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 Domínguez Lorente
Grupo de prácticas y profesor de prácticas: Christian Morillas (B1)
Fecha de entrega:
Fecha evaluación en clase:

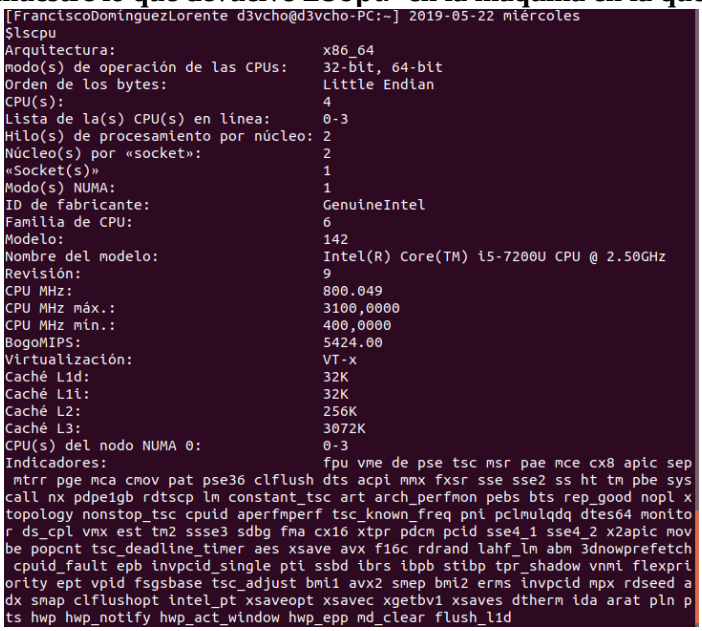
Antes de comenzar a realizar el trabajo de este cuaderno consultar el fichero con los normas de prácticas que se encuentra en SWAD

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz*

Sistema operativo utilizado: *Ubuntu 18.04*

Versión de gcc utilizada: *gcc (Ubuntu 7.4.0-1ubuntu1~18.04) 7.4.0*

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



```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~] 2019-05-22 miércoles
$lscpu
Arquitectura:                    x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:              Little Endian
CPU(s):                          4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:          2
«Socket(s)»                      1
Modo(s) NUMA:                    1
ID de fabricante:                GenuineIntel
Familia de CPU:                  6
Modelo:                          142
Nombre del modelo:               Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Revisión:                        9
CPU MHz:                         800.049
CPU MHz máx.:                    3100,0000
CPU MHz mín.:                    400,0000
BogoMIPS:                        5424.00
Virtualización:                  VT-x
Caché L1d:                       32K
Caché L1i:                       32K
Caché L2:                        256K
Caché L3:                        3072K
CPU(s) del nodo NUMA 0:          0-3
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 sys
call nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl x
topology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monito
r ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic mov
be popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch
 cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpri
ority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid mpx rdseed a
dx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln p
ts hwp hwp_notify hwp_act_window hwp_epp md_clear 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 <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char **argv)  {
   int i, j, k;
   double t1, total;

   if(argc < 2){
      fprintf(stderr,"Falta tamaño de la matriz (debe de ser cuadrada)\n");
      exit(-1);
    }

    // Tamaño de la matriz pasado por parámetro
    unsigned int tam = atoi(argv[1]);

    if(tam < 2){
        fprintf(stderr, "El tamaño no puede ser menor a 2");
        exit(-1);
    }

    // Declaramos con memoria dinámica
    double **m1, **m2, **m3;

    m1 = (double**) malloc(tam*sizeof(double*));
    m2 = (double**) malloc(tam*sizeof(double*));
    m3 = (double**) malloc(tam*sizeof(double*));


    // Reservamos memoria ahora para las componentes de las matrices
    for(i=0; i<tam; i++){
        m1[i] = (double*) malloc(tam*sizeof(double));
        m2[i] = (double*) malloc(tam*sizeof(double));
        m3[i] = (double*) malloc(tam*sizeof(double));
    }
```

```
    // Inicializamos la matriz y los vectores
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            m1[i][j] = 0;
            m2[i][j] = 2;
            m3[i][j] = 2;
        }
    }

    // Inicializamos la primera variable de tiempo
    t1 = omp_get_wtime();

    // Calculamos el producto m1 = m2*m3
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j++){
            for(k=0; k<tam; k++){
                m1[i][j] += (m2[i][k]*m3[k][j]);
            }
        }
    }

    // Obtenemos el tiempo total transcurrido
    total = omp_get_wtime() - t1;

    // Para tamaños pequeños (hasta tam=11), imprimimos todas las componentes del vector resultante
    // y el tiempo de ejecución
    if(tam <= 11){
        printf("Tamaño matriz: %i\n Tiempo de ejecución: %f\n", tam, total);

        for(i=0; i<tam; i++){
            for(j=0; j<tam; j++){
                printf("m1[%i][%i] = %f\n", i, j, m1[i][j]);
            }
        }
    }
```

```
    // Para tamaños superiores, imprimimos el tiempo de ejecución y la primera y última componente del vector
    else{
        printf("Tamaño vectores: %i\n Tiempo de ejecución: %f\n Primera componente: %f\n Última componente: %f\n",
        tam, total, m1[0][0], m1[tam-1][tam-1]);
    }

    // Liberamos memoria
    for(i=0; i<tam; i++){
        free(m1[i]);
        free(m2[i]);
        free(m3[i]);
    }

    free(m1);
    free(m2);
    free(m3);

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrrollado de bucles de reserva de memoria e inicialización de matrices.

Modificación b) –explicación-: Desenrrollado de bucle del producto de matrices

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

