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): Christian Andrades Molina
Grupo de prácticas: B2
Fecha de entrega: 10/06/14
Fecha evaluación en clase:

Versión de gcc utilizada: ***gcc (Ubuntu/Linaro 4.8.1-10ubuntu9) 4.8.1***

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

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 774.000
cache size  : 3072 KB
physical id : 0
siblings    : 4
core id                 : 0
cpu cores   : 2
apicid                  : 0
initial apicid          : 0
fpu                     : yes
fpu_exception           : yes
cpuid level : 13
wp                      : yes
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 rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1
sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
bogomips    : 3591.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 1
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
```

```
model name  : Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 774.000
cache size  : 3072 KB
physical id : 0
siblings    : 4
core id                 : 1
cpu cores   : 2
apicid                  : 2
initial apicid          : 2
fpu                     : yes
fpu_exception           : yes
cpuid level : 13
wp                      : yes
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 rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1
sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
bogomips    : 3591.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 2
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 774.000
cache size  : 3072 KB
physical id : 0
siblings    : 4
core id                 : 0
cpu cores   : 2
apicid                  : 1
initial apicid          : 1
fpu                     : yes
fpu_exception           : yes
cpuid level : 13
wp                      : yes
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 rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1
sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
```

```
bogomips    : 3591.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:

processor   : 3
vendor_id   : GenuineIntel
cpu family  : 6
model                   : 58
model name  : Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
stepping    : 9
microcode   : 0x19
cpu MHz                 : 774.000
cache size  : 3072 KB
physical id : 0
siblings    : 4
core id                 : 1
cpu cores   : 2
apicid                  : 3
initial apicid          : 3
fpu                     : yes
fpu_exception           : yes
cpuid level : 13
wp                      : yes
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 rdtscp lm constant_tsc arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq
dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1
sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
bogomips    : 3591.60
clflush size            : 64
cache_alignment         : 64
address sizes           : 36 bits physical, 48 bits virtual
power management:
```

1. Para el núcleo que se muestra en la Figura 1, y para un programa que implemente la multiplicación de matrices:

    a. Modifique el código C para reducir el tiempo de ejecución del mismo. Justifique los tiempos obtenidos a partir de la modificación realizada.

    b. Genere los programas en ensamblador para los programas modificados obtenidos en el punto anterior considerando las distintas opciones de optimización del compilador (-O1, -O2,...). Compare los tiempos de ejecución de las versiones de código ejecutable obtenidas con las distintas opciones de optimización y explique las diferencias en tiempo a partir de las características de dichos códigos.

c. (Ejercicio EXTRA) Intente mejorar los resultados obtenidos transformando el código ensamblador del programa para el que se han conseguido las mejores prestaciones de tiempo

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

struct {
             int a;
             int b;
}
s[5000];
main()
{
int ii, i, X1, X2;
const int N=40000;

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


  int *R;
  R = (int *) malloc (N*sizeof(int*));


  for (ii=1; ii<=40000;ii++) {
      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;
   }

for (ii=0;ii<=9;ii++)
     printf("%d\n",R[ii]);


}
```

Figura 1: Núcleo de programa en C para el ejercicio 1.

A) MULTIPLICACIÓN DE MATRICES:

CÓDIGO FUENTE: `pmm-secuencial-modificado.c`
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
_____________________________________________

