2º curso / 2º cuatr.

Grados Ing. Inform.

# Arquitectura de Computadores (AC)

## Cuaderno de prácticas.
## Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): QUINTÍN MESA ROMERO
Grupo de prácticas y profesor de prácticas:

---

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo y se lista con lscpu): **Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz**

Sistema operativo utilizado: ***Ubuntu 20.04.3 LTS***

Versión de gcc utilizada: **gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0**

Volcado de pantalla que muestre lo que devuelve `lscpu` en la máquina en la que ha tomado las medidas:

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~$ lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
Lista de la(s) CPU(s) en línea:  0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:          4
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          126
Nombre del modelo:               Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
Revisión:                        5
CPU MHz:                         1500.000
CPU MHz máx.:                    3900,0000
CPU MHz mín.:                    400,0000
BogoMIPS:                        2995.20
Virtualización:                  VT-x
Caché L1d:                       192 KiB
Caché L1i:                       128 KiB
Caché L2:                        2 MiB
Caché L3:                        8 MiB
CPU(s) del nodo NUMA 0:          0-7
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabl
                                 ed via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __
                                 user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional
                                 , RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Indicadores:                     fpu vme de pse tsc msr pae mce cx8 apic sep
                                  mtrr pge mca cmov pat pse36 clflush dts ac
                                 pi mmx fxsr sse sse2 ss ht tm pbe syscall n
                                 x pdpe1gb rdtscp lm constant_tsc art arch_p
```

1. Modificar el código secuencial para la multiplicación de matrices disponible en SWAD (solo el trozo que calcula la multiplicación) para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre `-O2`) a partir de la modificación realizada. Incorporar los códigos modificados en el cuaderno.

**MODIFICACIONES REALIZADAS (al menos dos modificaciones):**

**Modificación A) –explicación-:**

Para reducir el tiempo de ejecución, podemos desenrollar los bucles en la variable j; 4 operaciones más por bucle. La ventaja que tiene el desenrollar bucles es que reduce el número de saltos, aumenta la oportunidad de encontrar instrucciones independientes, facilita la posibilidad de insertar instrucciones para ocultar las latencias. Como contrapartida, aumenta el tamaño del código.

**Modificación B) –explicación-:**

Empleamos la localidad de accesos mediante el intercambio de las variables j y k para aprovechar así la localidad dado que se cambiará el acceso a los datos según los almacena el compilador.

**CÓDIGOS FUENTE MODIFICACIONES**

**A) Captura de** `pmm-secuencial-modificado_A.c`

```
for(i = 0; i < N; i++){
    for (j = 0; j < N; j+=4)
        for (k = 0; k < N; k++)
            m3[i][j] += m1[i][k] * m2[k][j];
            m3[i][j+1] += m1[i][k]*m2[k][j+1];
            m3[i][j+2] += m1[i][k]*m2[k][j+2];
            m3[i][j+3] += m1[i][k]*m2[k][j+3];
}
```

**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ gcc -O2 pmm-secuencial-modificado_A.c -o pmm-secuencial_mod
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./pmm-secuencial_mod 1300
Tiempo: 1.877580975     Tamaño: 1300
 m1[0][0]:0.000000      m1[N-1][N-1]: 1689999.000000
 m2[0][0]:0.000000      m2[N-1][N-1]: 168999.900000
m3[0][0]: 73148855.000000       m3[N-1][N-1]: 371007243617855.000000
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$
```

**B) Captura de** `pmm-secuencial-modificado_B.c`

```
for(i = 0; i < N; i++){
    for (k = 0; k < N; k++)
        for (j = 0; j < N; j++)
            m3[i][j] += m1[i][k] * m2[k][j];
}
```

