2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Jose Teodosio Lorente Vallecillos
Grupo de prácticas y profesor de prácticas: A3 Mancia Anguita Lopez
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™ i7-4510U CPU @ 2.00GHz × 4

Sistema operativo utilizado: *Ubuntu 20.04.2 LTS*

Versión de gcc utilizada: *9.3.0*

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

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer1] 2021-05-31 lunes
$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):                              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:                              69
Nombre del modelo:                   Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
Revisión:                            1
CPU MHz:                             2879.680
CPU MHz máx.:                        3100,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5188.44
Virtualización:                      VT-x
Caché L1d:                           64 KiB
Caché L1i:                           64 KiB
Caché L2:                            512 KiB
Caché L3:                            4 MiB
CPU(s) del nodo NUMA 0:              0-3
Vulnerability Itlb multihit:         KVM: Mitigation: Split huge pages
Vulnerability L1tf:                  Mitigation; PTE Inversion; VMX conditional cache flushes, SMT vulner
able
Vulnerability Mds:                   Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown:              Mitigation; PTI
Vulnerability Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:            Mitigation; Full generic retpoline, IBPB conditional, IBRS_FW, STIBP
 conditio
                                     nal, RSB filling
Vulnerability Srbds:                 Mitigation; Microcode
Vulnerability Tsx async abort:       Not affected
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat ps
e36 clflu
                                     sh dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
 lm const
                                     ant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cp
uid aperf
                                     mperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma
 cx16 xtp
                                     r pdcm pcid sse4_1 sse4_2 movbe popcnt tsc_deadline_timer aes xsave
avx f16c
                                     rdrand lahf_lm abm cpuid_fault epb invpcid_single pti ssbd ibrs ibpb
 stibp tp
                                     r_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1
avx2 smep
                                      bmi2 erms invpcid xsaveopt dtherm ida arat pln pts md_clear flush_l
1d
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer1] 2021-05-31 lunes
```

1. **(a)** Implementar un código secuencial que calcule la multiplicación de dos matrices cuadradas. Utilizar como base el código de suma de vectores de BP0. Los datos se deben generar de forma aleatoria para un número de filas mayor que 8, como en el ejemplo de BP0, se puede usar `drand48()`).

   MULTIPLICACIÓN DE MATRICES:

   CAPTURA CÓDIGO FUENTE: `pmm-secuencial.c`

```c
int main(int argc, char** argv){

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

  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<2){
    printf("Faltan nº componentes del vector\n");
    exit(-1);
  }

  unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

  int **v1, **v2, **v3;
  v1 = (int**) malloc(N*sizeof(int*));// malloc necesita el tamaño en bytes
  v2 = (int**) malloc(N*sizeof(int*)); //si no hay espacio suficiente malloc devuelve NULL
  v3 = (int**) malloc(N*sizeof(int*));

  if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("No hay suficiente espacio para los vectores o matriz \n");
    exit(-2);
  }

  for (int i = 0; i < N; i++)
  {
    v3[i] = (int*) malloc(N*sizeof(int));
    v1[i] = (int*) malloc(N*sizeof(int));
    v2[i] = (int*) malloc(N*sizeof(int));

    if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
      printf("No hay suficiente espacio para la matriz \n");
      exit(-2);
    }
  }

  srand(time(NULL));
  for (int i = 0; i < N; i++)
  {
    for (int j = 0; j < N; j++)
    {
      v2[i][j] = j + 1;
      v1[i][j] = j + 2;
      v3[i][j] = 0;
    }
  }
```

```c
  clock_gettime(CLOCK_REALTIME,&cgt1);
  //Calcular suma de vectores
  for(int i=0; i<N; i++)
    for(int j=0; j<N; j++)
      for(int k=0; k<N; k++)
        v3[i][j] += v1[i][k] * v2[k][j];

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

  //Imprimir resultado de la suma y el tiempo de ejecución

  printf("Tiempo (seg.): %11.9f\n",ncgt);

  return 0;
}
```

(b) Modificar el código (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-: He realizado una fragmentación de el bucle de 4 en 4

Modificación B) –explicación-: He cambiado los valores de K y de J, realizando una inversión de el bucle, para estar mas cerca de las posiciones de memoria

...

CÓDIGOS FUENTE MODIFICACIONES

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

