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): Francisco Javier Bolívar Lupiáñez
Grupo de prácticas: B1
Fecha de entrega: 27/05/2014
Fecha evaluación en clase:

Versión de gcc utilizada: *4.8.2*

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                   : 30
model name      : Intel(R) Core(TM) i5 CPU         760  @ 2.80GHz
stepping        : 5
microcode       : 0x3
cpu MHz                 : 1200.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id                 : 0
cpu cores       : 4
apicid                  : 0
initial apicid          : 0
fpu                     : yes
fpu_exception   : yes
cpuid level     : 11
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 pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16
xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm ida dtherm tpr_shadow vnmi flexpriority
ept vpid
bogomips        : 5585.82
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                   : 30
model name      : Intel(R) Core(TM) i5 CPU         760  @ 2.80GHz
stepping        : 5
microcode       : 0x3
cpu MHz                 : 1200.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
```

```
core id                 : 1
cpu cores       : 4
apicid                  : 2
initial apicid          : 2
fpu                     : yes
fpu_exception   : yes
cpuid level     : 11
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 pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16
xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm ida dtherm tpr_shadow vnmi flexpriority
ept vpid
bogomips        : 5585.82
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                   : 30
model name      : Intel(R) Core(TM) i5 CPU         760  @ 2.80GHz
stepping        : 5
microcode       : 0x3
cpu MHz                 : 1200.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id                 : 2
cpu cores       : 4
apicid                  : 4
initial apicid          : 4
fpu                     : yes
fpu_exception   : yes
cpuid level     : 11
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 pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16
xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm ida dtherm tpr_shadow vnmi flexpriority
ept vpid
bogomips        : 5585.82
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                   : 30
model name      : Intel(R) Core(TM) i5 CPU         760  @ 2.80GHz
stepping        : 5
microcode       : 0x3
cpu MHz                 : 1200.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
```

```
core id                     : 3
cpu cores     : 4
apicid                      : 6
initial apicid              : 6
fpu                         : yes
fpu_exception : yes
cpuid level   : 11
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 pni dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16
xtpr pdcm sse4_1 sse4_2 popcnt lahf_lm ida dtherm tpr_shadow vnmi flexpriority
ept vpid
bogomips      : 5585.82
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

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

main()
{
   …
   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;
   }
   …
}
```

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

A) MULTIPLICACIÓN DE MATRICES:

CÓDIGO FUENTE: pmm-secuencial-modificadoA.c

```
// Modificacion A): Desenrollado de bucles

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

//#define TIMES
//#define PRINTF_ALL

main(int argc, char **argv) {
  // 1. Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  if(n%5!=0) {
    fprintf(stderr,"num debe ser divisible entre 5\n");
    exit(-1);
  }
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  // 2. Creación e inicialización de vector y matriz
  // 2.1. Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  // 2.2. Inicialización
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
      A[i][j]=0;
    }
  }

  // 3. Impresión de vector y matriz
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
```

```
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  int tmp0,tmp1,tmp2,tmp3,tmp4;

  // 4. Cálculo resultado
  clock_gettime(CLOCK_REALTIME,&ini);

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      tmp0=tmp1=tmp2=tmp3=tmp4=0;
      for (k=0; k<n; k+=5) {
        tmp0=tmp0+B[i][k]*C[k][j];
        tmp1=tmp1+B[i][k+1]*C[k+1][j];
        tmp2=tmp2+B[i][k+2]*C[k+2][j];
        tmp3=tmp3+B[i][k+3]*C[k+3][j];
        tmp4=tmp4+B[i][k+4]*C[k+4][j];
      }
      A[i][j]=tmp0+tmp1+tmp2+tmp3+tmp4;
    }
  }

  clock_gettime(CLOCK_REALTIME,&fin);
  transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

  // 5. Impresión de vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  // 6. Eliminar de memoria
  free(A);
  free(B);
  free(C);
}
```

CÓDIGO FUENTE: pmm-secuencial-modificadoB.c

```
// Modificacion B): Quitando desenrollado y cambiar el orden de los bucles
para mejorar el acceso a memoria por la localidad de los elemntos

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

//#define TIMES
//#define PRINTF_ALL

main(int argc, char **argv) {
  // 1. Lectura valores de entrada
  if(argc < 2) {
    fprintf(stderr,"Falta num\n");
    exit(-1);
  }
  int n = atoi(argv[1]);
  if(n%5!=0) {
    fprintf(stderr,"num debe ser divisible entre 5\n");
    exit(-1);
  }
  int i,j,k;
  struct timespec ini,fin; double transcurrido;