//OPTIMIZACION 2

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

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

```
int i,j;

if (argc<2){
            printf("Faltan componentes del vector\n");
                        exit(1);
}

const int N = atoi(argv[1]);

//////// DECLARACIÓN DE VECTORES Y MATRIZ

int **matriz = (int **) malloc (N*sizeof(int *));
int **matriz2 = (int **) malloc (N*sizeof(int *));
int **matriz3 = (int **) malloc (N*sizeof(int *));

 for (i=0;i<N;i+=4){
            matriz[i]=(int *) malloc (N*sizeof(int));
            matriz[i+1]=(int *) malloc (N*sizeof(int));
            matriz[i+2]=(int *) malloc (N*sizeof(int));
            matriz[i+3]=(int *) malloc (N*sizeof(int));
}

 for (i=0;i<N;i+=4){
            matriz2[i]=(int *) malloc (N*sizeof(int));
            matriz2[i+1]=(int *) malloc (N*sizeof(int));
            matriz2[i+2]=(int *) malloc (N*sizeof(int));
            matriz2[i+3]=(int *) malloc (N*sizeof(int));
}

 for (i=0;i<N;i+=4){
            matriz3[i]=(int *) malloc (N*sizeof(int));
            matriz3[i+1]=(int *) malloc (N*sizeof(int));
            matriz3[i+2]=(int *) malloc (N*sizeof(int));
            matriz3[i+3]=(int *) malloc (N*sizeof(int));
}


if ( (matriz==NULL) || (matriz2==NULL) || (matriz3==NULL) ){
printf("Error en la reserva de espacio para matrices\n");
exit(-2);
}

////////// INICIALIZAR MATRIZ2 y 3  *quito la primera
 srand(time(NULL));

for (i=0; i<N; i++)
     for(j=0;j<N;j+=4){
            matriz2[i][j]=i+j;
                matriz2[i][j+1]=i+j+1;
                matriz2[i][j+2]=i+j+2;
                matriz2[i][j+3]=i+j+3;
            }

for (i=0; i<N; i++)
     for(j=0;j<N;j+=4){
                matriz3[i][j]=i+j;
```

```
            matriz3[i][j+1]=i+j+1;
            matriz3[i][j+2]=i+j+2;
            matriz3[i][j+3]=i+j+3;
         }


/*printf("\nMATRIZ:\n");
  for (i=0;i<N;i++){
            printf("|");
            for (j=0;j<N;j++) {
                        printf(" %d ",matriz3[i][j]);
            }
            printf("|");
            printf("\n");
  }         */


////////// MULTIPLICACIÓN MATRIZ2 X MATRIZ3

int k=0;
int x, y;

for (i=0;i<N;i++)
            for (j=0;j<N;j++){
                        for (k=0;k<N;k+=4) {

                                                x=matriz2[i][k];
                              y=matriz3[k][i];
                              matriz[i][j]+=x*y;

                              x=matriz2[i][k+1];
                              y=matriz3[k+1][i];
                              matriz[i][j]+=x*y;

                              x=matriz2[i][k+2];
                              y=matriz3[k+2][i];
                              matriz[i][j]+=x*y;

                              x=matriz2[i][k+3];
                              y=matriz3[k+3][i];
                              matriz[i][j]+=x*y;
                        }
        }

printf("\nResultado \n");
printf("\nPrimer valor: %d y ultimo valor: %d ",matriz[0]
[0],matriz[N-1][N-1]);
printf("\n\n");

return 0;

}
```

OPTIMIZACION 1

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

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

int i=0;int j=0;

 if ((argc)<2)
            exit(1);


 const int N = atoi(argv[1]);

struct col {
            int a, b;
} col [N];

int **matriz = (int **) malloc (N*sizeof(int *));
int **matriz2 = (int **) malloc (N*sizeof(int *));
int **matriz3 = (int **) malloc (N*sizeof(int *));

 for (i=0;i<N;i+=4){
            matriz[i]=(int *) malloc (N*sizeof(int));
            matriz[i+1]=(int *) malloc (N*sizeof(int));
            matriz[i+2]=(int *) malloc (N*sizeof(int));
            matriz[i+3]=(int *) malloc (N*sizeof(int));
}

for (i=0;i<N;i+=4) {
            matriz3[i]=(int *) malloc (N*sizeof(int));
            matriz3[i+1]=(int *) malloc (N*sizeof(int));
            matriz3[i+2]=(int *) malloc (N*sizeof(int));
            matriz3[i+3]=(int *) malloc (N*sizeof(int));
}

for (i=0;i<N;i+=4) {
            matriz2[i]=(int *) malloc (N*sizeof(int));
            matriz2[i+1]=(int *) malloc (N*sizeof(int));
            matriz2[i+2]=(int *) malloc (N*sizeof(int));
            matriz2[i+3]=(int *) malloc (N*sizeof(int));

