menos la actual y dividiendo el resultado entre 2.

Multiplicamos el contador de positivos por el vector de Sub2 y el resultado lo guardamos en el vectorAux2.

And: creamos dos punteros de enteros que referencien a los vectores: vectorAux2 y vectorT. Guardamos el resultado de hacer la operación and entre los valores de los 2 vectores en una variable entero, pasamos es valor a flotante y guardamos su valor en el vectorRes.

2 Aplicación Singlethread-SIMD

Usamos las funciones intrínsecas de 256 que son las más óptimas para nuestro trabajo.

_mm256_set1_ps la usamos para convertir cualquier elemento que necesitemos como un paquete de 8 elementos. La función toma un argumento. La usamos en la “funciónCountPos”

_mm256_cmp_ps la usamos para comparar los elementos de un vector con el cero que previamente lo introducimos mediante la función _mm256_set1_ps. Usando la extensión _CMP_GT_OQ, queremos obtener los números reales mayores o iguales que cero. La función toma 3 argumentos. Un vector de 256 pasado a través de un puntero, el cero pasado como mascara y formando un paquete de 8 elementos de 256 bits y la extensión _CMP_GT_OQ. Esta función nos devuelve el 0xFFFFFFFF en caso de que la comparación se haya hecho bien y cero en caso de que se haya hecho mal.

Una vez formado el vectorS donde albergamos nuestras comparaciones, mediante un puntero casteado a enteros y apuntando al vectorS, lo recorremos buscando todos los elementos del vector distintos de cero.

_mm256_sub_ps va restando la posición i del vector con la posición i+1, para poder llevar a cabo esta operación necesitamos alinear el vector, como tenemos un vector de 256 necesitamos llamar dos veces a la función _mm_alignr_epi8 ya que esta solo funciona con elementos de 128i. La función alignr toma tres argumentos, dos punteros apuntando al mismo vector y un contador de bytes, esta función concatena bloques de 16 bytes en un resultado temporal de 32 bytes, cambia el resultado con el contador de bytes y almacena los 16 bytes más bajos, el segundo alignr forma los siguientes 16 bytes formando el paquete de 8 elementos para hacer las operaciones asignadas.

_mm256_mul_ps aquí multiplicamos el resultado de la resta en cada posición por una máscara cuyo valor es de 0.5 ya que es lo mismo que si dividimos por 2.

La función sub recorre el vector hasta SIZE-1 por tanto el último paquete de 8 elementos tiene que ser calculado a mano.

La funciónMult utiliza la función intrínseca _mm256_mul_ps para multiplicar mediante paquetes de 8 elementos de 32 bits cada uno los elementos del vectorS que previamente han sido asignados al puntero mul256 con el contador de positivos.

La funcionAnd usa la función intrínseca _mm256_and_ps donde le pasa la puerta lógica and para el vector resultante de la funciónMult con un vector de números reales aleatorios.

3 Toma de tiempos

Los tiempos están en la hoja Excel añadida al proyecto.

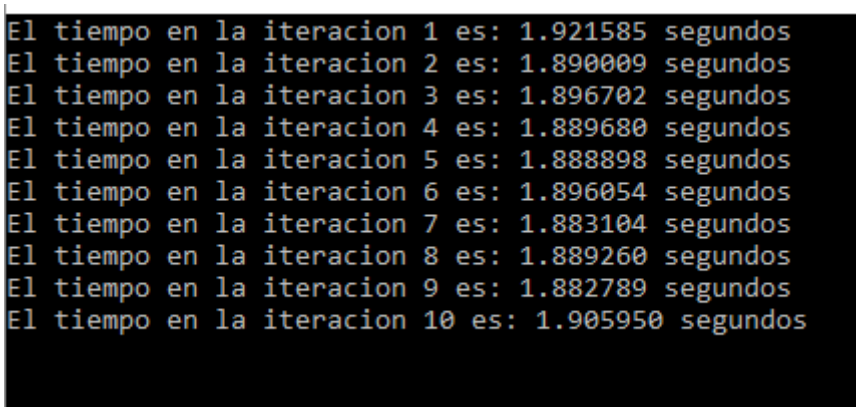
Hemos tomado tiempos de los métodos por separado porque aunque la fase 2 nos va un pelín más lenta que la fase 1 no quiere decir que todos los métodos estén mal.

