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): Juan Manuel Salcedo Serrano
Grupo de prácticas: D1
Fecha de entrega: 25/5/2017

Denominación de marca del chip de procesamiento o procesador : *GenuineIntel*

Sistema operativo utilizado: *Deepin Linux 15.3 Desktop*

Versión de gcc utilizada: *gcc (Debian 6.2.0-11) 6.2.0 20161103*

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`

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

void main(int argc,char **argv) {
               if(argc < 2) {
                              fprintf(stderr,"falta el número de filas/columnas\n");
                              exit(-1);
               }
               int N=atoi(argv[1]), i, j, k;
               struct timespec cgt1,cgt2;
               double ncgt;
               int **matriz;//matriz resultante
               int **matriz1;
               int **matriz2;
               //reservar memoria
               matriz = (int **)malloc (N*sizeof(int *));
               for(i=0; i<N; i++)
                              matriz[i] = (int *)malloc (N*sizeof(int));

               matriz1 = (int **)malloc (N*sizeof(int *));
               for(i=0; i<N; i++)
                              matriz1[i] = (int *)malloc (N*sizeof(int));
```

```
                matriz2 = (int **)malloc (N*sizeof(int *));
                for(i=0; i<N; i++)
                                matriz2[i] = (int *)malloc (N*sizeof(int));

                //inicializamos la matriz
                for(i=0; i<N; i++)
                                for(j=0; j<N; j++)
                                                matriz[i][j] = 0;

                for(i=0; i<N; i++)
                                for(j=0; j<N; j++)
                                                matriz1[i][j] = i+1;

                for(i=0; i<N; i++)
                                for(j=0; j<N; j++)
                                                matriz2[i][j] = 1;//para facilitar la
comprobacion

                //multiplicacion
                clock_gettime(CLOCK_REALTIME,&cgt1);
                for (i = 0; i < N; ++i) {
          for (j = 0; j < N; ++j) {
            for (k = 0; k < N; ++k) {
                matriz[i][j] = matriz[i][j] + matriz1[i][k] * matriz2[k][j];
            }
          }
        }
                clock_gettime(CLOCK_REALTIME,&cgt2);
                ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));

                //mostrar resultado
                if(N<10){
                                printf("matriz1:\n");
                                for(i=0; i<N; i++){
                                                for(j=0; j<N; j++)
                                                                printf("%d ",matriz1[i][j]);
                                                printf("\n");
                                }
                                printf("matriz2:\n");
                                for(i=0; i<N; i++){
                                                for(j=0; j<N; j++)
                                                                printf("%d ",matriz2[i][j]);
                                                printf("\n");
                                }
                                printf("\nResultado de multiplicación:\n");
                                for(i=0; i<N; i++){
                                                for(j=0; j<N; j++)
                                                                printf("%d ",matriz[i][j]);
                                                printf("\n");
                                }
                }else{
                                printf("matriz[0][0]:%d\n",matriz[0][0]);
                                printf("matriz[N-1][N-1]:%d\n",matriz[N-1][N-1]);
                }
                printf("\nTiempo en ejecutar matriz1xmatriz2:%8.6f\n",ncgt);

                //libera memoria
                for(i=0; i<N; i++)
                  free(matriz1[i]);
                free(matriz1);
                for(i=0; i<N; i++)
                  free(matriz2[i]);
                free(matriz2);
                for(i=0; i<N; i++)
                  free(matriz[i]);
                free(matriz);
}
```

```
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ gcc -O2 -o pmmsec pmm-secuencial.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./pmmsec 500 500
matriz[0][0]:500
matriz[N-1][N-1]:250000