 }

  srand(time(NULL));
 for (i=0; i<N; i++)
     for(j=0;j<N;j+=4){
                matriz[i][j]=0;
                matriz[i][j+1]=0;
                matriz[i][j+2]=0;
                matriz[i][j+3]=0;
            }

            for (i=0; i<N; i++)
     for(j=0;j<N;j+=4){
```

```
            matriz2[i][j]=i+j;
                matriz2[i][j+1]=i+j+1;
                matriz2[i][j+2]=i+j+2;
                matriz2[i][j+3]=i+j+3;
            }

            for (i=0; i<N; i++)
     for(j=0;j<N;j+=4){
                matriz3[i][j]=i+j;
            matriz3[i][j+1]=i+j+1;
            matriz3[i][j+2]=i+j+2;
            matriz3[i][j+3]=i+j+3;
         }

  int k=0;
  for (i=0;i<N;i++)
            for (j=0;j<N;j++){
                        for (k=0;k<N;k+=4) {
                        matriz[i][j]+=(((matriz2[i][k])*(matriz3[k]
[j]))+
                        ((matriz2[i][k+1])*(matriz3[k+1][j]))+
                        ((matriz2[i][k+2])*(matriz3[k+2][j]))+
                        ((matriz2[i][k+3])*(matriz3[k+3][j])));
                        }
        }

  printf("\nResultado \n");


            printf("\n %d %d primero y ultimo",matriz[0]
[0],matriz[N-1][N-1]);
  printf("\n\n");



return (0);

}
```

MODIFICACIONES REALIZADAS:

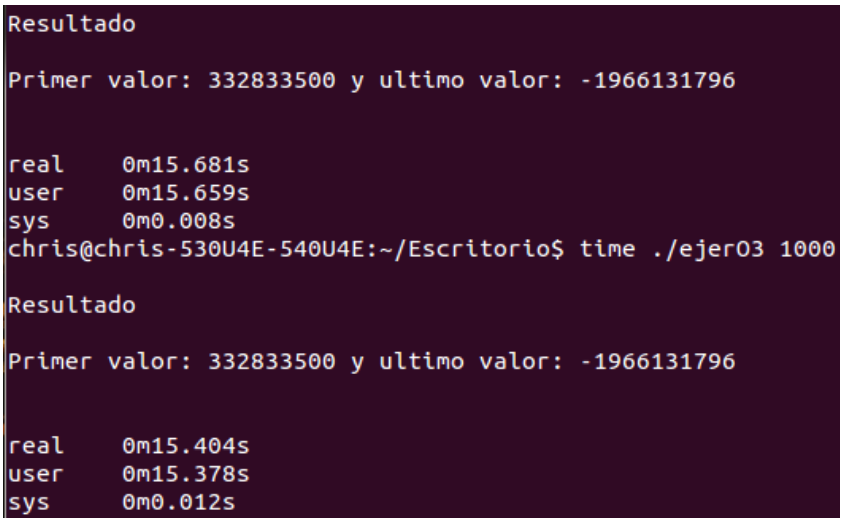
Modificación 1) –explicación-:
Desenrollados simples de los bucles en la inicialización de las matrices 1, 2 y 3 y multiplicación de las matrices 2 y 3 para dar lugar a 1.

Modificación 2) –explicación-:
Desenrollados de bucles en la inicialización y multiplicación de las matrices 2 y 3.
Dos variables “x” e “y” donde guardo el valor de la matriz 2 y 3 que necesito y multiplicarlos para dar lugar a A, reduciendo el tiempo de carga y búsqueda en memoria de los datos.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	18.987s	12.112s	11.306s	10.553s	10.434s
Modificación 1)	11.795s	9.132s	9.137s	9.208s	8.979s
Modificación 2)	9.935s	8.974s	9.067s	9.080s	8.865s

COMENTARIOS SOBRE LOS RESULTADOS: Los cambios implementados en la modificación 2 mejoran sustancialmente los tiempos de ejecución del programa sin mejora y con los cambios introducidos en la modificación 1.

CAPTURAS DE PANTALLA:



B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-modificado.c`

```
_____________________________________

OPTIMIZACIÓN b

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

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

main(){

int ii, i, X1, X2;
const int N=40000;
int tmp0, tmp1, tmp2, tmp3; //para a
int tmp4, tmp5, tmp6, tmp7; //para b
```

```
int *R;
R = (int *) malloc (N*sizeof(int*));

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


for (ii=1; ii<=40000;ii++) {
            for(i=0; i<5000;i+=4){
                                    tmp0+=2*s[i].a+ii;
                                    tmp4+=3*s[i].b-ii;
                                    tmp1+=2*s[i+1].a+ii;
                                    tmp6+=3*s[i+2].b-ii;
                                    tmp2+=2*s[i+2].a+ii;
                                    tmp5+=3*s[i+1].b-ii;
                                    tmp3+=2*s[i+3].a+ii;
                                    tmp7+=3*s[i+3].b-ii;
                                    X1+=tmp0+tmp1+tmp2+tmp3;
                                    X2+=tmp0+tmp1+tmp2+tmp3;