Las funciones de multiplicar y la AND con intrínsecas van mucho más rápidas que las de la fase 1, entendemos que al usar operaciones no intrínsecas en las funciones tengamos tiempos peores, sobre todo en el Sub2 que es donde más se dispara el tiempo.

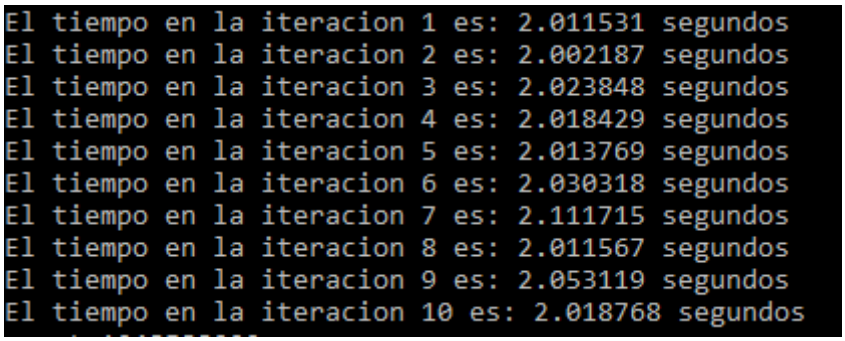
La función contar positivos tiene unos tiempos semejantes a la primera versión y es entendible que al usar el segundo bucle FOR con la comparación para sacar los elementos distintos a cero sin utilizar una función intrínseca pues ralentice el tiempo, pero es que tampoco hemos encontrado una solución porque si metiésemos otra intrínseca para buscar los distintos de cero seguiríamos teniendo que usar el puntero tmp para poder contar los reales positivos.

