

2º curso / 2º cuatr.
Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.

Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Clara María Romero Lara

Grupo de prácticas y profesor de prácticas: D1, Niceto Luque

Fecha de entrega: 12-05-2020

Fecha evaluación en clase: 22-05-2020

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) i5-8250U CPU @ 1.60GHz

Sistema operativo utilizado: Arch Linux 5.5.13-arch2-1

Versión de gcc utilizada: gcc (Arch Linux 9.3.0-1) 9.3.0

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

```
+ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:          39 bits physical, 48 bits virtual
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):              1
NUMA node(s):          1
Vendor ID:              GenuineIntel
CPU family:             6
Model:                 142
Model name:             Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
Stepping:              10
CPU MHz:               2635.982
CPU max MHz:           3400.0000
CPU min MHz:           400.0000
BogoMIPS:              3601.00
Virtualization:        VT-x
L1d cache:             128 KiB
L1i cache:             128 KiB
L2 cache:              1 MiB
L3 cache:              6 MiB
NUMA node0 CPU(s):    0-7
Vulnerability Itlb multihit: KVM: Mitigation: Split huge pages
Vulnerability Lltf:      Mitigation; PTE Inversion; VMX conditional cache flushes, SMT vulnerable
Vulnerability Mds:       Vulnerable: Clear CPU buffers attempted, no microcode; 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 conditional, RSB filling
Vulnerability Tsx async abort: Not affected
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr s
```

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

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

#define MAX 1024
int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

int main(int argc, char** argv){
    struct timespec cgt1, cgt2; //Medicion de tiempo
    double ncgt;
    if(argc < 2) {
        fprintf(stderr, "Falta el tamaño de la matriz\n");
        exit(-1);
    }
    unsigned int n = atoi(argv[1]);
    int i, j;
    if(n > MAX){
        n = MAX;
    }
}
```

```

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            A[i][j] = n + (2*j) - i + 1;
            B[i][j] = n + (2*j) - i + 1;
            C[i][j] = 0;
        }
    }

    if(n <= 11){
        printf("\nMatriz A:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", A[i][j]);
            }
            printf("\n");
        }

        printf("\nMatriz B:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", B[i][j]);
            }
            printf("\n");
        }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            for(int k = 0; k < n; k++){
                C[i][j] += A[i][k] * B[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));

    printf("\nTiempo matriz tamaño %d: %11.9f", n, ncgt);
    if(n <= 11){
        printf("\nMatriz resultado:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", C[i][j]);
            }
            printf("\n");
        }
    }
    return 0;
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):**Modificación a) –explicación-: Desenrollado de bucle: 4 operaciones por bucle****Modificación b) –explicación-: Trasponer la matriz B para aprovechar la localidad de los accesos****1.1. CÓDIGOS FUENTE MODIFICACIONES****a) Captura de pmm-secuencial-modificado_a.c**

```

#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX 1024
int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

int main(int argc, char** argv){
    struct timespec cgt1, cgt2;                //Medicion de tiempo
    double ncgt;

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

    unsigned int n = atoi(argv[1]);
    int i, j;

    if(n > MAX){
        n = MAX;
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            A[i][j] = n + (2*j) - i + 1;
            B[i][j] = n + (2*j) - i + 1;
            C[i][j] = 0;
        }
    }

    if(n <= 11){
        printf("\nMatriz A:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", A[i][j]);
            }
            printf("\n");
        }

        printf("\nMatriz B:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){

