2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Francisco Javier Bolívar Expósito
Grupo de prácticas y profesor de prácticas: D1 Francisco Barranco
Fecha de entrega:
Fecha evaluación en clase:

Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel® Core™ i7-4720HQ CPU @ 2.60GHz*

Sistema operativo utilizado: *Manjaro KDE*

Versión de gcc utilizada: *8.3.0-1*

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~] 2019-05-21 martes
$lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
Tamaños de las direcciones:          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:                              60
Nombre del modelo:                   Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz
Revisión:                            3
CPU MHz:                             878.448
CPU MHz máx.:                        3600,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5189.90
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            6144K
CPU(s) del nodo NUMA 0:              0-7
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
 dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good no
pl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr p
dcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm cpuid_fault epb
invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 s
mep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts flush_l1d
```

1. Para el núcleo que se muestra en el Figura 1, y para un programa que implemente la multiplicación de matrices con datos flotantes en doble precisión (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución (evalúe el tiempo y modifique sólo el trozo que hace la multiplicación y el trozo que se muestra en la Figura 1). Justifique los tiempos obtenidos (use –O2) a partir de la modificación realizada. Incorpore los códigos modificados en el cuaderno.

1.2 Genere los códigos en ensamblador con -O2 para el original y dos códigos modificados obtenidos en el punto anterior (incluido el que supone menor tiempo de ejecución) e incorpórelos al cuaderno de prácticas. Destaque las diferencias entre ellos en el código ensamblador.

1.3 (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

Figura 1 . Código C++ que suma dos vectores

```
struct {
        int a;
        int b;
}  s[5000];

main()
{
   …
   for (ii=0; ii<40000;ii++) {
      X1=0; X2=0;
      for(i=0; i<5000;i++)  X1+=2*s[i].a+ii;
      for(i=0; i<5000;i++)  X2+=3*s[i].b-ii;

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

A) MULTIPLICACIÓN DE MATRICES:
CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#ifdef _OPENMP
   #include <omp.h>
#else
   #define omp_get_thread_num() 0
#endif

#define MAX 20000
double m1[MAX][MAX], m2[MAX][MAX], m3[MAX][MAX];

int main(int argc, char** argv)
{
   if (argc<2)
   {
      printf("Faltan nº columnas de la matriz\n");
      exit(-1);
   }

   unsigned int N = atoi(argv[1]);
   if (N>MAX) N=MAX;

   struct timespec cgt1,cgt2;
   double ncgt;

   printf("Tamaño Matriz:%u x %u (%u B)\n", N, N, sizeof(unsigned int));
   //Inicializacion
   for (int i=0; i<N; ++i){
      for (int j=0; j<N; ++j)
      {
         m1[i][j] = N*0.1+i*0.1;
         m2[i][j] = N*0.1+i*0.1;
      }
   }
   //Multiplicacion
   clock_gettime(CLOCK_REALTIME,&cgt1);

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

   clock_gettime(CLOCK_REALTIME,&cgt2);

   //Salida del resultado y el tiempo de ejecucion
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

   printf("Tiempo:%11.9f\t / Tamaño columnas:%u\t/ m3[0] = %8.6f / / m3[%d] = %8.6f /\n", ncgt, N,
   m3[0][0], N-1, m3[N-1][N-1]);

   return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrollado del bucle más exterior de la multiplicación de matrices