Contar Positivos

Singlethread

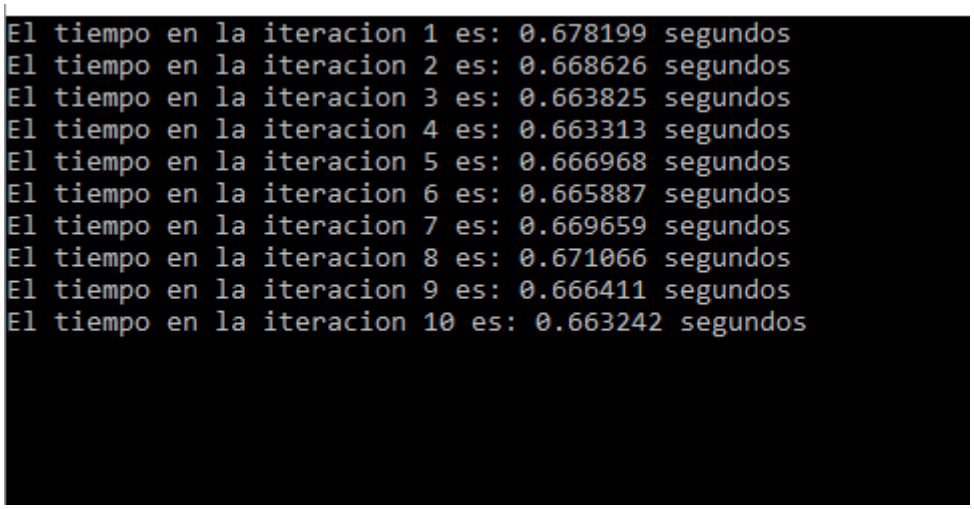


Singlethread-SIMD

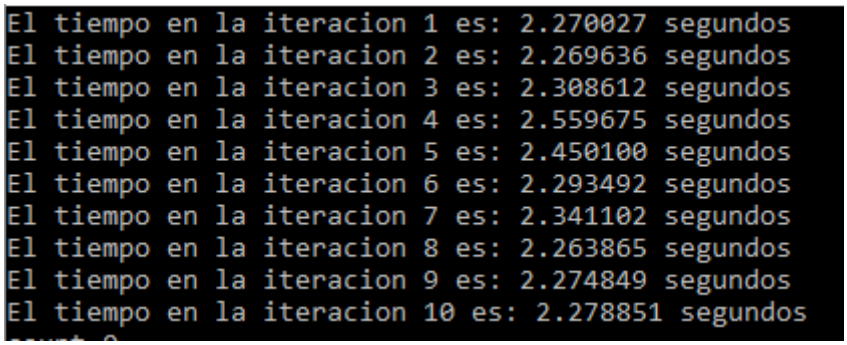


Sub2

Singlethread



Singlethread-SIMD



Multiplicación

Singlethread

```
El tiempo en la iteracion 1 es: 0.663273 segundos
El tiempo en la iteracion 2 es: 0.671605 segundos
El tiempo en la iteracion 3 es: 0.667811 segundos
El tiempo en la iteracion 4 es: 0.665840 segundos
El tiempo en la iteracion 5 es: 0.669576 segundos
El tiempo en la iteracion 6 es: 0.664791 segundos
El tiempo en la iteracion 7 es: 0.653932 segundos
El tiempo en la iteracion 8 es: 0.664477 segundos
El tiempo en la iteracion 9 es: 0.659290 segundos
El tiempo en la iteracion 10 es: 0.664421 segundos
```

Singlethread-SIMD

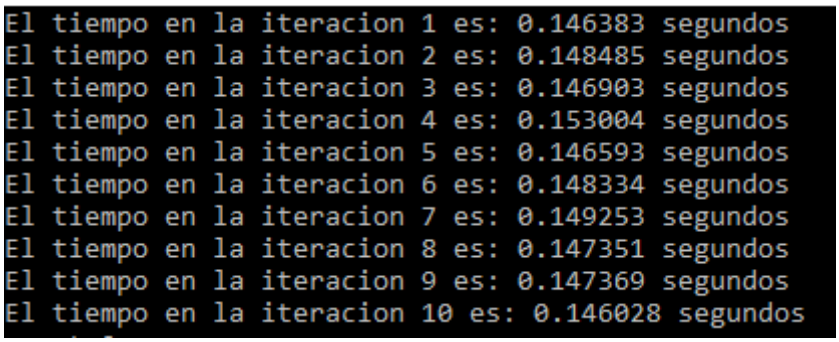
```
El tiempo en la iteracion 1 es: 0.106770 segundos
El tiempo en la iteracion 2 es: 0.111375 segundos
El tiempo en la iteracion 3 es: 0.109111 segundos
El tiempo en la iteracion 4 es: 0.109630 segundos
El tiempo en la iteracion 5 es: 0.108293 segundos
El tiempo en la iteracion 6 es: 0.112898 segundos
El tiempo en la iteracion 7 es: 0.110181 segundos
El tiempo en la iteracion 8 es: 0.112684 segundos
El tiempo en la iteracion 9 es: 0.107936 segundos
El tiempo en la iteracion 10 es: 0.107315 segundos
```

Función AND

Singlethread

```
El tiempo en la iteracion 1 es: 0.731835 segundos
El tiempo en la iteracion 2 es: 0.723933 segundos
El tiempo en la iteracion 3 es: 0.734033 segundos
El tiempo en la iteracion 4 es: 0.723505 segundos
El tiempo en la iteracion 5 es: 0.727785 segundos
El tiempo en la iteracion 6 es: 0.732501 segundos
El tiempo en la iteracion 7 es: 0.724948 segundos
El tiempo en la iteracion 8 es: 0.730947 segundos
El tiempo en la iteracion 9 es: 0.727476 segundos
El tiempo en la iteracion 10 es: 0.734656 segundos
```

Singlethread-SIMD



4 Aportación individual

Cristóbal Solar Fernández: aportación a la parte Singlethread-SIMD ayudando a elegir que funciones intrínsecas eran las más óptimas para la creación de los métodos propuestos. Y creación de la memoria.

José Antonio García García: creación de la versión sin funciones intrínsecas completa salvo la operación and, creación del esqueleto de la versión SIMD, creación del Excel, ayuda en la elección de las funciones intrínsecas para el countpos y el and. Revisión de la memoria.

Hugo Leite Pérez.: ampliacion de la parte Singlethread-SIMD pasando la versión sin funciones intrínsecas a la nueva versión SIMD, salvo la parte de la alineación de vectores. Revision de la versión Singlethread y revisión final de la parte de Singlethread-SIMD.