2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 4. Optimización de código

Estudiante (nombre y apellidos): Elena Cantero Molina
Grupo de prácticas: E2
Fecha de entrega:
Fecha evaluación en clase:

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel® Core™ i5-5200U CPU @ 2.20GHz × 4*

Sistema operativo utilizado: *Ubuntu 16,04 LTS*

Versión de gcc utilizada: *5.4.0*

Adjunte el contenido del fichero /proc/cpuinfo de la máquina en la que ha tomado las medidas

1. Para el núcleo que se muestra en la Figura 1 (ver guion de prácticas), y para un programa que implemente la multiplicación de matrices (use variables globales):

   1.1 Modifique el código C para reducir el tiempo de ejecución del mismo. 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

A) MULTIPLICACIÓN DE MATRICES:
CÓDIGO FUENTE: `pmm-secuencial.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

//producto A=B*C
unsigned int N = 1000;

 //Si el array es declarado como global o estático en una función, entonces,
 //todos los elementos se inicializan a cero en caso de que no estén aún inicializados.

int A[1000][1000],B[1000][1000],C[1000][1000];

int main() {

    int i,j,k;
```

```
    struct timespec cgt1,cgt2;
    double ncgt;
    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (i = 0; i < N; ++i)
      for (j = 0; j < N; ++j)
        for (k = 0; k < N; ++k)
          A[i][j] += B[i][k] * C[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 resultados
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);

return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a): Sólo he invertido los bucles j y k.

Modificación b): La modificación b) se ha realizado con un desenrollado de los bucles.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_1.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

//producto A=B*C
unsigned int N = 1000;

 //Si el array es declarado como global o estático en unas función, entonces,
 //todos los elementos se inicializan a cero en caso de que no estén aún inicializados.

int A[1000][1000],B[1000][1000],C[1000][1000];

int main() {

    int i,j,k;
    struct timespec cgt1,cgt2;
    double ncgt;
    clock_gettime(CLOCK_REALTIME,&cgt1);

    //inicializamos las matrices
    for (i = 0; i<N; i++) {
        for (j = 0; j<N; j++){
            B[i][j] = i;
            C[i][j] = j;
            A[i][j] = 0;
        }
    }

    //Calculamos el producto de ambas
    for (i = 0; i < N; i++)
      for (k = 0; k < N; k++)
        for (j = 0; j < N; j++){
          A[i][j] += B[i][k] * C[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 resultados
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
```

```
return 0;
}
```

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

```
usuario@asus:~/Escritorio/2.Practica 4$ gcc -O2 -o pmm1 pmm-secuencial-modificado_1.c
usuario@asus:~/Escritorio/2.Practica 4$ ./pmm1
Tiempo(seg.):0.799979664         / Tamaño Vectores:1000
```

b) pmm-secuencial-modificado_2.c

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

//producto A=B*C
unsigned int N = 1000;

 //Si el array es declarado como global o estático en unas función, entonces,
 //todos los elementos se inicializan a cero en caso de que no estén aún inicializados.

int A[1000][1000],B[1000][1000],C[1000][1000];

int main() {

    int i,j,k;
    struct timespec cgt1,cgt2;
    double ncgt;
    clock_gettime(CLOCK_REALTIME,&cgt1);

    //inicializamos las matrices
    for (i = 0; i<N; i++) {
        for (j = 0; j<N; j++){
            B[i][j] = i;
            C[i][j] = j;
        }
    }

    //Calculamos el producto de ambas
    for (i = 0; i < N; i+=4)
      for (j = 0; j < N; j+=4)
        for (k = 0; k < N; k+=4){
          A[i][j] += B[i][k] * C[k][j];
          A[i+1][j+1] += B[i+1][k+1] * C[k+1][j+1];
          A[i+2][j+2] += B[i+2][k+2] * C[k+2][j+2];
          A[i+3][j+3] += B[i+3][k+3] * C[k+3][j+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 resultados
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);

return 0;
}
```

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

```
usuario@asus:~/Escritorio/2.Practica 4$ gcc -O2 -o pmm2 pmm-secuencial-modificado_2.c
usuario@asus:~/Escritorio/2.Practica 4$ ./pmm2
Tiempo(seg.):0.141267124         / Tamaño Vectores:1000
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*1.306837811*
Modificación a)	*0.799979664*
Modificación b)	*0.141267124*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Podemos observar que con un simple cambio en los bucles se puede optimizar con respecto al código original, pero si hacemos un desenrollado de bucles, la optimización será mayor.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):

(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

pmm-secuencial.s