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ gcc -O2 pmm-secuencial-modificado_B.c -o pmm-secuencial_modB
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./pmm-secuencial_modB 1300
Tiempo: 1.889313880     Tamaño: 1300
 m1[0][0]:0.000000      m1[N-1][N-1]: 1689999.000000
 m2[0][0]:0.000000      m2[N-1][N-1]: 168999.900000
m3[0][0]: 73148855.000000       m3[N-1][N-1]: 371007243617855.000000
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$
```

**TIEMPOS:**

| Modificación | Breve descripción de las modificaciones | -O2 |
|---|---|---|
| Sin modificar | | *3.843726036* |
| Modificación A) | Desenrollar bucle de la variable j en 4 operaciones /buc | 2.082862845 |
| Modificación B) | Cambiar j por k | 1.889313880 |

2. Usar en este ejercicio el programa secuencial disponible en SWAD que utiliza como base el código de la Figura 1. Modificar en el programa el código mostrado en la Figura 1 para reducir el tiempo de ejecución. Justificar los tiempos obtenidos (usando siempre -O2) a partir de la modificación realizada. En las ejecuciones de evaluación usar valores de N y M mayores que 1000. Incorporar los códigos modificados en el cuaderno.

**Figura 1** . Código C++ que suma dos vectores. `M` y `N` deben ser parámetros de entrada al programa, usar valores mayores que 1000 en la evaluación.

```
struct nombre {
        int a;
        int b;
}  s[N];

main()
{
   …
   for (ii=0; ii<M;ii++) {
      X1=0; X2=0;
      for(i=0; i<N;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<N;i++)  X2+=3*s[i].b-ii;

      if (X1<X2)  R[ii]=X1  else  R[ii]=X2;
   }
   …
}
```

**MODIFICACIONES REALIZADAS (al menos dos modificaciones):**

**Modificación A) –explicación-:**

Una primera modificación del código podría ser la unificación de los bucles for anidados en el cálculo para así ahorrarnos recorrer dos veces el mismo bucle.

**Modificación B) –explicación-:**

La segunda modificación que podemos hacerle al programa es la referida a la localidad de los accesos ya que la forma en la que se declaran los arrays determina cómo se almacenan en memoria, por lo tanto, almacenaremos todas las a y b juntas en vez de un array con ambas variables.

## CÓDIGOS FUENTE MODIFICACIONES

**A) Captura** figura1-modificado_A.c

```
for (ii=0; ii<M;ii++){
        X1=0; X2=0;
        for(i=0; i<N;i++){
          X1 += 2*s[i].a + ii;
          X2 += 3*s[i].b - ii;
        }

        if (X1<X2) {R[ii]=X1;} else {R[ii]=X2;}
}
```

**Capturas de pantalla (que muestren la compilación y que el resultado es correcto):**

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ gcc -O2 figura1-original_A.c -o figura1-originalA
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./figura1-originalA 3000 30000
Tiempo: 0.065762377
Elemento 0 y 29999 de R: -609222, -89557713
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$
```

**B)**

```
    for (ii=0; ii<M;ii++){
            X1=0; X2=0;
            for(i=0; i<N;i++) X1 += 2*s.a[i] + ii;
            for(i=0; i<N;i++) X2 += 3*s.b[i] - ii;

            if (X1<X2) {R[ii]=X1;} else {R[ii]=X2;}
    }
```

```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ gcc -O2 figura1-original_B.c -o figura1-originalB
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./figura1-originalB 3000 30000
Tiempo: 0.102598364
Elemento 0 y 29999 de R: -602016, -89554710
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informática/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$
```