Tiempo en ejecutar matriz1xmatriz2:0.175694
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$
```

1.1. MODIFICACIONES REALIZADAS :

Modificación a): He desenrollado teniendo en cuenta los valores que puede tomar N para evitar cores. Baja mejora.

Modificación b): He utilizado un algoritmo de multiplicación por bloques con subbloques de N/8 ya que eran los que mejores resultados daban

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) `pmm-secuencial-modificadoA.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main(int argc,char **argv) {
		if(argc < 2) {
				fprintf(stderr,"falta el número de filas/columnas\n");
				exit(-1);
		}
		int N=atoi(argv[1]), i, j, k;
		struct timespec cgt1,cgt2;
		double ncgt;
		int **matriz;//matriz resultante
		int **matriz1;
		int **matriz2;
		//reservar memoria
		matriz = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz[i] = (int *)malloc (N*sizeof(int));
		matriz1 = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz1[i] = (int *)malloc (N*sizeof(int));
		matriz2 = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz2[i] = (int *)malloc (N*sizeof(int));
		//inicializamos la matriz
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz[i][j] = 0;
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz1[i][j] = i+1;
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz2[i][j] = 1;//para facilitar la
comprobacion
		//multiplicacion
		if(N%6 == 0){
				clock_gettime(CLOCK_REALTIME,&cgt1);
				for (i = 0; i < N; ++i) {
					for (j = 0; j < N; ++j) {
					  for (k = 0; k < N; k= k +6) {
						matriz[i][j] = matriz[i][j] + matriz1[i][k] *
matriz2[k][j];
						matriz[i][j] = matriz[i][j] + matriz1[i][k+1] *
matriz2[k+1][j];
						matriz[i][j] = matriz[i][j] + matriz1[i][k+2] *
matriz2[k+2][j];
						matriz[i][j] = matriz[i][j] + matriz1[i][k+3] *
```

```
matriz2[k+3][j];
                                    matriz[i][j] = matriz[i][j] + matriz1[i][k+4] *
matriz2[k+4][j];
                                    matriz[i][j] = matriz[i][j] + matriz1[i][k+5] *
matriz2[k+5][j];
                                  }
                                }
                              }
                              clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                else if(N%5 == 0){
                              clock_gettime(CLOCK_REALTIME,&cgt1);
                              for (i = 0; i < N; ++i) {
                                    for (j = 0; j < N; ++j) {
                                      for (k = 0; k < N; k= k +5) {
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k] *
matriz2[k][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+1] *
matriz2[k+1][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+2] *
matriz2[k+2][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+3] *
matriz2[k+3][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+4] *
matriz2[k+4][j];
                                      }
                                    }
                              }
                              clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                else if(N%4 == 0){
                              clock_gettime(CLOCK_REALTIME,&cgt1);
                              for (i = 0; i < N; ++i) {
                                    for (j = 0; j < N; ++j) {
                                      for (k = 0; k < N; k= k +4) {
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k] *
matriz2[k][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+1] *
matriz2[k+1][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+2] *
matriz2[k+2][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+3] *
matriz2[k+3][j];
                                       }
                                    }
                              }
                              clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                else if(N%3 == 0){
                              clock_gettime(CLOCK_REALTIME,&cgt1);
                              for (i = 0; i < N; ++i) {
                                    for (j = 0; j < N; ++j) {
                                      for (k = 0; k < N; k= k +3) {
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k] *
matriz2[k][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+1] *
matriz2[k+1][j];
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k+2] *
matriz2[k+2][j];
                                      }
                                    }
                              }
                              clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                else if(N%2 == 0){
                              clock_gettime(CLOCK_REALTIME,&cgt1);
                              for (i = 0; i < N; ++i) {
                                    for (j = 0; j < N; ++j) {
                                      for (k = 0; k < N; k= k +2) {
                                         matriz[i][j] = matriz[i][j] + matriz1[i][k] *
```

```
matriz2[k][j];
                                        matriz[i][j] = matriz[i][j] + matriz1[i][k+1] *
matriz2[k+1][j];
                                    }
                                }
                            }
                            clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                else{
                            clock_gettime(CLOCK_REALTIME,&cgt1);
                            for (i = 0; i < N; ++i) {
                                  for (j = 0; j < N; ++j) {
                                    for (k = 0; k < N; k++) {
                                        matriz[i][j] = matriz[i][j] + matriz1[i][k] *
matriz2[k][j];
                                    }
                                }
                            }
                            clock_gettime(CLOCK_REALTIME,&cgt2);
                }
                printf("Ultimo valor de i para N=%d: %d\n\n", N, i-1);
                ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-cgt1.tv_nsec)/
(1.e+9));

                //mostrar resultado
                if(N<10){
                            printf("matriz1:\n");
                            for(i=0; i<N; i++){
                                        for(j=0; j<N; j++)
                                                    printf("%d ",matriz1[i][j]);
                                        printf("\n");
                            }
                            printf("matriz2:\n");
                            for(i=0; i<N; i++){
                                        for(j=0; j<N; j++)
                                                    printf("%d ",matriz2[i][j]);
                                        printf("\n");
                            }
                            printf("\nResultado de multiplicación:\n");
                            for(i=0; i<N; i++){
                                        for(j=0; j<N; j++)
                                                    printf("%d ",matriz[i][j]);
                                        printf("\n");
                            }
                }else{
                            printf("matriz[0][0]:%d\n",matriz[0][0]);
                            printf("matriz[N-1][N-1]:%d\n",matriz[N-1][N-1]);
                }
                printf("\nTiempo en ejecutar matriz1xmatriz2:%8.6f\n",ncgt);

                //libera memoria
                for(i=0; i<N; i++)
                  free(matriz1[i]);
                free(matriz1);
                for(i=0; i<N; i++)
                  free(matriz2[i]);
                free(matriz2);
                for(i=0; i<N; i++)
                  free(matriz[i]);
                free(matriz);
}
```