```
.L4:
   movl %ecx, B(%rdx,%rax,4)
   movl %eax, C(%rdx,%rax,4)
   addq $1, %rax
   cmpq %rsi, %rax
   jne .L4
   addl $1, %ecx
   addq $4000, %rdx
   cmpl %r8d, %ecx
   jne .L5
   movl %edi, %eax
   movl $B, %r10d
   leaq 1(%rax), %rdx
   imulq $4004, %rdx, %rcx
   leaq A(,%rdx,4), %r9
   imulq $4000, %rdx, %rdx
   leaq A(%rcx), %r12
   movq %rax, %rcx
   imulq $-4000, %rax, %rax
   negq %rcx
   leaq C(%rdx), %rbp
   leaq -4(,%rcx,4), %rbx
   leaq -4000(%rax), %r11
.L9:
   leaq (%rbx,%r9), %r8
   movq %rbp, %rdi
   .p2align 4,,10
   .p2align 3
.L10:
   movl (%r8), %esi
   leaq (%r11,%rdi), %rax
   movq %r10, %rcx
   .p2align 4,,10
   .p2align 3
.L7:
   movl (%rcx), %edx
   addq $4000, %rax
   addq $4, %rcx
   imull -4000(%rax), %edx
   addl %edx, %esi
   cmpq %rax, %rdi
   jne .L7
   movl %esi, (%r8)
   addq $4, %r8
   addq $4, %rdi
   cmpq %r9, %r8
   jne .L10
   addq $4000, %r9
   addq $4000, %r10
   cmpq %r12, %r9
   jne .L9
```

pmm-secuencial-modificado_1.s

```
.L4:
   movl %esi, B(%rcx,%rdx,4)
   movl %edx, C(%rcx,%rdx,4)
   movl $0, A(%rcx,%rdx,4)
   addq $1, %rdx
   cmpq %rdx, %rdi
   jne .L4
   addl $1, %esi
   addq $4000, %rcx
   cmpl %esi, %r8d
   jne .L5
   movl %eax, %r8d
   xorl %edx, %edx
   addq $1, %r8
   imulq $4000, %r8, %r10
   salq $2, %r8
.L9:
   leaq        B(%rdx), %r9
   xorl        %esi, %esi
   .p2align 4,,10
   .p2align 3
.L10:
   movl (%r9), %edi
   xorl %eax, %eax
   .p2align 4,,10
   .p2align 3
.L7:
   movl C(%rsi,%rax), %ecx
   imull %edi, %ecx
   addl %ecx, A(%rdx,%rax)
   addq $4, %rax
   cmpq %rax, %r8
   jne .L7
   addq $4000, %rsi
   addq $4, %r9
   cmpq %r10, %rsi
   jne .L10
   addq $4000, %rdx
   cmpq %r10, %rdx
   jne .L9
```

pmm-secuencial-modificado_2.s

```
.L4:
   movl %ecx, B(%rdx,%rax,4)
   movl %eax, C(%rdx,%rax,4)
   addq $1, %rax
   cmpq %rax, %rsi
   jne .L4
   addl $1, %ecx
   addq $4000, %rdx
   cmpl %r15d, %ecx
   jne .L5
   xorl %r13d, %r13d
   movl $12000, %r12d
   movl $8000, %ebp
   movl $4000, %ebx
   movq $0, (%rsp)
   movl $0, 28(%rsp)
.L9:
   movq (%rsp), %rax
   movq $C, 8(%rsp)
   xorl %r14d, %r14d
   addq $B, %rax
   movq %rax, 16(%rsp)
   .p2align 4,,10
   .p2align 3
.L10:
   movq (%rsp), %rax
   movl A+4(%rbx,%r14,4),
%r10d
   xorl %edi, %edi
   movl A+8(%rbp,%r14,4),
%r9d
   movl A+12(%r12,%r14,4),
%r8d
   movq 16(%rsp), %rdx
   movl A(%rax,%r14,4),
%r11d
   movq 8(%rsp), %rax
   .p2align 4,,10
   .p2align 3
.L7:
   movl (%rdx), %ecx
   addl $4, %edi
   addq $16000, %rax
   imull -16000(%rax), %ecx
   addl %ecx, %r11d
   leaq 0(%r13,%rdx), %rcx
   addq $16, %rdx
   movl 4(%rcx,%rbx), %esi
   imull -11996(%rax), %esi
   addl %esi, %r10d
   movl 8(%rcx,%rbp), %esi
   movl 12(%rcx,%r12), %ecx
   imull -7992(%rax), %esi
```

		imull -3988(%rax), %ecx
		addl %esi, %r9d
		addl %ecx, %r8d
		cmpl %r15d, %edi
		jb .L7
		movq (%rsp), %rax
		addq $16, 8(%rsp)
		movl %r11d, A(%rax, %r14,4)
		movl %r10d, A+4(%rbx, %r14,4)
		movl %r9d, A+8(%rbp, %r14,4)
		movl %r8d, A+12(%r12,%r14,4)
		addq $4, %r14
		cmpl %r14d, %r15d
		ja .L10
		addl $4, 28(%rsp)
		addq $16000, (%rsp)
		addq $16000, %rbx
		movl 28(%rsp), %eax
		addq $16000, %rbp
		addq $16000, %r12
		subq $16000, %r13
		cmpl %eax, %r15d
		ja .L9

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

int main(){
  int ii, i, X1, X2;
  int R[40000];

  struct timespec cgt1, cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME, &cgt1);

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

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

  printf("R[0] = %i, R[39999] = %i \n", R[0], R[39999]);
```

```
  printf("\nTiempo(seg.) = %11.9f \n", ncgt);

  return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –explicación-: Sólo he quitado un bucle y el else de la condición if else, ya que para el compilador es más pesado hacer dos condiciones que una.

Modificación b) –explicación-: La modificación b sólo tiene un desenrollado de bucles.

1.1. CÓDIGOS FUENTE MODIFICACIONES

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

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

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

int main(){
  int ii, i, X1, X2;
  int R[40000];

  struct timespec cgt1, cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME, &cgt1);

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

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

  printf("R[0] = %i, R[39999] = %i \n", R[0], R[39999]);
  printf("\nTiempo(seg.) = %11.9f \n", ncgt);

  return 0;
}
```

