Práctica SIMD

*en todas las compilaciones se ha utilizado la opción “-lm” para incluir la librería math.h en la implementación ya que no compilaba incluso añadiéndola en el código fuente.

Sin vectorización

Se ha compilado el programa sin utilizar vectorización, en la siguiente figura se muestran los resultados obtenidos

```
arqp008@aloe:~/practicaSIMD$ gcc -O2 dist.c -o dist -lm
arqp008@aloe:~/practicaSIMD$ ./dist
r: 1199998.500000 1199996.500000 1199994.500000 1199992.500000 1197999.000000 1197996.875000
max: 1199998.500000
Exec time:    0.00550 sec.
arqp008@aloe:~/practicaSIMD$
```

Vectorización automática

En la siguiente figura se ha compilado el algoritmo con la opción “-ftee-vectorize” explicada por el profesor en el laboratorio y se le ha añadido la opción “-fopt-info-vec-optimized” para mostrar los bucles autovectorizados por el compilador.

También podemos observar en el siguiente comando de compilación la opción “-fopt-info-vec-missed” la cual nos da información de los datos que no han sido vectorizados.

```
arqp008@aloe:~/practicaSIMD$ gcc -fopt-info-vec-optimized -S -O2 -ftree-vectorize dist.c -o distv -lm
dist.c:23:5: note: loop vectorized
arqp008@aloe:~/practicaSIMD$ gcc -fopt-info-vec-missed -S -O2 -ftree-vectorize dist.c -o distv -lm
dist.c:11:5: note: not vectorized: not enough data-refs in basic block.
dist.c:32:5: note: not vectorized: control flow in loop.
dist.c:32:5: note: bad loop form.
dist.c:23:5: note: misalign = 0 bytes of ref a[i_67]
dist.c:23:5: note: misalign = 0 bytes of ref b[_10]
dist.c:23:5: note: virtual phi. skip.
dist.c:15:5: note: not vectorized: not enough data-refs in basic block.
dist.c:25:15: note: not vectorized: no vectype for stmt: MEM[(float *)vectp_a.12_71] = vect__8.11_73;
 scalar_type: vector(4) float
dist.c:25:15: note: not vectorized: not enough data-refs in basic block.
dist.c:15:5: note: not vectorized: not enough data-refs in basic block.
dist.c:11:5: note: not vectorized: not enough data-refs in basic block.
dist.c:34:27: note: SLP: step doesn't divide the vector-size.
dist.c:34:27: note: Unknown alignment for access: a[i_68]
dist.c:34:27: note: SLP: step doesn't divide the vector-size.
dist.c:34:27: note: Unknown alignment for access: b[i_68]
dist.c:34:27: note: Failed to SLP the basic block.
dist.c:34:27: note: not vectorized: failed to find SLP opportunities in basic block.
dist.c:15:5: note: not vectorized: not enough data-refs in basic block.
dist.c:32:25: note: not vectorized: not enough data-refs in basic block.
dist.c:15:5: note: not vectorized: not enough data-refs in basic block.
dist.c:11:5: note: not vectorized: not enough data-refs in basic block.
arqp008@aloe:~/practicaSIMD$
```

En ambos comandos se ha utilizado la opción -S para obtener también el código ensamblador de nuestro programa.

Como podemos observar se ha vectorizado solamente el vector localizado en la línea 23, el cual podemos ver en la siguiente imagen.

```
for (i = 0; i < N; i++)
{
   a[i] = (float) i;
   b[N-i-1] = (float) 2*i;
}
```

Esto quiere decir que el bucle principal de nuestro programa no se ha llegado a vectorizar, solo el bucle de inicialización de matrices de entrada.

Si nos fijamos en los datos de la captura del CLI podemos ver que no ha vectorizado debido a un flujo de control, por lo que probablemente se deba a la sentencia "if" que hay dentro del bucle principal

```
for (i = 0; i < N; i++)
{
   r[i] = sqrtf(a[i]*a[i] + b[i]*b[i]) + 0.5;
   if (r[i] > max) max = r[i];
}
```

Tras la ejecución del programa, podemos observar que el tiempo es menor pero la diferencia es ínfima (5,3ms frente a 5,5ms) debido a que no ha vectorizado el bucle principal, el cuál requiere el mayor tiempo de cómputo.

```
arqp008@aloe:~/practicaSIMD$ ./distv
r: 1199998.500000 1199996.500000 1199994.500000 1199992.500000 1197999.000000 1197996.875000
max: 1199998.500000
Exec time:   0.00528 sec.
arqp008@aloe:~/practicaSIMD$
```

Vectorización manual

Para poder vectorizar el bucle principal, y por lo tanto, optimizar el tiempo de ejecución del programa, hemos utilizado intrínsecas del compilador "-msse4" después de modificar nuestro bucle original. En concreto la opción recién mencionada hace referencia al conjunto de instrucciones más optimo que puede utilizar nuestra máquina "aloe" (SSE4.2).