Modificación b) –explicación-: Cambio del orden al que se accede a la matriz para hacerlo primero por filas y mejorar la localidad espacial.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura de `pmm-secuencial-modificado_a.c`

```
38    »   //Multiplicacion
39    »   clock_gettime(CLOCK_REALTIME,&cgt1);
40    »
41    »   for (int i=0; i<N; i+=4)
42    »   »   for (int j=0; j<N; ++j)
43    »   »   »   for (int k=0; k < N; ++k)
44    »   »   »   {
45    »   »   »   »   m3[i][j] += m1[i][k] * m2[k][j];
46    »   »   »   »   m3[i+1][j] += m1[i+1][k] * m2[k][j];
47    »   »   »   »   m3[i+2][j] += m1[i+2][k] * m2[k][j];
48    »   »   »   »   m3[i+3][j] += m1[i+3][k] * m2[k][j];
49    »   »   »   }
50
51    »   clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 pmm-secuencial-modificado_a.c -o pmm-modA
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./pmm-modA 1000
Tamaño Matriz:1000 x 1000 (4 B)
Tiempo:1.037592148       / Tamaño columnas:1000 / m3[0][0] = 14995000.000000 / / m3[999][999] = 29975005.000000 /
```

(Resultado original)

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 pmm-secuencial.c -o pmm-secuencial
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./pmm-secuencial 1000
Tamaño Matriz:1000 x 1000 (4 B)
Tiempo:2.647424575       / Tamaño columnas:1000 / m3[0][0] = 14995000.000000 / / m3[999][999] = 29975005.000000 /
```

b) Captura de `pmm-secuencial-modificado_b.c`

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-27 lunes
$g++ -O2 pmm-secuencial-modificado_b.c -o pmm-modB
```

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-05-27 lunes
$./pmm-modB 500
Tamaño Matriz:500 x 500 (4 B)
Tiempo:0.082006836       / Tamaño columnas:500  / m3[0][0] = 1873750.000000 / / m3[499][499] = 3743752.500000 /
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-05-27 lunes
$./pmm-modB 1000
Tamaño Matriz:1000 x 1000 (4 B)
Tiempo:0.796330151       / Tamaño columnas:1000 / m3[0][0] = 14995000.000000 / / m3[999][999] = 29975005.000000 /
```

1.1. TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*De 0,12 a 2.38 seg.*
Modificación a)	Desenrollado del bucle más exterior de la multiplicación de matrices	*De 0,06 a 1 seg.*
Modificación b)	Cambio del orden al que se accede a la matriz para hacerlo primero por filas	*De 0,08 a 0,8 seg.*
...		

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Al desenrollar los bucles se reducen las iteraciones y por lo tanto se reduce el número de saltos y aumenta la oportunidad de encontrar instrucciones independientes.

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

```c
#include <time.h>
#include <stdio.h>

struct
{
   int a;
   int b;
} s[5000];

int X1, X2, R[40000];

int main()
{
   struct timespec cgt1,cgt2;
   double ncgt;
   int i, ii;

   for (i=0; i<5000; ++i)
   {
      s[i].a = i*0.1;
      s[i].b = i*0.1;
   }

   clock_gettime(CLOCK_REALTIME,&cgt1);

   for (ii=0; ii<40000;ii++)
   {
      X1=0; X2=0;
      for(i=0; i<5000;i++) X1+=2*s[i].a+ii;
      for(i=0; i<5000;i++) X2+=3*s[i].b-ii;

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

   clock_gettime(CLOCK_REALTIME,&cgt2);

   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

   printf("Tiempo:%11.9f\t R[0] = %d, R[39999] = %d\n", ncgt, R[0], R[39999]);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrollado de los dos bucles interiores reduciendo así el número de saltos, y aumentando la oportunidad de encontrar instrucciones independientes.

Modificación b) –explicación-: Realizar las operaciones de los dos bucles interiores en un solo bucle reduciendo el número de saltos.

...

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1**`-modificado_a.c`**

```
clock_gettime(CLOCK_REALTIME,&cgt1);

for (ii=0; ii<40000;++ii)
{
»   X1=0; X2=0;
»   for(i=0; i<5000; i+=4)
»   {
»   »   X1+=2*s[i].a+ii;
»   »   X1+=2*s[i+1].a+ii;
»   »   X1+=2*s[i+2].a+ii;
»   »   X1+=2*s[i+3].a+ii;
»   }
»   for(i=0; i<5000; i+=4)
»   {
»   »   X2+=3*s[i].b-ii;
»   »   X2+=3*s[i+1].b-ii;
»   »   X2+=3*s[i+2].a-ii;
»   »   X2+=3*s[i+3].a-ii;
»   }
»
»   if (X1<X2) R[ii]=X1; else R[ii]=X2;
}

clock_gettime(CLOCK_REALTIME,&cgt2);
```