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

```
usuario@asus:~/Escritorio/2.Practica 4$ gcc -O2 -o figura1a figura1-modificado-a.c
usuario@asus:~/Escritorio/2.Practica 4$ ./figura1a
R[0] = 0, R[39999] = -199995000

Tiempo(seg.) = 0.188823645
```

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

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

struct {
```

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

int main(){
  int ii, i, X1, X2;
  int R[40000];

  struct timespec cgt1, cgt2;
  double ncgt;

  clock_gettime(CLOCK_REALTIME, &cgt1);

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

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

  printf("R[0] = %i, R[39999] = %i \n", R[0], R[39999]);
  printf("\nTiempo(seg.) = %11.9f \n", ncgt);

  return 0;
}
```

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

```
usuario@asus:~/Escritorio/2.Practica 4$ gcc -O2 -o figura1b figura1-modificado-b.c
usuario@asus:~/Escritorio/2.Practica 4$ ./figura1b
R[0] = 0, R[39999] = -199995000

Tiempo(seg.) = 0.154517376
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	*0.274833497*
Modificación a)	*0.188823645*
Modificación b)	*0.154517376*

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Podemos observar como el hecho de haber desenrollado bucles en la segunda modificación, no supone mucha mejor de tiempo con respecto a la primera modificación.

1.2. CÓDIGO EN ENSAMBLADOR DEL ORIGINAL Y DE DOS MODIFICACIONES (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR EVALUADA, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

Figura1-original.s

```
.L2:
   movl %r9d, %edi
   movl $s, %eax
   xorl %esi, %esi
   .p2align 4,,10
   .p2align 3
.L3:
   movl (%rax), %edx
   addq $8, %rax
   leal (%rdi,%rdx,2), %edx
   addl %edx, %esi
   cmpq $s+40000, %rax
   jne .L3
   movl $s+4, %eax
   xorl %ecx, %ecx
   .p2align 4,,10
   .p2align 3
.L4:
   movl (%rax), %edx
   addq $8, %rax
   leal (%rdx,%rdx,2), %edx
   subl %edi, %edx
   addl %edx, %ecx
   cmpq %rax, %r8
   jne .L4
   cmpl %ecx, %esi
   cmovl %esi, %ecx
   movl %ecx, 48(%rsp,%r9,4)
   addq $1, %r9
   cmpq $40000, %r9
   jne .L2
   leaq 32(%rsp), %rsi
   xorl %edi, %edi
```

Figura1-modificado-a.s

```
.L2:
   movl %r9d, %edi
   movl $s, %eax
   xorl %esi, %esi
   xorl  %ecx, %ecx
   .p2align 4,,10
   .p2align 3
.L3:
   movl (%rax), %edx
   addq $8, %rax
   leal (%rdi,%rdx,2), %edx
   addl %edx, %ecx
   movl -4(%rax), %edx
   leal (%rdx,%rdx,2), %edx
   subl %edi, %edx
   addl %edx, %esi
   cmpq %rax, %r8
   jne .L3
   cmpl %esi, %ecx
   cmovge %esi, %ecx
   movl %ecx, 48(%rsp,%r9,4)
   addq $1, %r9
   cmpq $40000, %r9
   jne .L2
   leaq 32(%rsp), %rsi
   xorl %edi, %edi
```

Figura1-modificado-b.s

