2º curso / 2º cuatr.

Grado Ing. Inform.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Félix Ramírez García
Grupo A3 con profesor Christian Morillas

Fecha de entrega: 27-05-2019
Fecha evaluación en clase: 27-05-2019

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo):

model name : Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz

Sistema operativo utilizado: Ubuntu 18.04 (virtualización nativa de windows)

Versión de gcc utilizada: gcc (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0

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

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

CPU(s): 6

On-line CPU(s) list: 0-5

Thread(s) per core: 1

Core(s) per socket: 6

Socket(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 158

Model name: Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz

Stepping: 12

CPU MHz: 3696.000

CPU max MHz: 3696.0000

BogoMIPS: 7392.00

Virtualization: VT-x

Hypervisor vendor: Windows Subsystem for Linux

Virtualization type: container

Flags: 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 pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave osxsave avx f16c rdrand

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.

   A continuación se muestra el código de `figura1-original.c`:

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

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

int main(){
    int ii, X1, X2, i;
    int R[40000];
    struct timespec ini, fin;
    double time;

    clock_gettime(CLOCK_REALTIME, &ini);
    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, &fin);

    time=(double)(fin.tv_sec-ini.tv_sec)+
        (double)((fin.tv_nsec-ini.tv_nsec)/1.e+9);

    printf("R[0]=%d\tR[39999]=%d\n",R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}
```

1.1. MODIFICACIONES REALIZADAS :

Modificación a) –explicación-: **Realizar los dos bucles distintos en uno solo**

Modificación b) –explicación-: Reducir el número de iteraciones del bucle más interno

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) A continuación se muestran en orden los codigos de figura1**-modificado_a.c**

```c
#include <stdio.h>
#include <time.h>
#include "figura1_a.h"

int main(){
    int ii, X1, X2, i;
    int R[40000];
    struct timespec ini, fin;
    double time;

    clock_gettime(CLOCK_REALTIME, &ini);
    auxiliar(ii,X1,X2,i,R); //definicion en figura1_a.h
    clock_gettime(CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec-ini.tv_sec)+(double)((fin.tv_nsec-ini.tv_nsec)/

    printf("R[0]=%d\tR[39999]=%d\n",R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}
```

y de figura1_a.h :

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

void auxiliar(int ii, int X1, int X2 , int i, int R[]){
    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;
    }
}
```

A continuación se muestra la compilación y resultado de `figura1-modificado_a.c`

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$gcc -o figura1-modificado_a figura1-modificado_a.c -lrt
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$gcc -S figura1-modificado_a.c -lrt
```

b) A continuación se muestran en orden los códigos de `figura1-modificado_b.c` y de `figura1_b.b` :

```c
#include <stdio.h>
#include <time.h>
#include "figura1_a.h"

int main(){
    int ii, X1, X2, i;
    int R[40000];
    struct timespec ini, fin;
    double time;

    clock_gettime(CLOCK_REALTIME, &ini);
    auxiliar(ii,X1,X2,i,R);
    clock_gettime(CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec-ini.tv_sec)+(double)((fin.tv_nsec-ini.tv_nsec)/

    printf("R[0]=%d\tR[39999]=%d\n",R[0], R[39999]);
    printf("Time: %11.9f\n", time);
}
```

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

void auxiliar(int ii, int X1, int X2 , int i, int R[]){
        for(ii=0;ii<40000;ii++){
        X1=0; X2=0;
        for(i=0;i<5000;i+=10) {
            X1+=2*s[i].a+ii; X2+=3*s[i].b-ii;
            X1+=2*s[i+1].a+ii; X2+=3*s[i+1].b-ii;
            X1+=2*s[i+2].a+ii; X2+=3*s[i+2].b-ii;
            X1+=2*s[i+3].a+ii; X2+=3*s[i+3].b-ii;
            X1+=2*s[i+4].a+ii; X2+=3*s[i+4].b-ii;
            X1+=2*s[i+5].a+ii; X2+=3*s[i+5].b-ii;
            X1+=2*s[i+6].a+ii; X2+=3*s[i+6].b-ii;
            X1+=2*s[i+7].a+ii; X2+=3*s[i+7].b-ii;
            X1+=2*s[i+8].a+ii; X2+=3*s[i+8].b-ii;
            X1+=2*s[i+9].a+ii; X2+=3*s[i+9].b-ii;
        }

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

A continuación se muestra la compilación y resultado de `figura1-modificado_b.c`

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$gcc -o figura1-modificado_b figura1-modificado_b.c -lrt
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entrega/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$gcc -S figura1-modificado_b.c -lrt
```