Capturas de pantalla (que muestren la compilación y que el resultado es correcto):
(Ejecución del original)

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 figura1-original.c -o figura1-original
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./figura1-original
Tiempo:0.220244330      R[0] = 2495000, R[39999] = -196252500
```

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 figura1-modificado_a.c -o figura1-modA
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./figura1-modA
Tiempo:0.137680171      R[0] = 2495000, R[39999] = -196252500
```

b) Captura figura1**`-modificado_b.c`**

```
clock_gettime(CLOCK_REALTIME,&cgt1);

for (ii=0; ii<40000;ii++)
{
»   X1=0; X2=0;
»   for(i=0; i<5000;i++)
»   {
»   »   X1+=2*s[i].a+ii;
»   »   X2+=3*s[i].b-ii;
»   }
»
»   if (X1<X2) R[ii]=X1; else R[ii]=X2;
}

clock_gettime(CLOCK_REALTIME,&cgt2);
```

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 figura1-modificado_b.c -o figura1-modB
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./figura1-modB
Tiempo:0.147871186      R[0] = 2495000, R[39999] = -196252500
```

1.1. TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*0,221*
Modificación a)	Desenrollado de los dos bucles interiores	*0,140*
Modificación b)	Realizar las operaciones de los dos bucles interiores en un solo bucle	*0,148*
...		

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO:

Al reducir las iteraciones de los bucles ya sea desenrollandolos o juntando los dos en uno se reduce el número de saltos y aumenta la oportunidad de encontrar instrucciones independientes.

2. 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=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

2.1. Genere 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. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos. Sólo se debe evaluar el tiempo del núcleo DAXPY

2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en

coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MAX 100000000

double x[MAX], y[MAX];

int main(int argc, char** argv)
{
  if (argc<2)
  {
    printf("Faltan nº columnas de la matriz\n");
    exit(-1);
  }
  unsigned int N = atoi(argv[1]);
  if (N>MAX) N=MAX - 1;


  struct timespec cgt1,cgt2;
  double ncgt;

  //Inicializacion
  for (int i=0; i<=N; ++i)
  {
    y[i] = N*0.1+i*0.1;
    x[i] = N*0.1+i*0.1;
  }

  //Nucleo
  clock_gettime(CLOCK_REALTIME,&cgt1);
  for (int i=1;i<=N;i++)
    y[i] = 2*x[i] + y[i];
  clock_gettime(CLOCK_REALTIME,&cgt2);

  ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

  printf("Tiempo:%11.9f\t / N:%u\t/ y[0] = %8.6f / y[%d] = %8.6f /\n", ncgt, N, y[0], N, y[N]);

  return 0;
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
	0,23sg	0,13sg	0,139sg	0,130sg

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

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O0 daxpy.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./a.out 100000000
Tiempo:0.230705444       / N:99999999   / y[0] = 9999999.900000 / y[99999999] = 59999999.400000 /
```

```
$g++ -Os daxpy.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./a.out 100000000
Tiempo:0.128661778       / N:99999999   / y[0] = 9999999.900000 / y[99999999] = 59999999.400000 /
```

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O2 daxpy.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./a.out 100000000
Tiempo:0.139414029       / N:99999999   / y[0] = 9999999.900000 / y[99999999] = 59999999.400000 /
```

```
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$g++ -O3 daxpy.c
[FranciscoJavierBolívarExpósito jakerxd@jakerxd-pc:~/Nextcloud/Apuntes_Universidad/AC/Practicas/bp4/ejer1] 2019-0
5-23 jueves
$./a.out 100000000
Tiempo:0.130890561       / N:99999999   / y[0] = 9999999.900000 / y[99999999] = 59999999.400000 /
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR: El código O0 es sin optimización y es una traducción literal a ensamblador del código original. El Os solo utiliza las instrucciones necesarias y O2 optimiza el acceso a memoria con alineaciones e introduce la suma de 8 en un calculo de de desplazamiento. La optimización O3 utiliza movimientos condicionales para reducir los saltos y optimiza el acceso a memoria.