```
.L2:
   movl %r10d, %r8d
   movl $s, %eax
   xorl %ecx, %ecx
   xorl %edx, %edx
   .p2align 4,,10
   .p2align 3
.L3:
   movl (%rax), %esi
   addq $32, %rax
   leal (%r8,%rsi,2), %esi
   addl %esi, %edx
   movl -28(%rax), %esi
   leal (%rsi,%rsi,2), %edi
   movl -24(%rax), %esi
   subl %r8d, %edi
   leal (%r8,%rsi,2), %esi
   addl %edi, %ecx
   addl %esi, %edx
   movl -20(%rax), %esi
   leal (%rsi,%rsi,2), %esi
   subl %r8d, %esi
   leal (%rsi,%rcx), %edi
   movl -16(%rax), %ecx
   leal (%r8,%rcx,2), %ecx
   addl %ecx, %edx
   movl -12(%rax), %ecx
   leal (%rcx,%rcx,2), %ecx
   subl %r8d, %ecx
   leal (%rcx,%rdi), %esi
   movl -8(%rax), %ecx
   leal (%r8,%rcx,2), %ecx
   addl %ecx, %edx
   movl -4(%rax), %ecx
   leal (%rcx,%rcx,2), %ecx
   subl %r8d, %ecx
   addl %esi, %ecx
   cmpq %rax, %r9
   jne .L3
   cmpl %ecx, %edx
   cmovge %ecx, %edx
   movl %edx, 48(%rsp,
%r10,4)
   addq $1, %r10
   cmpq $40000, %r10
   jne .L2
   leaq 32(%rsp), %rsi
   xorl %edi, %edi
```

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 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 opciones de optimización del compilador (-O0, -O2, -O3) y explique las diferencias que se observan en

el código justificando las mejoras en velocidad que acarrean. Incorpore los códigos al cuaderno de prácticas y destaque las diferencias entre ellos.

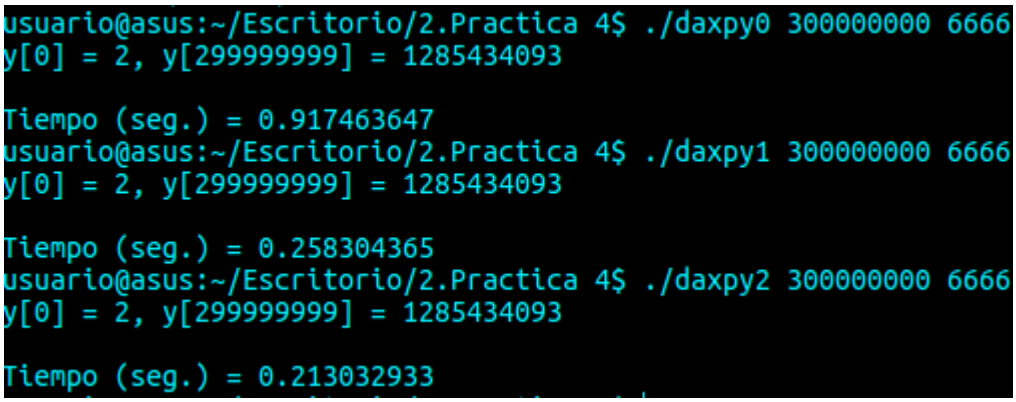
2.2. (Ejercicio EXTRA) Para la mejor de las opciones, obtenga los tiempos de ejecución con distintos valores de N y determine para su sistema los valores de Rmax (valor máximo del número de operaciones en coma flotante por unidad de tiempo), Nmax (valor de N para el que se consigue Rmax), y N1/2 (valor de N para el que se obtiene Rmax/2). Estime el valor de la velocidad pico (Rpico) del procesador (consulte en [4] el número de ciclos por instrucción punto flotante para la familia y modelo de procesador que está utilizando) y compárela con el valor obtenido para Rmax. -Consulte la Lección 3 del Tema 1.

CÓDIGO FUENTE: `daxpy.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

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

void daxpy(int *y, int *x, int a, unsigned n, struct timespec *cgt1, struct timespec
*cgt2)
{
    clock_gettime(CLOCK_REALTIME,cgt1);
    unsigned i;
    for (i=0; i<n; i++)
        y[i] += a*x[i];
    clock_gettime(CLOCK_REALTIME,cgt2);
}

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "ERROR: falta tam del vector y constante\n");
        exit(1);
    }

    unsigned n = strtol(argv[1], NULL, 10);
    int a = strtol(argv[2], NULL, 10);
    int *y, *x;
    y = (int*) malloc(n*sizeof(int));
    x = (int*) malloc(n*sizeof(int));

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

    struct timespec cgt1,cgt2; double ncgt;


    daxpy(y, x, a, n, &cgt1, &cgt2);

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

    printf("y[0] = %i, y[%i] = %i\n", y[0], n-1, y[n-1]);
    printf("\nTiempo (seg.) = %11.9f\n", ncgt);

    free(y);
    free(x);

    return 0;
}
```