```c
int main(int argc, char const * argv[]){

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

  //Leer argumento de entrada (nº de componentes del vector)
  if (argc<2){
    fprintf(stderr,"Faltan nº componentes del vector\n");
    exit(-1);
  }

  unsigned int n = atoi(argv[1]); // Máximo N =2^32-1=4294967295 (sizeof(unsigned int) = 4 B)

  int **v1, **v2, **v3;
  v1 = (int**) malloc(n*sizeof(int*));// malloc necesita el tamaño en bytes
  v2 = (int**) malloc(n*sizeof(int*)); //si no hay espacio suficiente malloc devuelve NULL
  v3 = (int**) malloc(n*sizeof(int*));

  if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
    printf("No hay suficiente espacio para los vectores o matriz \n");
    exit(-2);
  }

  for (int i = 0; i < n; i++)
  {
    v3[i] = (int*) malloc(n*sizeof(int));
    v1[i] = (int*) malloc(n*sizeof(int));
    v2[i] = (int*) malloc(n*sizeof(int));

    if ( (v1==NULL) || (v2==NULL) || (v3==NULL) ){
      printf("No hay suficiente espacio para la matriz \n");
      exit(-2);
    }
  }

  srand(time(NULL));
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      v2[i][j] = j + 1;
      v1[i][j] = j + 2;
      v3[i][j] = 0;
    }
  }
```

```
  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+=4){
        v3[i][k]   += v1[i][j] * v2[j][k];
        v3[i][k+1] += v1[i][j] * v2[j][k+1];
        v3[i][k+2] += v1[i][j] * v2[j][k+2];
        v3[i][k+3] += v1[i][j] * v2[j][k+3];
      }
    }
  }

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

  //Imprimir resultado de la suma y el tiempo de ejecución

  printf("Tiempo (seg.): %11.9f\n",ncgt);
  printf("v3[%d] =  %d\n",0,v3[0][0]);
  printf("v3[%d] =  %d\n",n-1,v3[0][n-1]);

  return 0;
}
```

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

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer1] 2021-06-02 miércoles
$./pmm-secuencial 1000
Tiempo (seg.): 6.781225217
v3[0] =  501500
v3[999] =  501500000
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer1] 2021-06-02 miércoles
$./pmm-secuencial-modificado_A 1000
Tiempo (seg.): 1.157066537
v3[0] =  501500
v3[999] =  501500000
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer1] 2021-06-02 miércoles
$
```

B) ...

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*6.781225217*
Modificación A)	Desenrollar el bucle de 4 en 4	
Modificación B)	Intercambio de los valores k y j	1.157066537
...		

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

Como se puede observar la diferencia de tiempo de la +modificación final a la original es muy notable, ya que al hacer que el bucle realice menos vueltas tal que n/4, siendo n el tamaño, en la captura se

observa el ejemplo de 1000, por lo que, 1000/4=250, de esta manera nos ahorramos 750 vueltas.

2. **(a)** Usando como base el código de BP0, generar un programa para evaluar un código de la Figura 1. M y N deben ser parámetros de entrada al programa. Los datos se deben generar de forma aleatoria para valores de M y N mayores que 8, como en el ejemplo de BP0.

CÓDIGO FIGURA 1:

CAPTURA CÓDIGO FUENTE: `figura1-original.c`

```c
#include <stdlib.h> // biblioteca con funciones atoi(), rand(), srand(), malloc() y free()
#include <stdio.h> // biblioteca donde se encuentra la función printf()
#include <time.h>     // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define M 40000
#define N 5000

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

int main(int argc, char** argv){

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

  int ii,i,x1,x2;
  int R[M];

  srand(time(NULL));
  for(ii=0; ii<N; ii++){
    s[ii].a= rand() % 10;
    s[ii].b= rand() % 10;
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  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;
  }

  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 (seg.): %11.9f\n",ncgt);

  return 0;
}
```

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 {
        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;
   }
   …
}
```

(b) Modificar el código C (solo el trozo a evaluar) 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.

MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación A) –explicación-: Realizo una modificación a el struct para que sea lineal acceder a los datos, do forma que tenga un bucle para cada dato, A y B.

Modificación B) –explicación-: Realizo una sustitución de las multiplicaciones por operadores de desplazamiento, consiguiendo así que las multiplicaciones sean mas rápidas.

...

CÓDIGOS FUENTE MODIFICACIONES

A) Captura figura1-modificado_A.c

```c
#include <time.h>     // biblioteca donde se encuentra la función clock_gettime()
#include <omp.h>