  // 2. Creación e inicialización de vector y matriz
  // 2.1. Creación
  int **A, **B, **C;
  A = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    A[i] = (int*)malloc(n*sizeof(int));

  B = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    B[i] = (int*)malloc(n*sizeof(int));

  C = (int**) malloc(n*sizeof(int*));
  for(i=0;i<n;i++)
    C[i] = (int*)malloc(n*sizeof(int));

  // 2.2. Inicialización
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      B[i][j]=n*i+j;
      C[i][j]=n*i+j;
      A[i][j]=0;
    }
  }

  // 3. Impresión de vector y matriz
  #ifndef TIMES
    #ifdef PRINTF_ALL
      printf("Matriz inicial B:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(B[i][j]<10) printf(" %d ",B[i][j]);
          else printf("%d ",B[i][j]);
        }
        printf("\n");
      }
      printf("Matriz inicial C:\n");
      for (i=0; i<n; i++) {
```

```
        for (j=0; j<n; j++) {
          if(C[i][j]<10) printf(" %d ",C[i][j]);
          else printf("%d ",C[i][j]);
        }
        printf("\n");
      }
    #endif
  #endif

  // 4. Cálculo resultado
  clock_gettime(CLOCK_REALTIME,&ini);

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

  // 5. Impresión de vector resultado
  #ifdef TIMES
    printf("%d %11.9f\n",n,transcurrido);
  #else
    #ifdef PRINTF_ALL
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("Matriz resultado A=B*C:\n");
      for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
          if(A[i][j]<10) printf(" %d ",A[i][j]);
          else printf("%d ",A[i][j]);
        }
        printf("\n");
      }
    #else
      printf("Tiempo: \%11.9f\n",transcurrido);
      printf("A[0][0]: %d, A[n-1][n-1]: %d\n",A[0][0],A[n-1][n-1]);
    #endif
  #endif

  // 6. Eliminar de memoria
  free(A);
  free(B);
  free(C);
}
```

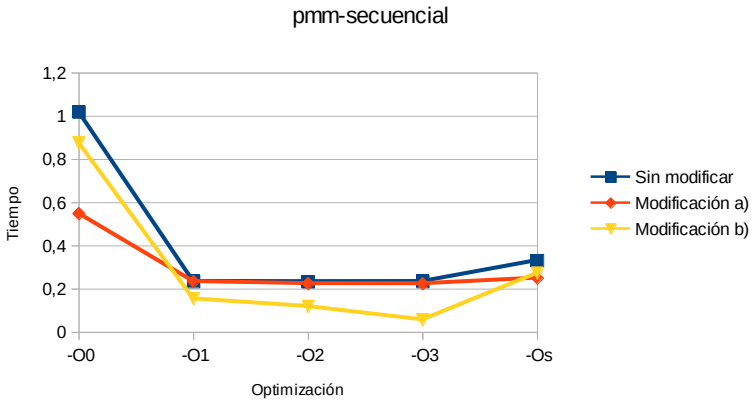
MODIFICACIONES REALIZADAS:

Modificación a) –explicación-: Desenrollado de bucle. En vez de realizar un salto en cada iteración, se realizará un salto cada cinco iteraciones reduciendo así el número de instrucciones.

Modificación b) –explicación-: Se elimina el desenrollado de bucle, pero a cambio se hace un cambio en el orden de ejecución de los bucles (i, k, j en lugar de i, j, k). Con este cambio, se accede a elementos consecutivos cuando se llama a M[k][j].

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	0,1255774	0,0245569	0,0223201	0,0223654	0,0393739
Modificación a)	0,0647983	0,0199654	0,0196289	0,0198285	0,0253895
Modificación b)	0,87603339	0,15455048	0,11883410	0,05700969	0,27288225

**tama 500*



COMENTARIOS SOBRE LOS RESULTADOS: En el caso de la **primera modificación**, la mejora en rendimiento más notable se observa con los niveles -O0 y -Os. En el resto de niveles, se asimila bastante el tiempo que se ha obtenido con y sin el desenrollado, por lo que podríamos pensar que el compilador hace una especie de desenrollado para mejorar el código ensamblador generado. Además, entre los niveles -O1, -O2 y -O3 apenas se obtiene una mejora significativa. En el caso de la segunda modificación, la mejora más significativa se encuentra en los niveles -O1, -O2 y -O3. En los otros dos niveles, pese a mejorarse el rendimiento con respecto al código sin modificar, se empeora con respecto a la primera modificación.

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-modificadoA.c`