1.1. TIEMPOS:

Para la toma de tiempos se han ejecuta los 3 programas como muestra la siguiente imagen:

```
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entre
ga/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$./figura1-original
R[0]=0  R[39999]=-199995000
Time: 0.615945500
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entre
ga/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$./figura1-modificado_a
R[0]=0  R[39999]=-199995000
Time: 0.399110700
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/felix/Desktop/Google_drive/AC/Practicas/Entre
ga/bp4_RamirezGarciaFelix/ejer1] 2019-05-23 Thursday
$./figura1-modificado_b
R[0]=0  R[39999]=-199995000
Time: 0.400449300
```

Modificación	Tiempo
Sin modificar	*0.615945500*
Modificación a)	0.399110700
Modificación b)	0.400449300

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

En la estructura los datos están seguidos unos de otros (a1, b1, a2, b2, a3, b3…). Por lo que en si lo ejecutamos seguido (el calculo con la variable ‘a’ y después el calculo con la variable ‘b’) vemos bastante mejora en el tiempo. La segunda mejora también ha sido bastante efectiva ya que nos hemos quitado 5000/10 comparaciones y saltos del bucle for con respecto a la ejecución anterior. Los archivos se han divido en .c y .h respectivamente por la comodidad de localizar la función en el código ensamblador generado..

1.2 Genere los códigos en ensamblador 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.

figura1-original	figura1-modificado_a	figura1-modificado_b

.L3:
movl (%rax), %edx
addq $8, %rax
leal (%rdi,%rdx,2), %edx
addl %edx, %ecx
cmpq $s+40000, %rax
jne .L3
movl $s+4, %eax
xorl %esi, %esi
.p2align 4,,10
.p2align 3
 	
.L4:
movl (%rax), %edx
addq $8, %rax
leal (%rdx,%rdx,2), %edx
subl %edi, %edx
addl %edx, %esi
cmpq $s+40004, %rax
jne .L4
	.L3:
	movl (%rax), %edx
	leal (%rdi,%rdx,2), %edx
	addl %edx, %ecx
	movl 4(%rax), %edx
	addq $8, %rax
	leal (%rdx,%rdx,2), %edx
	subl %edi, %edx
	addl %edx, %esi
	cmpq $s+40000, %rax
	jne .L3
	
	Como vemos, aquí a parte de realizar muchos menos saltos, estos son más pequeños. En la versión original realiza el salto de 8 posiciones 5000*2 veces, mientras que aquí lo realizamos la mitad de veces, además de evitarnos los saltos de ejecutar dos bucles.
		movl 8(%rax), %r10d
		addl %esi, %ecx
		movl 12(%rax), %esi
		leal (%rdx,%r10,2), %r10d
		leal (%rsi,%rsi,2), %esi
		addl %r10d, %r9d
		movl 16(%rax), %r10d
		subl %edx, %esi
		addl %esi, %ecx
		
		….
		
		movl 68(%rax), %ecx
		leal (%rcx,%rcx,2), %ecx
		subl %edx, %ecx
		addl %ecx, %esi
		movl 76(%rax), %ecx
		addq $80, %rax
		leal (%rcx,%rcx,2), %ecx
		subl %edx, %ecx
		addl %esi, %ecx
		cmpl $5000, %edi
		jne .L3
		
		Con este algoritmo, en vez de estar haciendo saltos y comparaciones en la cabecera del for en cada iteración, nos ahorramos algunas iteraciones, ya que como vemos esos cálculos los hace seguidos dentro del bucle.

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

Para optimizar algo mas el código podríamos eliminar los saltos sustituyendo las instrucciones jne por instrucciones cmovxx o setxx , asi como usar instrucciones de prefech para cargar zonas de memoria en cache antes de que sean solicitadas, evitándonos así atascos por acceso a memoria.