#define M 4
#define N 5

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

int main(int argc, char** argv){

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

  int ii,i,x1,x2;
  int R[M];

  srand(time(NULL));
  for(ii=0; ii<N; ii++){
    s.a[ii]= rand() % 10;
  }
  for(ii=0; ii<N; ii++){
    s.b[ii]= rand() % 10;
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(ii=0; ii<M; ii++){
    x1=0;
    x2=0;
    for(i=0; i<N; i++)
      x1+=((s.a[i])<<1)+ii;
    for(i=0; i<N; i++)
      x2+=((s.b[i]<<1)+s.b[ii])-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 (seg.): %11.9f\n",ncgt);

  return 0;
}
```

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

```
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer2] 2021-06-03 jueves
$./figura1-original && ./figura1-modificado_A
Tiempo (seg.): 0.000000125
Tiempo (seg.): 0.000000063
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer2] 2021-06-03 jueves
$
```

B) ...

TIEMPOS:

Modificación	**Breve descripción de las modificaciones**	**-O2**
Sin modificar		*0.000000125*
Modificación A)	Modificación del struct	0.000000063
Modificación B)	Modificación de las operaciones * a <<	0.000000063

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

Como se puede observar en la captura del resultado final se reuce considerablemente, ya que nos ahorramos las multiplicaciones por operadores de desplazamiento, al realizarse secuencialmente priemro todos los valores de A y lugo todos los valores de B, ahorrando ciclos de reloj.

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];
```

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.

CAPTURA CÓDIGO FUENTE: `daxpy.c`

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

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

extern double *x,*y,a;
extern unsigned int N;

unsigned int N=0;
double a=0.0;
double *x;
double *y;

int main(int argc, char **argv){

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

  if(argc<3){
    fprintf(stderr,"Faltan recibir por parametro el tamaño y un numero \n");
    exit(-1);
  }

  N=(unsigned int)atoi(argv[1]);
  a=atof(argv[2]);

  x=(double*)malloc(N*sizeof(double));
  y=(double*)malloc(N*sizeof(double));

  for(int i=0; i<N; i++){
    x[i]=i+1;
    y[i]=i+2;
  }

  clock_gettime(CLOCK_REALTIME,&cgt1);

  for(int i=0; i<N; i++)
    y[i]=a*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 (seg.): %11.9f\n",ncgt);

  return 0;
}
```

Tiempos ejec. Longitud vectores=XXXX	**-O0**	**-Os**	**-O2**	**-O3**
	0.478891125	0.195234847	0.180594392	0.146786745

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

```
#!/bin/bash

gcc -O0 -fopenmp -o daxpy_O0 daxpy.c
echo "DAXPY O0:"
./daxpy_O0 100000000 2.25

gcc -Os -fopenmp -o daxpy_Os daxpy.c
echo "DAXPY Os:"
./daxpy_Os 100000000 2.25

gcc -O2 -fopenmp -o daxpy_O2 daxpy.c
echo "DAXPY O2:"
./daxpy_O2 100000000 2.25

gcc -O3 -fopenmp -o daxpy_O3 daxpy.c
echo "DAXPY O3:"
./daxpy_O3 100000000 2.25
```

```
joseteo@joseteo-X550LD: ~/bp4
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer3] 2021-06-03 jueves
$sh script.sh
DAXPY O0:
Tiempo (seg.): 0.478891125
DAXPY Os:
Tiempo (seg.): 0.195234847
DAXPY O2:
Tiempo (seg.): 0.180594392
DAXPY O3:
Tiempo (seg.): 0.146786745
[JoséTeodosioLorenteVallecillos joseteo@joseteo-X550LD:~/bp4/ejer3] 2021-06-03 jueves
$
```

COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

Una de las diferencias mas apreciables es la extensión de los códigos ensamblador O0, Os y O2 en comparación con O3 el cual tiene 300 lineas, y esto es debido al desenrollado de los bucles. Ademas se observa es que en el código ensamblador con la instrucción Os es que realiza primero la comprobación, si la cumple salta a L11, sin embargo si no, se realiza la operación , el incremento y un salto a L5 de nuevo para repetirlo todo, sin embargo con O0 realiza la comprobación después de hacer la operación y el incremento, lo curioso es que realiza la comprobación antes de calcular en la primera vuelta, es decir, entra en L3 y salta a L5, comprueba y luego salta a L6 y se repite L5 y L6 hasta que la condición se haya cumplido, en el caso de O2 sucede un caso similar, salvo que tiene 2 comprobaciones, una primera en L3 la cual comprueba con los valores inicializados, si se cumple salta a L5 y ejecuta el resto del código sin hacer bucle, pero si no entra en L6 calcula, y si no cumple la comprobación repite L6 hasta que se cumpla.

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)**:**

O0