El bucle optimizado ha quedado de la siguiente manera:

```
__m128 matmax;
for (i = 0; i < N; i+=4) {
    __m128 av = _mm_load_ps(&a[i]);
    __m128 bv = _mm_load_ps(&b[i]);
           av = _mm_mul_ps(av,av);
     bv = _mm_mul_ps(bv,bv);
    __m128 vfinal = _mm_add_ps(av,bv);
           vfinal = _mm_sqrt_ps(vfinal);
     vfinal = _mm_add_ps(vfinal, _mm_set1_ps(0.5));
           matmax = _mm_max_ps(vfinal, matmax);
     _mm_store_ps(&r[i],vfinal);
}
_mm_store_ps(&maxvector[0],matmax);

for (i = 0; i<4; i++) {
  if (maxvector[i] > max) {
   max = maxvector[i];
  }
}
```

Para crear este código se ha utilizado como referencia el código mm-v.c de la clase de prácticas.

Básicamente se ha traducido el bucle original a uno con instrucciones que permitan el paralelismo SIMD.

Como podemos observar, utilizamos vectores del tipo __m128 el cual es un registro que almacena 4 valores de float de 4 bytes cada uno.

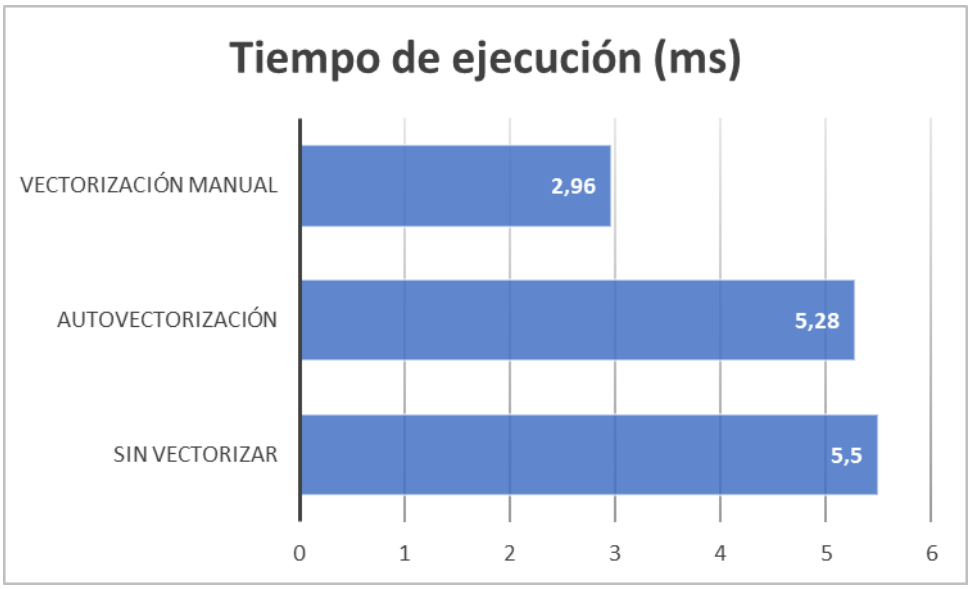
Primero creamos 2 vectores av y bv los cuales cargamos con los vectores originales a y b. Debido a que no podemos utilizar operadores con el conjunto de instrucciones SEE4.2, después de cargarlos los multiplicamos por ellos mismos (función_mm_mul_ps) tal y como se hace en el bucle original y luego creamos otro vector “vfinal” auxiliar para almacenar la suma de estos dos anteriores (mm_add_ps), hacerle la raíz a sus elementos (_mm_sqrt_ps) y después sumarle una constante 0.5 (_mm_set1_ps) a todos los valores del vector.

Por último, para finalizar el bucle, utilizamos el vector matmax para almacenar los máximos del vector vfinal y lo guardamos en el array de resultados r.

Para calcular el máximo, guardamos en el array maxvector (empezando desde la posición 0) el vector matmax y calculamos el máximo valor que contiene en el siguiente bucle comparándolo con la variable max inicializada a 0.0 al principio del código.

Ahora sí que podemos ver una mejora considerable en el tiempo de ejecución comparándolo con los apartados anteriores, de hecho se ha reducido el tiempo casi en 3 milisegundos. Todo esto ha sido debido a la optimización del bucle principal.

```
arqp008@aloe:~/practicaSIMD$ gcc -O2 -msse4 dist-v.c -o dist-v -lm
arqp008@aloe:~/practicaSIMD$ ./dist-v
r: 1199998.500000 1199996.500000 1199994.500000 1199992.500000 1197999.000000 1197996.875000
max: 1199998.500000
Exec time:    0.00296 sec.
```



Como conclusión podríamos decir que, aunque los compiladores hagan un muy buen trabajo optimizando el código, se sigue requiriendo de la mano del programador en muchos casos como este.

Este tipo de optimizaciones suponen menos tiempo de cómputo, lo que se traduce en menos consumo de energía, de uso de CPU y por lo tanto, un beneficio para la persona u organización que pueda estar encargado de ejecutar el código.