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;
   }
   …
}
```

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):

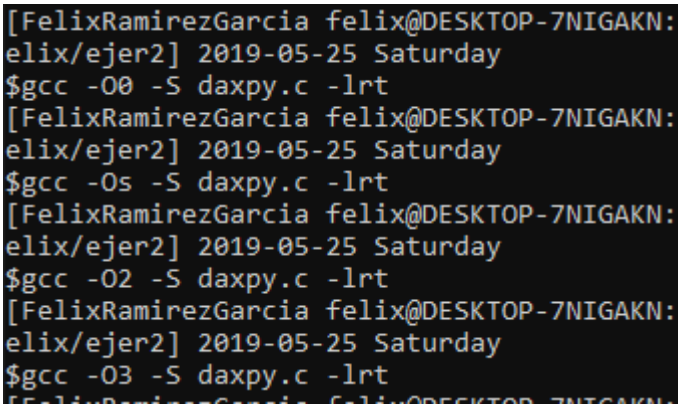
```
for (i=1;i<=N,i++) y[i]= a*x[i] + y[i];
```

A continuación se muestra el código de `daxpy.c`:

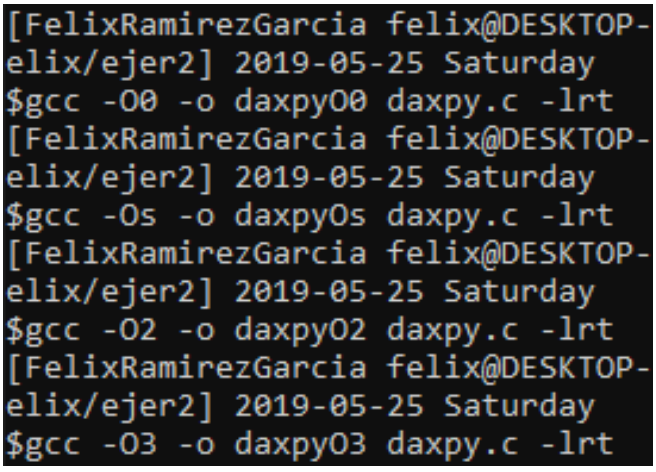
```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define max 33554432

int x[max], y[max];

int main(int argc, char *argv[]){
    int n, constante, i;
    struct timespec ini, fin;
    double time;

    if(argc<3){
        fprintf(stderr, "Falta el tamaño del vector y la constante");
        exit(-1);
    }

    n=atoi(argv[1]);
    constante=atoi(argv[2]);
    if(n>max) n=max;

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

    clock_gettime(CLOCK_REALTIME, &ini);
    for(i=0;i<n;i++) y[i]= constante*x[i]+y[i];
    clock_gettime(CLOCK_REALTIME, &fin);

    time=(double)(fin.tv_sec-ini.tv_sec)+
        (double)((fin.tv_nsec-ini.tv_nsec)/1.e+9);

    printf("Time=%11.9f\ty[0]=%d, y[%d]=%d\n", time, y[0], n-1, y[n-1]);
}
```

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

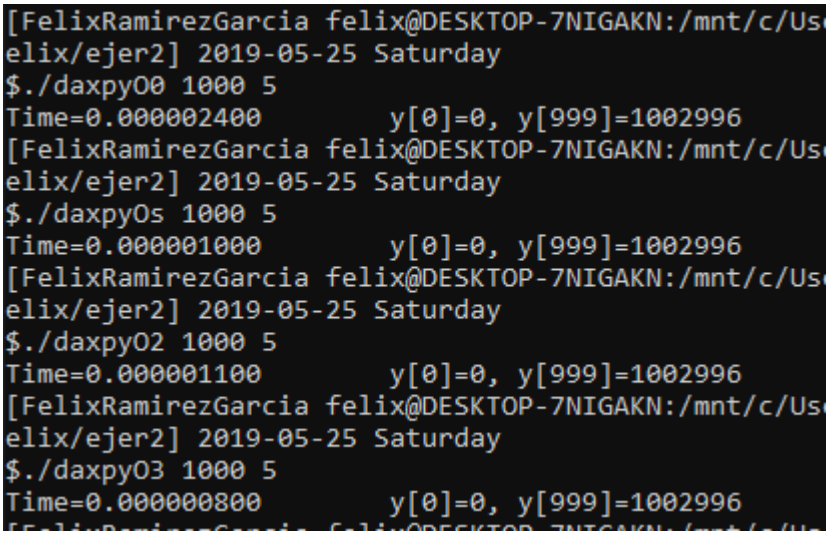
A continuación se muestra una captura con la obtención del código ensamblador de daxpy con las diferentes optimizaciones:



A continuación se muestra una captura con la compilación de daxpy con las diferentes optimizaciones:



A continuación se muestra una captura con la ejecución de las diferentes optimizaciones y una tabla con los resultados del tiempo:



Tiempos ejec.	**-O0**	**-Os**	**-O2**	**-O3**
	0.000002400	0.000001000	0.000001100	0.000000800

A continuación se muestra una tabla con el código ensamblador de daxpy en sus diferentes optimizaciones:

daxpyO0.s	daxpyOs.s	daxpyO2.s	daxpyO3.s
.file "daxpy.c" .text .comm x,134217728,32 .comm y,134217728,32 .section .rodata .align 8 .LC0: .string "Falta el tama\303\261o del vector y la constante" .align 8 .LC2: .string "Time= %11.9f\ty[0]=%d, y[%d]=%d\n" .text .globl main .type main, @function main: .LFB5: .cfi_startproc pushq %rbp .cfi_def_cfa_offset 16 .cfi_offset 6, -16 movq %rsp, %rbp	.file "daxpy.c" .text .section .rodata.str1.1, "aMS",@progbits,1 .LC0: .string "Falta el tama\303\261o del vector y la constante" .LC2: .string "Time= %11.9f\ty[0]=%d, y[%d]=%d\n" .section .text.startup,"ax",@progbits .globl main .type main, @function main: .LFB23: .cfi_startproc pushq %r12 .cfi_def_cfa_offset 16 .cfi_offset 12, -16 pushq %rbp .cfi_def_cfa_offset 24 .cfi_offset 6, -24 pushq %rbx .cfi_def_cfa_offset 32	.text .section .rodata.str1.8,"aMS",@progbits,1 .align 8 .LC0: .string "Falta el tama\303\261o del vector y la constante" .align 8 .LC2: .string "Time=%11.9f\ty[0]=%d, y[%d]=%d\n" .section .text.startup,"ax",@progbits .p2align 4,,15 .globl main .type main, @function main: .LFB41: .cfi_startproc pushq %r13 .cfi_def_cfa_offset 16 .cfi_offset 13, -16 pushq %r12 .cfi_def_cfa_offset 24 .cfi_offset 12, -24 pushq %rbp	.file "daxpy.c" .text .section .rodata.str1.8,"aMS",@progbits,1 .align 8 .LC1: .string "Falta el tama\303\261o del vector y la constante" .align 8 .LC4: .string "Time=%11.9f\ty[0]=%d, y[%d]=%d\n" .section .text.startup,"ax",@progbits .p2align 4,,15 .globl main .type main, @function main: .LFB41: .cfi_startproc pushq %r14 .cfi_def_cfa_offset 16 .cfi_offset 14, -16 pushq %r13 .cfi_def_cfa_offset 24 .cfi_offset 13, -24