}

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

for (ii=0;ii<=9;ii++)
            printf("%d\n",R[ii]);
}
```

OPTIMIZACIÓN a

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

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


main() {


int ii, i, X1, X2;
const int N=40000;

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

```
  int *R;
  R = (int *) malloc (N*sizeof(int*));


  for (ii=1; ii<=40000;ii++) {
                          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;
   }

for (ii=0;ii<=9;ii++)
     printf("%d\n",R[ii]);


}
                  X2+=3*s[i].b-ii;
            }
      if (X1<X2)  R[ii]=X1;  else  R[ii]=X2;
   }

for (ii=0;ii<=9;ii++)
     printf("%d\n",R[ii]);


}
```

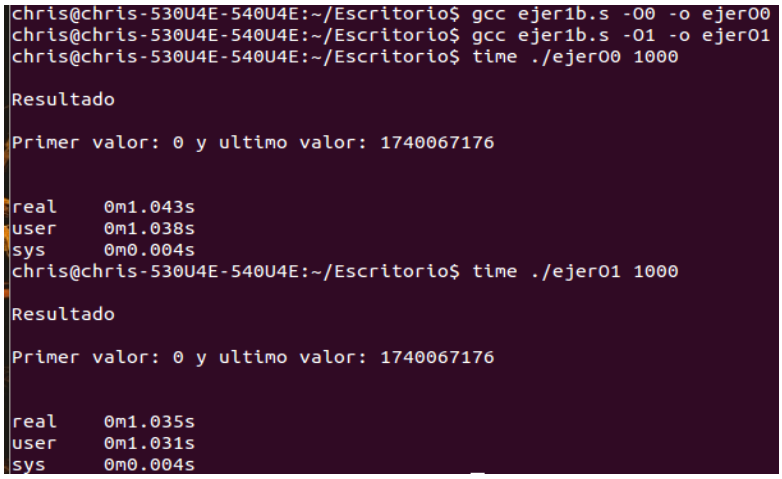
MODIFICACIONES REALIZADAS:

Modificación a) –explicación-: Desenrollado de bucles y unión de ambos bucles

Modificación b) –explicación-: Unión de ambos bucles sin desenrollar.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	1.325s	0.384s	0.371s	0.173s	0.386s
Modificación a)	1.078s	0.359s	0.365s	0.141s	0.303s
Modificación b)	1.042s	0.196s	0.185s	0.181s	0.228s

CAPTURAS DE PANTALLA:



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];
```

   a. Genere los programas en ensamblador para cada una de las opciones de optimización del compilador (-O1, -O2,..) y explique las diferencias que se observan en el código justificando las mejoras en velocidad que acarrean.

   b. (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) y compárela con el valor obtenido para Rmax.

CÓDIGO FUENTE: `daxpy.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>
int main(int argc, char ** argv) {

if (argc<2){
            printf("Faltan componentes del vector\n");
                        exit(1);
}

const int N = atoi(argv[1]);

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

for (i=0;i<N;i++) {
            x[i]=i;
            y[i]=i*100;
}

double a;

for (i=0;i<N;i++)
            a += x[i] * N / 0.093432;
```

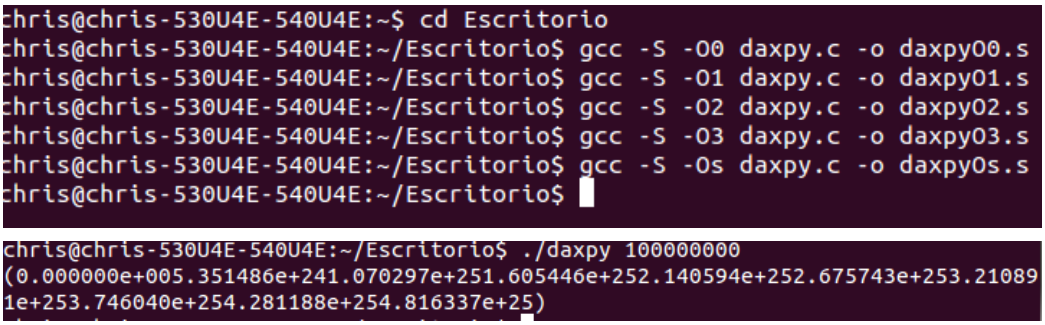
```
for(i=0;i<N;i++)
  y[i]= (a*x[i] + y[i]);