```

```

        printf(" %d ", B[i][j]);
    }
    printf("\n");
}

clock_gettime(CLOCK_REALTIME,&cgt1);
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        for(int k = 0; k < n; k += 4){
            C[i][j] += A[i][k] * B[k][j] + A[i][k+1] * B[k+1]
[j] + A[i][k+2] * B[k+2][j] + A[i][k+3] * B[k+3][j];
        }
    }
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
cgt1.tv_nsec) / (1.e+9));

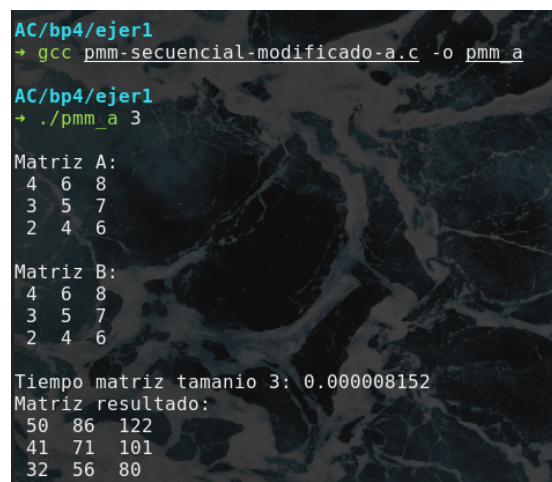
printf("\nTiempo matriz tamaño %d: %11.9f", n, ncgt);

if(n <= 11){
    printf("\nMatriz resultado:\n");
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf(" %d ", C[i][j]);
        }
        printf("\n");
    }
}

return 0;
}

```

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



```

AC/bp4/ejer1
→ gcc pmm-secuencial-modificado-a.c -o pmm_a

AC/bp4/ejer1
→ ./pmm_a 3

Matriz A:
4 6 8
3 5 7
2 4 6

Matriz B:
4 6 8
3 5 7
2 4 6

Tiempo matriz tamaño 3: 0.000008152
Matriz resultado:
50 86 122
41 71 101
32 56 80

```

b) Captura de pmm-secuencial-modificado_b.c

```

#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX 1024
double A[MAX][MAX], B[MAX][MAX], B_T[MAX][MAX], C[MAX][MAX];

int main(int argc, char** argv){
    struct timespec cgt1, cgt2;           //Medicion de tiempo
    double ncgt;

    if(argc < 2) {
        fprintf(stderr, "Falta el tamaño de la matriz\n");
        exit(-1);
    }
    unsigned int n = atoi(argv[1]);
    int i, j;
    if(n > MAX){
        n = MAX;
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            A[i][j] = n + (2*j) - i + 1;
            B[i][j] = n + (2*j) - i + 1;
            C[i][j] = 0;
        }
    }

    if(n <= 11){
        printf("\nMatriz A:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", A[i][j]);
            }
            printf("\n");
        }

        printf("\nMatriz B:\n");
        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                printf(" %d ", B[i][j]);
            }
            printf("\n");
        }
    }
}

```

```

clock_gettime(CLOCK_REALTIME,&cgt1);
for(i = 0; i < n; i++){
    for(j = 0; j < n; j++){
        B_T[i][j] = B[j][i];
    }
}


for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        for(int k = 0; k < n; k++){
            C[i][j] += A[i][k] * B[j][k];
        }
    }
}
clock_gettime(CLOCK_REALTIME,&cgt2);
ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec) + (double) ((cgt2.tv_nsec-
cgt1.tv_nsec) / (1.e+9));

printf("\nTiempo matriz tamaño %d: %11.9f", n, ncgt);

if(n <= 11){
    printf("\nMatriz resultado:\n");
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf(" %d ", C[i][j]);
        }
        printf("\n");
    }
}
return 0;
}

```

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



```

AC/bp4/ejer1
→ gcc pmm-secuencial-modificado-b.c -o pmm_b

AC/bp4/ejer1
→ ./pmm_b 3

Matriz A:
4 6 8
3 5 7
2 4 6

Matriz B:
4 6 8
3 5 7
2 4 6

Tiempo matriz tamaño 3: 0.000040352
Matriz resultado:
50 86 122
41 71 101
32 56 80

```

1.1. TIEMPOS: tamaño matriz 1000 para poder apreciar las diferencias

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	/	4.027477799
Modificación a)	Desenrollado de bucle	3.949253156
Modificación b)	Trasposición de la matriz B	0.623872194

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

La modificación b es notablemente mejor que la a y la no modificada. Esto se debe a que, tan solo con transponer la matriz b y por como se realiza la multiplicación de matrices aprovechamos mucho mejor la localidad de los accesos, mientras que la primera modificación nos ahorramos solamente un bucle.

