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): Daniel Bolaños Martínez
Grupo de prácticas: A1
Fecha de entrega: 01/06/2017
Fecha evaluación en clase: 02/06/2017

Denominación de marca del chip de procesamiento o procesador (se encuentra en /proc/cpuinfo): *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz x4*

Sistema operativo utilizado: *Ubuntu 16.04.2 LTS*

Versión de gcc utilizada: *gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609*

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)

```
/* pmm-secuencial.c
 Multiplicación de matriz por matriz (cuadradas): M3 = M1*M2
 Para compilar usar (-lrt: real time library):
 gcc -O2 pmm-secuencial.c -o pmm-secuencial -lrt

 Para ejecutar use: pmm-secuencial longitud
*/

 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
 #include <stdio.h> // biblioteca donde se encuentra la función printf()
 #include <time.h> // biblioteca donde se encuentra la función clock_gettime()
```

```
 //#define PRINTF_ALL // comentar para quitar el printf ...
                                 // que imprime todos los componentes

 //Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
 //un defines siguientes puede estar descomentado):
 #define PMV_GLOBAL // descomentar para que los vector/matriz sean
variables ...
                                   // globales (su longitud no estará limitada
por el ...
                                  // tamaño de la pila del programa)
 #ifdef PMV_GLOBAL
 #define MAX 8192 // 2^13
 double M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX];
 #endif

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

    int i, j, k;
    struct timespec cgt1,cgt2;   //para tiempo de ejecución

    //Leer argumento de entrada (no de componentes del vector/matriz)
    if (argc<2){
       printf("Faltan no componentes del vector/matriz\n");
       exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

    #ifdef PMV_GLOBAL
    if (N>MAX) N=MAX;
    #endif

    //Inicializar Matrices
    for(i=0; i<N; i++){
       for(j=0; j<N; j++){
          M1[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
          M2[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
          M3[i][j] = 0; //los valores dependen de N
       }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Calcular producto de matrices

    for(i=0; i<N; i++){
       for(j=0; j<N; j++){
          for(k=0; k<N; k++){
            M3[i][j] += M1[i][k] * M2[k][j];
          }
       }
    }

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

    //Imprimir resultado de la suma y el tiempo de ejecución
     #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\n",ncgt,N);
```

```
    for(i=0; i<N; i++){
       for(j=0; j<N; j++)
          printf("%8.6f ", i, j, M3[i][j]);
    printf("\n");
    }
    #else
       printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t/ M3[0][0]=%8.6f / /
M3[%d][%d]=%8.6f /\n", ncgt,N, M3[0][0], N-1, N-1, M3[N-1][N-1]);
    #endif

    return 0;
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –Desenrollado de bucle-: Desenrrollado del bucle de multiplicación de las matrices.
- Versión 1: Desenrollado del bucle de la iteración j en 8 bloques.
- Versión 2: Desenrollado del bucle de la iteración k en 8 bloques.

Modificación b) –Localidad de los accesos-: Intercambio del orden de los bucles j y k para aprovechar la localidad + Multiplicar una matriz por la otra traspuesta.
- Versión 1: Utilización de una matriz auxiliar para la realización de las trasposición
- Versión 2: Utilización de una variable auxiliar para la realización de las trasposición

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificado_a1.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* pmm-secuencial-modificado_a.c
 Multiplicación de matriz por matriz (cuadradas): M3 = M1*M2
 Para compilar usar (-lrt: real time library):
 gcc -O2 pmm-secuencial-modificado_a.c -o pmm-secuencial_a -lrt

 Para ejecutar use: pmm-secuencial_a longitud
*/

 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y free()
 #include <stdio.h> // biblioteca donde se encuentra la función printf()
 #include <time.h> // biblioteca donde se encuentra la función clock_gettime()

 //#define PRINTF_ALL // comentar para quitar el printf ...
                                // que imprime todos los componentes

 //Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno de
los ...
 //un defines siguientes puede estar descomentado):
 #define PMV_GLOBAL // descomentar para que los vector/matriz sean
variables ...
                                 // globales (su longitud no estará limitada
por el ...
                                // tamaño de la pila del programa)

 #ifdef PMV_GLOBAL
 #define MAX 8192 // 2^13
 double M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX];
 #endif

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

```
    int i, j, k;
    double s0=0, s1=0, s2=0, s3=0, s4=0, s5=0, s6=0, s7=0;
    struct timespec cgt1,cgt2;   //para tiempo de ejecución

    //Leer argumento de entrada (no de componentes del vector/matriz)
    if (argc<2){
       printf("Faltan no componentes del vector/matriz\n");
       exit(-1);
    }
    if (atoi(argv[1]) % 8 != 0){
       printf("La matriz no es múltiplo del desenrollado de los bucles. Pruebe
con un múltiplo de 8.\n");
       exit(-1);
    }