for (i=0;i<10;i++)
            printf(" %e", y[i]);

return (0);
}
```

Tiempos ejec.	-O0	-O1	-O2	-O3	-Os
	3.111s	2.073s	2.023s	1.891s	2.615s

CAPTURAS DE PANTALLA:



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

'-O0': el compilador intenta reducir el tamaño del código y tiempo de ejecución, sin realizar ninguna optimización que toman una gran cantidad de tiempo de compilación, utilizando los registros mucho más en nivel ensamblador. Traducción directa de C a ensamblador.

'-O1': se intenta reducir tanto el tamaño del código, como el tiempo de ejecución. Conseguimos optimizaciones en bloques repetitivos, operaciones con coma flotante, acceso al dato exacto del vector, etc.

'-O2': Con “-O2” conseguimos todas las optimizaciones de “-O1”. Aquí si se activa el desenrollado de bucle y reduce el acceso a pila

'-O3': conseguimos todas las optimizaciones de “-O2” como el desenrollado de bucles y otras prestaciones muy vinculadas con el tipo de procesador.

'-Os': Similar a “-O2” pero intenta reducir las lineas de código de las operaciones mediante métodos como utilizar inc para incrementar el índice en vez de usar addl, etc.

CÓDIGO EN ENSAMBLADOR:

daxpyO0.s

```
.L7:
            movl        36(%esp), %eax
            leal        0(,%eax,8), %edx
            movl        40(%esp), %eax
            addl        %edx, %eax
            movl        36(%esp), %edx
```

```
                leal            0(,%edx,8), %ecx
                movl            44(%esp), %edx
                addl            %ecx, %edx
                fldl            (%edx)
                fmull           48(%esp)
                movl            36(%esp), %edx
                leal            0(,%edx,8), %ecx
                movl            40(%esp), %edx
                addl            %ecx, %edx
                fldl            (%edx)
                faddp           %st, %st(1)
                fstpl           (%eax)
                addl            $1, 36(%esp)
.L6:
                fildl           36(%esp)
                fldl            56(%esp)
                fucomip         %st(1), %st
                fstp            %st(0)
                ja              .L7
                movl            $0, 36(%esp)
                jmp             .L8
```

daxpyO1.s

```
.L10:
                fldl            16(%esp)
                fmull           (%eax,%edx,8)
                faddl           (%edi,%edx,8)
                fstpl           (%edi,%edx,8)
                addl            $1, %edx
                movl            %edx, 44(%esp)
                fildl           44(%esp)
                fldl            24(%esp)
                fucomip         %st(1), %st
                fstp            %st(0)
                ja              .L10
                jmp             .L12
```

daxpyO2.s

```
.L10:
                fldl            (%eax,%edx,8)
                fmul            %st(2), %st
                faddl           (%edi,%edx,8)
                fstpl           (%edi,%edx,8)
                addl            $1, %edx
                movl            %edx, 44(%esp)
                fildl           44(%esp)
                fxch            %st(1)
                fucomi          %st(1), %st
                fstp            %st(1)
                ja              .L10
                fstp            %st(0)
                fstp            %st(0)
                jmp             .L12
.L27:
                fstp            %st(0)
                fstp            %st(0)
                .p2align 4,,7
                .p2align 3
```

daxpyO3.s

```
.L10:
            fldl        (%eax,%edx,8)
            fmul        %st(2), %st
            faddl       (%edi,%edx,8)
            fstpl       (%edi,%edx,8)
            addl        $1, %edx
            movl        %edx, 44(%esp)
            fildl       44(%esp)
            fxch        %st(1)
            fucomi      %st(1), %st
            fstp        %st(1)
            ja          .L10
            fstp        %st(0)
            fstp        %st(0)
            jmp         .L12
```

daxpyOs.s

```
.L6:
            movl        %edx, -28(%ebp)
            fildl       -28(%ebp)
            fxch        %st(1)
            fucomi      %st(1), %st
            fstp        %st(1)
            jbe         .L14
            fldl        (%eax,%edx,8)
            fmul        %st(2), %st
            faddl       (%edi,%edx,8)
            fstpl       (%edi,%edx,8)
            incl        %edx
            jmp         .L6
```