```
        .cfi_def_cfa_register 6
        subq    $112, %rsp
        movl    %edi, -84(%rbp)
        movq    %rsi, -96(%rbp)
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        cmpl    $2, -84(%rbp)
        jg      .L2
        movq    stderr(%rip), %rax
        movq    %rax, %rcx
        movl    $42, %edx
        movl    $1, %esi
        leaq    .LC0(%rip), %rdi
        call    fwrite@PLT
        movl    $-1, %edi
        call    exit@PLT
.L2:
        movq    -96(%rbp), %rax
        addq    $8, %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        call    atoi@PLT
        movl    %eax, -68(%rbp)
        movq    -96(%rbp), %rax
        addq    $16, %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        call    atoi@PLT
        movl    %eax, -60(%rbp)
        cmpl    $33554432, -68(%rbp)
        jle     .L3
        movl    $33554432, -68(%rbp)
.L3:
        movl    $0, -64(%rbp)
        jmp     .L4
.L5:
        movl    -64(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rcx
        leaq    x(%rip), %rax
        movl    -64(%rbp), %edx
        movl    %edx, (%rcx, %rax)
        movl    -64(%rbp), %eax
        imull   -64(%rbp), %eax
        movl    %eax, %edx
        movl    -64(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rcx
        leaq    y(%rip), %rax
        movl    %edx, (%rcx, %rax)
        .cfi_offset 3, -32
        subq    $48, %rsp
        .cfi_def_cfa_offset 80
        movq    %fs:40, %rax
        movq    %rax, 40(%rsp)
        xorl    %eax, %eax
        cmpl    $2, %edi
        jg      .L2
        movq    stderr(%rip), %rsi
        leaq    .LC0(%rip), %rdi
        call    fputs@PLT
        orl     $-1, %edi
        call    exit@PLT
.L2:
        movq    8(%rsi), %rdi
        movq    %rsi, %rbp
        call    atoi@PLT
        movq    16(%rbp), %rdi
        movl    %eax, %ebx
        leaq    y(%rip), %rbp
        call    atoi@PLT
        cmpl    $33554432, %ebx
        movl    $33554432, %ecx
        leaq    x(%rip), %rdx
        movl    %eax, %r12d
        cmovg   %ecx, %ebx
        xorl    %eax, %eax
.L3:
        cmpl    %ebx, %eax
        jge     .L10
        movl    %eax, %ecx
        movl    %eax, (%rdx, %rax,4)
        imull   %eax, %ecx
        movl    %ecx, 0(%rbp,%rax,4)
        incq    %rax
        jmp     .L3
.L10:
        leaq    8(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        leaq    x(%rip), %rdx
        xorl    %eax, %eax
.L5:
        cmpl    %eax, %ebx
        jle     .L11
        movl    (%rdx, %rax,4), %ecx
        imull   %r12d, %ecx
        addl    %ecx, 0(%rbp,%rax,4)
        incq    %rax
        jmp     .L5
.L11:
        leaq    24(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    32(%rsp), %rax
        subq    16(%rsp), %rax
        .cfi_def_cfa_offset 32
        .cfi_offset 6, -32
        pushq   %rbx
        .cfi_def_cfa_offset 40
        .cfi_offset 3, -40
        subq    $56, %rsp
        .cfi_def_cfa_offset 96
        movq    %fs:40, %rax
        movq    %rax, 40(%rsp)
        xorl    %eax, %eax
        cmpl    $2, %edi
        jle     .L15
        movq    8(%rsi), %rdi
        movq    %rsi, %rbp
        movl    $10, %edx
        xorl    %esi, %esi
        call    strtol@PLT
        movq    16(%rbp), %rdi
        movq    %rax, %rbx
        xorl    %esi, %esi
        movl    $10, %edx
        movl    $33554432, %ebp
        call    strtol@PLT
        cmpl    $33554432, %ebx
        movl    %eax, %r13d
        cmovle  %ebx, %ebp
        testl   %ebx, %ebx
        jle     .L3
        leaq    x(%rip), %r12
        leaq    y(%rip), %rbx
        xorl    %edx, %edx
        .p2align 4,,10
        .p2align 3
.L4:
        movl    %edx, %ecx
        movl    %edx, (%r12,%rdx,4)
        imull   %edx, %ecx
        movl    %ecx, (%rbx, %rdx,4)
        addq    $1, %rdx
        cmpl    %edx, %ebp
        jg      .L4
        movq    %rsp, %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        xorl    %edx, %edx
        .p2align 4,,10
        .p2align 3
.L6:
        movl    (%r12,%rdx,4), %ecx
        imull   %r13d, %ecx
        addl    %ecx, (%rbx, %rdx,4)
        addq    $1, %rdx
        cmpl    %edx, %ebp
        jg      .L6
.L7:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    24(%rsp), %rax
        subq    8(%rsp), %rax
        leal    -1(%rbp), %ecx
        pxor    %xmm0, %xmm0
        pushq   %r12
        .cfi_def_cfa_offset 32
        .cfi_offset 12, -32
        pushq   %rbp
        .cfi_def_cfa_offset 40
        .cfi_offset 6, -40
        pushq   %rbx
        .cfi_def_cfa_offset 48
        .cfi_offset 3, -48
        subq    $64, %rsp
        .cfi_def_cfa_offset 112
        movq    %fs:40, %rax
        movq    %rax, 56(%rsp)
        xorl    %eax, %eax
        cmpl    $2, %edi
        jle     .L23
        movq    8(%rsi), %rdi
        movq    %rsi, %rbx
        movl    $10, %edx
        xorl    %esi, %esi
        call    strtol@PLT
        movq    16(%rbx), %rdi
        movq    %rax, %r14
        xorl    %esi, %esi
        movl    $10, %edx
        movl    $33554432, %ebx
        call    strtol@PLT
        cmpl    $33554432, %r14d
        movq    %rax, %r13
        movl    %eax, 12(%rsp)
        cmovle  %r14d, %ebx
        testl   %r14d, %r14d
        jle     .L3
        cmpl    $3, %r14d
        jle     .L15
        movl    %ebx, %ecx
        movdqa  .LC0(%rip), %xmm1
        leaq    x(%rip), %r12
        leaq    y(%rip), %rbp
        shrl    $2, %ecx
        movdqa  .LC2(%rip), %xmm3
        xorl    %eax, %eax
        xorl    %edx, %edx
        .p2align 4,,10
        .p2align 3
.L5:
        movdqa  %xmm1, %xmm2
        movdqa  %xmm1, %xmm0
        addl    $1, %edx
        pmuludq %xmm1, %xmm2
        psrlq   $32, %xmm0
        pshufd  $8, %xmm2, %xmm2
        pmuludq %xmm0, %xmm0
        pshufd  $8, %xmm0, %xmm0
        movaps  %xmm1, (%r12,%rax)
        paddd   %xmm3, %xmm1
```