Capturas de pantalla:

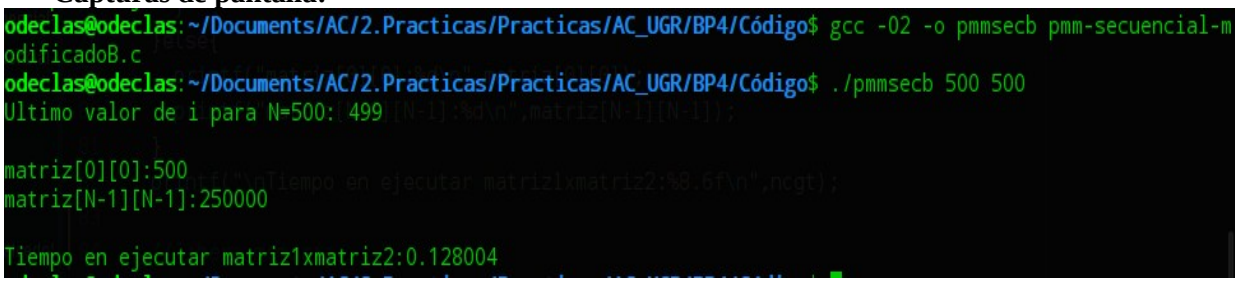
```
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ gcc -O2 -o pmmsecA pmm-secuencial-m
odificadoA.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./pmmsecA 500 500
Ultimo valor de i para N=500: 499

matriz[0][0]:500
matriz[N-1][N-1]:250000

Tiempo en ejecutar matriz1xmatriz2:0.159462
```

b) pmm-secuencial-modificadoA.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main(int argc,char **argv) {
		if(argc < 2) {
				fprintf(stderr,"falta el número de filas/columnas\n");
				exit(-1);
		}
		int N=atoi(argv[1]), i, j, k;
		struct timespec cgt1,cgt2;
		double ncgt;
		int **matriz;//matriz resultante
		int **matriz1;
		int **matriz2;
		int jj, kk, temp, s;
		//reservar memoria
		matriz = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz[i] = (int *)malloc (N*sizeof(int));
		matriz1 = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz1[i] = (int *)malloc (N*sizeof(int));
		matriz2 = (int **)malloc (N*sizeof(int *));
		for(i=0; i<N; i++)
				matriz2[i] = (int *)malloc (N*sizeof(int));

		//inicializamos la matriz
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz[i][j] = 0;
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz1[i][j] = i+1;
		for(i=0; i<N; i++)
				for(j=0; j<N; j++)
						matriz2[i][j] = 1;//para facilitar la
comprobacion

		//multiplicacion
		s=N/8;
				clock_gettime(CLOCK_REALTIME,&cgt1);
		for( jj=0;jj<N;jj+= s){
				for( kk=0;kk<N;kk+= s){
						for( i=0;i<N;i++){
								for( j = jj; j<((jj+s)>N?N:(jj+s)); j++){
										temp = 0;
										for( k = kk; k<((kk+s)>N?N:
(kk+s)); k++){
												temp += matriz1[i]
[k]*matriz2[k][j];
										}
										matriz[i][j] += temp;
								}
						}
				}
		}
		clock_gettime(CLOCK_REALTIME,&cgt2);
		printf("Ultimo valor de i para N=%d: %d\n\n", N, i-1);
		ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.tv_nsec-
cgt1.tv_nsec)/(1.e+9));