    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

    #ifdef PMV_GLOBAL
    if (N>MAX) N=MAX;
    #endif

    //Inicializar Matrices
    for(i=0; i<N; i++){
       for(j=0; j<N; j++){
          M1[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
          M2[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
          M3[i][j] = 0; //los valores dependen de N
       }
    }

    clock_gettime(CLOCK_REALTIME,&cgt1);
    //Calcular producto de matrices

    /*Versión 1: Desenrollado del bucle j*/
    for(i=0; i<N; i++){
       s0=s1=s2=s3=s4=s5=s6=s7=0;
       for(j=0; j<N; j+=8){
          for (k=0; k < N; k++){
             M3[i][j] += M1[i][k]*M2[k][j];
             M3[i][j+1] += M1[i][k]*M2[k][j+1];
             M3[i][j+2] += M1[i][k]*M2[k][j+2];
             M3[i][j+3] += M1[i][k]*M2[k][j+3];
             M3[i][j+4] += M1[i][k]*M2[k][j+4];
             M3[i][j+5] += M1[i][k]*M2[k][j+5];
             M3[i][j+6] += M1[i][k]*M2[k][j+6];
             M3[i][j+7] += M1[i][k]*M2[k][j+7];

          }
       }
    }

    /*Versión 2: Desenrollado del bucle k
    for(i=0; i<N; i++){
       for(j=0; j<N; j++){
          s0=s1=s2=s3=s4=s5=s6=s7=0;
          for (k=0; k < N; k+=8){
             s0 += M1[i][k]*M2[k][j];
             s1 += M1[i][k+1]*M2[k+1][j];
             s2 += M1[i][k+2]*M2[k+2][j];
             s3 += M1[i][k+3]*M2[k+3][j];
```

```
            s4 += M1[i][k+4]*M2[k+4][j];
            s5 += M1[i][k+5]*M2[k+5][j];
            s6 += M1[i][k+6]*M2[k+6][j];
            s7 += M1[i][k+7]*M2[k+7][j];

         }
         M3[i][j] = s0 + s1 + s2 + s3 + s4 + s5 + s6 + s7;
      }
   }*/

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

    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\n",ncgt,N);
    for(i=0; i<N; i++){
       for(j=0; j<N; j++)
          printf("%8.6f ", i, j, M3[i][j]);
    printf("\n");
    }
    #else
       printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t/ M3[0][0]=%8.6f / /
M3[%d][%d]=%8.6f /\n", ncgt,N, M3[0][0], N-1, N-1, M3[N-1][N-1]);
    #endif

    return 0;
}
```

b) `pmm-secuencial-modificado_b2.c`

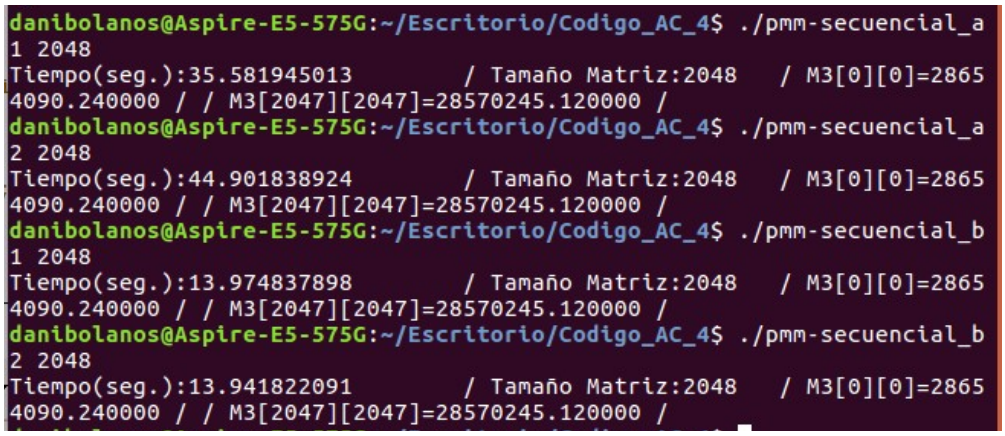
(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
/* pmm-secuencial-modificado_b.c
 Multiplicación de matriz por matriz (cuadradas): M3 = M1*M2
 Para compilar usar (-lrt: real time library):
 gcc -O2 pmm-secuencial-modificado_b.c -o pmm-secuencial_b -lrt

 Para ejecutar use: pmm-secuencial_b longitud
*/

 #include <stdlib.h> // biblioteca con funciones atoi(), malloc() y
free()
 #include <stdio.h> // biblioteca donde se encuentra la función
printf()
 #include <time.h> // biblioteca donde se encuentra la función
clock_gettime()

 //#define PRINTF_ALL // comentar para quitar el printf ...
                              // que imprime todos los componentes

 //Sólo puede estar definida uno de los dos constantes PMV_ (sólo uno
de los ...
 //un defines siguientes puede estar descomentado):
 #define PMV_GLOBAL // descomentar para que los vector/matriz sean
variables ...
                               // globales (su longitud no estará
limitada por el ...
```

```
                                // tamaño de la pila del programa)
#ifdef PMV_GLOBAL
#define MAX 8192 // 2^13
double M1[MAX][MAX], M2[MAX][MAX], M3[MAX][MAX], M2T[MAX][MAX];
#endif

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

   int i, j, k;

   struct timespec cgt1,cgt2;   //para tiempo de ejecución

   double aux;

   //Leer argumento de entrada (no de componentes del vector/matriz)
   if (argc<2){
      printf("Faltan no componentes del vector/matriz\n");
      exit(-1);
   }

   unsigned int N = atoi(argv[1]); // Máximo N =2^32-1=4294967295
(sizeof(unsigned int) = 4 B)

   #ifdef PMV_GLOBAL
   if (N>MAX) N=MAX;
   #endif

   //Inicializar Matrices
   for(i=0; i<N; i++){
      for(j=0; j<N; j++){
         M1[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
         M2[i][j] = N*0.1-i*0.1-j*0.1; //los valores dependen de N
         M3[i][j] = 0; //los valores dependen de N
      }
   }

   clock_gettime(CLOCK_REALTIME,&cgt1);

   //Calcular traspuesta de matriz

   /*Versión 1: Utilizando una matriz temporal
   for(i=0; i<N; i++){
      for(j=0; j<N; j++){
           M2T[i][j] = M2[j][i];
      }
   }*/

   /*Versión 2: Utilizando una variable temporal*/
   for(i=0; i<N; i++){
      for(j=i+1; j<N; j++){
           aux = M2[i][j];
               M2[j][i] = M2[i][j];
               M2[j][i] = aux;
      }
   }
```

```
    //Calcular producto de matrices

    // Intercambio del orden de los bucles j y k para aprovechar la
localidad

    for(i=0; i<N; i++){
       for(k=0; k<N; k++){
          for(j=0; j<N; j++){
            M3[i][j] += M1[i][k] * M2[k][j]; //M2T si utilizamos la
versión 1
          }
       }
    }

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

    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\n",ncgt,N);
    for(i=0; i<N; i++){
       for(j=0; j<N; j++)
          printf("%8.6f ", i, j, M3[i][j]);
    printf("\n");
    }
    #else
       printf("Tiempo(seg.):%11.9f\t / Tamaño Matriz:%u\t/ M3[0][0]=
%8.6f / / M3[%d][%d]=%8.6f /\n", ncgt,N, M3[0][0], N-1, N-1, M3[N-1]
[N-1]);
    #endif

    return 0;
}
```