**TIEMPOS:**

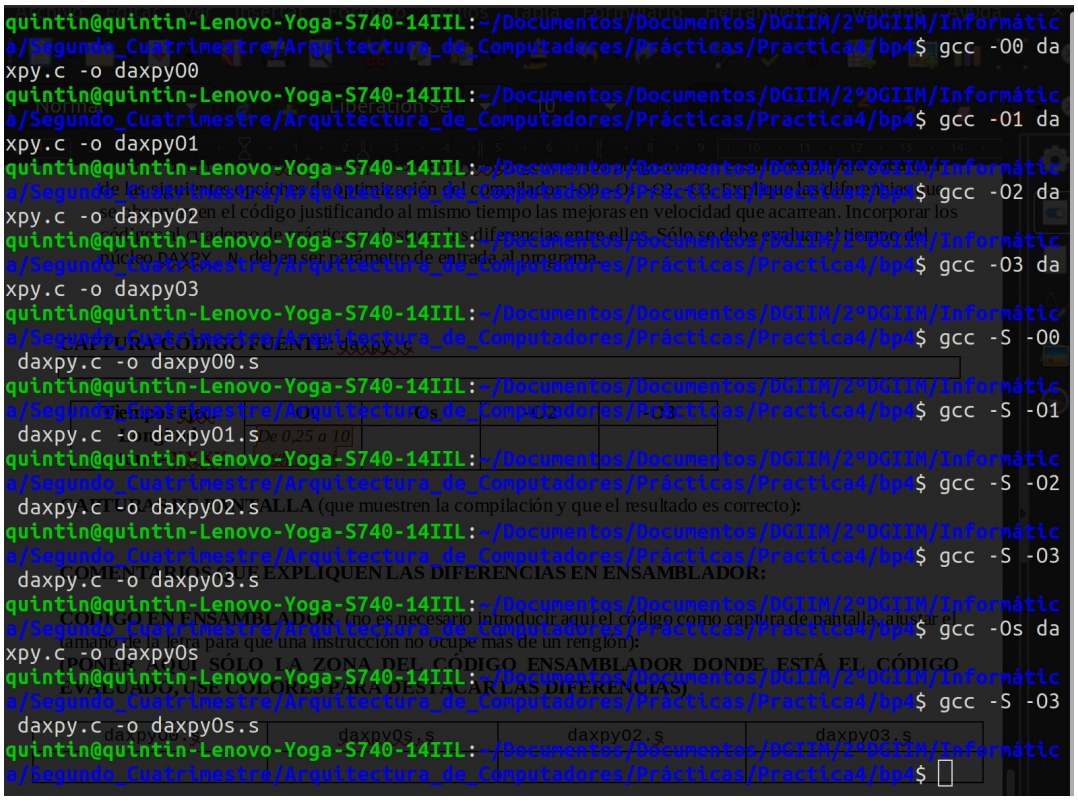
| Modificación | Breve descripción de las modificaciones | -O2 |
|---|---|---|
| Sin modificar | | *0.108498667* |
| Modificación A) | unificación de los bucles for anidados | 0.065762377 |
| Modificación B) | almacenamos todas las a y b juntas en vez de un array con ambas variables | 0.102598364 |

3. El benchmark Linpack ha sido uno de los programas más ampliamente utilizados para evaluar las prestaciones de los computadores. De hecho, se utiliza como base en la lista de los 500 computadores más rápidos del mundo (el Top500 Report). El núcleo de este programa es una rutina que opera con flotantes de doble precisión denominada `DAXPY` (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector (Lección 3/Tema 1):

```
for (i=0;i<N;i++) y[i]= a*x[i] + y[i];
```

A partir del programa `DAXPY` disponible en SWAD, generar los programas en ensamblador para cada una de las siguientes opciones de optimización del compilador: -O0, -Os, -O2, -O3. Explique las diferencias que se observan en el código justificando al mismo tiempo las mejoras en velocidad que acarrean. Incorporar los códigos al cuaderno de prácticas y destacar las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo `DAXPY`. `N` deben ser parámetro de entrada al programa.