```
108     call  clock_gettime@PLT
109     movl  $0, -60(%rbp)
110     jmp .L5
111   .L6:
112     movq  x(%rip), %rax
113     movl  -60(%rbp), %edx
114     movslq  %edx, %rdx
115     salq  $3, %rdx
116     addq  %rdx, %rax
117     movsd (%rax), %xmm1
118     movsd a(%rip), %xmm0
119     mulsd %xmm0, %xmm1
120     movq  y(%rip), %rax
121     movl  -60(%rbp), %edx
122     movslq  %edx, %rdx
123     salq  $3, %rdx
124     addq  %rdx, %rax
125     movsd (%rax), %xmm0
126     movq  y(%rip), %rax
127     movl  -60(%rbp), %edx
128     movslq  %edx, %rdx
129     salq  $3, %rdx
130     addq  %rdx, %rax
131     addsd %xmm1, %xmm0
132     movsd %xmm0, (%rax)
133     addl  $1, -60(%rbp)
134   .L5:
135     movl  -60(%rbp), %edx
136     movl  N(%rip), %eax
137     cmpl  %eax, %edx
138     jb  .L6
139     leaq  -32(%rbp), %rax
140     movq  %rax, %rsi
141     movl  $0, %edi
142     call  clock_gettime@PLT
```

Os

```
69      call  clock_gettime@PLT
70      movl  N(%rip), %ecx
71      movq  x(%rip), %rsi
72      xorl  %eax, %eax
73      movq  y(%rip), %rdx
74    .L5:
75      cmpl  %eax, %ecx
76      jbe .L11
77      movsd (%rsi,%rax,8), %xmm0
78      mulsd a(%rip), %xmm0
79      addsd (%rdx,%rax,8), %xmm0
80      movsd %xmm0, (%rdx,%rax,8)
81      incq  %rax
82      jmp .L5
83    .L11:
84      xorl  %edi, %edi
85      leaq  24(%rsp), %rsi
86      call  clock_gettime@PLT
```

O2

O3

```
75    call  clock_gettime@PLT
76    movl  N(%rip), %eax
77    testl %eax, %eax
78    je  .L5
79    movq  x(%rip), %rdi
80    movq  y(%rip), %rdx
81    leal  -1(%rax), %esi
82    xorl  %eax, %eax
83    .p2align 4,,10
84    .p2align 3
85  .L6:
86    movsd (%rdi,%rax,8), %xmm0
87    mulsd a(%rip), %xmm0
88    movq  %rax, %rcx
89    addsd (%rdx,%rax,8), %xmm0
90    movsd %xmm0, (%rdx,%rax,8)
91    addq  $1, %rax
92    cmpq  %rsi, %rcx
93    jne .L6
94  .L5:
95    xorl  %edi, %edi
96    leaq  16(%rsp), %rsi
97    call  clock_gettime@PLT
```

```
121   call  clock_gettime@PLT
122   movl  N(%rip), %edi
123   testl %edi, %edi
124   je  .L7
125   movq  x(%rip), %rcx
126   movq  y(%rip), %rdx
127   leal  -1(%rdi), %eax
128   leaq  15(%rcx), %rsi
129   subq  %rdx, %rsi
130   cmpq  $30, %rsi
131   seta  %r8b
132   cmpl  $1, %eax
133   seta  %sil
134   testb %sil, %r8b
135   je  .L8
136   movl  %edi, %esi
137   leaq  (%rdx,%rsi,8), %r8
138   leaq  a(%rip), %rsi
139   cmpq  %rsi, %r8
140   setbe %r8b
141   addq  $8, %rsi
142   cmpq  %rsi, %rdx
143   setnb %sil
144   orb %sil, %r8b
145   je  .L8
146   movsd a(%rip), %xmm1
147   movl  %edi, %esi
148   xorl  %eax, %eax
149   shrl  %esi
150   unpcklpd  %xmm1, %xmm1
151   salq  $4, %rsi
152   .p2align 4,,10
153   .p2align 3
154 .L9:
155   movupd  (%rcx,%rax), %xmm0
156   movupd  (%rdx,%rax), %xmm7
157   mulpd %xmm1, %xmm0
158   addpd %xmm7, %xmm0
159   movups  %xmm0, (%rdx,%rax)
160   addq  $16, %rax
161   cmpq  %rax, %rsi
162   jne .L9
163   movl  %edi, %eax
164   andl  $-2, %eax
165   andl  $1, %edi
166   je  .L7
167   cltq
168   movsd (%rcx,%rax,8), %xmm0
169   leaq  (%rdx,%rax,8), %rdx
170   mulsd a(%rip), %xmm0
171   addsd (%rdx), %xmm0
172   movsd %xmm0, (%rdx)
173 .L7:
174   xorl  %edi, %edi
175   leaq  16(%rsp), %rsi
176   call  clock_gettime@PLT
```

(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s