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



```
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial 8
Tiempo(seg.):0.000002562         / Tamaño Matriz:8
2.040000 1.680000 1.320000 0.960000 0.600000 0.240000 -0.120000 -0.480000
1.680000 1.400000 1.120000 0.840000 0.560000 0.280000 -0.000000 -0.280000
1.320000 1.120000 0.920000 0.720000 0.520000 0.320000 0.120000 -0.080000
0.960000 0.840000 0.720000 0.600000 0.480000 0.360000 0.240000 0.120000
0.600000 0.560000 0.520000 0.480000 0.440000 0.400000 0.360000 0.320000
0.240000 0.280000 0.320000 0.360000 0.400000 0.440000 0.480000 0.520000
-0.120000 -0.000000 0.120000 0.240000 0.360000 0.480000 0.600000 0.720000
-0.480000 -0.280000 -0.080000 0.120000 0.320000 0.520000 0.720000 0.920000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial_a1 8
Tiempo(seg.):0.000005460         / Tamaño Matriz:8
2.040000 1.680000 1.320000 0.960000 0.600000 0.240000 -0.120000 -0.480000
1.680000 1.400000 1.120000 0.840000 0.560000 0.280000 -0.000000 -0.280000
1.320000 1.120000 0.920000 0.720000 0.520000 0.320000 0.120000 -0.080000
0.960000 0.840000 0.720000 0.600000 0.480000 0.360000 0.240000 0.120000
0.600000 0.560000 0.520000 0.480000 0.440000 0.400000 0.360000 0.320000
0.240000 0.280000 0.320000 0.360000 0.400000 0.440000 0.480000 0.520000
-0.120000 -0.000000 0.120000 0.240000 0.360000 0.480000 0.600000 0.720000
-0.480000 -0.280000 -0.080000 0.120000 0.320000 0.520000 0.720000 0.920000
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial_b2 8
Tiempo(seg.):0.000010673         / Tamaño Matriz:8
2.040000 1.680000 1.320000 0.960000 0.600000 0.240000 -0.120000 -0.480000
1.680000 1.400000 1.120000 0.840000 0.560000 0.280000 -0.000000 -0.280000
1.320000 1.120000 0.920000 0.720000 0.520000 0.320000 0.120000 -0.080000
0.960000 0.840000 0.720000 0.600000 0.480000 0.360000 0.240000 0.120000
0.600000 0.560000 0.520000 0.480000 0.440000 0.400000 0.360000 0.320000
0.240000 0.280000 0.320000 0.360000 0.400000 0.440000 0.480000 0.520000
-0.120000 -0.000000 0.120000 0.240000 0.360000 0.480000 0.600000 0.720000
-0.480000 -0.280000 -0.080000 0.120000 0.320000 0.520000 0.720000 0.920000
```