```
    // Reservamos memoria ahora para las componentes de las matrices
    for(i=0; i<tam; i+=4){
        m1[i] = (double*) malloc(tam*sizeof(double));
        m1[i+1] = (double*) malloc(tam*sizeof(double));
        m1[i+2] = (double*) malloc(tam*sizeof(double));
        m1[i+3] = (double*) malloc(tam*sizeof(double));
        m2[i] = (double*) malloc(tam*sizeof(double));
        m2[i+1] = (double*) malloc(tam*sizeof(double));
        m2[i+2] = (double*) malloc(tam*sizeof(double));
        m2[i+3] = (double*) malloc(tam*sizeof(double));
        m3[i] = (double*) malloc(tam*sizeof(double));
        m3[i+1] = (double*) malloc(tam*sizeof(double));
        m3[i+2] = (double*) malloc(tam*sizeof(double));
        m3[i+3] = (double*) malloc(tam*sizeof(double));
    }

    // Inicializamos la matriz y los vectores
    for(i=0; i<tam; i++){
        for(j=0; j<tam; j+=4){
            m1[i][j] = 0;
            m1[i][j+1] = 0;
            m1[i][j+2] = 0;
            m1[i][j+3] = 0;
            m2[i][j] = 2;
            m2[i][j+1] = 2;
            m2[i][j+2] = 2;
            m2[i][j+3] = 2;
            m3[i][j] = 2;
            m3[i][j+1] = 2;
            m3[i][j+2] = 2;
            m3[i][j+3] = 2;
        }
    }
```

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

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$gcc -fopenmp -o pmm-secuencial-modificado_a pmm-secuencial-modificado_a.c
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$./pmm-secuencial-modificado_a 900
Tamaño vectores: 900
 Tiempo de ejecución: 6.748976
 Primera componente: 3600.000000
 Última componente: 3600.000000
```

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$gcc -fopenmp -o pmm-secuencial-modificado_b pmm-secuencial-modificado_b.c
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$./pmm-secuencial-modificado_b 900
Tamaño vectores: 900
 Tiempo de ejecución: 7.207586
 Primera componente: 3600.000000
 Última componente: 3600.000000
```

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

```
67      // Calculamos el producto m1 = m2*m3
68      for(i=0; i<tam; i++){
69          for(j=0; j<tam; j++){
70              for(k=0; k<tam; k+=4){
71                  m1[i][j] += (m2[i][k]*m3[k][j]);
72                  m1[i][j] += (m2[i][k+1]*m3[k+1][j]);
73                  m1[i][j] += (m2[i][k+2]*m3[k+2][j]);
74                  m1[i][j] += (m2[i][k+3]*m3[k+3][j]);
75              }
76          }
77      }
```

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar		*7.568746*
Modificación a)	Desenrrollado de los bucles de reserva de memoria y de inicialización de las matrices.	*6.748976*
Modificación b)	Desenrrollado del bucle del producto de las matrices.	*7.207586*

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO: En mi caso, aplicar la optimización a) hace que se reduzca el tiempo de ejecución consideramente. Con la optimización b) se mejora, pero en menor medida que la primera.

B) CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

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

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

int main(int argc, char** argv) {
    int i, ii, X1, X2;
    int R[40000];
    struct timespec cgt1, cgt2;
    double tiempo;

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

    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);
    tiempo = (double) (cgt2.tv_sec - cgt1.tv_sec)+ (double) ((cgt2.tv_nsec - cgt1.tv_nsec) / (1.e+9));
    printf("a: %i - b: %i", R[5000], s[4999].a);
    printf(" Tiempo(seg.): %11.9f\n", tiempo);
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Desenrrollado bucle de inicialización

Modificación b) –explicación-: Unión y desenrrollado de bucles internos

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1`-modificado_a.c`

```c
int main(int argc, char** argv) {
    int i, ii, X1, X2;
    int R[40000];
    struct timespec cgt1, cgt2;
    double tiempo;

    for(i=0; i<5000;i+=4)  {
        s[i].a = i;
        s[i+1].a = i+1;
        s[i+2].a = i+2;
        s[i+3].a = i+3;
        s[i].b = i;
        s[i+1].b = i+1;
        s[i+2].b = i+2;
        s[i+3].b = i+3;
    }
```

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

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$gcc -fopenmp -o figura1-modificado_a figura1-modificado_a.c
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$./figura1-modificado_a
a: 12492500 - b: 4999 Tiempo(seg.): 0.868834923
```