**CAPTURAS DE PANTALLA** (que muestren la compilación y que el resultado es correcto)**:**



```
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./daxpyO0
10000000 5
Tiempo:0.040902721       / Tamaño Vectores:10000000     / alpha*x[0]+y[0]=z[0](5.000000
*0.000000+0.000985=0.000985) / / alpha*x[9999999]+y[9999999]=z[9999999](5.000000*0.1726
82+0.185006=1.048418) /
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./daxpyO1
10000000 5
Tiempo:0.025557891       / Tamaño Vectores:10000000     / alpha*x[0]+y[0]=z[0](5.000000
*0.000000+0.000985=0.000985) / / alpha*x[9999999]+y[9999999]=z[9999999](5.000000*0.1726
82+0.185006=1.048418) /
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./daxpyO2
10000000 5
Tiempo:0.024480771       / Tamaño Vectores:10000000     / alpha*x[0]+y[0]=z[0](5.000000
*0.000000+0.000985=0.000985) / / alpha*x[9999999]+y[9999999]=z[9999999](5.000000*0.1726
82+0.185006=1.048418) /
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./daxpyO3
10000000 5
Tiempo:0.022411154       / Tamaño Vectores:10000000     / alpha*x[0]+y[0]=z[0](5.000000
*0.000000+0.000985=0.000985) / / alpha*x[9999999]+y[9999999]=z[9999999](5.000000*0.1726
82+0.185006=1.048418) /
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$ ./daxpyOs
10000000 5
Tiempo:0.024499511       / Tamaño Vectores:10000000     / alpha*x[0]+y[0]=z[0](5.000000
*0.000000+0.000985=0.000985) / / alpha*x[9999999]+y[9999999]=z[9999999](5.000000*0.1726
82+0.185006=1.048418) /
quintin@quintin-Lenovo-Yoga-S740-14IIL:~/Documentos/Documentos/DGIIM/2ºDGIIM/Informátic
a/Segundo_Cuatrimestre/Arquitectura_de_Computadores/Prácticas/Practica4/bp4$
```

| **Tiempos ejec. Longitud vectores=XXXX** | **-O0** | **-Os** | **-O2** | **-O3** |
|---|---|---|---|---|
| | *0.040902721* | 0.024499511 | 0.024480771 | 0.022411154 |

**COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:**

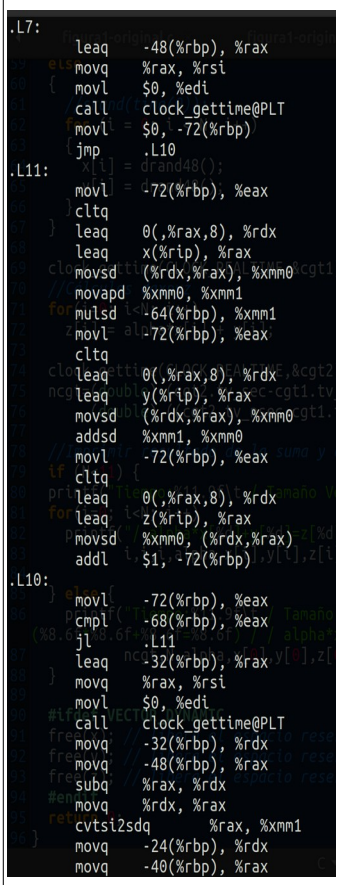
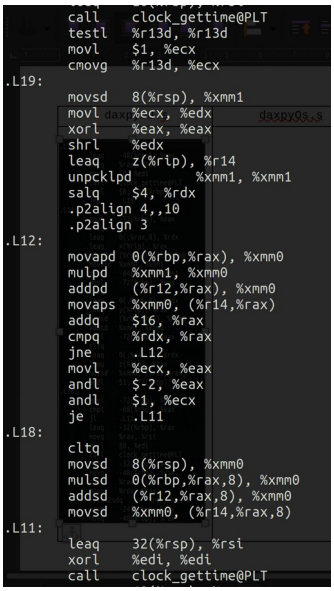
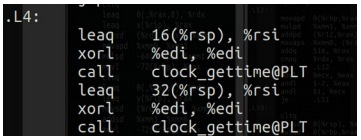
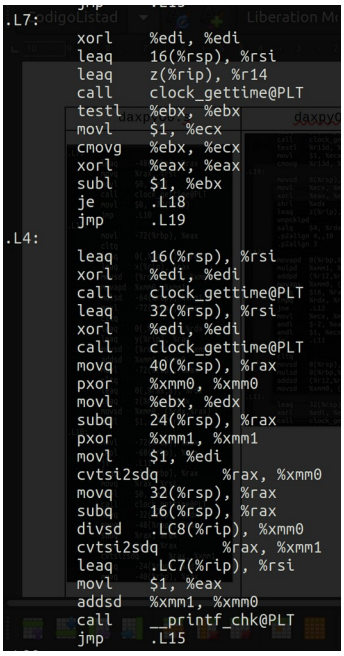
**- O0:** No optimización.