1.1. TIEMPOS:

```
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ gcc -O2 pmm-secuenc
ial.c -o pmm-secuencial -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ gcc -O2 pmm-secuenc
ial-modificado_a1.c -o pmm-secuencial_a1 -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ gcc -O2 pmm-secuenc
ial-modificado_b2.c -o pmm-secuencial_b2 -lrt
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial 10
24
Tiempo(seg.):12.972813289        / Tamaño Matriz:1024   / M3[0][0]=35843
84.000000 / / M3[1023][1023]=3563432.960000 /
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial_a1
 1024
Tiempo(seg.):4.416346236         / Tamaño Matriz:1024   / M3[0][0]=35843
84.000000 / / M3[1023][1023]=3563432.960000 /
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./pmm-secuencial_b2
 1024
Tiempo(seg.):3.475491966         / Tamaño Matriz:1024   / M3[0][0]=35843
84.000000 / / M3[1023][1023]=3563432.960000 /
```

Modificación	**-O2**
Sin modificar	12.972813289
Modificación a1)	4.416346236
Modificación b2)	3.475491966

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Para la resolución de este ejercicio, he realizado dos tipos de modificaciones para optimizar el código secuencial de la multiplicación de matrices. Primeramente he dividido las mejoras en dos, una orientada al desenrollado de bucles y otra a la localidad de los accesos a memoria.

El desenrollado lo he hecho de 8 `bloques` y he probado con dos versiones, una con el bucle de la `j` y otro con el de la `k`. Probado para una matriz de orden 2048, la versión 1, obtiene mejores

resultados que la 2. Una diferencia de:

44.901838924 - 35.581945013 = **9.319893911 s**

Lo que obviamente me hace decantar por la primera opción. Desenrollado del bucle `j` (más externo).

La localidad de acceso a memoria la he hecho utilizando varias modificaciones. La primera, una aconsejada en clase, que consiste en trasponer la segunda matriz antes de multiplicarla y la segunda modificación la he realizado invirtiendo el orden de los bucles `j` y `k` en el código. La segunda modificación la probé en clase y obtenía mejores tiempos que la versión secuencial normal por lo que la incluí en el código.

Pero la primera (trasponer la matriz) me hizo dudar entre dos versiones: trasponer la matriz utilizando una matriz auxiliar o sobre la misma matriz utilizando una variable temporal. Probado para una matriz de orden 2048, la versión 2, obtiene mejores resultados que la 1. Una diferencia de:

13.974837898– 13.941822091 = **0.033015807 s**

No es tanta diferencia como en el primer caso, pero prefiero quedarme con la versión 2, la que traspone sobre la misma matriz usando la variable temporal.

Finalmente para matrices de orden 1024, podemos observar que la mejor opción es la b2, es decir, la que utiliza localidad de acceso a memoria, seguida por la que utiliza desenrollado de bucles. Por tanto llegamos a la conclusión de que optimizar teniendo en cuenta los accesos a memoria es muy buena opción para obtener códigos más eficientes.

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	pmm-secuencial-modificado_a1.s