```
        addl    $1, -64(%rbp)
.L4:
        movl    -64(%rbp),
%eax
        cmpl    -68(%rbp),
%eax
        jl      .L5
        leaq    -48(%rbp),
%rax
        movq    %rax, %rsi
        movl    $0, %edi
        call
        clock_gettime@PLT
        movl    $0, -64(%rbp)
        jmp     .L6
.L7:
        movl    -64(%rbp),
%eax
        cltq
        leaq    0(,%rax,4),
%rdx
        leaq    x(%rip), %rax
        movl    (%rdx,%rax),
%eax
        imull   -60(%rbp),
%eax
        movl    %eax, %edx
        movl    -64(%rbp),
%eax
        cltq
        leaq    0(,%rax,4),
%rcx
        leaq    y(%rip), %rax
        movl    (%rcx,%rax),
%eax
        leal    (%rdx,%rax),
%ecx
        movl    -64(%rbp),
%eax
        cltq
        leaq    0(,%rax,4),
%rdx
        leaq    y(%rip), %rax
        movl    %ecx,   (%rdx,
%rax)
        addl    $1, -64(%rbp)
.L6:
        movl    -64(%rbp),
%eax
        cmpl    -68(%rbp),
%eax
        jl      .L7
        leaq    -32(%rbp),
%rax
        movq    %rax, %rsi
        movl    $0, %edi
        call
        clock_gettime@PLT
        movq    -32(%rbp),
%rdx
        movq    -48(%rbp),
%rax
        subq    %rax, %rdx
        movq    %rdx, %rax
        cvtsi2sdq %rax, %xmm1
        movq    -24(%rbp),
%rdx
        movq    -40(%rbp),
%rax
        subq    %rax, %rdx
        movq    %rdx, %rax
        leal    -1(%rbx),
%ecx
        leaq    .LC2(%rip),
%rsi
        movl    $1, %edi
        movslq  %ecx, %rdx
        movl    0(%rbp,
%rdx,4), %r8d
        movl    y(%rip),
%edx
        cvtsi2sdq %rax,
%xmm0
        movq    24(%rsp),
%rax
        subq    8(%rsp),
%rax
        cvtsi2sdq %rax,
%xmm1
        movb    $1, %al
        divsd   .LC1(%rip),
%xmm0
        addsd   %xmm1,
%xmm0
        call
        __printf_chk@PLT
        xorl    %eax, %eax
        movq    40(%rsp),
%rdx
        xorq    %fs:40, %rdx
        je      .L7
        call
        __stack_chk_fail@PLT
.L7:
        addq    $48, %rsp
        .cfi_def_cfa_offset 32
        popq    %rbx
        .cfi_def_cfa_offset 24
        popq    %rbp
        .cfi_def_cfa_offset 16
        popq    %r12
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE23:
        .size   main, .-main
        .comm
        y,134217728,32
        .comm
        x,134217728,32
        .section    .rodata.cst8,"a
M",@progbits,8
        .align 8
.LC1:
        .long   0
        .long   1104006501
        .ident  "GCC:
(Ubuntu  7.3.0-27ubuntu1~18.04)
7.3.0"
        .section    .note.GNU-
stack,"",@progbits
        leaq    .LC2(%rip), %rsi
        pxor    %xmm1,
%xmm1
        movslq  %ecx, %rdx
        movl    (%rbx,%rdx,4),
%r8d
        movl    y(%rip), %edx
        movl    $1, %edi
        cvtsi2sdq %rax, %xmm0
        movq    16(%rsp), %rax
        subq    (%rsp), %rax
        cvtsi2sdq %rax, %xmm1
        movl    $1, %eax
        divsd   .LC1(%rip),
%xmm0
        addsd   %xmm1,
%xmm0
        call
        __printf_chk@PLT
        xorl    %eax, %eax
        movq    40(%rsp), %rsi
        xorq    %fs:40, %rsi
        jne     .L16
        addq    $56, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 40
        popq    %rbx
        .cfi_def_cfa_offset 32
        popq    %rbp
        .cfi_def_cfa_offset 24
        popq    %r12
        .cfi_def_cfa_offset 16
        popq    %r13
        .cfi_def_cfa_offset 8
        ret
.L3:
        .cfi_restore_state
        movq    %rsp, %rsi
        xorl    %edi, %edi
        leaq    y(%rip), %rbx
        call
        clock_gettime@PLT
        jmp     .L7
.L16:
        call
        __stack_chk_fail@PLT
.L15:
        movq    stderr(%rip),
%rcx
        leaq    .LC0(%rip), %rdi
        movl    $42, %edx
        movl    $1, %esi
        call    fwrite@PLT
        orl     $-1, %edi
        call    exit@PLT
        .cfi_endproc
.LFE41:
        .size   main, .-main
        .comm   y,134217728,32
        .comm   x,134217728,32
        .section    .rodata.cst8,"aM"
,@progbits,8
        .align 8
.LC1:
        .long   0
        .long   1104006501
        .ident  "GCC:  (Ubuntu
7.3.0-27ubuntu1~18.04) 7.3.0"
                .section    .note.GNU-
        stack,"",@progbits
        punpckldq
        %xmm0, %xmm2
        movaps  %xmm2,
0(%rbp,%rax)
        addq    $16, %rax
        cmpl    %edx, %ecx
        ja      .L5
        movl    %ebx, %eax
        andl    $-4, %eax
        cmpl    %eax, %ebx
        je      .L24
.L4:
        movl    %eax, %ecx
        movslq  %eax, %rdx
        imull   %eax, %ecx
        movl    %eax,
(%r12,%rdx,4)
        movl    %ecx,  0(%rbp,
%rdx,4)
        leal    1(%rax), %edx
        cmpl    %edx, %ebx
        jle     .L7
        movslq  %edx, %rcx
        addl    $2, %eax
        movl    %edx,
(%r12,%rcx,4)
        imull   %edx, %edx
        cmpl    %eax, %ebx
        movl    %edx,  0(%rbp,
%rcx,4)
        jle     .L7
        movslq  %eax, %rdx
        movl    %eax,
(%r12,%rdx,4)
        imull   %eax, %eax
        movl    %eax,  0(%rbp,
%rdx,4)
.L7:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call
        clock_gettime@PLT
        cmpl    $3, %r14d
        jle     .L16
.L13:
        movd    12(%rsp),
%xmm4
        movl    %ebx, %ecx
        leaq    x(%rip), %rdx
        leaq    y(%rip), %rax
        shrl    $2, %ecx
        xorl    %esi, %esi
        pshufd  $0,     %xmm4,
%xmm2
        movdqa  %xmm2,
%xmm3
        psrlq   $32, %xmm3
        .p2align 4,,10
        .p2align 3
.L9:
        movdqa  (%rdx),
%xmm1
        addl    $1, %esi
        movdqa  %xmm2,
%xmm0
        addq    $16, %rdx
        psrlq   $32, %xmm1
        pmuludq %xmm3,
%xmm1
        pshufd  $8,     %xmm1,
%xmm1
```