B) CÓDIGO FIGURA 1:**CAPTURA CÓDIGO FUENTE:** figura1-original.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM_S 5000
#define TAM_R 40000

struct{
    int a;
    int b;
} s[TAM_S];
int R[TAM_R];

int main(){
    int i, ii, X1, X2;

    struct timespec cgt1, cgt2;           //Medicion de tiempo
    double ncgt;

    for (i = 0; i < TAM_S; i++ ) {
        s[i].a = i;
        s[i].b = 2*i;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for ( ii = 0; ii < TAM_R; ii++ ) {
        X1 = 0;
        X2 = 0;
        for(i=0; i < TAM_S;i++) X1+=2*s[i].a+ii;
        for(i=0; i < TAM_S;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("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
    printf("Tiempo: %11.9f\n", ncgt);

    return 0;
}

```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: reducción de bucles y operador ternario

Modificación b) –explicación-: desenrollado de bucle y operador ternario

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) Captura figura1-modificado_a.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM_S 5000
#define TAM_R 40000

struct{
    int a;
    int b;
} s[TAM_S];
int R[TAM_R];

int main(){
    int i, ii, X1, X2;
    int sumatorio;

    struct timespec cgt1, cgt2; //Medicion de tiempo
    double ncgt;

    for (i = 0; i < TAM_S; i++ ) {
        s[i].a = i;
        s[i].b = 2*i;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for ( ii = 0; ii < TAM_R; ii++ ) {
        X1 = 0;
        X2 = 0;

        for ( i = 0; i < TAM_S; i++ ) {
            X1 += s[i].a;
            X2 += s[i].b;
        }
        sumatorio = TAM_S * ii;
    }
}

```

```

        X1 = (X1*2) + sumatorio;
        X2 = (X2*3) - sumatorio;

        R[ii] = X1 < X2 ? X1 : 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("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
    printf("Tiempo: %11.9f\n", ncgt);

    return 0;
}

```

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



```

AC/bp4/ejer1
→ gcc figura1-modificado-a.c -o fig1_a

AC/bp4/ejer1
→ ./fig1_a

Vector R:
R[0]=24995000 R[39999]=-125010000
Tiempo: 0.474866935

```

b) Captura de figura1-modificado_b.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define TAM_S 5000
#define TAM_R 40000

struct{
    int a;
    int b;
} s[TAM_S];
int R[TAM_R];

int main(){
    int i, ii, X1, X2;

    struct timespec cgt1, cgt2; //Medicion de tiempo
    double ncgt;

```

```

    for (i = 0; i < TAM_S; i++ ) {
        s[i].a = i;
        s[i].b = 2*i;
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    for ( ii = 0; ii < TAM_R; ii++ ) {
        X1 = 0;
        X2 = 0;

        for ( i = 0; i < TAM_S; i += 5 ) {
            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;
            X1 += 2*s[i+4].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;
            X2 += 3*s[i+4].b - ii;
        }
        R[ii] = X1 < X2 ? X1 : 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("\nVector R: \n");
    printf("R[0]=%d R[39999]=%d\n", R[0], R[TAM_R-1]);
    printf("Tiempo: %11.9f\n", ncgt);

    return 0;
}

```

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



```

AC/bp4/ejer1
→ gcc figural-modificado-b.c -o fig1_b

AC/bp4/ejer1
→ ./fig1_b

Vector R:
R[0]=24995000 R[39999]=-125010000
Tiempo: 0.417505936

```

1.1. TIEMPOS:

Modificación	Breve descripción de las modificaciones	-O2
Sin modificar	/	0.230489733
Modificación a)	Reducción de bucles y operador ternario	0.097326707
Modificación b)	Desenrollado de bucle y operador ternario	0.112365086

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

En este caso la mejor opción es la a: en ambas modificaciones juntamos todo dentro de un sólo bucle for interno, pero funciona mejor llamar al struct el menor número de veces posible acumulando los resultados que ahorrarnos bucles con multiples llamadas dentro.

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 (*Double precision- real Alpha X Plus 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 acarreen. 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

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

int main (int argc, char **argv){
    struct timespec cgt1, cgt2;           //Medicion de tiempo
    double ncgt;
    const int A = 150;                   //Constante arbitraria

    if ( argc < 2 ) {
        fprintf(stderr,"Introduzca un num\n");
        exit(-1);
    }

    int n = atoi(argv[1]);
    int x[n], y[n];

    for (int i = 0; i <= n; i++){
        x[i] = i;
        y[i] = 2*i;
```

```

    }

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (int i = 1; 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("\ny[0]: %d, y[%d]: %d\n", y[0], n, y[n]);
    printf("Tiempo: %11.9f\n", ncgt);

    return 0;
}