leaq 0(,
%rbx,8), %rax
salq
$16, %rbx
leaq -
8(,%r13,8), %r10
movl
$M1, %r8d
leaq
M3(%rax,%rbx),
%r11
leaq
M3(%rax), %rcx
movq
%r13, %rax
salq
$16, %rax
addq
$M2, %rbx
leaq -
65536(%rax), %r9
.L7:
leaq
	leaq M1(%rbx), %rax
	movq $M1, 8(%rsp)
	movq $M3, 16(%rsp)
	movq %rax, 32(%rsp)
	leaq 8(,%rbp,8), %rax
	movq %rax, 40(%rsp)
	.L8:
	movq 40(%rsp), %r12
	addq 8(%rsp), %r12
	movl $M2, %r15d
	movq 16(%rsp), %r13
	movq $-131072, %rdi
	xorl %r14d, %r14d
	.p2align 4,,10
	.p2align 3
	.L12:
	movq 8(%rsp), %rdx
	movl $M2, %ecx
	movq $-65536, %rbp
	movl $65536, %ebx
	movl $131072, %r11d
	movl $196608, %r10d
	movl $262144, %r9d

```
                (%r10,%rcx), %rdx
                movq
                %rbx, %rdi
                .p2align 4,,10
                .p2align 3
.L11:
                movsd
                (%rdx), %xmm1
                leaq
                (%r9,%rdi), %rax
                movq
                %r8, %rsi
                .p2align 4,,10
                .p2align 3
.L8:
                movsd
                (%rsi), %xmm0
                addq
                $65536, %rax
                addq            $8,
%rsi
                mulsd           -
65536(%rax), %xmm0
                cmpq
                %rax, %rdi
                addsd
                %xmm0, %xmm1
                jne             .L8
                movsd
                %xmm1, (%rdx)
                addq            $8,
%rdx
                addq            $8,
%rdi
                cmpq
                %rcx, %rdx
                jne
                .L11
                addq
                $65536, %rcx
                addq
                $65536, %r8
                cmpq
                %r11, %rcx
                jne             .L7
.L10:
                leaq
                16(%rsp), %rsi
                xorl
                %edi, %edi
```

```
                movl            $327680, %r8d
                subq            %rdi, %rcx
                movsd           0(%r13), %xmm8
                movq            %r15, %rsi
                movsd           8(%r13), %xmm7
                subq            %rdi, %rbp
                movsd           16(%r13), %xmm6
                subq            %rdi, %rbx
                movsd           24(%r13), %xmm5
                subq            %rdi, %r11
                movsd           32(%r13), %xmm4
                subq            %rdi, %r10
                movsd           40(%r13), %xmm3
                subq            %rdi, %r9
                movsd           48(%r13), %xmm2
                subq            %rdi, %r8
                movsd           56(%r13), %xmm1
                .p2align 4,,10
                .p2align 3
.L9:
                movsd           (%rdx), %xmm0
                leaq            (%rdi,%rcx), %rax
                addq            $8, %rdx
                movsd           (%rsi), %xmm9
                addq            $8, %rcx
                addq            $65536, %rsi
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm8
                movsd           (%rax,%rbp), %xmm9
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm7
                movsd           -8(%rcx), %xmm9
                cmpq            %r12, %rdx
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm6
                movsd           (%rax,%rbx), %xmm9
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm5
                movsd           (%rax,%r11), %xmm9
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm4
                movsd           (%rax,%r10), %xmm9
                mulsd           %xmm0, %xmm9
                addsd           %xmm9, %xmm3
                movsd           (%rax,%r9), %xmm9
                mulsd           %xmm0, %xmm9
                mulsd           (%rax,%r8), %xmm0
                addsd           %xmm9, %xmm2
                addsd           %xmm0, %xmm1
                jne             .L9
                movsd           %xmm8, 0(%r13)
                addl            $8, %r14d
                addq            $64, %r13
                movsd           %xmm7, -56(%r13)
                addq            $64, %r15
                subq            $524288, %rdi
                movsd           %xmm6, -48(%r13)
                movsd           %xmm5, -40(%r13)
                movsd           %xmm4, -32(%r13)
                movsd           %xmm3, -24(%r13)
                movsd           %xmm2, -16(%r13)
                movsd           %xmm1, -8(%r13)
                cmpl            %r14d, 24(%rsp)
                ja              .L12
                addq            $65536, 8(%rsp)
                addq            $65536, 16(%rsp)
                movq            8(%rsp), %rax
                cmpq            %rax, 32(%rsp)
                jne             .L8
```

```
                                        .L11:
                                                    leaq            64(%rsp), %rsi
                                                    xorl            %edi, %edi
```

pmm-secuencial-modificado_b2.s

```
                    xorl            %esi, %esi
                    cmpl            %ebx, %esi
                    je              .L28
.L10:
                    leal            1(%rsi), %r8d
                    cmpl            %r8d, %ebx
                    jbe             .L9
                    movq            %rsi, %rdi
                    movl            %r8d, %edx
                    salq            $13, %rdi
                    .p2align 4,,10
                    .p2align 3
.L8:
                    movslq          %edx, %rax
                    addl            $1, %edx
                    leaq            (%rdi,%rax), %rcx
                    salq            $13, %rax
                    addq            %rsi, %rax
                    cmpl            %edx, %ebx
                    movsd           M2(,%rcx,8), %xmm0
                    movsd           %xmm0, M2(,%rax,8)
                    ja              .L8
.L9:
                    movslq          %r8d, %rsi
                    cmpl            %ebx, %esi
                    jne             .L10
.L28:
                    testl           %ebx, %ebx
                    je              .L12
                    movl            %ebp, %eax
                    xorl            %esi, %esi
                    leaq            1(%rax), %r8
                    leaq            8(,%rax,8), %r9
                    movq            %r8, %r10
                    salq            $3, %r8
                    salq            $16, %r10
.L15:
                    leaq            M1(%rsi), %rdx
                    leaq            M1(%rsi,%r9), %rcx
                    xorl            %edi, %edi
                    .p2align 4,,10
                    .p2align 3
.L16:
                    movsd           (%rdx), %xmm1
                    xorl            %eax, %eax
                    .p2align 4,,10
                    .p2align 3
.L13:
                    movsd           M2(%rdi,%rax), %xmm0
                    mulsd           %xmm1, %xmm0
                    addsd           M3(%rsi,%rax), %xmm0
                    movsd           %xmm0, M3(%rsi,%rax)
                    addq            $8, %rax
                    cmpq            %rax, %r8
                    jne             .L13
                    addq            $8, %rdx
                    addq            $65536, %rdi
                    cmpq            %rcx, %rdx
                    jne             .L16
                    addq            $65536, %rsi
                    cmpq            %rsi, %r10
```

```
                jne             .L15
.L12:
                leaq            16(%rsp), %rsi
                xorl            %edi, %edi
```

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: `figura1-original.c`

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// figura1-original.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;

    srand(time(NULL));

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

    clock_gettime(CLOCK_REALTIME,&cgt1);

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

    printf("Tiempo(seg.):%11.9f / / R[0]=%i, R[39999]=%i\n", ncgt,
R[0], R[39999]);
    return 0;
```