```
// Modificacion A): Desenrollado de bucles

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

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

main()
{
    int ii,i,X1,X2;
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
```

```
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=2*s[i].a+ii;
            tmpX1_1+=2*s[i+1].a+ii;
            tmpX1_2+=2*s[i+2].a+ii;
            tmpX1_3+=2*s[i+3].a+ii;
            tmpX1_4+=2*s[i+4].a+ii;
             }
        for(i=0; i<5000;i+=5) {
            tmpX2_0+=3*s[i].b-ii;
            tmpX2_1+=3*s[i+1].b-ii;
            tmpX2_2+=3*s[i+2].b-ii;
            tmpX2_3+=3*s[i+3].b-ii;
            tmpX2_4+=3*s[i+4].b-ii;
        }
        // sumar temporales
        X1=tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4;
        X2=tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4;
        // comprobacion
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

CÓDIGO FUENTE: `figura1-modificadoB.c`

```
// Modificacion B): Eliminacion de un bucle innecesario

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

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

main()
{
    int ii,i,X1,X2;
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
```

```
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=2*s[i].a+ii;
            tmpX1_1+=2*s[i+1].a+ii;
            tmpX1_2+=2*s[i+2].a+ii;
            tmpX1_3+=2*s[i+3].a+ii;
            tmpX1_4+=2*s[i+4].a+ii;

            tmpX2_0+=3*s[i].b-ii;
            tmpX2_1+=3*s[i+1].b-ii;
            tmpX2_2+=3*s[i+2].b-ii;
            tmpX2_3+=3*s[i+3].b-ii;
            tmpX2_4+=3*s[i+4].b-ii;
        }
        // sumar temporales
        X1=tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4;
        X2=tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4;
        // comprobacion
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

CÓDIGO FUENTE: `figura1-modificadoC.c`

```
// Modificacion C): Sacar multiplicacion del interior de un bucle

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

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

main()
{
    int ii,i,X1,X2;
    int tmpX1_0,tmpX1_1,tmpX1_2,tmpX1_3,tmpX1_4;
    int tmpX2_0,tmpX2_1,tmpX2_2,tmpX2_3,tmpX2_4;
    int R[40000];

    struct timespec ini,fin;
    double transcurrido;

    clock_gettime(CLOCK_REALTIME,&ini);

    for (ii=1; ii<=40000;ii++) {
        // iniciar temporales
        tmpX1_0=tmpX1_1=tmpX1_2=tmpX1_3=tmpX1_4=0.0;
        tmpX2_0=tmpX2_1=tmpX2_2=tmpX2_3=tmpX2_4=0.0;
        // calcular en temporales
        for(i=0; i<5000;i+=5) {
            tmpX1_0+=s[i].a+ii;
            tmpX1_1+=s[i+1].a+ii;
```

```
            tmpX1_2+=s[i+2].a+ii;
            tmpX1_3+=s[i+3].a+ii;
            tmpX1_4+=s[i+4].a+ii;

            tmpX2_0+=s[i].b-ii;
            tmpX2_1+=s[i+1].b-ii;
            tmpX2_2+=s[i+2].b-ii;
            tmpX2_3+=s[i+3].b-ii;
            tmpX2_4+=s[i+4].b-ii;
        }
        // sumar temporales
        X1=(tmpX1_0+tmpX1_1+tmpX1_2+tmpX1_3+tmpX1_4)*2;
        X2=(tmpX2_0+tmpX2_1+tmpX2_2+tmpX2_3+tmpX2_4)*3;
        // comprobacion
        if (X1<X2)  R[ii]=X1;
        else  R[ii]=X2;
    }