```

	-O0	-Os	-O2	-O3
Tiempos ejec.	0.000002823	0.000000675	0.000003571	0.000000385

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



COMENTARIOS QUE EXPLIQUEN LAS DIFERENCIAS EN ENSAMBLADOR:

- O0: sin optimización alguna.
- Os: se reduce el tamaño del archivo respecto a O2. Bastante parecido a O2.
- O2: se reduce el tamaño del archivo respecto a O0. Se emplean instrucciones más eficientes que las usadas en O0.
- O3: el tamaño del archivo aumenta. Se genera un código más complejo con más llamadas a subrutinas.

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
<pre> .file "daxpy.c" .text .section .rodata .LC0: .string "Introduzca un num\n" .LC2: .string "\ny[0]: %d, y[%d]: %d\n" .LC3: .string "Tiempo: %11.9f\n" .text .globl main .type main, @function main: .LFB6: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %r15 pushq %r14 pushq %r13 pushq %r12 pushq %rbx subq \$168, %rsp .cfi_offset 15, -24 .cfi_offset 14, -32 .cfi_offset 13, -40 .cfi_offset 12, -48 .cfi_offset 3, -56 movl %edi, -164(%rbp) movq %rsi, -176(%rbp) movq %fs:40, %rax movq %rax, -56(%rbp) </pre>	<pre> .file "daxpy.c" .text .section .rodata.str1 .LC0: .string "Introduzca un num\n" .LC2: .string "\ny[0]: %d, y[%d]: %d\n" .LC3: .string "Tiempo: %11.9f\n" .section .text.startup p,"ax",@progbits .globl main .type main, @function main: .LFB6: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %r15 pushq %r14 pushq %r13 pushq %r12 pushq %rbx subq \$72, %rsp .cfi_offset 15, -24 .cfi_offset 14, -32 .cfi_offset 13, -40 .cfi_offset 12, -48 .cfi_offset 3, -56 movq %fs:40, %rax movq %rax, -56(%rbp) xorl %eax, %eax decl </pre>	<pre> .file "daxpy.c" .text .section .rodata.str1 .LC0: .string "Introduzca un num\n" .LC2: .string "\ny[0]: %d, y[%d]: %d\n" .LC3: .string "Tiempo: %11.9f\n" .section .text.startup p,"ax",@progbits .p2align 4 .globl main .type main, @function main: .LFB22: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %r15 pushq %r14 pushq %r13 pushq %r12 pushq %rbx subq \$72, %rsp .cfi_offset 15, -24 .cfi_offset 14, -32 .cfi_offset 13, -40 .cfi_offset 12, -48 .cfi_offset 3, -56 movq %fs:40, %rax movq %rax, -56(%rbp) xorl %eax, %eax cmpl </pre>	<pre> .file "daxpy.c" .text .section .rodata.str1 .LC1: .string "Introduzca un num\n" .LC4: .string "\ny[0]: %d, y[%d]: %d\n" .LC5: .string "Tiempo: %11.9f\n" .section .text.startup p,"ax",@progbits .p2align 4 .globl main .type main, @function main: .LFB22: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp .cfi_def_cfa_register 6 pushq %r15 pushq %r14 pushq %r13 pushq %r12 pushq %rbx subq \$72, %rsp .cfi_offset 15, -24 .cfi_offset 14, -32 .cfi_offset 13, -40 .cfi_offset 12, -48 .cfi_offset 3, -56 movq %fs:40, %rax movq %rax, -56(%rbp) xorl %eax, %eax </pre>