**-O2:** El número de instrucciones es reducido y observamos cambios positivos considerables en la eficiencia de las instrucciones que se usen.

**- O3:** El código en ensamblador es mucho más complejo y difícil de entender, aumenta también el número de subrutinas de las instrucciones que se usen.

**-Os:** Disminuye el tamaño del ejecutable y el tamaño de archivo también es menor.

**CÓDIGO EN ENSAMBLADOR** (no es necesario introducir aquí el código como captura de pantalla, ajustar el tamaño de la letra para que una instrucción no ocupe más de un renglón)**:**
**(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)**

| daxpy00.s | daxpy0s.s | daxpy02.s | daxpy03.s |
|---|---|---|---|
|  |  |  |  |

```
.L7:
        leaq    -48(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime@PLT
        movl    $0, -72(%rbp)
        jmp     .L10
.L11:
        movl    -72(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    x(%rip), %rax
        movsd   (%rdx,%rax), %xmm0
        movapd  %xmm0, %xmm1
        mulsd   -64(%rbp), %xmm1
        movl    -72(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    y(%rip), %rax
        movsd   (%rdx,%rax), %xmm0
        addsd   %xmm1, %xmm0
        movl    -72(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    z(%rip), %rax
        movsd   %xmm0, (%rdx,%rax)
        addl    $1, -72(%rbp)
.L10:
        movl    -72(%rbp), %eax
        cmpl    -68(%rbp), %eax
        jl      .L11
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime@PLT
        movq    -32(%rbp), %rdx
        movq    -48(%rbp), %rax
        subq    %rax, %rdx
        movq    %rdx, %rax
        cvtsi2sdq       %rax, %xmm1
        movq    -24(%rbp), %rdx
        movq    -40(%rbp), %rax
```

```
        call    clock_gettime@PLT
        testl   %r13d, %r13d
        movl    $1, %ecx
        cmovg   %r13d, %ecx
.L19:
        movsd   8(%rsp), %xmm1
        movl    %ecx, %edx
        xorl    %eax, %eax
        shrl    %edx
        leaq    z(%rip), %r14
        unpcklpd        %xmm1, %xmm1
        salq    $4, %rdx
        .p2align 4,,10
        .p2align 3
.L12:
        movapd  0(%rbp,%rax), %xmm0
        mulpd   %xmm1, %xmm0
        addpd   (%r12,%rax), %xmm0
        movaps  %xmm0, (%r14,%rax)
        addq    $16, %rax
        cmpq    %rdx, %rax
        jne     .L12
        movl    %ecx, %eax
        andl    $-2, %eax
        andl    $1, %ecx
        je      .L11
.L18:
        cltq
        movsd   8(%rsp), %xmm0
        mulsd   0(%rbp,%rax,8), %xmm0
        addsd   (%r12,%rax,8), %xmm0
        movsd   %xmm0, (%r14,%rax,8)
.L11:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

```
.L4:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
```

```
.L7:
        xorl    %edi, %edi
        leaq    16(%rsp), %rsi
        leaq    z(%rip), %r14
        call    clock_gettime@PLT
        testl   %ebx, %ebx
        movl    $1, %ecx
        cmovg   %ebx, %ecx
        xorl    %eax, %eax
        subl    $1, %ebx
        je      .L18
        jmp     .L19
.L4:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    40(%rsp), %rax
        pxor    %xmm0, %xmm0
        movl    %ebx, %edx
        subq    24(%rsp), %rax
        pxor    %xmm1, %xmm1
        movl    $1, %edi
        cvtsi2sdq       %rax, %xmm0
        movq    32(%rsp), %rax
        subq    16(%rsp), %rax
        divsd   .LC8(%rip), %xmm0
        cvtsi2sdq       %rax, %xmm1
        leaq    .LC7(%rip), %rsi
        movl    $1, %eax
        addsd   %xmm1, %xmm0
        call    __printf_chk@PLT
        jmp     .L15
```