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    printf("Tiempo(seg): %f\nR[0]=%d, R[39999]=%d
\n",transcurrido,R[0],R[39999]);
}
```

MODIFICACIONES REALIZADAS:

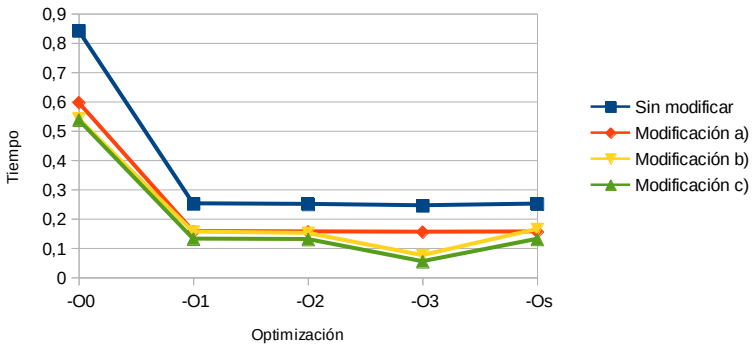
Modificación a) –explicación-: Desenrollado de bucle. En vez de realizar un salto en cada iteración, se realizará un salto cada cuatro iteraciones reduciendo así el número de instrucciones.

Modificación b) –explicación-: Además de incluir la modificación anterior, se unen dos bucles, para evitar recorrer dos veces nuestra estructura y hacerlo solo una vez haciendo dos operaciones por índice.

Modificación c) –explicación-: Además de incluir todo lo anterior, se hace una multiplicación que se hacía en el interior del bucle fuera para no tener que realizar n operaciones y hacerla solo una vez.

Modificación	-O0	-O1	-O2	-O3	-Os
Sin modificar	0,851999	0,254306	0,250665	0,247389	0,252845
Modificación a)	0,548735	0,161289	0,151445	0,069772	0,165477
Modificación b)	0,5419	0,155548	0,15099	0,075354	0,164118
Modificación c)	0,537638	0,131626	0,130334	0,054444	0,130836

figura1



COMENTARIOS SOBRE LOS RESULTADOS: En este caso, con la **primera modificación** si se observa una mejora de rendimiento general en todos los niveles. Con la **segunda modificación**, además, con el nivel **-**O3 se obtiene una mejora significativa con respecto al nivel -O2, algo que no se produjo en el caso anterior. Ya con la **tercera modificación** se sigue mejorando para obtener tiempos finales muy inferiores a los obtenidos sin el código optimizado.

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.

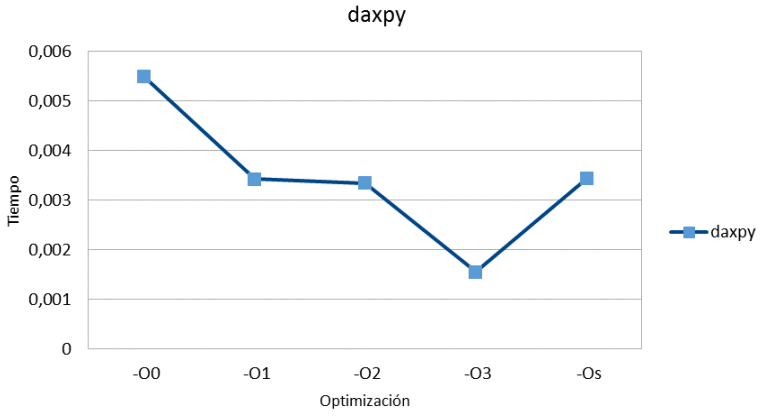
CÓDIGO FUENTE: `daxpy.c`

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

main(int argc, char **argv)
{
    // 1. Lectura valores de entrada
    if(argc < 2) {
        fprintf(stderr,"Falta num\n");
        exit(-1);
    }
    int N = atoi(argv[1]);

    struct timespec ini,fin;
    double transcurrido;

    // 2. Creación e inicialización de vector y matriz
    // 2.1. Creación
    int i,a=47;
    int x[N], y[N];

    // 2.2. Inicialización
    for (i=1; i<=N;i++) {
        x[i]=i;
        y[i]=i;
```

```
    }

    // 3. Cálculo resultado
    clock_gettime(CLOCK_REALTIME,&ini);

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

    clock_gettime(CLOCK_REALTIME,&fin);
    transcurrido=(double) (fin.tv_sec-ini.tv_sec)+(double) ((fin.tv_nsec-
ini.tv_nsec)/(1.e+9));

    // 4. Impresión de vector resultado
    printf("Tiempo(seg): %f\ny[0]=%d, y[N-1]=%d \n",transcurrido,y[0],y[N-1]);
}
```