152(%rbp)	xorl %eax, %eax movq %rsp, %rax movq %rax, %rbx movl \$150, -	%edi jg .L2 movq	\$1, %edi jle .L14 movq 8(%rsi), %rdi movl \$10, %edx xorl %esi, %esi call strtol@PLT movslq %eax, %r12 movq %rax, %r15 leaq 15(,%r12,4),	%rdi	cmpl \$1, %edi jle .L22 movq 8(%rsi), movl \$10, %edx xorl %esi, %esi call strtol@PLT movslq %eax, %r12 movq %rax, %r15 movq leaq 15(,%r12,4),
164(%rbp)	cmpl \$1, -	call fputs@PLT orl \$-1, %edi call exit@PLT	shrq \$4, %rax salq \$4, %rax subq %rax, %rsp movq %rsp, %r14 subq %rax, %rsp leaq 3(%rsp), %rbx movq %rbx, %r13 andq \$-4, %rbx shrq \$2, %r13 testl %r15d, %r15d js .L3 movl %r15d, %edx xorl %eax, %eax .p2align 4,,10 .p2align 3	%rax	shrq \$4, %rax salq \$4, %rax subq %rax, %rsp movq %rsp, %r14 subq %rax, %rsp leaq 3(%rsp), movq %rbx, %r13 andq \$-4, %rbx shrq \$2, %r13 testl %r15d, %r15d js .L3 leal 1(%r15),
stderr(%rip), %rax	movq %rax, %rcx movl \$18, %edx movl \$1, %esi leaq .LC0(%rip),	movq 8(%rsi),	%rax	%rax	shrq \$4, %rax salq \$4, %rax subq %rax, %rsp movq %rsp, %r14 subq %rax, %rsp leaq 3(%rsp), %rbx movq %rbx, %r13 andq \$-4, %rbx shrq \$2, %r13 testl %r15d, %r15d js .L3 movl %r15d, %edx xorl %eax, %eax .p2align 4,,10 .p2align 3
%rdi	call fwrite@PLT movl \$-1, %edi call exit@PLT	movq %r12, %r14 shrq \$4, %rax salq \$4, %rax subq %rax, %rsp movq %rsp, %r15 subq %rax, %rsp xorl %eax, %eax leaq 3(%rsp),	%rbx	%rbx	movq %rbx, %r13 andq \$-4, %rbx shrq \$2, %r13 testl %r15d, %r15d js .L3 leal 1(%r15),
.L2:	movq -176(%rbp),	movq %rbx, %r13 andq \$-4, %rbx shrq \$2, %r13	.L4:	.L4:	cmpl \$2, %r15d jbe .L15 movl %ecx, %edx movdqa .LC0(%rip),
%rax	addq \$8, %rax movq (%rax), %rax movq %rax, %rdi call atoi@PLT movl %eax, -	cmpl %eax, %r14d jl .L10 leal (%rax,%rax),	(%r14,%rax,4)	(%r14,%rax)	movdqa .LC2(%rip), xorl %eax, %eax shrl \$2, %edx salq \$4, %rdx .p2align
140(%rbp)	movl -140(%rbp),	movq %rdx, -	%ecx	%ecx	movdqa .LC2(%rip), xorl %eax, %eax shrl \$2, %edx salq \$4, %rdx .p2align
%eax	movslq %eax, %rdx subq \$1, %rdx movq %rdx, -	movl %edx, (%rbx,	%xmm1	%xmm1	movdqa .LC2(%rip), xorl %eax, %eax shrl \$2, %edx salq \$4, %rdx .p2align
136(%rbp)	movslq %eax, %rdx movq %rdx, -	incq %rax jmp .L3	%xmm2	%xmm2	movdqa .LC2(%rip), xorl %eax, %eax shrl \$2, %edx salq \$4, %rdx .p2align
192(%rbp)	movq \$0, -	leaq -88(%rbp),	4,,10	.L5:	movdqa %xmm1, %xmm0 padd %xmm2, %xmm1 movups %xmm0,
184(%rbp)	movslq %eax, %rdx movq %rdx, -	xorl %edi, %edi call	(%r14,%rax)	(%r14,%rax)	pslld \$1, %xmm0 movups %xmm0,
208(%rbp)	movq \$0, -	xorl %eax, %eax	clock_gettime@PLT	clock_gettime@PLT	pslld \$1, %xmm0 movups %xmm0,
200(%rbp)	cltq leaq 0(,%rax,4),	clock_gettime@PLT	testl %r15d, %r15d jle .L7 leal -1(%r15), %edx movl	testl %r15d, %r15d jle .L7 leal -1(%r15), %edx movl	pslld \$1, %xmm0 movups %xmm0,
%rdx	movl \$16, %eax	.L5:			