		//mostrar resultado
		if(N<10){
				printf("matriz1:\n");
				for(i=0; i<N; i++){
						for(j=0; j<N; j++)
								printf("%d ",matriz1[i][j]);
						printf("\n");
```

```
                                }
                                printf("matriz2:\n");
                                for(i=0; i<N; i++){
                                        for(j=0; j<N; j++)
                                                printf("%d ",matriz2[i][j]);
                                        printf("\n");
                                }
                                printf("\nResultado de multiplicación:\n");
                                for(i=0; i<N; i++){
                                        for(j=0; j<N; j++)
                                                printf("%d ",matriz[i][j]);
                                        printf("\n");
                                }
                        }else{
                                printf("matriz[0][0]:%d\n",matriz[0][0]);
                                printf("matriz[N-1][N-1]:%d\n",matriz[N-1][N-1]);
                        }
                        printf("\nTiempo en ejecutar matriz1xmatriz2:%8.6f\n",ncgt);

                        //libera memoria
                        for(i=0; i<N; i++)
                          free(matriz1[i]);
                        free(matriz1);
                        for(i=0; i<N; i++)
                          free(matriz2[i]);
                        free(matriz2);
                        for(i=0; i<N; i++)
                          free(matriz[i]);
                        free(matriz);
}
```

Capturas de pantalla:



1.1. TIEMPOS:

Modificación	-Sin modificar	Modificación a)	Modificación b)
-O2	0.175694	0.159462	0.128004

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Tamaño 500x500

B) CÓDIGO FIGURA 1:

CÓDIGO FUENTE: figura1-original.c

```
#include <time.h>
#include <stdio.h>
struct {
            int a;
            int b;
}s[5000];

int main(){
            struct timespec cgt1,cgt2; double ncgt;
            int ii, i, X1, X2;
            int R[40000];

            for(ii=0; ii < 5000; ii++){
                        s[ii].a = ii+1;
                        s[ii].b = ii+2;
            }
            //..
            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);
            ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-
            cgt1.tv_nsec)/(1.e+9));
            printf("Tiempo(seg.):%11.9f\t\n",ncgt);
            //..
}
```

```
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ gcc -O2 -o fig figura1.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./fig 100
Tiempo(seg.):0.000000254
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./fig 1000
Tiempo(seg.):0.000000206
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./fig 10000
Tiempo(seg.):0.000000397
```

1.1. MODIFICACIONES REALIZADAS:

Modificación a): Se fusiona en uno los 2 bucles que rellenaban X1 y X2.

Modificación b): Se desenrolla el bucle obtenido anteriormente.

1.1. CÓDIGOS FUENTE MODIFICACIONES

a) figura1-modificadoa.c

```
#include <time.h>
#include <stdio.h>
struct {
            int a;
            int b;
}s[5000];

int main(){
            struct timespec cgt1,cgt2; double ncgt;
            int ii, i, X1, X2;
            int R[40000];

            for(ii=0; ii < 5000; ii++){
                        s[ii].a = ii+1;
                        s[ii].b = ii+2;
            }
```

```
                //..
                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;
/*_mm_prefetch(s[ii+10],_MM_HINT_T0);*/}
                                                if (X1<X2)     R[ii]=X1;
                                                else           R[ii]=X2;
                }
                clock_gettime(CLOCK_REALTIME,&cgt2);
                ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-
                cgt1.tv_nsec)/(1.e+9));
                printf("Tiempo(seg.):%11.9f\t\n",ncgt);
                //..
}
```

Capturas de pantalla :

```
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ gcc -O2 -o figa figura1-modificadoa.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figa 100
Tiempo(seg.):0.000000481
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figa 1000
Tiempo(seg.):0.000000333
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figa 10000
Tiempo(seg.):0.000000313
```

b) figura1-modificadob.c