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

```
    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;
            X2+=3*s[i].b-ii;
            X2+=3*s[i+1].b-ii;
            X2+=3*s[i+2].b-ii;
            X2+=3*s[i+3].b-ii;
        }

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

```
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$gcc -fopenmp -o figura1-modificado_b figura1-modificado_b.c
[FranciscoDomínguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer1] 2019-
05-22 miércoles
$./figura1-modificado_b
a: 12492500 - b: 4999 Tiempo(seg.): 0.460817337
```

1.1. TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*0.866842420*
Modificación a)	Desenrrollado de los bucles de inicialización	*0.868834923*
Modificación b)	Unión y desenrrollado de los bucles internos	*0.460817337*

1.1. COMENTARIOS SOBRE LOS RESULTADOS Y JUSTIFICACIÓN DE LAS MEJORAS EN TIEMPO: En este caso, al aplicar la optimización b), se obtienen mejoras sustanciales de tiempos, casi en la mitad.

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>

int main(int argc,char **argv) {
   unsigned int N,i,j;
   double k = 33;

   struct timespec cgt1,cgt2;
   double ncgt; //para tiempo de ejecución

   if (argc<2) {
      printf("Faltan no componentes del vector\n");
      exit(-1);
   }
   N=atoi(argv[1]);
   double *A,*B,*C;

   A=(double *) malloc(N*sizeof(double));
   B=(double *) malloc(N*sizeof(double));
   C=(double *) malloc(N*sizeof(double));

   for(i=0; i<N; i++) {
      B[i]=2;
      C[i]=3;
   }

   clock_gettime(CLOCK_REALTIME,&cgt1);

   for (i=0; i<N; i++){
      A[i]= k*B[i] + C[i];
   }

   clock_gettime(CLOCK_REALTIME,&cgt2);
   ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));
```

```c
   //Imprimir resultado de la suma y el tiempo de ejecución
   printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
   printf("A[0] = %d // A[N-1] = %d.\n",A[0],A[N-1]);

   free(B);
   free(C);
   free(A);
}
```

Tiempos ejec.	-O0	-Os	-O2	-O3
	0.000448215	*0.000240967*	*0.000239256*	*0.000225882*

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

```
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$gcc -O0 -o daxpy daxpy.c
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$./daxpy 20000
Tiempo(seg.):0.000448215         / Tamaño Vectores:20000
A[0] = 69.000000 // A[N-1] = 69.000000.
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$gcc -Os -o daxpy daxpy.c
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$./daxpy 20000
Tiempo(seg.):0.000240967         / Tamaño Vectores:20000
A[0] = 69.000000 // A[N-1] = 69.000000.
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$gcc -O2 -o daxpy daxpy.c
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$./daxpy 20000
Tiempo(seg.):0.000239256         / Tamaño Vectores:20000
A[0] = 69.000000 // A[N-1] = 69.000000.
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$gcc -O3 -o daxpy daxpy.c
[FranciscoDominguezLorente d3vcho@d3vcho-PC:~/Escritorio/Uni/AC/bp4/ejer2] 2019-
05-22 miércoles
$./daxpy 20000
Tiempo(seg.):0.000225882         / Tamaño Vectores:20000
A[0] = 69.000000 // A[N-1] = 69.000000.
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

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)

daxpyO0.s

```
.L6:
        movl    -96(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -72(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm0
        mulsd   -88(%rbp), %xmm0
        movl    -96(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -64(%rbp), %rax
        addq    %rdx, %rax
        movsd   (%rax), %xmm1
        movl    -96(%rbp), %eax
        leaq    0(,%rax,8), %rdx
        movq    -80(%rbp), %rax
        addq    %rdx, %rax
        addsd   %xmm1, %xmm0
        movsd   %xmm0, (%rax)
        addl    $1, -96(%rbp)
```

daxpyOs.s

```
.L5:
        cmpl    %eax, %ebx
        jbe     .L11
        movsd   0(%r13,%rax,8), %xmm0
        mulsd   %xmm1, %xmm0
        addsd   (%r12,%rax,8), %xmm0
        movsd   %xmm0, 0(%rbp,%rax,8)
        incq    %rax
        jmp     .L5
```

daxpyO2.s

```
.L6:
        movsd   0(%rbp, %rax), %xmm0
        mulsd   %xmm1, %xmm0
        addsd   (%r12,%rax), %xmm0
        movsd   %xmm0, (%r15,%rax)
        addq    $8, %rax
        cmpq    %rax, %rbx
        jne     .L6
```

daxpyO3.s

```
.L10:
        movapd  (%r8,%rax), %xmm0
        addl    $1, %ecx
        mulpd   %xmm1, %xmm0
        movupd  (%rdx, %rax), %xmm2
        addpd   %xmm2, %xmm0
        movups  %xmm0, (%rsi,%rax)
        addq    $16, %rax
        cmpl    %r10d, %ecx
        jb      .L10
        movl    %r9d, %eax
        andl    $-2, %eax
        addl    %eax, %edi
        cmpl    %eax, %r9d
        je      .L14
```