%rax	subq \$1, %rax addq %rdx, %rax movl \$16, %esi movl \$0, %edx divq %rsi imulq \$16, %rax,	incq %rax cmpl %eax, %r14d jl .L11 imull \$150, (%r15,%rax,4), %edx addl %edx, (%rbx, %rax,4)	.L6: imull \$150, (%r14,%rax,4), %ecx addl %ecx, (%rbx, %rax,4)	(%rbx,%rax) addq \$16, %rax cmpq %rdx, %rax jne .L5 movl %ecx, %eax andl \$-4, %eax andl \$3, %ecx je .L6
	subq %rax, %rsp movq %rsp, %rax addq \$3, %rax shrq \$2, %rax salq \$2, %rax movq %rax, - 128(%rbp)	.L11: xorl %edi, %edi leaq -72(%rbp), %rsi call clock_gettime@PLT movq -64(%rbp), %rax	addq \$1, %rax cmpq %rax, %rdx jne .L6 .L7: xorl %edi, %edi leaq -80(%rbp), %rsi call clock_gettime@PLT movq -72(%rbp), %rax	.L4: movslq %eax, %rdx leal (%rax,%rax), %ecx movl %eax, (%r14,%rdx,4) movl %ecx, (%rbx, %rdx,4)
%eax	movl -140(%rbp), movslq %eax, %rdx subq \$1, %rdx movq %rdx, - 120(%rbp)	subq -80(%rbp), %rax movl %r14d, %edx cvtsi2sdq %rax, %xmm0 movq -72(%rbp), %rax	pxor %xmm0, %xmm0 movl %r15d, %edx subq -88(%rbp), %rax	leal 1(%rax), %edx cmpl %edx, %r15d jl .L6 movslq %edx, %rcx addl \$2, %eax movl %edx, (%r14,%rcx,4)
%rdx	movslq %eax, %rdx movq %rdx, %r14 movl \$0, %r15d movslq %eax, %rdx movq %rdx, %r12 movl \$0, %r13d cltq leaq 0(,%rax,4), movl \$16, %eax subq \$1, %rax addq %rdx, %rax movl \$16, %esi movl \$0, %edx divq %rsi imulq \$16, %rax,	subq -88(%rbp), %rax divsd .LC1(%rip), %xmm0 cvtsi2sdq %rax, %xmm1 movl (%rbx, %r12,4), %ecx leaq .LC2(%rip), %rdi xorl %eax, %eax movl 0(,%r13,4), %esi addsd %xmm1, %xmm0 movsd %xmm0, - 104(%rbp) call printf@PLT movsd -104(%rbp), %xmm0 leaq .LC3(%rip), %rdi movb \$1, %al call printf@PLT movq -56(%rbp), %rax	pxor %xmm1, %xmm1 movl (%rbx,%r12,4), %ecx leaq .LC2(%rip), %rdi cvtsi2sdq %rax, %xmm0 movq -80(%rbp), %rax subq -96(%rbp), %rax divsd .LC1(%rip), %xmm0 cvtsi2sdq %rax, %xmm1 movl 0(,%r13,4), %esi xorl %eax, %eax addsd %xmm1, %xmm0 movsd %xmm0, - 104(%rbp) call printf@PLT movsd -104(%rbp), %xmm0 movl \$1, %eax leaq .LC3(%rip), %rdi call	addl %edx, %rcx addl \$2, %eax movl %edx, (%r14,%rcx,4) addl %edx, %edx cmpl %eax, %r15d jl .L7 movslq %eax, %rdx movl %eax, (%r14,%rdx,4) addl %eax, %eax movl %eax, (%rbx, %rdx,4) .L7: leaq -96(%rbp), %rsi xorl %edi, %edi call clock_gettime@PLT .L8: leal -1(%r15), %eax cmpl \$2, %eax jbe .L16 movl %r15d, %edx
%rax	subq %rax, %rsp movq %rsp, %rax addq \$3, %rax shrq \$2, %rax salq \$2, %rax movq %rax, - 120(%rbp)	xorq %fs:40, %rax je .L7	call	