```
#include <time.h>
#include <stdio.h>
#include <xmmintrin.h>
struct {
                int a;
                int b;
}s[5000];

int main(){
                struct timespec cgt1,cgt2; double ncgt;
                int ii, i, X1, X2;
                int R[40000];

                for(ii=0; ii < 5000; ii++){
                                s[ii].a = ii+1;
                                s[ii].b = ii+2;
                }
                //..
                clock_gettime(CLOCK_REALTIME,&cgt1);
                for (ii=1; ii<=40000;ii++) {
                                X1=0; X2=0;
                                for(i=0; i<4998;i=i+4){
                                                X1= X1 + 2*s[i].a+ii + 2*s[i+1].a+ii +
2*s[i+2].a+ii + 2*s[i+3].a+ii;
                                                X2= X2 + 3*s[i].b-ii + 3*s[i+1].b-ii +
3*s[i+2].b-ii + 3*s[i+3].b-ii;
                                }
                                                if (X1<X2)     R[ii]=X1;
                                                else R[ii]=X2;
                }
                clock_gettime(CLOCK_REALTIME,&cgt2);
                ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+ (double) ((cgt2.tv_nsec-
                cgt1.tv_nsec)/(1.e+9));
                printf("Tiempo(seg.):%11.9f\t\n",ncgt);

                //..
}
```

Capturas de pantalla :

```
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ gcc -O2 -o figb figura1-modificadob.c
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figb 100
Tiempo(seg.):0.000000381
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figb 1000
Tiempo(seg.):0.000000137
odeclas@odeclas:~/Documents/AC/2.Practicas/Practicas/AC_UGR/BP4/Código$ ./figb 10000
Tiempo(seg.):0.000000486
```

1.1. TIEMPOS:

Modificación	Sin modificar	Modificación a)	Modificación b)
-O2	0.000000397	0.000000313	0.000000486

1.1. COMENTARIOS SOBRE LOS RESULTADOS: Tamaño 10000

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.

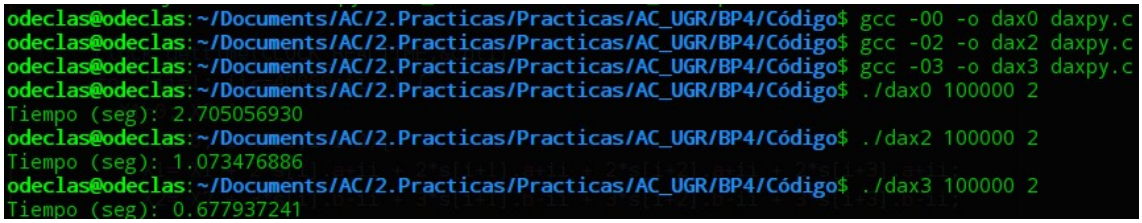
CÓDIGO FUENTE: `daxpy.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 16000
#define BILLION  1000000000L;
int main(int argc, char **argv){
              if(argc < 3) {
                            fprintf(stderr,"Sintaxis de ejecucion: ./daxpy <Tam_vector>
<Constante_multiplicativa>\n");
              exit(-1);
              }
              struct timespec start, stop;
              double accum;

              int tam_vector=atoi(argv[1]);
              int constante=atoi(argv[2]);
              int sumador[tam_vector];
              int multiplicador[tam_vector];

              int i;
              int j;
              int t;
              clock_gettime(CLOCK_REALTIME, &start);
              for(t=0;t<5000;t++){
                            for(i=0;i<tam_vector;i++){
                                          sumador[i]=1;
                                          multiplicador[i]=1;
                            }
                            for(i=0;i<tam_vector;i++){
                                          sumador[i]=constante*multiplicador[i]
+sumador[i];
                            }
              }
```

```
                clock_gettime(CLOCK_REALTIME, &stop);
                accum = ( stop.tv_sec - start.tv_sec ) + (double)( stop.tv_nsec -
start.tv_nsec ) / (double)BILLION;
                printf("Tiempo (seg): %11.9f\n", accum);
}
```

Tiempos ejec.	-O0	-O2	-O3
	2.705056930	1.073476886	0.677937241

CAPTURAS DE PANTALLA:



COMENTARIOS SOBRE LAS DIFERENCIAS EN ENSAMBLADOR: Al avanzar la optimizacion la cantidad de codigo ensamblador gcc disminuye. Tambien el numero de instrucciones de salto y comparación.