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

```
        call    clock_gettime@PLT
        movl    $1, -60(%rbp)
.L11:
        movl    -60(%rbp), %eax
        cmpl    %eax, -68(%rbp)
        jb      .L10
        movl    -60(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    x(%rip), %rax
        movsd   (%rdx,%rax), %xmm0
        movapd  %xmm0, %xmm1
        addsd   %xmm0, %xmm1
        movl    -60(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    y(%rip), %rax
        movsd   (%rdx,%rax), %xmm0
        addsd   %xmm1, %xmm0
        movl    -60(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        leaq    y(%rip), %rax
        movsd   %xmm0, (%rdx,%rax)
        addl    $1, -60(%rbp)
        jmp     .L11
.L10:
        leaq    -32(%rbp), %rax
        movq    %rax, %rsi
        movl    $0, %edi
        call    clock_gettime@PLT
```

daxpy0s.s

```
        call    clock_gettime@PLT
        leaq    y(%rip), %rax
        xorl    %edx, %edx
.L5:
        incq    %rdx
        addq    $8, %rax
        cmpl    %edx, %ebx
        jb      .L4
        movsd   (%r12,%rdx,8), %xmm0
        addsd   %xmm0, %xmm0
        addsd   (%rax), %xmm0
        movsd   %xmm0, (%rax)
        jmp     .L5
.L4:
        xorl    %edi, %edi
        leaq    24(%rsp), %rsi
        call    clock_gettime@PLT
```

daxpy02.s

```
        call    clock_gettime@PLT
        testl   %r14d, %r14d
        je      .L4
        movl    $1, %eax
        .p2align 4,,10
        .p2align 3
.L5:
        movsd   0(%rbp,%rax,8), %xmm0
        addsd   %xmm0, %xmm0
        addsd   (%rbx,%rax,8), %xmm0
        movsd   %xmm0, (%rbx,%rax,8)
        addq    $1, %rax
        cmpl    %eax, %r12d
        jnb     .L5
.L4:
        xorl    %edi, %edi
        leaq    16(%rsp), %rsi
        call    clock_gettime@PLT
```

daxpy03.s

```
        call    clock_gettime@PLT
        testl   %r13d, %r13d
        je      .L6
        cmpl    $1, %r14d
        movl    $1, %esi
        leal    -2(%rbp), %eax
        cmova   %ebp, %esi
        cmpl    $-4, %eax
        ja      .L12
        leaq    8+x(%rip), %rax
        leaq    8+y(%rip), %rdx
        movl    %esi, %ecx
        shrl    %ecx
        salq    $4, %rcx
        addq    %rax, %rcx
        .p2align 4,,10
        .p2align 3
.L8:
        movupd  (%rax), %xmm0
        movupd  (%rdx), %xmm5
        addq    $16, %rax
        addq    $16, %rdx
        addpd   %xmm0, %xmm0
        addpd   %xmm5, %xmm0
        movups  %xmm0, -16(%rdx)
        cmpq    %rcx, %rax
        jne     .L8
        movl    %esi, %edx
        andl    $-2, %edx
        leal    1(%rdx), %eax
        cmpl    %esi, %edx
        je      .L6
.L7:
        cltq
        movsd   (%r12,%rax,8), %xmm0
        addsd   %xmm0, %xmm0
        addsd   (%rbx,%rax,8), %xmm0
        movsd   %xmm0, (%rbx,%rax,8)
.L6:
        xorl    %edi, %edi
        leaq    16(%rsp), %rsi
        call    clock_gettime@PLT
```