112(%rbp)	movl \$0, -	call __stack_chk_fail@PLT	printf@PLT movq -56(%rbp),	movl \$4, %eax shrl \$2, %edx salq \$4, %rdx addq \$4, %rdx .p2align
144(%rbp)	jmp .L3	.L7:	%rax xorq %fs:40, %rax jne .L15 leaq -40(%rbp),	4,,10 .p2align 3
.L4:	movq -128(%rbp),	%rsp xorl %eax, %eax popq %rbx popq %r12 popq %r13 popq %r14 popq %r15 popq %rbp .cfi_def_cfa	%rsp xorl %eax, %eax popq %rbx popq %r12 popq %r13 popq %r14 popq %r15 popq %rbp .cfi_restore_s	.L11:
%rax	movl -144(%rbp),	ret .cfi_endproc	state 7, 8	%xmm1 movdqu (%r14,%rax),
%edx	movslq %edx, %rdx movl -144(%rbp),	.LFE6:	.size main, .-main .section .rodata.cst8	%xmm3 movdqu (%r14,%rax),
%ecx	movl %ecx, (%rax,	, "aM", @progbits, 8	.align 8	%xmm4 movdqu (%rbx,%rax),
%rdx, 4)	movl -144(%rbp),	.LC1:	.long 0 .long 1104006501 .ident "GCC: (Arch	pslld \$2, %xmm1 padd %xmm3, %xmm1 movdqa %xmm1, %xmm0 pslld \$4, %xmm0 psubd %xmm1, %xmm0 pslld \$1, %xmm0 padd %xmm4, %xmm0 movups %xmm0,
%eax	leal (%rax,%rax),	Linux 9.3.0-1) 9.3.0"	.section .note.GNU-	addq \$16, %rax cmpq %rdx, %rax jne .L11 movl %r15d, %edx andl \$-4, %edx leal 1(%rdx),
%ecx	movq -112(%rbp),	stack,"", @progbits	clock_gettime@PLT jmp .L7	%eax cmpl %edx, %r15d je .L9
%rax	movl -144(%rbp),		.L3:	.L10:
%edx	movslq %edx, %rdx movl %ecx, (%rax,		tate %rsi xorl %edi, %edi call	movslq %eax, %rdx imull \$150,
%rdx, 4)	addl \$1, -		__stack_chk_fail@PLT .L14:	(%r14,%rdx, 4), %ecx addl %ecx, (%rbx,
144(%rbp)	movl -144(%rbp),		movq stderr(%rip),	leal 1(%rdx),
.L3:	cmpl -140(%rbp),		%rcx movl \$18, %edx movl \$1, %esi leaq .LC0(%rip),	cmpl %edx, %r15d je .L9
%eax	jle .L4 leaq -96(%rbp),		%rdi call fwrite@PLT orl \$-1, %edi call exit@PLT .cfi_endproc	.L10:
%rax	movq %rax, %rsi movl \$0, %edi call		.LFE22:	%edx cmpl %r15d, %edx jg .L9 movslq %edx, %rdx addl \$2, %eax imull \$150,
clock_gettime@PLT	movl \$1, -		.size main, .-main .section .rodata.cst8,"	(%r14,%rdx, 4), %ecx addl %ecx, (%rbx,
148(%rbp)	jmp .L5		.align 8	leal 1(%rax),
.L6:	movq -128(%rbp),		.LC1:	%edx cmpl %r15d, %edx jg .L9 movslq %edx, %rdx addl \$2, %eax imull \$150,
%rax	movl -148(%rbp),		.long 0 .long 1104006501 .ident "GCC: (Arch	(%r14,%rdx, 4), %ecx addl
%edx	movslq %edx, %rdx movl (%rax,			
%rdx, 4), %eax	imull -152(%rbp),			