```
}
```

1.1. MODIFICACIONES REALIZADAS (al menos dos modificaciones):

Modificación a) –Localidad accesos: Utilización de los siguientes arreglos para mejorar la optimización.

1. Unión de ambos bucles for en uno solo para mejorar la localidad de acceso a memoria.
2. Utilización del operador ?/: en lugar de if/else, para reducir el número de instrucciones de salto.

Modificación b) –Localidad accesos + Desenrollado-: Utilización de los dos arreglos anteriores, más la utilización de desenrollado del bucle, iteración i, en bloques de 5 (múltiplo de las iteraciones).

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1**-modificado_a.c**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// figura1-modificado-a.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;

    srand(time(NULL));

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

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (ii = 1; 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] = ( X1 < X2 ) ? X1 : X2;
    }

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

    printf("Tiempo(seg.):%11.9f / / R[0]=%i, R[39999]=%i\n", ncgt,
R[0], R[39999]);
    return 0;
}
```

b) figura1**-modificado_b.c**

(ADJUNTAR CÓDIGO FUENTE AL .ZIP)

```
// 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;

    srand(time(NULL));

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

    clock_gettime(CLOCK_REALTIME,&cgt1);

    for (ii = 1; ii <= 40000; ii++)
    {
        X1 = 0; X2 = 0;

        for (i = 0; i < 5000; i+=5){

           X1 += 2 * s[i].a + ii;
           X1 += 2 * s[i+1].a + ii;
```

```
            X1 += 2 * s[i+2].a + ii;
            X1 += 2 * s[i+3].a + ii;
            X1 += 2 * s[i+4].a + ii;
            X2 += 3 * s[i].b - ii;
            X2 += 3 * s[i+1].b - ii;
            X2 += 3 * s[i+2].b - ii;
            X2 += 3 * s[i+3].b - ii;
            X2 += 3 * s[i+4].b - ii;
        }

        R[ii] = ( X1 < X2 ) ? X1 : X2;
    }

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

    printf("Tiempo(seg.):%11.9f / / R[0]=%i, R[39999]=%i\n", ncgt,
R[0], R[39999]);
    return 0;
}
```

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

```
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./figura1-original
Tiempo(seg.):0.238554693 / / R[0]=0, R[39999]=-1351828066
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./figura1-modifica
do-a
Tiempo(seg.):0.179428811 / / R[0]=0, R[39999]=-1458964565
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./figura1-modifica
do-b
Tiempo(seg.):0.132804392 / / R[0]=0, R[39999]=-2123666158
```

1.1. TIEMPOS:

Modificación	**-O2**
Sin modificar	0.238554693
Modificación a)	0.179428811
Modificación b)	0.132804392

1.1. COMENTARIOS SOBRE LOS RESULTADOS:

Como podemos observar de forma evidente, los arreglos realizados al código original, aumentan la velocidad de los programas.

Es claro, como ya vimos en el ejercicio de las matrices, que el desenrollado de bucles es muy buena idea para optimizar el código, por lo que la modificación `b`, será mejor que la `a`, ya que además de los cambios de `a`, aplica el desenrollado.

Sería curioso comprobar, si el programa se optimiza mejor aplicando los dos cambios de la modificación `a` o solo el desenrollado de bucles (dejando el doble bucle `for` y el `if else`).

```
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./figura1-modifica
do-a
Tiempo(seg.):0.178411230 / / R[0]=0, R[39999]=-777080020
danibolanos@Aspire-E5-575G:~/Escritorio/Codigo_AC_4$ ./figura1-modifica
do-c
Tiempo(seg.):0.143301279 / / R[0]=0, R[39999]=-438768682
```

Como podemos ver, la modificación con desenrollado del bucle sola, es incluso mejor que la que contiene los dos primeros arreglos.

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.c

```
                leaq            36(%rsp),
%r8
                movl            $1, %ecx
                .p2align 4,,10
                .p2align 3
.L3:
                movl            $s, %eax
                xorl            %esi, %esi
                .p2align 4,,10
                .p2align 3
.L4:
                movl            (%rax), %edx
                addq            $8, %rax
                leal            (%rcx,
%rdx,2), %edx
                addl            %edx, %esi
                cmpq            $s+40000,
%rax
                jne             .L4
                movl            $s+4, %eax
                xorl            %edi, %edi
                .p2align 4,,10
                .p2align 3
.L5:
                movl            (%rax), %edx
                addq            $8, %rax
                leal            (%rdx,
%rdx,2), %edx
                subl            %ecx, %edx
                addl            %edx, %edi
                cmpq            $s+40004,
%rax
                jne             .L5
                cmpl            %edi, %esi
                jge             .L6
                movl            %esi, (%r8)
.L7:
                addl            $1, %ecx
                addq            $4, %r8
                cmpl            $40001, %ecx
                jne             .L3
                leaq            16(%rsp),
%rsi
                xorl            %edi, %edi
```