Tiempos ejec.	**-O0**	**-O2**	**-O3**
	0.917463647	*0.258304365*	*0.213032933*

CAPTURAS DE PANTALLA:



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:
Con O0 usamos direcciones relativas a la pila y con O2, registros de la arquitectura. Así ahorramos muchas operaciones move innecesarias y como se puede ver abajo obtenemos un código mucho más reducido para O2 que para O0, donde estamos moviendo a registros de la arquitectura direcciones relativas a la pila y operando con esas direcciones.

Por último, en O3, el compilador ha hecho un desenrollado del bucle, dándonos un código mas largo, y en este caso un poco mas rápido que O2.

CÓDIGO EN ENSAMBLADOR (ADJUNTAR AL .ZIP):
(PONER AQUÍ SÓLO LA ZONA DEL CÓDIGO ENSAMBLADOR DONDE ESTÁ EL CÓDIGO EVALUADO, USE COLORES PARA DESTACAR LAS DIFERENCIAS)

daxpyO0.s

```
   movq %rax, -64(%rbp)
   movl $0, -84(%rbp)
   jmp .L6
.L7:
   movl -84(%rbp), %eax
   leaq 0(,%rax,4), %rdx
   movq -72(%rbp), %rax
   addq %rdx, %rax
   movl -84(%rbp), %edx
   addl $2, %edx
   movl %edx, (%rax)
   movl -84(%rbp), %eax
   leaq 0(,%rax,4), %rdx
   movq -64(%rbp), %rax
   addq %rdx, %rax
   movl -84(%rbp), %edx
   addl %edx, %edx
   movl %edx, (%rax)
   addl $1, -84(%rbp)
.L6:
   movl -84(%rbp), %eax
   cmpl -80(%rbp), %eax
   jb .L7
   leaq -32(%rbp), %r8
   leaq -48(%rbp), %rdi
   movl -80(%rbp), %ecx
   movl -76(%rbp), %edx
   movq -64(%rbp), %rsi
   movq -72(%rbp), %rax
   movq %r8, %r9
   movq %rdi, %r8
   movq %rax, %rdi
```

daxpyO2.s

```
   xorl %edi, %edi
   xorl %esi, %esi
   movl $2, %eax
   .p2align 4,,10
   .p2align 3
.L13:
   movl %eax, (%rbx,%rsi)
   addl $1, %eax
   movl %edi, 0(%rbp,%rsi)
   addq $4, %rsi
   addl $2, %edi
   cmpl %edx, %eax
   jne .L13
.L14:
   leaq 32(%rsp), %r9
   leaq 16(%rsp), %r8
   movl %r12d, %ecx
               movl
               %r13d, %edx
               movq
               %rbp, %rsi
movq %rbx, %rdi
```

daxpyO3.s

```
   movl %r13d, %eax
.L33:
   cmpl $1, %eax
   movl $2, (%rbx)
   movl $0, (%r12)
   je .L46
   cmpl $2, %eax
   movl $3, 4(%rbx)
   movl $2, 4(%r12)
   je .L47
   cmpl $3, %eax
   movl $4, 8(%rbx)
   movl $4, 8(%r12)
   je .L48
   cmpl $5, %eax
   movl $5, 12(%rbx)
   movl $6, 12(%r12)
   jne .L49
   movl $6, 16(%rbx)
   movl $8, 16(%r12)
   movl $5, %edx
.L35:
   cmpl %eax, %r13d
   leal -1(%r13), %r14d
   je .L42
.L34:
   movl %r13d, %r8d
   leal -1(%r13), %r14d
   movl %eax, %esi
   subl %eax, %r8d
   leal -4(%r8), %ecx
   movl %r14d, %edi
```

		subl %eax, %edi
		shrl $2, %ecx
		addl $1, %ecx
		cmpl $2, %edi
		leal 0(,%rcx,4), %r9d
		jbe .L37
		movl %edx, 8(%rsp)
		salq $2, %rsi
		xorl %eax, %eax