%eax		Linux 9.3.0-1) 9.3.0" .section .note.GNU- stack,"",@progbits	%rdx,4) %r14,%rax,4), %edx %rax,4) .L9: %rsi clock_gettime@PLT %rax %rax %r12,4), %ecx %rdi %rax %rax %xmm0 %esi 104(%rbp) %xmm0 %rdi %rax	%ecx, (%rbx, cmpl %r15d, %eax jg .L9 cltq imull \$150, addl %edx, (%rbx, xorl %edi, %edi leaq -80(%rbp), call movq -72(%rbp), pxor %xmm0, %xmm0 movl %r15d, %edx subq -88(%rbp), pxor %xmm1, %xmm1 movl (%rbx, leaq .LC4(%rip), cvtsi2sdq %rax, %xmm0 movq -80(%rbp), subq -96(%rbp), divsd .LC3(%rip), cvtsi2sdq %rax, %xmm1 movl 0(,%r13,4), xorl %eax, %eax addsd %xmm1, %xmm0 movsd %xmm0, - call printf@PLT movsd -104(%rbp), movl \$1, %eax leaq .LC5(%rip), call printf@PLT movq -56(%rbp), xorq
------	--	---	---	--

			<pre> %fs:40, %rax jne .L23 leaq -40(%rbp), %rsp xorl %eax, %eax popq %rbx popq %r12 popq %r13 popq %r14 popq %r15 popq %rbp .cfi_remembe r_state .cfi_def_cfa 7, 8 ret .L6: .cfi_restore _state xorl %edi, %edi leaq -96(%rbp), %rsi call clock_gettime@PLT testl %r15d, %r15d jg .L8 jmp .L9 .L3: leaq -96(%rbp), %rsi xorl %edi, %edi call clock_gettime@PLT jmp .L9 .L16: movl \$1, %eax jmp .L10 .L15: xorl %eax, %eax jmp .L4 .L23: call __stack_chk_fail@PLT .L22: movq stderr(%rip), %rcx movl \$18, %edx movl \$1, %esi leaq .LC1(%rip), %rdi call </pre>
--	--	--	---

			<pre> fwrite@PLT orl \$-1, %edi call exit@PLT .cfi_endproc .LFE22: .size main, .-main .section .rodata.cst1 6,"aM",@progbits,16 .align 16 .LC0: .long 0 .long 1 .long 2 .long 3 .align 16 .LC2: .long 4 .long 4 .long 4 .long 4 .section .rodata.cst8 ,"aM",@progbits,8 .align 8 .LC3: .long 0 .long 1104006501 .ident "GCC: (Arch Linux 9.3.0-1) 9.3.0" .section .note.GNU- stack,"",@progbits </pre>
--	--	--	--

	movl %eax, %ecx movq -112(%rbp), %rax			
	movl -148(%rbp), %edx			
	movslq %edx, %rdx movl (%rax, %rdx, 4), %eax			
	addl %eax, %ecx movq -112(%rbp), %rax			
	movl -148(%rbp), %edx			
	movslq %edx, %rdx movl %ecx, (%rax, %rdx, 4)			
	addl \$1, - 148(%rbp) .L5:			
	movl -148(%rbp), %eax			
	cmpl -140(%rbp), %eax			
	jle .L6 leaq -80(%rbp), %rax			
	movq %rax, %rsi movl \$0, %edi call			
	clock_gettime@PLT			
	movq -80(%rbp), %rdx			
	movq -96(%rbp), %rax			
	subq %rax, %rdx movq %rdx, %rax cvtsi2sdq %rax, %xmm1 movq -72(%rbp), %rdx			
	movq -88(%rbp), %rax			
	subq %rax, %rdx movq %rdx, %rax cvtsi2sdq %rax, %xmm0 movsd .LC1(%rip), %xmm2			
	divsd %xmm2, %xmm0 addsd %xmm1, %xmm0 movsd %xmm0, - %xmm0			

--	--	--	--