figura1-modificado-a.c

```
                leaq            36(%rsp), %r8
                movl            $1, %esi
                .p2align 4,,10
                .p2align 3
.L3:
                movl            $s, %eax
                xorl            %edi, %edi
                xorl            %ecx, %ecx
                .p2align 4,,10
                .p2align 3
.L4:
                movl            (%rax), %edx
                addq            $8, %rax
                leal            (%rsi,
%rdx,2), %edx
                addl            %edx, %ecx
                movl            -4(%rax),
%edx
                leal            (%rdx,
%rdx,2), %edx
                subl            %esi, %edx
                addl            %edx, %edi
                cmpq            %rax, %rbx
                jne             .L4
                cmpl            %edi, %ecx
                cmovg           %edi, %ecx
                addl            $1, %esi
                addq            $4, %r8
                movl            %ecx, -4(%r8)
                cmpl            $40001, %esi
                jne             .L3
                leaq            16(%rsp),
%rsi
                xorl            %edi, %edi
```

figura1-modificado-b.c

```
                leaq            36(%rsp), %r11
                movl            $1, %r8d
                .p2align 4,,10
                .p2align 3
.L3:
                movl            $s, %eax
                xorl            %edx, %edx
                xorl            %r9d, %r9d
                .p2align 4,,10
                .p2align 3
.L4:
                movl            (%rax), %ecx
                addq            $40, %rax
                leal            (%r8,%rcx,2), %r10d
                movl            -32(%rax), %ecx
                addl            %r10d, %r9d
                leal            (%r8,%rcx,2), %ecx
                addl            %ecx, %r9d
                movl            -24(%rax), %ecx
                leal            (%r8,%rcx,2), %ecx
                addl            %ecx, %r9d
                movl            -16(%rax), %ecx
                leal            (%r8,%rcx,2), %ecx
                addl            %ecx, %r9d
                movl            -8(%rax), %ecx
                leal            (%r8,%rcx,2), %ecx
                addl            %ecx, %r9d
                movl            -36(%rax), %ecx
                leal            (%rcx,%rcx,2), %esi
                subl            %r8d, %esi
                leal            (%rsi,%rdx), %edi
                movl            -28(%rax), %edx
                leal            (%rdx,%rdx,2), %ecx
                movl            -20(%rax), %edx
                subl            %r8d, %ecx
                leal            (%rdx,%rdx,2), %edx
                leal            (%rcx,%rdi), %esi
                subl            %r8d, %edx
                leal            (%rdx,%rsi), %ecx
                movl            -12(%rax), %edx
                leal            (%rdx,%rdx,2), %edx
                subl            %r8d, %edx
                addl            %ecx, %edx
                movl            -4(%rax), %ecx
                leal            (%rcx,%rcx,2), %edi
                subl            %r8d, %edi
                addl            %edi, %edx
                cmpq            %rax, %rbx
                jne             .L4
                cmpl            %edx, %r9d
                cmovg           %edx, %r9d
                addl            $1, %r8d
                addq            $4, %r11
                movl            %r9d, -4(%r11)
                cmpl            $40001, %r8d
                jne             .L3
                leaq            16(%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)

```
// daxpy.c

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

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

   if(argc < 2){
      printf("Falta tamaño de vector\n");
      exit(1);
   }

   srand(time(NULL));

   unsigned int  n = atoi(argv[1]);
   int k = rand();

   double *x, *y;
   y = (double*) malloc(n*sizeof(double));
   x = (double*) malloc(n*sizeof(double));

   struct timespec cgt1,cgt2;

   for(int i=0; i<n; i++){
      x[i] = i+1;
      y[i] = i+3;
   }

   clock_gettime(CLOCK_REALTIME,&cgt1);

   for(int i=0; i<n; i++) y[i] += k*x[i]; //rutina denominada DAXPY

   clock_gettime(CLOCK_REALTIME,&cgt2);

   double ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double)
```

```
((cgt2.tv_nsec-cgt1.tv_nsec)/(1.e+9));

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

   free(x);
   free(y);

   return 0;
}
```

Tiempos ejec.	-O0	-O2	-O3
	0.163560002	0.100635916	0.064010977

CAPTURAS DE PANTALLA:

COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR:

He implementado el programa dejando la constante como un número aleatorio y he realizado la media de dos mediciones para el valor de 2^{25} componentes.

La optimización O2 es mejor que la O0, como podemos comprobar en los ensambladores, además de ser casi la mitad de longitud uno que otro, la optimización O2 simplifica muchas operaciones de suma `addsd` y `lea` en `mulsd`, que como ya vimos en algunos casos puede ser ineficiente, pero al ser tan corto el programa, lo optimiza bien. También podemos notar que O0 abusa de los registros de pila mientras que O2, usa los registros del computador.

La optimización O3 es mejor en este caso que la O2, ya vimos en clase, que la versión O2 siempre es mejor que la O0, pero la O3 puede empeorar, no es el caso. El compilador ha realizado un desenrollado del bucle para optimizar la versión O3 y parece que lo ha hecho respecto a la versión 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	daxpyO2.s

movl $0, -84(%rbp)
jmp .L5
.L6:
movl -84(%rbp), %eax
	pxor %xmm1, %xmm1
	xorl %eax, %eax
	cvtsi2sd %r12d, %xmm1