Tiempos ejec.	-O0	-O1	-O2	-O3	-Os
	0,005494	0,003425	0,003351	0,001556	0,003287

**tama 1000000*



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

- **-O0:** Con este nivel se desconecta por completo la optimización y el código, por tanto, no se optimizará.
- **-O1:** El tiempo en compilar es algo mayor aunque no se nota diferencia aquí. La diferencia se encuentra en el código ensamblador, donde se reducen el número de líneas de forma notoria (de 200 a 127) generándose más subrutinas pero de un número de instrucciones mucho menor. En el apartado de la suma de vectores, el número de instrucciones se reduce de forma notoria.
- **-O2:** Este es el nivel de optimización recomendado. El número de instrucciones sigue siendo similar al que hay en el nivel anterior, pero hay una mayor mejora. En comparación con el nivel -O1, el número de instrucciones en la parte de la suma de vectores sigue siendo más o menos el mismo, pero las instrucciones no son las mismas.
- **-O3:** Este es el nivel de optimización más alto, activa opciones que son caras en términos de tiempo de compilación y uso de memoria y, aunque este no es el caso, hay ocasiones en las que no se obtiene una mejora de rendimiento. La parte de la suma de vectores, usando este nivel, es mucho más ilegible que el resto. Crea muchas subrutinas a las que llama (aquí vemos lo que acabo de comentar del uso de memoria), pero al fin y al cabo y viendo la tabla de tiempos anterior, el rendimiento es mayor.

- **-Os:** Con este nivel se optimiza el tamaño del ejecutable usándose todas las opciones de -O2 que no aumenten el tamaño del código. En la suma de vectores, el código se parece mucho al generado por los niveles -O1 y -O2.

CÓDIGO EN ENSAMBLADOR:

daxpyO0.s

```
            jmp         .L5
.L6:
            movq        -112(%rbp), %rax
            movl        -132(%rbp), %edx
            movslq      %edx, %rdx
            movl        (%rax,%rdx,4), %eax
            imull       -124(%rbp), %eax
            movl        %eax, %ecx
            movq        -96(%rbp), %rax
            movl        -132(%rbp), %edx
            movslq      %edx, %rdx
            movl        (%rax,%rdx,4), %eax
            addl        %eax, %ecx
            movq        -96(%rbp), %rax
            movl        -132(%rbp), %edx
            movslq      %edx, %rdx
            movl        %ecx, (%rax,%rdx,4)
            addl        $1, -132(%rbp)
.L5:
            movl        -132(%rbp), %eax
            cmpl        -128(%rbp), %eax
            jle         .L6
```

daxpyO1.s

```
.L7:
            movslq      %eax, %rdx
            imull       $47, (%r12,%rdx,4), %ecx
            addl        %ecx, (%rbx,%rdx,4)
            addl        $1, %eax
            cmpl        %r13d, %eax
            jle         .L7
```

daxpyO2.s

```
.L7:
            movl        4(%r12,%rdx), %ecx
            movl        $47, %eax
            imull       %eax, %ecx
            addl        %ecx, 4(%rbx,%rdx)
            addq        $4, %rdx
            cmpq        %rsi, %rdx
            jne         .L7
```

daxpyO3.s

```
.L7:
            leaq        -80(%rbp), %rsi
            xorl        %edi, %edi
            call        clock_gettime
            leaq        4(%r12), %rax
            andl        $15, %eax
            shrq        $2, %rax
            negq        %rax
            andl        $3, %eax
            cmpl        %ebx, %eax
            cmova       %ebx, %eax
```

```
                cmpl            $4, %ebx
                cmovbe          %ebx, %eax
                testl           %eax, %eax
                je              .L33
                imull           $47, 4(,%r13,4), %edx
                addl            %edx, 4(%r12)
                cmpl            $1, %eax
                jbe             .L34
                imull           $47, 8(,%r13,4), %edx
                addl            %edx, 8(%r12)
                cmpl            $2, %eax
                jbe             .L35
                imull           $47, 12(,%r13,4), %edx
                addl            %edx, 12(%r12)
                cmpl            $3, %eax
                jbe             .L36
                imull           $47, 16(,%r13,4), %edx
                addl            %edx, 16(%r12)
                movl            $5, %edx
.L36:
                movl            $4, %edx
                jmp             .L20
.L34:
                movl            $2, %edx
                jmp             .L20
.L35:
                movl            $3, %edx
                jmp             .L20
.L3:
                leaq            -80(%rbp), %rsi
                xorl            %edi, %edi
                call            clock_gettime
                jmp             .L24
```

daxpyOs.s

```
.L5:
                cmpl            %ebx, %eax
                jg              .L10
                movslq          %eax, %rdx
                incl            %eax
                imull           $47, (%r14,%rdx,4), %ecx
                addl            %ecx, 0(%r13,%rdx,4)
                jmp             .L5
```