```
call clock_gettime@PLT
movl $0, -60(%rbp)
jmp .L5
.L6:
movq x(%rip), %rax
movl -60(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rdx, %rax
movsd (%rax), %xmm1
movsd a(%rip), %xmm0
mulsd %xmm0, %xmm1
movq y(%rip), %rax
movl -60(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rdx, %rax
movsd (%rax), %xmm0
movq y(%rip), %rax
movl -60(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rdx, %rax
addsd %xmm1, %xmm0
movsd %xmm0, (%rax)
addl $1, -60(%rbp)
.L5:
movl -60(%rbp), %edx
movl N(%rip), %eax
cmpl %eax, %edx
jb .L6
leaq -32(%rbp), %rax
movq %rax, %rsi
movl $0, %edi
call clock_gettime@PLT
```

daxpyOs.s

```
call clock_gettime@PLT
movl N(%rip), %eax
testl %eax, %eax
je .L5
movq x(%rip), %rdi
movq y(%rip), %rdx
leal -1(%rax), %esi
xorl %eax, %eax
.p2align 4,,10
.p2align 3
.L6:
movsd (%rdi,%rax,8),
%xmm0
mulsd a(%rip), %xmm0
movq %rax, %rcx
addsd (%rdx,%rax,8),
%xmm0
movsd %xmm0, (%rdx,
%rax,8)
addq $1, %rax
cmpq %rsi, %rcx
jne .L6
.L5:
xorl %edi, %edi
leaq 16(%rsp), %rsi
call clock_gettime@PLT
```

daxpyO2.s

```
call clock_gettime@PLT
movl N(%rip), %edi
testl %edi, %edi
je .L7
movq x(%rip), %rcx
movq y(%rip), %rdx
leal -1(%rdi), %eax
leaq 15(%rcx), %rsi
subq %rdx, %rsi
cmpq $30, %rsi
seta %r8b
cmpl $1, %eax
seta %sil
testb %sil, %r8b
je .L8
movl %edi, %esi
leaq (%rdx,%rsi,8), %r8
leaq a(%rip), %rsi
cmpq %rsi, %r8
setbe %r8b
addq $8, %rsi
cmpq %rsi, %rdx
setnb %sil
orb %sil, %r8b
je .L8
movsd a(%rip), %xmm1
movl %edi, %esi
xorl %eax, %eax
shrl %esi
unpcklpd %xmm1, %xmm1
salq $4, %rsi
.p2align 4,,10
.p2align 3
.L9:
movupd (%rcx,%rax), %xmm0
movupd (%rdx,%rax), %xmm7
mulpd %xmm1, %xmm0
addpd %xmm7, %xmm0
movups %xmm0, (%rdx,%rax)
addq $16, %rax
cmpq %rax, %rsi
jne .L9
movl %edi, %eax
andl $-2, %eax
andl $1, %edi
```

daxpyO3.s

```
call clock_gettime@PLT
movl N(%rip), %ecx
movq x(%rip), %rsi
xorl %eax, %eax
movq y(%rip), %rdx
.L5:
cmpl %eax, %ecx
jbe .L11
movsd (%rsi,%rax,8), %xmm0
mulsd a(%rip), %xmm0
addsd (%rdx,%rax,8), %xmm0
movsd %xmm0, (%rdx,%rax,8)
incq %rax
jmp .L5
.L11:
xorl %edi, %edi
leaq 24(%rsp), %rsi
call clock_gettime@PLT
```

		je .L7
		cltq
		movsd (%rcx,%rax,8), %xmm0
		leaq (%rdx,%rax,8), %rdx
		mulsd a(%rip), %xmm0
		addsd (%rdx), %xmm0
		movsd %xmm0, (%rdx)
		.L7:
		xorl %edi, %edi
		leaq 16(%rsp), %rsi
		call clock_gettime@PLT	