```
        cvtsi2sdq %rax, %xmm0
        movsd   .LC1(%rip), %xmm2
        divsd   %xmm2, %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -56(%rbp)
        movl    -68(%rbp), %eax
        subl    $1, %eax
        cltq
        leaq    0(,%rax,4), %rdx
        leaq    y(%rip), %rax
        movl    (%rdx,%rax), %edx
        movl    -68(%rbp), %eax
        leal    -1(%rax), %edi
        movl    y(%rip), %esi
        movq    -56(%rbp), %rax
        movl    %edx, %ecx
        movl    %edi, %edx
        movq    %rax, -104(%rbp)
        movsd   -104(%rbp), %xmm0
        leaq    .LC2(%rip), %rdi
        movl    $1, %eax
        call    printf@PLT
        movl    $0, %eax
        movq    -8(%rbp), %rsi
        xorq    %fs:40, %rsi
        je      .L9
        call
        __stack_chk_fail@PLT
.L9:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE5:
        .size   main, .-main
        .section .rodata
        .align 8
.LC1:
        .long   0
        .long   1104006501
        .ident  "GCC: (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0"
        .section .note.GNU-stack,"",@progbits
```

```
        pmuludq -16(%rdx), %xmm0
        pshufd  $8, %xmm0, %xmm0
        addq    $16, %rax
        punpckldq %xmm1, %xmm0
        paddd   -16(%rax), %xmm0
        movaps  %xmm0, -16(%rax)
        cmpl    %ecx, %esi
        jb      .L9
        movl    %ebx, %eax
        andl    $-4, %eax
        cmpl    %ebx, %eax
        je      .L12
.L11:
        movslq  %eax, %rdx
        movl    (%r12,%rdx,4), %ecx
        imull   %r13d, %ecx
        addl    %ecx, 0(%rbp, %rdx,4)
        leal    1(%rax), %edx
        cmpl    %ebx, %edx
        jge     .L12
        movslq  %edx, %rdx
        addl    $2, %eax
        movl    (%r12,%rdx,4), %ecx
        imull   %r13d, %ecx
        addl    %ecx, 0(%rbp, %rdx,4)
        cmpl    %ebx, %eax
        jge     .L12
        cltq
        imull   (%r12,%rax,4), %r13d
        addl    %r13d, 0(%rbp,%rax,4)
.L12:
        leaq    32(%rsp), %rsi
        xorl    %edi, %edi
        call    clock_gettime@PLT
        movq    40(%rsp), %rax
        pxor    %xmm0, %xmm0
        subq    24(%rsp), %rax
        pxor    %xmm1, %xmm1
        leal    -1(%rbx), %ecx
        leaq    .LC4(%rip), %rsi
        movl    $1, %edi
        movslq  %ecx, %rdx
        cvtsi2sdq %rax, %xmm0
        movq    32(%rsp), %rax
        subq    16(%rsp), %rax
        movl    0(%rbp, %rdx,4), %r8d
        movl    y(%rip), %edx
        cvtsi2sdq %rax, %xmm1
        movl    $1, %eax
        divsd   .LC3(%rip),
```