```
                cltq
                leaq            0(,%rax,8),
%rdx
                movq            -72(%rbp),
%rax
                addq            %rdx, %rax
                movl            -84(%rbp),
%edx
                movslq          %edx, %rdx
                leaq            0(,%rdx,8),
%rcx
                movq            -72(%rbp),
%rdx
                addq            %rcx, %rdx
                movsd           (%rdx), %xmm1
                pxor            %xmm0, %xmm0
                cvtsi2sd        -76(%rbp),
%xmm0
                movl            -84(%rbp),
%edx
                movslq          %edx, %rdx
                leaq            0(,%rdx,8),
%rcx
                movq            -64(%rbp),
%rdx
                addq            %rcx, %rdx
                movsd           (%rdx), %xmm2
                mulsd           %xmm2, %xmm0
                addsd           %xmm1, %xmm0
                movsd           %xmm0, (%rax)
                addl            $1, -84(%rbp)
.L5:
                movl            -84(%rbp),
%eax
                cmpl            -80(%rbp),
%eax
                jb              .L6
                leaq            -32(%rbp),
%rax
                movq            %rax, %rsi
                movl            $0, %edi
                .p2align 4,,10
                .p2align 3
.L5:
                movsd
                0(%r13,%rax,8), %xmm0
                mulsd           %xmm1,
%xmm0
                addsd           (%rbx,
%rax,8), %xmm0
                movsd           %xmm0,
(%rbx,%rax,8)
                addq            $1, %rax
                cmpl            %eax, %r14d
                ja              .L5
.L6:
                leaq            16(%rsp),
%rsi
                xorl            %edi, %edi
```

daxpy03.s

```
                movq            %rbp, %rax
                pxor            %xmm1, %xmm1
                salq            $60, %rax
                movl            %ebx, %edx
                shrq            $63, %rax
                cmpl            %ebx, %eax
                cvtsi2sd        %r13d, %xmm1
                cmova           %ebx, %eax
                cmpl            $3, %ebx
                ja              .L51
.L20:
                movsd           (%r14), %xmm0
                cmpl            $1, %edx
                mulsd           %xmm1, %xmm0
                addsd           0(%rbp), %xmm0
                movsd           %xmm0, 0(%rbp)
                je              .L31
                movsd           8(%r14), %xmm0
                cmpl            $3, %edx
                mulsd           %xmm1, %xmm0
                addsd           8(%rbp), %xmm0
                movsd           %xmm0, 8(%rbp)
                jne             .L32
```

```
                movsd           16(%r14), %xmm0
                movl            $3, %ecx
                mulsd           %xmm1, %xmm0
                addsd           16(%rbp), %xmm0
                movsd           %xmm0, 16(%rbp)
.L14:
                cmpl            %edx, %ebx
                je              .L19
                subl            %edx, %ebx
                movl            %edx, %edi
                leal            -2(%rbx), %esi
                shrl            %esi
                addl            $1, %esi
                cmpl            $1, %ebx
                leal            (%rsi,%rsi), %r9d
                jne             .L22
.L16:
                movslq          %ecx, %rcx
                mulsd           (%r14,%rcx,8), %xmm1
                leaq            0(%rbp,%rcx,8), %rax
                addsd           (%rax), %xmm1
                movsd           %xmm1, (%rax)
.L19:
                leaq            32(%rsp), %rsi
                xorl            %edi, %edi
```

2.2 Ejercicio Extra.

He obtenido los tiempos del daxpy para valores entre 512 a 262144, para el programa con mejor tiempo, el de optimización -O3.

```
Tiempo(seg.):0.006401555 / / y[0]=501908833.000000, y
[1048575]=526289554374658.000000
Tiempo(seg.):0.006534478 / / y[0]=501908833.000000, y
[2097151]=1052579108749314.000000
Tiempo(seg.):0.012669475 / / y[0]=501908833.000000, y
[4194303]=2105158217498626.000000
Tiempo(seg.):0.025129284 / / y[0]=501908833.000000, y
[8388607]=4210316434997250.000000
Tiempo(seg.):0.050473680 / / y[0]=501908833.000000, y
[16777215]=8420632869994498.000000
Tiempo(seg.):0.107837769 / / y[0]=501908833.000000, y
[33554431]=16841265739988994.000000
Tiempo(seg.):0.215217759 / / y[0]=1273826332.000000, y
[67108863]=85485037939589120.000000
Tiempo(seg.):0.403330032 / / y[0]=1273826332.000000, y
[134217727]=170970075879178240.000000
Tiempo(seg.):0.806931361 / / y[0]=966849074.000000, y
[268435455]=259536571525496832.000000
```

R = Nº instrucciones coma flotantes*Componentes / T_{CPU} x 10^9

Componentes	Tiempo	R
512	0,000004875	0,2100512821
1024	0,000005357	0,3823035281
2048	0,000010264	0,3990646921
4096	0,00002037	0,4021600393
8192	0,000026941	0,6081437215
16384	0,000055888	0,5863154881
32768	0,000083693	0,7830523461
65536	0,000166429	0,7875550535
131072	0,000284437	0,9216241206
262144	0,000562497	0,9320725266

$\mathbf{R_{max}}$ = 0,9320725266
$\mathbf{N_{max}}$ = 262144
$\mathbf{R_{max}/2}$ = 0,9320725266/2= 0,4660362
$\mathbf{N_{1/2}}$ = **4096**