```
%xmm0
        addsd   %xmm1,
%xmm0
        call
        __printf_chk@PLT
        xorl    %eax, %eax
        movq    56(%rsp), %rdi
        xorq    %fs:40, %rdi
        jne     .L25
        addq    $64, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 48
        popq    %rbx
        .cfi_def_cfa_offset 40
        popq    %rbp
        .cfi_def_cfa_offset 32
        popq    %r12
        .cfi_def_cfa_offset 24
        popq    %r13
        .cfi_def_cfa_offset 16
        popq    %r14
        .cfi_def_cfa_offset 8
        ret
.L24:
        .cfi_restore_state
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        call
        clock_gettime@PLT
        jmp     .L13
.L3:
        leaq    16(%rsp), %rsi
        xorl    %edi, %edi
        leaq    y(%rip), %rbp
        call
        clock_gettime@PLT
        jmp     .L12
.L15:
        xorl    %eax, %eax
        leaq    x(%rip), %r12
        leaq    y(%rip), %rbp
        jmp     .L4
.L16:
        xorl    %eax, %eax
        jmp     .L11
.L25:
        call
        __stack_chk_fail@PLT
.L23:
        movq    stderr(%rip),
%rcx
        leaq    .LC1(%rip),
%rdi
        movl    $42, %edx
        movl    $1, %esi
        call    fwrite@PLT
        orl     $-1, %edi
        call    exit@PLT
        .cfi_endproc
.LFE41:
        .size   main, .-main
        .comm
        y,134217728,32
        .comm
        x,134217728,32
        .section    .rodata.cst16,"
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: (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0" .section .note.GNU-stack,"",@progbits

Se puede apreciar claramente que la optimizacion O3 genera un numero de instrucciones mayor, ya que usa instrucciones con mayor complejidad para obtener un mejor rendimiento, como es el caso de usar instrucciones 'pmuludq' que es una instruccion del conjunto SSE2 y intrucciones MMX con una extension de hasta 128 bits.

O0: No hay optimización con respecto al código original, es la opción por defecto.

Os: Se optimiza usando las opciones de O2 que no aumenten el tamaño del código.

O2: Se optimiza el número de instrucciones en comparación a O0, pero aquí si usa las opciones de optimización que aumentan el tamaño del código, aunque el aumento de tamaño es insignificante.

O3: Activa las opciones que hacen que el tiempo de compilación y el uso de memoria aumente

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.

Para proceder al calculo vamos a ejecutar el programa como se muestra en la siguiente imagen:

```
$./daxpyO3 1000 10
Time=0.000001000        y[0]=0, y[999]=1007991
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/fe
elix/ejer2] 2019-05-25 Saturday
$./daxpyO3 10000 10
Time=0.000002700        y[0]=0, y[9999]=100079991
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/fe
elix/ejer2] 2019-05-25 Saturday
$./daxpyO3 100000 10
Time=0.000026300        y[0]=0, y[99999]=1410865399
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/fe
elix/ejer2] 2019-05-25 Saturday
$./daxpyO3 1000000 10
Time=0.000438900        y[0]=0, y[999999]=-719379977
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/fe
elix/ejer2] 2019-05-25 Saturday
$./daxpyO3 10000000 10
Time=0.004160100        y[0]=0, y[9999999]=356447223
[FelixRamirezGarcia felix@DESKTOP-7NIGAKN:/mnt/c/Users/fe
elix/ejer2] 2019-05-25 Saturday
$./daxpyO3 100000000 10
Time=0.012837700        y[0]=0, y[33554431]=268435447
```

Considerando los tiempos de la imagen:

R = N.º instrucciones en coma flotante / TCPU * 10^9

N = 1000 R= $3*10^8$ / $0.000001000*10^9$ = 300000
N = 4000 R= $3*10^8$ / $0.000002000*10^9$ = 150000
N = 10000 R= $3*10^8$ / $0.000002700*10^9$ = 111111,11
N = 100000 R= $3*10^8$ / $0.000026300*10^9$ = 11406,84
N = 1000000 R= $3*10^8$ / $0.000438900*10^9$ = 683,52
N = 10000000 R= $3*10^8$ / $0.004160100*10^9$ = 72,11

Rmax = 300000
Nmax = 1000
N1/2 = 4000

La velocidad pico de mi procesador es de 3.70GHz , que se corresponde con el valor obtenido para N1/2 , que es muy inferior al valor de Rmax.